Efficient Retrieval for Browsing Large Image Databases*

Daniel Wu Divyakant Agrawal Amr El Abbadi
Ambuj Singh Terrence R. Smith
{danielw, agrawal, amr, ambuj, smithtr}@cs.ucsb.edu
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

The management of large image databases poses several interesting and challenging problems. These problems range from ingesting the data and extracting meta-data to the efficient storage and retrieval of the data. Of particular interest are the retrieval methods and user interactions with an image database during browsing. In image databases, the response to a given query is not an exact well-defined set, rather, the user poses a query and expects a set of responses that should contain many possible candidates from which the user chooses the answer set. We first present the browsing model in Alexandria, a digital library for maps and satellite images. Designed for content-based retrieval, the relevant information in an image is encoded in the form of a multi-dimensional feature vector. Various techniques have been previously proposed for the efficient retrieval of such vectors by reducing the dimensionality of such vectors. We show that for even moderately large databases (in fact, only 1856 texture images), these approaches do not scale well for exact retrieval. However, as a browsing tool, these dimensionality reduction techniques hold much promise.

1 Introduction

The advent of image databases poses a new set of problems and challenges. Unlike traditional relational databases which offer elegant theoretical formulations and reductions, image databases are not so amenable. They have to be approached as an engineering effort. Among the many difficulties encountered in image databases are: massive storage requirements, reasonable response times, and practical query formulations.

Traditional databases view data as a set of records with fields that can be processed and queried. Queries are formulated by specifying these field values. Eg: *Locate an employee between the age of 45–55 whose salary is greater than $55,000* (to be targeted for early retirement). The fields *age* and *salary* drive the query. In image databases, however, the record is the actual image itself. Processing queries such as *Locate a photograph of a river in Colorado with trees nearby* (to scout for a fishing hole) presents a formidable challenge because the data is not organized in a form in which one can simply specify the field values for retrieval. In the absence of such an organization,

*Work supported by a research grant from NSF/ARPA/NASA IRI9411330.

there is no exact means to determine if the image is in the database or not. The user must browse the database to locate an image of interest.

Some databases organize their data based upon the content of the images, and permit fuzzy searches for an image by describing general properties of the image. For example, the research project QBIC at IBM Almaden provides users with the ability to formulate queries based on colors, shapes, and sketches against a database of color photographs [NBE+93, FBF+94]. The Digital Library project at UC Berkeley takes a similar approach to catalog photographic collections of California Department of Water resources [OS95].

These are examples of *similarity queries* or *query-by-example*: The user specifies a sample image and initiates a search to retrieve similar images from the database. With a good user interface, the user can browse the database by clicking on images of further interest, following the similarity theme of these images in the same manner as hypertext links.

The Alexandria project at UC Santa Barbara has been building its digital library of map and image data with the goal of supporting a wide range of similarity queries. The current implementation provides similarity queries for texture data [MM96]. The user selects from a library of standard textures, and issues a query for images containing similar textures. The many applications for this technology abound. Scientists may want to retrieve images with specific topological features for modeling purposes. Real estate developers can search territories for vacant lots and building sites.

These queries can be implemented either by having a human being (notably a domain expert) read each image and manually catalog the information in the image, or by applying automated image processing techniques. The manual approach in addition to being tedious and error prone (it is very easy to misspell or mislabel information in the photograph) is too time-consuming. Images from a variety of sources come in every day; a team of full-time cataloguers would not be able to keep up with the stream of data. Furthermore, if a new method of cataloguing were introduced, every image would have to be re-examined. The cost of such a task is too prohibitively expensive to be considered. For these reasons, the automated process is preferred.

When images are entered into the database, image processing and texture analysis is performed to extract the salient features and characteristics of each image. These features are summarized in a multi-dimensional feature vector and stored in the database. Using a feature vector to represent an image vastly simplifies the problem of how to query for the image. Since the feature vector is just a tuple of values, we can once again perform a query by specifying the component field values. To query for an image, the database query is first translated into a specification for a feature vector; once the feature vector is located in the database, the corresponding image can also be retrieved and delivered to the user.

This automated scheme also supports similarity queries. When image processing is performed to extract a feature vector from an image, or equivalently, to transform each image to a point in a multi-dimensional space, the transformations are chosen so that similar images are represented as points that are close to each other. Often the Euclidean distance is used as a measure of “closeness” or “similarity” but other measures may also be used. When a user queries for similar images, this translates into retrieving nearby points in the multi-dimensional space.

Although this technique reduces the information content from an entire image to a relatively small number of features, this number is still too large for efficient indexing and browsing of large image databases. For example, if the original image were $1k \times 1k$ pixels, the information in this image can be summarized in a vector with 48 features. Though the compression of information is quite spectacular from an image processing perspective, searching 48-dimension feature vectors is still far from trivial from a database indexing perspective.

When the image collection is in the order of a hundred to a thousand, sequential similarity matching of feature vectors may be acceptable. However, as we move towards image databases containing millions of images, new techniques are required to index and browse such databases. In this paper, we explore mathematical techniques that further reduce the dimensionality of the search space. Our approach is motivated primarily by the need to support browsing, rather than exact searches of image databases.

The paper is organized as follows: In Section 2, we describe the types of queries an image database supports. Section 3 introduces the problem of indexing a large image database, and proposes a solution using Discrete Fourier Transform and Singular Value Decomposition; we provide a brief mathematical treatment to show how DFT and SVD can be applied. In Section 4, we describe the criteria used to evaluate database queries for accuracy and correctness. We then describe our experimental design in Section 5, and present our results.

2 Similarity Queries of Feature Vectors

The dataset used in this paper is based on the collection of textures in the Brodatz album [Bro66]. This collection was originally used as a testbed to test the accuracy of the texture identification system in Alexandria. The work of [MM96] enabled feature extraction form this data. Each image from the Brodatz album was partitioned into 16 subimages. A Gabor transform was performed on each subimage at different scales and orientations. The result of this processing is a set of 48 feature measurements, represented as a 48-dimension feature vector, or as a point in a 48-dimensional space.

Using the geometric model of representing an image in the database as a feature vector in some n-dimensional space, the types of queries that we make also have geometric interpretation. We consider two types of similarity queries: *nearest-neighbor* queries and *epsilon*-queries [AFS93].

A nearest-neighbor query takes two arguments, an image specification (I) and the number of nearest neighbors (M) to retrieve M closest images of the specified image in the n-dimensional space.

$$GetNearestNeighbors(ImageSpecification\, I,\ int\, M)$$

The image specification is translated into a representative feature vector to compare against all other feature vectors in the database; the N closest neighbor vectors are returned by this query.

An epsilon-query takes as its arguments, an image specification (I) and a radius (R) and returns all images whose feature vectors are within the n-dimensional sphere (neighborhood) centered at the query image with radius R.

$$GetEpsilonNeighbors(ImageSpecification\, I,\ float\, R)$$

As before, the image specification is translated into a feature vector, and compared against other vectors in the database.

On the surface, it appears that nearest-neighbor queries are more appropriate for browsing. The user specifies the number of nearest neighbors M, and the database retrieves M closest vectors. With epsilon-queries, the user has to somehow specify a radius R. While R may have some semantic interpretation in image processing, this notion is not easily conveyed to the user. For simplicity, the user can just consider R to be some sort of "adjustable knob" that controls the number of images retrieved. The number of images that epsilon-queries retrieve is based on R. If R is too small, no images are returned; if R is too large the entire image database is returned. One advantage of epsilon-queries is that they are easier to implement in indexing structures such as an $R^\star$-tree. Recent papers, though, have described implementations of nearest-neighbor searches in $R^\star$-trees [RKV95].

In the browsing model that we have adopted [NBE+93, OS95], the user begins by specifying a sample image. For databases which store synthetic patterns, this sample image can be drawn on screen using a graphical user interface. For more complex databases storing detailed photographs, sample photographs and images may be loaded into the system; the user can modify and alter these images using an array of image-processing tools. Next, a feature vector, known as the query vector, is generated from this image specification. The database uses the query vector in nearest-neighbor or epsilon-queries as a reference from which to retrieve other feature vectors with similar content. Images corresponding to feature vectors of nearest geometric proximity are then displayed on screen as low resolution thumbnail images. On a workstation display the number of thumbnails that fit on screen is about 10–20. After reviewing the thumbnails, the interested user can retrieve the full resolution image associated with each thumbnail, if so desired.

3 Efficient Indexing and Retrieval of Feature Vectors

The texture image database for Alexandria currently holds 280,000 feature vectors and images, with long-term plans to store several millions. Due to the growing number of feature vectors in the database, the problem of how to efficiently retrieve these vectors must be addressed. A search structure such as an $R^\star$-tree [BKSS90] is fully capable of indexing feature vectors and other multi-dimensional data, but its performance degrades with increase in the number of dimensions [FRM94, AMAM93]. Feature vectors with number of dimensions 24 would not perform well using an $R^\star$-tree. We must examine other approaches to solving this problem.

One approach which we will explore is to reduce the dimension of the feature vectors. If each image can be represented by a smaller-dimension feature vector in an $R^\star$-tree, say 10 dimensions, then performance improves and the task of indexing millions of feature vectors is greatly reduced. The drawback to this scheme is that 2 feature vectors that are very close may be reduced to the same lower-dimension feature vector; this means we would have to provide some arbitration scheme to differentiate between the 2 vectors. We will overlook this problem for now as the likelihood of 2 vectors collapsing into 1 is very remote, especially since their components are real-valued.

To achieve this reduction in the dimension of the feature vectors, we consider two different compression techniques using the Discrete Fourier Transform (DFT) and the Singular Value Decomposition (SVD). These techniques have been used to reduce the dimensionality of information in several projects [NBE+93, PF94, FRM94, FL94]. For time-series data such as stock market prices, one may obtain a good approximation to the data by retaining only its low-frequency Fourier coefficients. That is, as an indexing strategy, it suffices to examine just the first few Fourier coefficients of the data to index to it and locate it in some indexing structure. Likewise SVD (or the equivalent K-L transform) may be applied to the data. By selecting the first few largest singular values, we can achieve a similar reduction in the dimension of each feature vector.

We will study both of these techniques as we seek to browse a database, and quickly present the user with a set of candidate images. The computational aspects of these techniques have been evaluated in prior studies but no comparison has been made to evaluate the relative advantages and disadvantages of these two popular techniques for image databases.

3.1 Discrete Fourier Transform

Following the approach taken by [FRM94], we begin by studying the properties of the DFT. The n-point Discrete Fourier Transform of a finite-length real or complex–valued sequence $\vec{x} = \{x_i\}_{i=0}^{n-1}$ is a sequence $\vec{X}$ of complex numbers $X_i, i = 0, \ldots, n-1$ given by:

$$X_i = 1/\sqrt{n} \sum_{i=0}^{n-1} x_i e^{-j2\pi/n}$$

$i = 0, \ldots, n-1$.

In our discussion above, we have been dealing with feature vectors, but it is just as easy to represent each vector of length n as a finite-length sequence of length n.

The norm of a complex-valued sequence $\vec{X} = \{X_i\}_{i=0}^{n-1}$ is defined as:

$$\|\vec{X}\| = \sqrt{\sum_{i=0}^{n-1} |x_i|^2}$$

A basic result from harmonic analysis is Parseval's Theorem [AFS93]: Given a sequence $\vec{x}$ and its Fourier transform $\vec{X}$ their norms must coincide:

$$\|\vec{x}\| = \|\vec{X}\|.$$

Equivalently,

$$\sum_{i=0}^{n-1} |x_i|^2 = \sum_{i=0}^{n-1} |X_i|^2$$

If we truncate the sequence $\vec{X}$ in the frequency domain to N terms, to form $\vec{X}_N$, then for $N \leq n$,

$$\|\vec{X}_N\|^2 = \sum_{i=0}^{N-1} |X_i|^2 \leq \sum_{i=0}^{n-1} |X_i|^2 = \|\vec{X}\|^2$$

so

$$\|\vec{X}_N\| \leq \|\vec{X}\|.$$

Applying this result, if we truncate the 2 feature vectors $\vec{U}$ and $\vec{V}$ to $\vec{U}_N$ and $\vec{V}_N$ by taking the first N components, then the distance between $\vec{U}_N$ and $\vec{V}_N$ is less than or equal to the distance between $\vec{u}$ and $\vec{v}$.

$$\|\vec{U}_N - \vec{V}_N\| \leq \|\vec{u} - \vec{v}\|.$$

The distances between the truncated vectors in the transformed domain *shrink* when compared to their dimensions in the original domain.

If we perform an epsilon-query to retrieve all the vectors $\{\vec{v}^1, \ldots \vec{v}^m\}$ in a neighborhood of radius r about $\vec{u}$, then the transformed vectors $\{\vec{V}^1, \ldots \vec{V}^m\}$ also lie in a neighborhood of radius r about $\vec{U}$. We can thus perform the same epsilon-query in a lower dimension feature vector space. Truncating the transformed vectors to N components equates to projecting each vector $\{\vec{V}^1, \ldots \vec{V}^m\}$ from an n-dimensional feature vector space to an N-dimensional space.

The advantage of doing this lies in the fact that indexing lower dimension feature vectors is much more efficient that indexing the full dimension feature vectors. The disadvantage is that while the truncated vectors $\{\vec{V}_N^1, \ldots \vec{V}_N^m\}$ will still lie in the neighborhood of radius r about $\vec{U}_N$, many other vectors may now lie in that neighborhood as well. Since distances between vectors shrink when they are projected to a lower dimensional space, there may be other vectors $\{\vec{Z}^j\}$, $j = 1 \ldots l$, such that $\|\vec{Z}^j - \vec{U}\| > r$, but $\|\vec{Z}_N^j - \vec{U}_N\| \leq r$. These vectors $\{\vec{Z}^j\}$ are the *false hits* of a query. By performing the epsilon-query on a truncated feature vector space we never dismiss an image that we would have retrieved had we performed the epsilon-query in the n-dimensional space, but we may retrieve some extraneous images that would not have been included in the original space (the false hits). By varying the number of vector components we truncate, we can vary the number of false hits that we encounter in an epsilon-query.

For nearest-neighbor queries, this result is not of immediate benefit. A nearest neighbor in the truncated space need not be the nearest neighbor in the full n-dimensional feature vector space.

3.2 Singular Value Decomposition

Suppose we have a set of m feature vectors $\{\vec{x}^1, \ldots, \vec{x}^m\}$ in a database. Collecting these m n-dimensional vectors into an $m \times n$ matrix A, the Singular Value Decomposition of A is given by:

$$A = U \Sigma V^T, \tag{1}$$

where U is an $m \times n$ matrix, and Σ and V are $n \times n$ matrices[Nas79]. In this decomposition, the matrix U comprises a set of column-orthonormal row vectors,

$$UU^T = I_m \qquad U^T U = I_n$$

while V is an orthonormal matrix

$$VV^T = V^T V = I_n$$

The matrix Σ is a diagonal matrix with non-negative elements, called singular values, along its diagonal.

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_n \end{pmatrix}$$

Details on how to to compute the SVD of a matrix are given in [PTVF92]. The important thing to note is that the SVD provides the best least squares fit to a set of data points. If each image can be mapped to a point in n-dimensional space, then the database consists of a collection of points scattered in n-dimensional space. The SVD finds a set of orthonormal vectors $\{\vec{v}^1, \ldots, \vec{v}^n\}$ such that the points are scattered and spread out most widely along the $\vec{v}^1$ direction, then along the $\vec{v}^2$ direction, and so on. Unlike the DFT which always chooses *sin* and *cos* functions for its basis independent of the data, the SVD calculates its orthonormal basis from the input data; the SVD is a data-dependent operation. For random processes, in which the input data are samples of random variables, the SVD has been shown to be an optimal linear estimate for the statistics of these random variables. For this reason, SVD has been widely used in many areas of communications and signal and image processing.

In this decomposition, the matrix V, called the *prediction matrix*, rotates the vectors in A to the directions of the best least squares fit. Since V is orthonormal, we can multiply both sides of equation (1) by V

$$AV = U\Sigma \tag{2}$$

The product $U\Sigma$ now contains a set of vectors $\{\vec{X}^1, \ldots, \vec{X}^m\}$, which are rotated from the original $\{\vec{x}^1, \ldots, \vec{x}^m\}$ vectors. Since they are merely rotated, their lengths are preserved,

$$\|\vec{x}^i\| = \|\vec{X}^i\|$$

for $i = 1, \ldots, m$.

This result is similar to that of the DFT preserving the norms of finite-length sequences. Just as for DFT, we can take advantage of this property to reduce the dimension of the feature vectors yet still make meaningful similarity queries. From (2), we have

$$\begin{pmatrix} \overrightarrow{X^1}^T \\ \vdots \\ \overrightarrow{X^m}^T \end{pmatrix} = \begin{pmatrix} \overrightarrow{U^1}^T \\ \vdots \\ \overrightarrow{U^m}^T \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_n \end{pmatrix}$$

We can reduce the number of singular values from n to N by discarding (or zeroing-out) the least significant singular values in Σ.

$$\Sigma_N = \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_N \end{pmatrix}$$

We also discard the corresponding entries in the A, U, and V matrices. We then have

$$\begin{pmatrix} \overrightarrow{X_N^1}^T \\ \vdots \\ \overrightarrow{X_N^m}^T \end{pmatrix} = \begin{pmatrix} \overrightarrow{U_N^1}^T \\ \vdots \\ \overrightarrow{U_N^m}^T \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_N \end{pmatrix} = U_N \Sigma_N$$

The product $U_N\Sigma_N$ now contains a collection $\{\vec{X}_N^1, \ldots, \vec{X}_N^m\}$, the truncated versions of the original feature vectors rotated to the directions of best least squares fit.

Just as with DFT-transformed vectors, the distances between the $\vec{X}_N^1, \ldots \vec{X}_N^m$ vectors are less than or equal to the corresponding distances between the $\vec{x}^1, \ldots \vec{x}^m$ vectors. That means that for epsilon-queries, we once again have the property that no image is falsely dismissed if we perform the epsilon-query using truncated feature vectors, though we may pick up some extraneous feature vectors.

4 Performance Criteria

After processing the data by performing DFT and SVD to reduce the number of dimensions, we proceed to make nearest-neighbor and epsilon-queries and examine the quality of these results. To determine whether DFT or SVD produces more accurate results from each query, we adopt the following 2 criteria: we will measure the *recall* and *precision* of each query [Sal89].

After the user makes an epsilon-query, the database constructs a reference feature vector $\vec{q}$ based upon the specifications of the query. A similarity query seeks to retrieve all vectors $\vec{x}$ that lie in a neighborhood of radius r about $\vec{q}$. The approach we describe above projects the vectors to a lower-dimension space, and a vector $\vec{X}_N$ is retrieved if it lies in the neighborhood of radius r about $\vec{Q}_N$. We are, however, subject to false hits in this lower-dimension space. We consider a truncated vector $\vec{X}_N$ in the neighborhood about $\vec{Q}_N$ to be *correct* if $\vec{x}$ lies in the neighborhood about $\vec{q}$ in the original n-dimensional space. A similar definition of correctness holds for nearest-neighbor queries.

In the information retrieval literature [Sal89], the concept of *relevant* rather than *correct* image is usually considered. However, the notion of relevance is difficult to ascertain. It involves human testing and cognition, and is very difficult to determine in an automated fashion. For this reason, we consider correctness based upon the geometric property of feature vector similarity, as opposed to relevance, to gauge the performance of DFT and SVD processing.

Suppose a user issues a similarity query and retrieves m images from the database. The number of correct images stored in the database is n. Of these only k satisfy the query and are considered correct. *Recall* is defined as k/n, the proportion of the number of correct results retrieved to the number of correct results stored in the database. *Precision* is defined as k/m, the proportion of the number of correct results retrieved to the number of images retrieved.

Both DFT and SVD processing provide 100% recall for epsilon-queries. Since there are never any false dismissals, all the correct feature vectors must be retrieved. For nearest-neighbor queries, the concepts of recall and precision coincide; since $m = n$ for nearest-neighbor queries, we have $k/n = k/m$.

There is a trade-off between recall and precision when we apply approximation techniques to speed up retrieval times. A high recall will usually have low precision since in order to ensure that all the correct feature vectors are returned, some incorrect ones are included. Likewise, a high precision will have poor recall. By being so selective and requiring that every feature vector in the result set is correct, the query overlooks the more questionable feature vectors though some may be correct. Recall and precision are both important criteria because they measure the accuracy and quality of responses to the user. A query with poor recall means that few correct answers are retrieved. A query with poor precision means that the correct answers are mixed with many more incorrect ones.

5 Experimental Setup

To compare whether truncated SVD components or DFT coefficients provide better retrieval performance, we simulate a series of queries on the 1856 feature vectors, each representing an image. Every vector is processed at the time of insertion into the database with SVD or DFT. It is helpful to discuss SVD-processing of feature vectors before proceeding to DFT-processing.

5.1 SVD Processed Feature Vectors

After performing SVD on matrix A, containing the feature vectors $\{\vec{x}^1, \ldots \vec{x}^m\}$, we obtain the decomposition matrices U, Σ, and V. Most SVD code such as the C routine `svdcmp` in [PTVF92] arrange the singular values of Σ in sorted order; i.e., σ_1 is the largest singular value, σ_2 is the next largest, and so on. This is a very convenient form, because when we vary the number of singular values in our approximation of A, we want to pick the largest singular values for the best approximation. Having the singular values sorted along the diagonal of the Σ facilitates this process.

The number of singular values corresponds to the dimension of the truncated vectors $\{\vec{X}^1_N, \ldots \vec{X}^m_N\}$. If $N = 1$, then we retain the largest singular value σ_1 in Σ so that the truncated vectors have only 1 component, which is the most significant component. Since the components are most-widely spread out in the $\vec{v}^1$ direction, similarity queries benefit because there are fewer other feature vectors in the immediate neighborhood to discern.

Similarly, for $N = 2$, we select the 2 largest singular values σ_1 and σ_2, and truncate the $\{\vec{X}^1, \ldots, \vec{X}^m\}$ vectors to 2 dimensions. The component values are spread out the most widely along the $\vec{v}^1$ direction then along the $\vec{v}^2$ direction. With 2 components instead of 1, the approximation improves and both nearest-neighbor and epsilon-queries improve in accuracy.

Clearly, as N increases we get increasingly more accurate results. When $N = n$, we achieve 100% recall and 100% precision, but no reduction of feature vector dimensions. In our experiments, we will make nearest-neighbor and epsilon-queries over a range of values of N to see how we can reconcile the greater accuracy of large N with the higher efficiency of small N.

5.2 DFT Processed Feature Vectors

For an n-length real sequence, there are n complex DFT coefficients:

$$X_0, X_1, \ldots, X_{n/2-1}, X_{n/2}, X_1^*, \ldots X_{n/2-1}^*$$

with X_0 and $X_{n/2}$ real-valued; the remaining $n-2$ coefficients $\{X_1, \ldots, X_{n/2-1}, X_1^*, \ldots, X_{n/2-1}^*\}$ occur in complex conjugates pairs.

Now we are in a slight quandary. In the case of SVD, for each truncated vector $\vec{Y}^i_N$, we can vary the number of components N from 1 to n. For DFT sequences $\vec{X}^i_N, i = 1, \ldots, m$, we would also like to vary the number of values from 1 to n so we can compare the performance of a truncated SVD feature vector $\vec{Y}^i_N$ to a truncated DFT feature vector $\vec{X}^i_N$ using the same value of components N. However the n DFT coefficients are complex numbers, with a total of $2n-2$ real and imaginary parts; the lengths of truncated SVD and DFT feature vectors would not correspond.

To resolve this matter, we re-formulate the components of each DFT sequence.

$$\begin{aligned} \|\vec{X}\|^2 &= |X_0|^2 + |X_1|^2 + |X_1^*|^2 + \ldots + |X_{n/2}|^2 \\ &= |X_0|^2 + 2|X_1|^2 + \ldots + 2|X_{n/2-1}|^2 + |X_{n/2}|^2 \\ &= |X_0|^2 + |\sqrt{2}X_1|^2 + \ldots + |\sqrt{2}X_{n/2-1}|^2 + |X_{n/2}|^2 \end{aligned}$$

Now we need only consider the n values:

$$\{X_0,\, \sqrt{2}\,Re(X_1),\, \sqrt{2}\,Im(X_1), \ldots, \sqrt{2}\,Re(X_{n/2-1}),\, \sqrt{2}\,Im(X_{n/2-1}),\, X_{n/2}\}$$

We have absorbed each pair of conjugates into a single coefficient by scaling its real and imaginary parts by $\sqrt{2}$. Not only are there now n values so that we can compare $\vec{X}^i_N$ with $\vec{Y}^i_N$, but these n coefficients are still meaningful. The first value X_0 is the DC coefficient; the next 2 are the real and imaginary parts of the first harmonic $\sqrt{2}\,Re(X_1)$, $\sqrt{2}\,Im(X_1)$; the next 2 form the second harmonic $\sqrt{2}\,Re(X_2)$, $\sqrt{2}\,Im(X_2)$, etc. Later, we show how identifying the real and imaginary components of each harmonic can improve the DFT approximation.

In our trials comparing DFT with SVD, we begin by taking $N = 1$, in which $\vec{X}^i_N = (X^i_0)$ and $\vec{Y}^i_N = (Y^i_1)$, to compare the effect of 1 Fourier coefficient with 1 SVD component. Next we take $N = 3$, where $\vec{X}^i_N = (X^i_0, \sqrt{2}\,Re(X^i_1), \sqrt{2}\,Im(X^i_1))$ and $\vec{Y}^i_N = (Y^i_1, Y^i_2, Y^i_3)$ to compare the effect of 2 DFT coefficients with 3 SVD components. We skip over $N = 2$, because we would then be comparing the DC component X^i_0 and only the real part of the first Fourier coefficient $\sqrt{2}\,Re(X^i_1)$, with 2 SVD vector components. We proceed in this manner as we increase N in order to keep intact the harmonic contribution from each Fourier coefficient.

5.3 Performance Evaluations

For each of the 1856 vectors, $\vec{x}^i$, $i = 1, \ldots, 1856$, we simulate an epsilon-query for a specified radius r on $\vec{x}^i$. The query vector $\vec{x}^i$ is processed by SVD or DFT and truncated to the appropriate length. The truncated transformed vector is then compared against every other truncated transformed vector in the database to see which database vectors are within distance of r from $\vec{x}^i_N$. We then check the resulting collection for false hits. From these measurements, we calculate the average precision for queries of radius r from DFT and SVD-processing. Likewise we simulate a nearest-neighbor query for each vector $\vec{x}^i$ for a specified number of neighbors M. After retrieving the response set and measuring false hits, we determine the average recall for M nearest-neighbor queries.

Since our epsilon-queries all have 100% recall, we plot only the average precision. For nearest-neighbor queries, recall and precision are the same, so we list only the average recall. One must be careful not to equate the average precision of an epsilon-query with the average recall (precision) of a nearest-neighbor query. Although we can consider a nearest-neighbor query for m nearest-neighbors as an epsilon-query in which the effective radius r' is the smallest radius containing m closest vectors, this r' is not consistent from feature vector to feature vector; each query vector has a different effective radius r' for its m nearest neighbors. Since we specify the radius r in epsilon-queries, each query vector does share a consistent radius r.

Furthermore, since distances between feature vectors *shrink* when these feature vectors are truncated, the effective radius r'_N corresponding to a query vector's m closest neighbors in a lower dimension is not the same as the effective radius r' for m closest neighbors in the original space. Indeed, $r'_N < r'$. In contrast to epsilon-queries, in which the same radius r is used to process queries in both original and lower dimension spaces, 2 different radii r' and r'_N are used for nearest-neighbor queries. Consequently, it makes no sense to compare the average precision of an epsilon-query to the average recall or precision of a nearest-neighbor query.

Graphs of epsilon-queries and nearest-neighbor queries are presented below. For epsilon-queries, the figures display the average precision over a wide range of radii; graphs for nearest-neighbor queries plot the average recall over a range of nearest-neighbors. The tables beneath each graph highlight data in a range suitable for browsing, which is an answer set consisting of about 20 candidate images. For epsilon-queries, each table lists the radius r, the average number of correct images expected from that query, the average number of images retrieved if DFT or SVD-processing were employed, and the average precision of the query. Tables for nearest-neighbor queries list the number of neighbors, the average number of correct images retrieved when processed with DFT or SVD, and the average recall.

For the time being we ignore the Sorted DFT curves in each graph, and the corresponding DFT* entries in each table. These will be examined later as we consider ways to improve the recall and precision results in the following graphs.

5.3.1 Epsilon Queries

Epsilon-queries measuring the average precision are shown in Figures 1–4. As our results indicate, processing the database with just 1 set of Fourier coefficients and singular values (Figure 1) is not very useful. The curves are nearly identical, but the precision is so poor that no useful information can be derived from an epsilon-query. At radius of 1.5, for example, an average of 4.83 correct images are expected from the query. DFT-processing would deliver 374.80 images yielding a precision of 0.0129, while SVD processing would retrieve 360.02 images at a precision of 0.0134. Since the vast majority of this collection are false hits, the information collected is more noise than actual data.

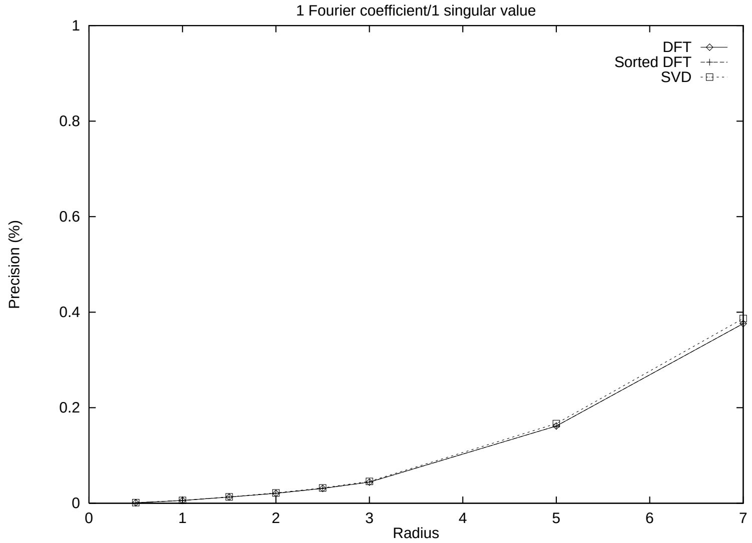
With the addition of more Fourier coefficients and components, the precision improves substantially, with SVD providing higher precision. An interesting phenomena takes place when using 7 components (Figure 2). While the SVD curve continues to be a monotonically increasing function, the DFT curve reaches a local maximum near a radius of 1.5. After radius 1.5, the DFT begins to decline; eventually, the curve must rise again (as the precision approaches 1.0 when the radius is large), but the presence of this local maximum is encouraging. It suggests that truncated DFT vectors share enough common harmonics that they can be grouped together in small clusters of average radius 1.5. Unfortunately, at this radius, DFT retrieves 37.03 images, too large to fit on screen, while SVD retrieves just 13.99.

As we increase the number of components, the maximum becomes more prominent and shifts to radius 2.0 for 17 components (Figure 4). Now DFT retrieves 20.02 images while SVD retrieves 13.95; both lie in the range that can be displayed on screen. The precision at this radius, however, is still higher for SVD (0.7327) than for DFT (0.5104).

5.3.2 Nearest-Neighbor Queries

Nearest-neighbor queries do not guarantee 100% recall, but they provide higher precision than epsilon-queries. As noted above, epsilon-queries maintain the same radius r before and after truncation. False hits arise when distances between the query vector and other extraneous vectors become less than r after truncation. For nearest-neighbor queries, we are interested in retrieving the m nearest neighbors before and after truncation; the distance between each of these m neighbors and the query vector shrinks as the vectors are truncated. The effective radius r'_N of these m nearest neighbors about the query vector after truncation is smaller than the effective radius r' of the untruncated m neighbors. Moreover, when distances among truncated feature vectors shrink, they tend to shrink in a manner such that a truncated vector will retain many of its same nearest neighbors in the lower-dimensional space. Our results bear this out.

Taking just 1 coefficient or component for nearest neighbors, DFT and SVD perform nearly identically in Figure 5. With the addition of 3 more Fourier coefficients or 6 more SVD components (Figure 6), both DFT and SVD make substantial improvements. In the range of interest, 20 nearest-neighbors, SVD's recall (0.7973) is higher than that of DFT (0.6109). SVD retrieves an average of 3.73 more correct images than DFT, a significant advantage in browsing. This advantage in SVD-processing is carried over to the remaining set of components. At 17 components (Figure 8), however, both DFT and SVD have very high recall (.8142 and .9341 respectively) for 20 nearest-neighbors, with SVD returning 2.40 more correct images.



Radius	Ave Correct	Ave Retrieved DFT	DFT*	SVD	Ave Precision DFT	DFT*	SVD
1.0	1.45	253.24	253.24	242.13	0.0057	0.0057	0.0060
1.5	4.83	374.80	374.80	360.02	0.0129	0.0129	0.0134
2.0	10.22	490.97	490.97	474.00	0.0208	0.0208	0.0216
2.5	18.70	604.45	604.45	583.21	0.0309	0.0309	0.0321
3.0	31.56	715.12	715.12	688.44	0.0441	0.0441	0.0458

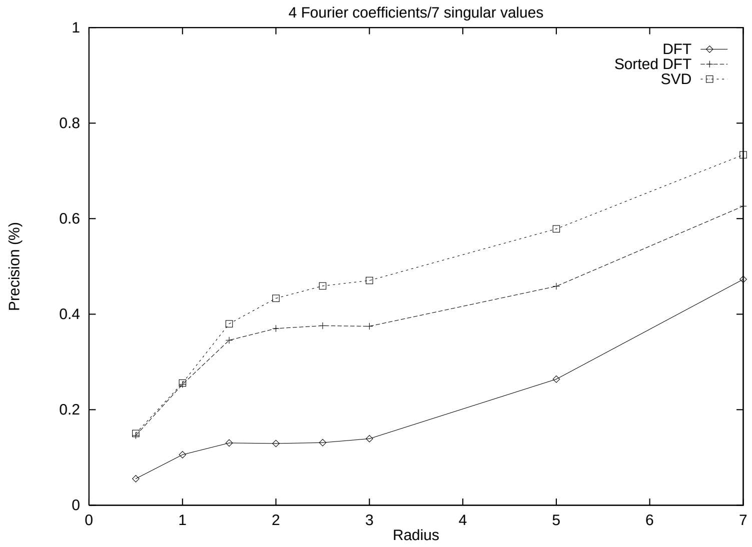
Figure 1: Epsilon-Queries using 1 component

In these graphs (Figures 6–8), we detect a relative maximum for recall of both DFT and SVD-processed queries at 15 nearest neighbors. This observation is consistent with what we would expect. Recall that during feature extraction, each image in the Brodatz album was sectioned into 16 subimages, from which a feature vector for each subimage was generated. Since 16 similar-looking subimages comprise an entire image, it makes sense that we would observe a high recall at 15 nearest neighbors: the 15 nearest neighbors of each feature vector are likely the 15 remaining subimages belonging to that texture pattern image.

As these plots (Figures 6–8) indicate, the recall for SVD-processing is much higher than for DFT-processing, by as much as .2384 (at 11 components) for browsing 20 nearest neighbors.

5.4 Evaluation

From these results, SVD overall outperforms DFT by a considerable margin. In the context of browsing, SVD provides higher precision and greater recall than DFT. There are still a few considerations that weigh in favor of DFT though. Calculating the SVD of m n-dimensional feature vectors requires $O(m\,n^2)$ operations. Taking the DFT of these m vectors also requires $O(m\,n^2)$ operations, but the constant of proportionality is much lower. If n is an even power of 2, the DFT can be calculated using Fast Fourier Transform (FFT) algorithms in $O(mn\,log\,n)$ operations[OS89].



Radius	Ave Correct	Ave Retrieved DFT	DFT*	SVD	Ave Precision DFT	DFT*	SVD
1.0	1.45	13.66	5.72	5.66	0.1060	0.2531	0.2559
1.5	4.83	37.03	13.99	12.72	0.1305	0.3455	0.3798
2.0	10.22	79.00	27.63	23.59	0.1294	0.3700	0.4332
2.5	18.70	142.74	49.75	40.73	0.1310	0.3758	0.4591
3.0	31.56	226.51	84.24	67.08	0.1393	0.3746	0.4704

Figure 2: Epsilon-Queries using 7 components

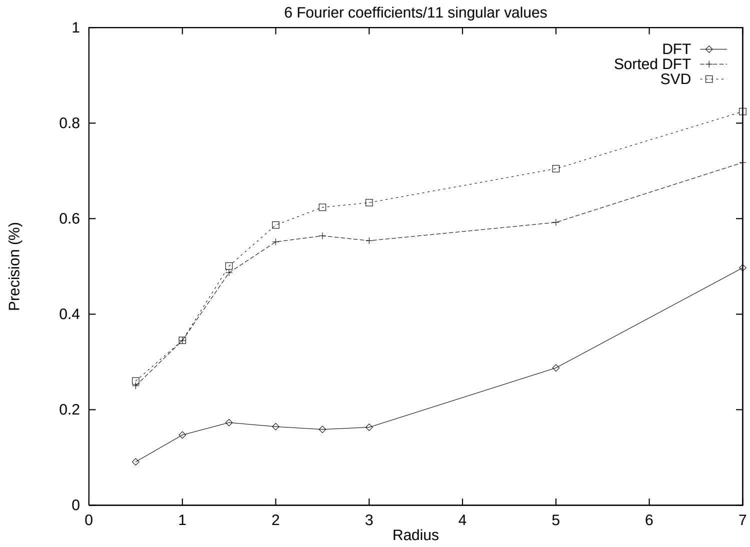
These advantages in efficient computation become very attractive as the number of feature vectors m gets large. With this in mind, we re-examine DFT-processing to see how we can improve its retrieval performance. One important point to consider is whether taking the first few Fourier coefficients is the most effective way of applying DFT. For SVD-processing, we have simply been taking the largest singular values, because these values provide the most contributions in this approximation scheme. For DFT, however, we simply take the coefficients in increasing harmonics.

In the experiments of Faloutsos et al [FRM94], the data consisting of stock market prices was very smooth, very slow-growing, and had very few high frequency harmonics. Consequently, keeping the first few Fourier coefficients provided good results. There is no reason to assume that this property holds for our texture images. Rather than take the first N Fourier coefficients, it is more reasonable to extract the first N *dominant* Fourier coefficients, just as we took the first N largest singular values for SVD.

The plot of the magnitudes of scaled Fourier coefficients

$$\{X_0,\, \sqrt{2}\,Re(X_1),\, \sqrt{2}\,Im(X_1), \ldots, \sqrt{2}\,Re(X_{n/2-1}),\, \sqrt{2}\,Im(X_{n/2-1}),\, X_{n/2}\}$$

in Figure 9 shows that these coefficients are not ordered in monotonically decreasing fashion. The DC coefficient is dominant, followed by the 2nd harmonic coefficient, then the 8th harmonic, the



Radius	Ave Correct	Ave Retrieved DFT	DFT*	SVD	Ave Precision DFT	DFT*	SVD
1.0	1.45	9.84	4.20	4.19	0.1471	0.3448	0.3452
1.5	4.83	27.93	9.93	9.65	0.1730	0.4868	0.5009
2.0	10.22	62.11	18.54	17.42	0.1646	0.5513	0.5866
2.5	18.70	117.69	33.15	29.97	0.1589	0.5640	0.6238
3.0	31.56	193.40	56.98	49.81	0.1632	0.5539	0.6335

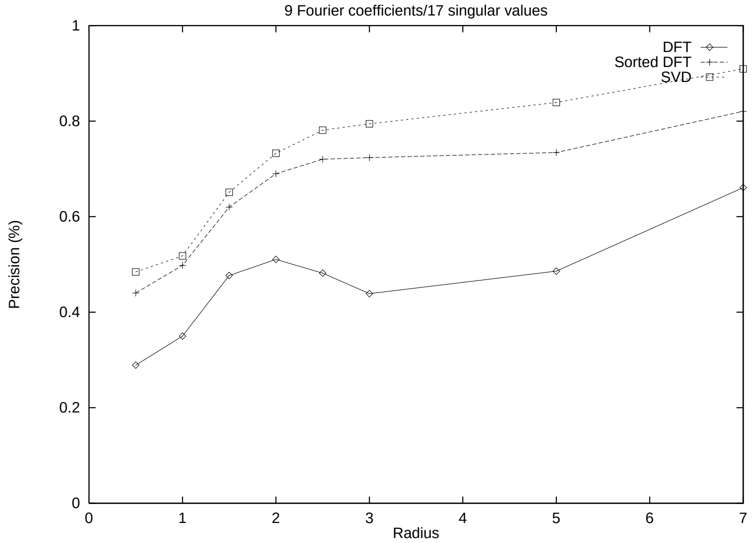
Figure 3: Epsilon-Queries using 11 components

16th harmonic, followed by the rest. Indeed, from this plot, we note that these first 4 dominant coefficients are much larger in magnitude than the others. After the 16th harmonic (magnitude 1.75), the next largest coefficient occurs at the 1st harmonic (magnitude 1.26); after that, the remaining coefficients have magnitudes less than unity. From this graph, it appears that the first 4–5 dominant coefficients, provide the most contribution. Beyond that there is less to gain from taking further Fourier coefficients.

5.4.1 Re-processed Epsilon Queries

Re-examining epsilon-queries (Figures 2–4), we see that selecting the dominant Fourier coefficients has a considerable effect on the precision. The Sorted DFT curves begin to approach that of SVD. In the plot with 11 components, the precision between the SVD and Sorted DFT curves are very close for radius 0.5 to 1.5. At radius 1.0, for example, the precision of SVD exceeds that of DFT by only 0.0004. In terms of images, SVD's slightly better precision amounts to 0.01 fewer images.

In Figures 10 and 11, we generate new plots, fixing the radius at 1.5 and 2.5, but varying the number of components used in the DFT and SVD computation. We note the difference in precision between Sorted DFT and SVD becomes larger as the size of the radius gets larger. That is, Sorted DFT better approximates SVD for small radius sizes than for large radius sizes. This observation



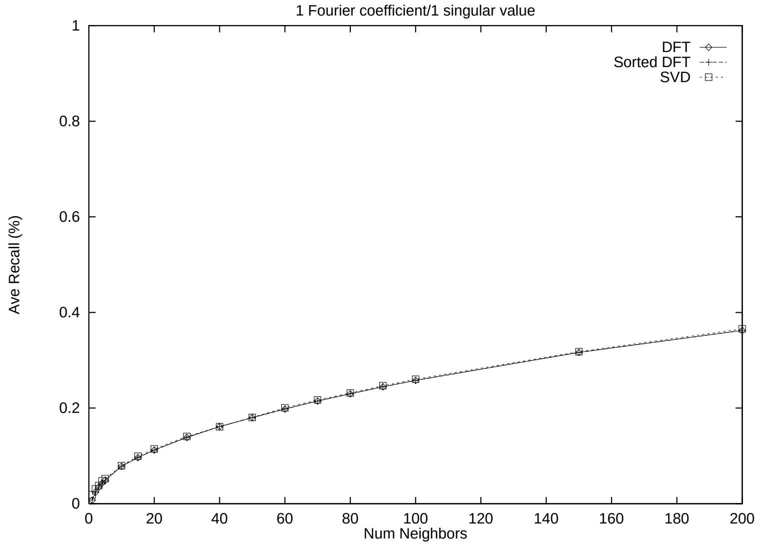
Radius	Ave Correct	Ave Retrieved DFT	DFT*	SVD	Ave Precision DFT	DFT*	SVD
1.0	1.45	4.13	2.91	2.79	0.3500	0.4980	0.5179
1.5	4.83	10.13	7.80	7.42	0.4769	0.6198	0.6510
2.0	10.22	20.02	14.82	13.95	0.5104	0.6898	0.7327
2.5	18.70	38.81	25.96	23.95	0.4818	0.7201	0.7808
3.0	31.56	71.91	43.62	39.73	0.4389	0.7234	0.7942

Figure 4: Epsilon-Queries using 17 components

is in line with general properties of the SVD and DFT. The SVD is calculated using every vector in the database simultaneously; it performs its computations using global aspects of the data. The DFT, on the other hand, is computed on each vector individually; it uses only local aspects of the data. Thus for larger radii, we would expect SVD to yield better performance, as large radii encompass a wider, more global view. We see this effect in Figures 10–11.

From these figures, we see that the difference in precision between Sorted DFT and SVD has a relative minimum using 11 components. At radius 1.5, the difference in precision is .0141; at radius 2.5 the difference is .0598. For radius 2.5, the difference in precision is even smaller for 19 components: .0587. However, as indexing 19 dimensional feature vectors in an R*-tree would not yield good performance, we discard 19 as the choice of number of components to truncate.

Taking 11 components, we note from Figure 3 that a radius of 2.0 yields comparable precisions between SVD (0.5866) and Sorted DFT (0.5513). The average number of images retrieved from SVD is 17.42. Of these 10.22 are correct images. Using Sorted DFT, we retrieve 18.54 images, which readily fit on a CRT. By suffering 1.12 more incorrect images, we achieve a major reduction in computational time required to process these feature vectors and browse the database.



Num Neighbors	Ave Correct			Ave Recall		
	DFT	DFT*	SVD	DFT	DFT*	SVD
1	0.01	0.01	0.02	0.0075	0.0075	0.0189
5	0.25	0.25	0.26	0.0491	0.0491	0.0525
10	0.78	0.78	0.79	0.0782	0.0782	0.0794
15	1.45	1.45	1.49	0.0964	0.0964	0.0994
20	2.24	2.24	2.29	0.1120	0.1120	0.1145
30	4.16	4.16	4.22	0.1386	0.1386	0.1406

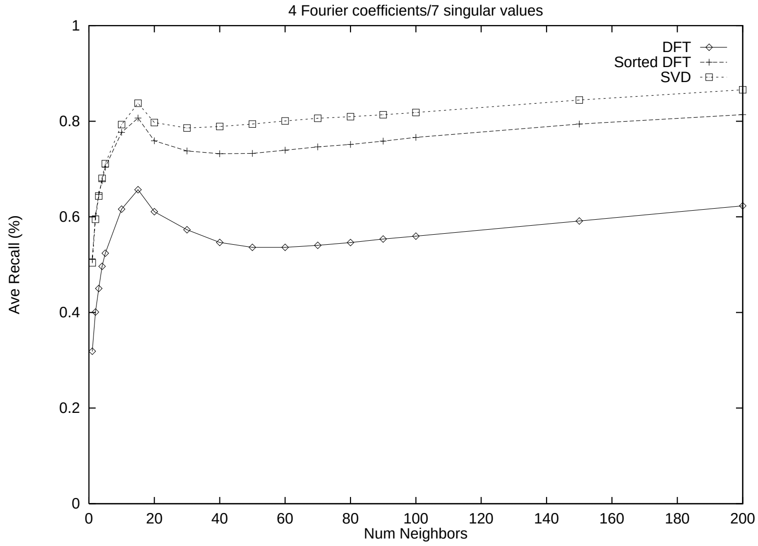
Figure 5: Nearest-Neighbor Queries using 1 component

5.4.2 Re-processed Nearest-Neighbor Queries

Likewise, we re-examine nearest neighbor queries (Figures 6–8) after Sorted DFT-processing. Again we note that the recall performance improves greatly over usual DFT processing. With 11 components, the average recall of Sorted-DFT (0.8640) slightly exceeds that of SVD (.8631) for the first 10 nearest neighbors.

Generating new plots (Figures 12 and 13) by fixing the number of nearest neighbors at 10 and 20, but varying the number of components, we again see that Sorted DFT's recall performance comes closest to SVD by taking 11 components or 6 Fourier coefficients. As the number of nearest neighbors increases from 10 to 20, SVD's recall gradually overtakes that of DFT, but the difference in recall between SVD and Sorted DFT still achieves a relative minimum at 11 components.

From Figure 7, retrieving 10 nearest neighbors to fit on screen yields, on the average, 17.18 correct images for Sorted DFT and 17.69 for SVD. Now we only expect 0.51 fewer correct images from DFT-processing than from SVD.



Num Neighbors	Ave Correct			Ave Recall		
	DFT	DFT*	SVD	DFT	DFT*	SVD
1	0.32	0.51	0.50	0.3190	0.5124	0.5038
5	2.62	3.52	3.56	0.5239	0.7039	0.7112
10	6.16	7.77	7.93	0.6161	0.7767	0.7931
15	9.85	12.10	12.56	0.6568	0.8066	0.8376
20	12.22	15.18	15.95	0.6109	0.7590	0.7973
30	17.19	22.14	23.57	0.5730	0.7379	0.7858

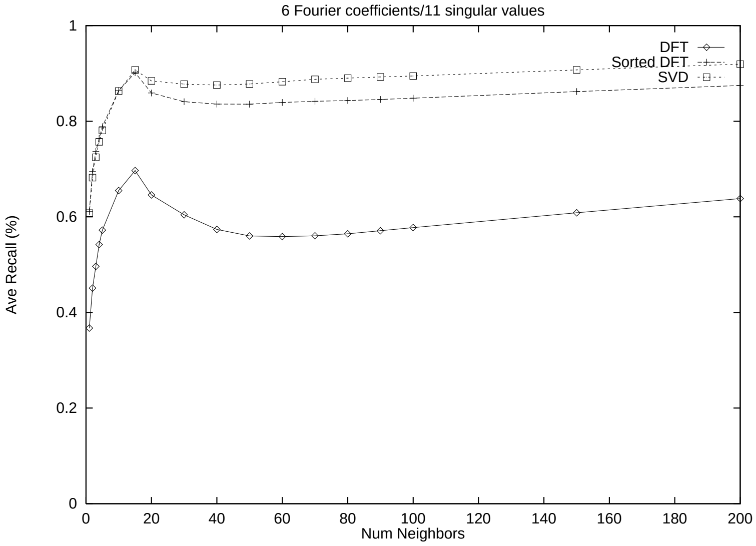
Figure 6: Nearest-Neighbor Queries using 7 components

6 Concluding Remarks

SVD is optimal in the sense that it provides the best linear least squares error to data; it outperforms DFT when we query for a large number of neighbors or take a large radius since SVD determines the optimal directions from a global view of the data.

In querying a database, a more local view of data often suffices. We seek to retrieve only those images that are closest to the specification; images that are distant and far away are of little benefit. Due to the level of difficulty formulating an exact query, we often must resort to browsing a database, to gain familiarity with its contents as well as to explore related material. When we browse, though, we are limited by the number of images we can display on screen, and by the number we can visually process. In browsing a database we want to retrieve a limited number of the most accurate images. It is in this setting that DFT compares favorably with SVD.

In this paper, we illustrated how well the retrieval performance of DFT approaches that of SVD; by re-arranging the Fourier coefficients and taking the dominant harmonics in decreasing order, we see a great improvements in both recall and precision. After taking just 11 coefficients, that is, compressing feature vectors from 48 dimensions down to 11 dimensions, we are able to



Num Neighbors	Ave Correct			Ave Recall		
	DFT	DFT*	SVD	DFT	DFT*	SVD
1	0.37	0.61	0.61	0.3675	0.6110	0.6078
5	2.86	3.94	3.90	0.5722	0.7886	0.7806
10	6.55	8.64	8.63	0.6549	0.8640	0.8631
15	10.45	13.52	13.61	0.6968	0.9014	0.9074
20	12.42	17.18	17.69	0.6459	0.8590	0.8843
30	18.13	25.22	26.33	0.6044	0.8408	0.8777

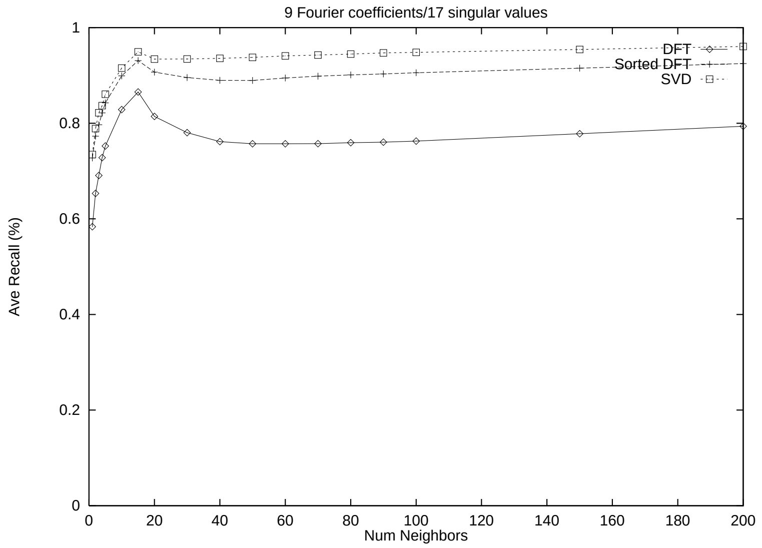
Figure 7: Nearest-Neighbor Queries using 11 components

achieve a recall of .8590 for 20 nearest neighbors. Though the recall for SVD is slightly higher, there are benefits to using the ubiquitous DFT or its very fast implementation, the FFT. Not only are hardware implementations of FFT chips commonplace, but with the advent of multi-processors, the FFT can be parallelized for extremely fast calculations. This, in addition to the competitive recall and precision of DFT makes the DFT a promising technique to consider for browsing large image databases.

References

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms. 4th Intl. Conf.*, pages 69–84, 1993.

[AMAM93] A. D. Alexandrov, W. Y. Ma, A. El Abbadi, and B. S. Manjunath. Adaptive filtering and indexing for image databases. In *Proc. of the SPIE Int. Conf. on Storage and*



Num Neighbors	Ave Correct			Ave Recall		
	DFT	DFT*	SVD	DFT	DFT*	SVD
1	0.58	0.73	0.73	0.5835	0.7274	0.7344
5	3.76	4.21	4.30	0.7524	0.8427	0.8696
10	8.28	8.99	9.15	0.8284	0.8991	0.9154
15	12.98	13.97	14.24	0.8654	0.9310	0.9494
20	16.28	18.14	18.68	0.8142	0.9070	0.9341
30	23.41	26.86	28.03	0.7803	0.8955	0.9343

Figure 8: Nearest-Neighbor Queries using 17 components

Retrieval for Image and Video Databases - III, pages 12–23, San Jose, CA, February 5993.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangle. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 1990.

[Bro66] P. Brodatz. *Textures: A Photographic Album for Artists & Designers.* Dover, New York, New York, 1966.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. In *Journal of Intelligent Information Systems*, volume 3, pages 231–262, 1994.

[FL94] C. Faloutsos and K. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. Technical Report CS-TR-3383, Univ. of Maryland Institute for Advanced Computer Studies, January 1994.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, Minneapolis, May 1994.

[MM96] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, November 1996.

[Nas79] J. C. Nash. *Compact Numerical Methods for Computers: linear algebra and function minimisation.* Adam Hilger Ltd, 1979.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, February 1993.

[OS89] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Inc., 1989.

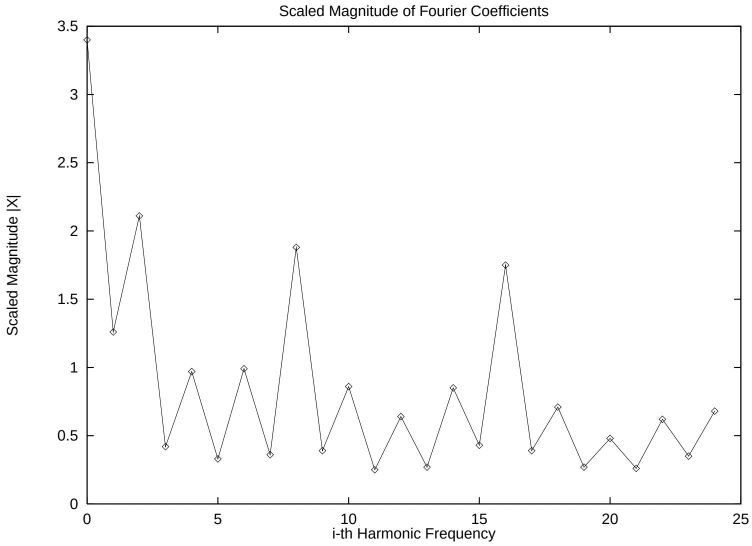
[OS95] V. E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, pages 41–48, September 1995.

[PF94] E. Petrakis and C. Faloutsos. Similarity searching in large image databases. Technical Report CS-TR-3388, Univ. of Maryland Institute for Advanced Computer Studies, December 1994.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, 2nd Ed.* Cambridge University Press, 1992.

[RKV95] N. Roussooulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, May 1995.

[Sal89] G. Salton. *Automatic Text Processing.* Addison-Wesley, 1989.



i-th Harmonic	Scaled $\lvert X_i \rvert$	i-th Harmonic	Scaled $\lvert X_i \rvert$
0	3.40	—	—
1	1.26	2	2.11
3	0.42	4	0.97
5	0.33	6	0.99
7	0.36	8	1.88
9	0.39	10	0.86
11	0.25	12	0.64
13	0.27	14	0.85
15	0.43	16	1.75
17	0.39	18	0.71
19	0.27	20	0.48
21	0.26	22	0.62
23	0.35	24	0.68

Figure 9: Scaled Magnitude of each Fourier Coefficient

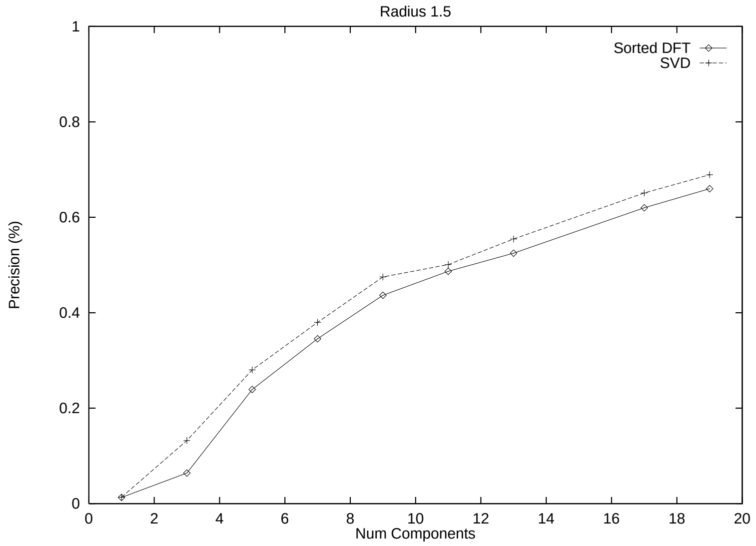


Figure 10: Epsilon-Queries at radius 1.5

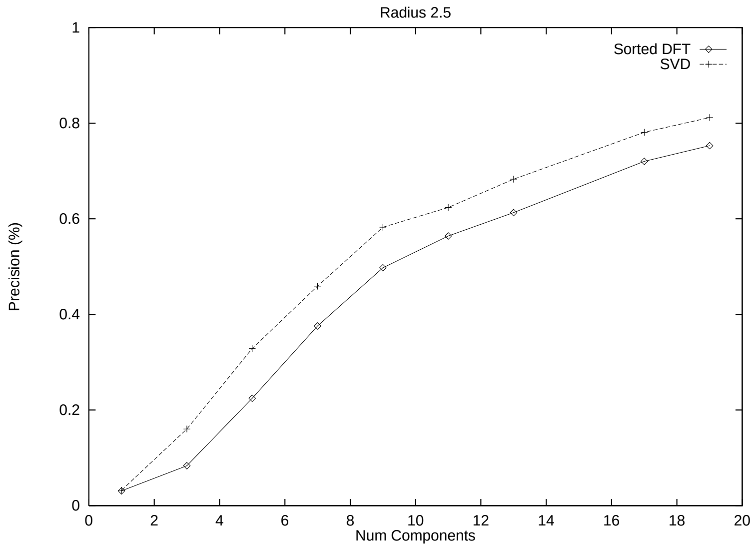


Figure 11: Epsilon-Queries at radius 2.5

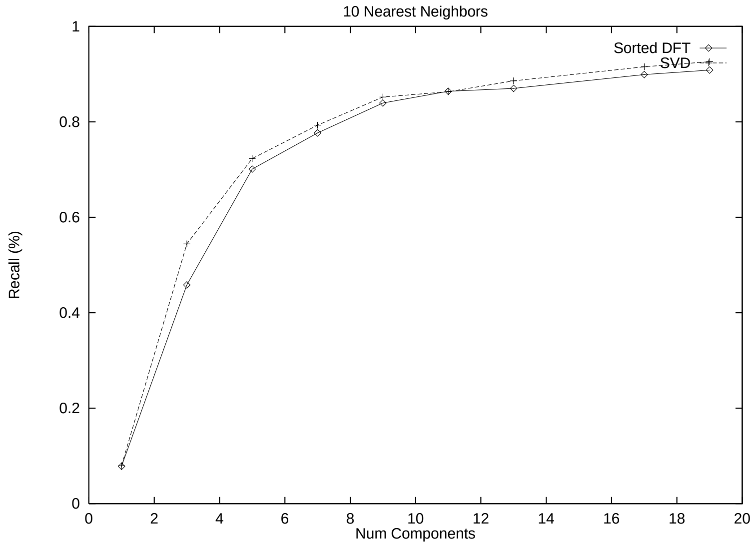


Figure 12: Queries for 10 Nearest-Neighbors

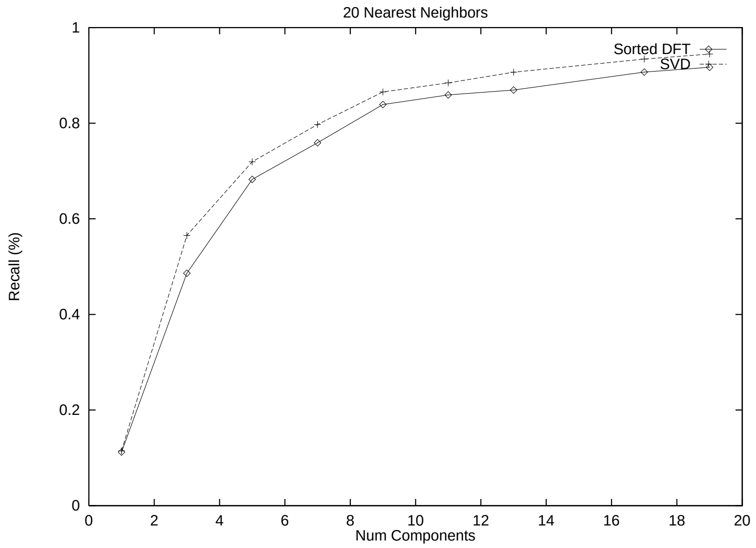


Figure 13: Queries for 20 Nearest-Neighbors