Texture Representations for Image Retrieval

by Scott A. Shafer

B.Sc. in Mathematics and Computer Science, January 2002, McGill University

A Thesis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Master of Science

May 20, 2012

Thesis directed by

Kie B. Eom
Professor of Engineering and Applied Science

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Acknowledgements

I would like to acknowledge and thank my thesis advisor, Professor Kie B. Eom, for his encouragement throughout the course of this research. I also thank the members of my thesis committee, Professors Milos Doroslovacki and H. Howie Huang, for their guidance and suggestions. I appreciate the hard work of the employees in the GW Veteran Services Office and the university's participation in the Yellow Ribbon Program, which helped fund this degree. I give a very special thanks to my wife, Jennifer, who encouraged me through the long hours and late nights in completing this degree.

Abstract of Thesis

Texture Representations for Image Retrieval

Image databases are used in medical research, hospitals, in scientific research, in museums to catalogue content, in security and biometrics and many other fields. The entire World Wide Web can be considered an image database from the point of view of a search engine. Image retrieval involves searching and retrieving specific or similar images from a database given a query image or textual descriptor. Content-based image retrieval uses low-level visual features, such as color, shape and texture, in digital images to search and retrieve content from an image database.

Texture plays a major role in the human visual system for image understanding and object recognition and is found, to some degree, in all natural images. The thesis explores texture representation in general and texture feature extraction methods for image retrieval. Three texture representations in the spatial, spatial-frequency and sparse domains are discussed and implemented to extract feature vectors from texture images. Sparse representations are invaluable to many signal processing and computer vision algorithms. Sparse coding and dictionary learning algorithms are discussed and implemented with the goal of evaluating the merit of sparse representation for image retrieval.

To compare the performance of these three texture feature methods a small texture image database is implemented using images from USC's SIPI and MIT's VisTex texture databases. Three measures are used to compare retrieval performance of the extraction methods. The results show that the sparse image retrieval method performed favorably but has room for improvement in both performance and complexity.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Image retrieval has been a popular computer vision research area for the past two decades [21, 26, 31, 20]. This research has been contributed from different fields, to include, image understanding, machine learning, and data mining, with the collective goal of providing a more accurate image retrieval system [6]. An image retrieval system is one that, given an input image, can search and retrieve one or more images from a database that are relevant to the input by some measure of similarity.

Describing similarity between images can be difficult to quantify, as perception may change from one viewer to another [32]. Further adding to the difficulty are differences in illumination, scale, rotation, occlusion and others. Early image retrieval systems, including many in use today, use textual descriptors to retrieve content and are similar to conventional information retrieval systems [33]. Textual descriptors require tagging each image or video with keyword metadata, which, in general, cannot be automated, and is therefore very time consuming. Further, textual descriptors cannot sufficiently characterize an image with a limited number of keywords. A more promising retrieval paradigm is content-based image retrieval (CBIR), which uses low-level visual data such as color, texture, and shape in place of text. Finding reliable and efficient methods to extract these low level features has been the focus of image retrieval, within the computer vision realm, for many years. In order to standardize CBIR feature extraction and meta-data organization the international standard MPEG-7 was created in 2001 [21, 25].

Given a query image, the goal of CBIR is to search the image database and provide a short list of ordered results which best match the input based on comparison of some feature vector(s) and a distance measure between the input and set of database vectors.

An obvious drawback is that many images in a database may be similar in low-level content to the query image, but may look completely different to a human observer. For example, two images might share a grass-like texture with strong enough similarity for a match. However, one of the images may be a close up of a long grain fabric and the other a natural landscape scene. Despite such shortcomings, MPEG-7, which employs CBIR, has a proven track record [21].

1.1 Texture in Content Based Image Retrieval

This thesis focuses on one low level feature, namely texture, and describes various mathematical models used in its description for the purpose of image retrieval. All natural images exhibit some notion of texture. Texture plays an important role in human perception regarding surface properties such as roughness and can help determine surface orientation. It has also been shown that texture analysis by the human visual system is an initial step in object identification [30]. On this basis, texture descriptors may provide powerful tools for image comparison and matching for the purpose of image retrieval.

While no definitive definition of texture exists, in the context of computer vision we may define texture as displaying some aspect of a pattern or arrangement of *texels*, or texture elements, and can therefore be approached statistically or structurally [13]. Natural textures display some amount of regularity and randomness. We may further define texture as an ordered collection of patterns forming a texture spectrum. The texture spectrum ranges from deterministic, tile-like, patterns on one end, to random noise on the other and are classified as: regular, near-regular, irregular, near-stochastic,

and stochastic. This spectrum is a commonly used notion in the field of texture synthesis [17].

Texture models can generally be categorized as statistical, structural or spectral [13]. A texture model seeks a simple, efficient and reliable approximation of the information in the texture region of the image. Many mathematical models have been applied to represent texture in the computer vision literature and some of the popular methods are discussed in the following chapter. MPEG-7 uses two texture descriptors, the homogeneous texture descriptor (HTD) and the non-homogeneous texture descriptor, also known as the edge histogram descriptor (EHD). The MPEG-7 texture descriptors can be applied individually or together. Homogeneity and object boundaries used together or individually can express many classes of texture. The MPEG-7 HTD uses a 30-channel Gabor filter bank of different scales and orientations to achieve a scale and rotation invariant descriptor [21]. The EHD partitions the image into 16 non-overlapping sub-images and the occurrences of edges, quantized to 5 orientations, are tabulated for each sub-image. Further details of the two texture descriptors can be found in [21], and chapter 3.

1.2 Applications of Texture Representation

Two very popular applications of texture representation are segmentation and synthesis. Texture segmentation involves the partitioning of an image into a set of disjoint texture regions. It is believed that object boundary detection and segmentation of different objects play a major role in image understanding and motion analysis in the human visual system [23]. Being able to identify the boundary between differing textures

is important in many applications. Everything from neural networks [22], and Markov random fields [14], to wavelets [38] have been applied to texture segmentation and can find application in remote sensing, quality control, biomedical image classification and many others. Texture segmentation, and segmentation techniques in general, can also be applied to image retrieval [33].

Texture synthesis is the process of creating digital textures of any size that are free of visual repetition. The primary use of texture synthesis is in computer graphics found in video games, image and video editing, and data visualization. Texture synthesis is generally obtained either by replicating a small texture image patch or generated by some process [17]. In general, the higher the quality of a texture synthesis technique for a particular class of texture, the more the technique lends itself to an analysis of the texture.

1.3 Thesis Objective

The objective of this thesis is to explore the performance of key texture feature extraction methods as well as to investigate the applicability of sparse representation for texture image retrieval. The MPEG-7 texture descriptors will be studied and their retrieval performance on a small texture database will be compared to that of the sparse texture model. The field of sparse representation, to include dictionary learning, has found many applications in image processing that provide better results than previous leading scale-frequency methods [9]. A compact representation, accompanied with impressive results in other applications could make the sparse model a candidate for high performance in image retrieval.

1.4 Thesis Outline

This thesis is organized as follows. Chapter two explores three classic texture representations that are still popular in many applications, namely, the Gray Level Co-occurrence Matrix and associated measures, the two dimensional causal auto-regressive model, and Markov Random Fields. Chapter three explores in more detail the three texture feature extraction methods used for image retrieval performance comparisons found in chapter 4. These methods include MPEG-7's edge histogram descriptor (EHD) and the homogeneous texture descriptor (HTD), and the sparse texture descriptor (STD). Chapter four describes the set up of the texture image database and specific details about how the three feature extraction methods are implemented. Chapter five presents and analyzes the obtained results. Lastly, chapter six concludes the thesis and explores possible future research.

Chapter 2:

Review of Research in Texture Representation

2.1 The Co-occurrence Matrix

A gray level image histogram is a discrete function that tabulates the number of pixels N_k, for each intensity value k, over the entire image. The histogram is often displayed in in a bar graph format and stored as a vector for further calculations. The image histogram is one of the most widely used features in image processing and computer vision [21]. Image histograms are easy to calculate and are invariant to image rotation. Moreover, a normalized histogram approximates the intensity distribution, thus enabling the calculation of useful image statistics. The major drawback to image histograms is that they do not convey any spatial information, which is a crucial property in texture representation due to the high correlation between adjacent pixels.

The gray-level co-occurrence matrix (GLCM) incorporates the relative positions of pixels and the distribution of pixel intensities. A GLCM can be considered a second order image histogram. The GLCM, denoted by $P_{ij}(d,\theta)$, is defined as a matrix of relative frequencies in which a pair of neighboring pixels, separated by distance d and along orientation θ, have gray levels i and j, respectively. For example, suppose d is one pixel, and $\theta = 0$, then the entries in this GLCM are determined by,

$$P_{ij}(d=1,\theta=0) = \#\{[(k,l),(m,n)],\ |k-m|=0,\ |l-n|=1,\ I(k,l)=i,\ I(m,n)=j\},$$

where # denotes the cardinality of the set and $I(k,l)$ is the gray level intensity of the pixel at position (k,l) in the image. For this example, the relative frequency of intensity pairs is calculated by observing a pixel's intensity and that of its rightmost neighbor, one

pixel away. The values of *i* and *j* range over all possible gray levels, thus for a N-bit image, the GLCM has dimension 2^N by 2^N. In [15], Haralick, who first proposed this matrix, introduces a set of GLCMs, where $\theta = \{0,45,90,135\}$, all using the same distance along the various orientations. With this set of matrices various texture features can be computed.

A GLCM for natural images is sparse with most non-zero entries clustered along the main diagonal. However, it is not symmetric since $P_{ij}(d,\theta)$ is not necessarily equal to $P_{ji}(d,\theta)$. In order to make use of the statistics of the GLCM it is advantageous to normalize the matrix by the number of non-zero entries, *N*, and make it symmetric. The GLCM can be added to its own transpose, creating a symmetric matrix. Once the symmetric GLCM is normalized, the entries can be thought of as probabilities. For example, the sum of the diagonal entries gives the probability of constant intensity regions of the image.

Statistics obtained from GLCMs of different textures can be used to measure the differences between such textures. The GLCM is not rotation invariant, a particularly useful characteristic for content-based image retrieval (CBIR). However, averaging $P_{ij}(d,\theta)$ along the different orientations of θ can increase rotational invariance. An additional drawback for CBIR is that the texture element's shape, commonly referred to as a *texel*, is lost in computing the GLCM. Haralick defined 14 texture features to be used in texture classification [15]. These include:

Uniformity $\sum_{i=1}^{K}\sum_{j=1}^{K} p_{ij}^2$

$$
\begin{aligned}
&\textit{Contrast} && \sum_{i=1}^{K}\sum_{j=1}^{K}(i-j)^2 p_{ij} \\
&\textit{Correlation} && \sum_{i=1}^{K}\sum_{j=1}^{K}\frac{(i-m_r)(j-m_c)p_{ij}}{\sigma_r\sigma_c} \\
&\textit{Entropy} && -\sum_{i=1}^{K}\sum_{j=1}^{K} p_{ij}\log_2 p_{ij} \\
&\textit{Homogeneity} && \sum_{i=1}^{K}\sum_{j=1}^{K}\frac{p_{ij}}{1+|i-j|},
\end{aligned}
\tag{2.1}
$$

where K is the number of intensity levels, $p_{ij} = \frac{P_{ij}}{N}$, for co-occurrence matrix P and normalizing constant N, means $m_r = \sum_{i,j}^{K} ip_{ij}$, and $m_c = \sum_{i,j}^{K} jp_{ij}$, and variances $\sigma_r^2 = \sum_{i,j}^{K}(i-m_r)^2 p_{ij}$, and $\sigma_c^2 = \sum_{i,j}^{K}(j-m_c)^2 p_{ij}$. Uniformity is maximized when an image has only one intensity level and therefore is a measure of how few intensity levels are distributed in the image. Contrast measures the difference in intensity between a pixel and its neighbor. Correlation measures how correlated a pixel is to its neighbor. Entropy measures the randomness of the entries in the GLCM and is 0 for a constant image. Lastly, homogeneity measures the spatial closeness of entries in the GLCM to the main diagonal.

2.2 Causal Auto-Regressive Model

The autoregressive (AR) process is a mathematical model based on the correlation of a signal's current value to some number of previous values. AR modeling assumes that the next value in a signal or time series is a linear combination of the past p output

values. Thus, AR can be used to predict the behavior of a process and provides a simple way to model spectral characteristics of signals. The number of past values, p, used in the regression is known as the model order. The weights, or coefficients, can be estimated and then the AR process can be used as an analysis or synthesis filter.

The autoregressive model is used extensively in one-dimensional signal processing of natural phenomena. For example, in speech processing, linear predictive coding (LPC) is based on the AR model. The autoregressive process can be stated more compactly by,

$$AR(p):\ Y(n) = \sum_{i=1}^{p} a_i Y(n-i) + w(n) + c \tag{2.2}$$

where Y is the previous output, a_i are the filter coefficients to be estimated, w is white noise and c is a constant (often omitted). Any estimation technique can be used to determine the filter coefficients a_i in equation (2.2), such as constructing the Yule-Walker linear system of equations and solving it using the Levinson-Durbin algorithm [16].

This model assumes that the signal is wide sense stationary (WSS) and causal. While a natural image is not WSS it can be modeled as such in small neighborhoods, however, the notion of causality can be cumbersome in two dimensions. Some artificial directed lattice convention must be assumed to enforce causality. One popular convention is the non-symmetric half plane (NSHP). Figure 2-1 gives a simple description of the NSHP, where time moves in a raster scan order. In Figure 2-1, the grey circles represent past pixel values, the black circle represents the pixel value currently being predicted, and the white circles denote future pixels. An equation for the two dimensional AR process is defined in equation (2.3),

$$AR(p): Y(n,m) = \sum_{(i,j)\in S}^{p} a_{ij} Y(n-i, m-j) + w(n,m), \tag{2.3}$$

where S is the support of the prediction filter.

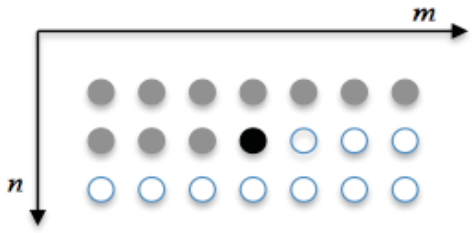

Figure 2-1: Example of non-symmetric half plane (NSHP).
In the figure above the grey pixels represent past values, black represents the value currently being predicted, and white represent future values.

Several methods exist to determine the best choice in model order, but the most commonly used is known as Akaike's Information Criterion (AIC). AIC is a statistical measure of goodness of fit of the *AR(p)* model to the given signal and is a function of the variance of the residual sequence. It is given by,

$$AIC(p) = N \log \sigma^2 + 2p \tag{2.4}$$

where p is the model order, N is the number of values and σ^2 is the variance of the AR residual or error term (in this case we assume the error is normally distributed).

2.3 Markov Random Field

A pixel's intensity is generally dependent upon all neighboring pixels. The restriction of causality in the autoregressive (AR) model imposes an unnatural and anisotropic dependency. The Markov random field (MRF) model drops causality to promote an isotropic pixel dependency. This is accomplished by defining pixel neighborhoods and cliques and treats the image as a set of lattice points S. The basic definition of a MRF is a

set of random variables with Markov property described by an undirected lattice [2]. Two common conventions for neighborhoods are the 4-point and 8-point neighborhoods as shown in Figure 2-2. The neighborhood of s is denoted by $N_s = \{r : r \text{ is a neighbor of } s\}$.

Figure 2-2: Pixel groupings defining commonly used neighborhoods with respect to the center. (Left) 4-point neighborhood. (Right) 8-point neighborhood.

We call $\mathcal{N} = \{N_s \ \forall s \in S\}$ a neighborhood system if $\forall s \in S$ we have the following two properties: $s \notin N_s$, that is, s cannot be a neighbor with itself, and neighborhoods must be symmetric, that is, $s \in N_r \Leftrightarrow r \in N_s$. Another useful definition is that of a clique $C \subset S$. A clique is a set of points C that are all neighbors with each other, that is $\forall\ a,b \in C,\ a \in N_b$.

A simple example of a MRF is the 1D Markov chain shown in Figure 2-3. The lattice point s with value x_s has neighborhood $N_s = \{s-1, s+1\}$. We say that a random object X on lattice S is a MRF with neighborhood system $\mathcal{N}$ if for all $s \in S$ we have $p(x_s \mid x_i,\ \forall\ i \in S \text{ and } i \neq s) = p(x_s \mid x_i,\ \forall i \in N_s)$. For the Markov chain example in Figure 2-3 we get: $\forall s \in S$, $p(x_s \mid x_i,\ \forall\ i \in N_s) = p(x_s \mid x_{s-1}, x_{s+1})$. In general, estimating parameters and calculating density functions for a 2D lattice can be difficult. However,

MRFs have been applied to texture synthesis [35], segmentation [14], restoration and many other image processing applications.

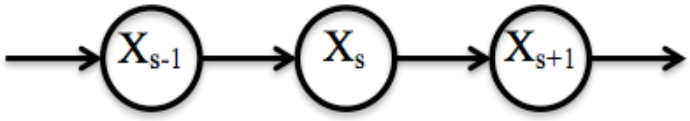

Figure 2-3: Example of a Markov chain Markov random field (MRF).

Chapter 3:

Texture Features for Image Retrieval

3.1 Edge Histogram

Edges play a critical role in distinguishing the boundaries between different objects and illumination conditions. Sudden changes in intensity can be described by step edges and more slowly varying intensity is described by ramp edges. In image processing, edge detection is often a crucial step in segmentation. In general, traditional edge detectors come in two flavors, one based on the gradient or first derivative approximation, and the other on the Laplacian operator or second derivative [13].

The gradient vector points in the direction of the greatest rate of change and is given in equation (3.1),

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} G_x \\ G_y \end{pmatrix}. \tag{3.1}$$

The magnitude of the gradient,

$$\text{mag}(\nabla f) = \sqrt{G_x^2 + G_y^2} \tag{3.2}$$

and its orientation,

$$\theta = \arctan\left(\frac{G_y}{G_x}\right), \tag{3.3}$$

can be computed to determine the greatest rate of change in intensity over an image patch. The direction of the edge is perpendicular to the direction of the gradient. The Laplacian operator, defined by equation (3.4),

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \tag{3.4}$$

is given by the divergence of the gradient and measures the flux density of a vector field. These two classes of edge detectors translate into approximating the first or second derivatives, respectively. The first derivative can be numerically approximated by a first difference equation such as

$$\frac{\partial f}{\partial x} \approx f(x+1,y) - f(x,y), \tag{3.5}$$

and similarly in the *y* direction. In general, a first derivative digital approximation must be 0 over constant intensity, be non-zero at the onset of a ramp edge, as well as non-zero along the ramp. In two dimensions, the digital derivative is best applied by convolving a spatial filter or mask, whose entries comply with the previously stated derivate rules. A popular and successful edge detector based on the gradient of a Gaussian mask is the Canny edge detector [13]. A much simpler, though less robust, edge detector comes from applying the Sobel masks shown in Figure 3-1.

A benefit of the Sobel masks is that they are separable, that is, for example, the mask in Figure 3-1 (a) can be computed by first convolving the image with $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$, followed by $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$. Another way of stating this is that the mask in Figure 3-1 (a) is the outer product of the two vectors respectively. This method of filtering is equivalent to two-dimensional convolution and is possible due the associative nature of convolution. This separability can significantly increase the computational advantage depending on the size of the mask. If the dimensions of the mask are *P* by *Q*, then the computational advantage is given by $PQ/(P+Q)$. In the case of a Sobel mask, this provides a

$PQ/(P+Q) = 9/6 = 1.5$ times faster convolution. The larger the mask used in the convolution, the greater the gain over applying a two dimensional mask. The Sobel masks also make use of the [1 2 1] vector that provides spatial smoothing to the image, which helps suppress noise.

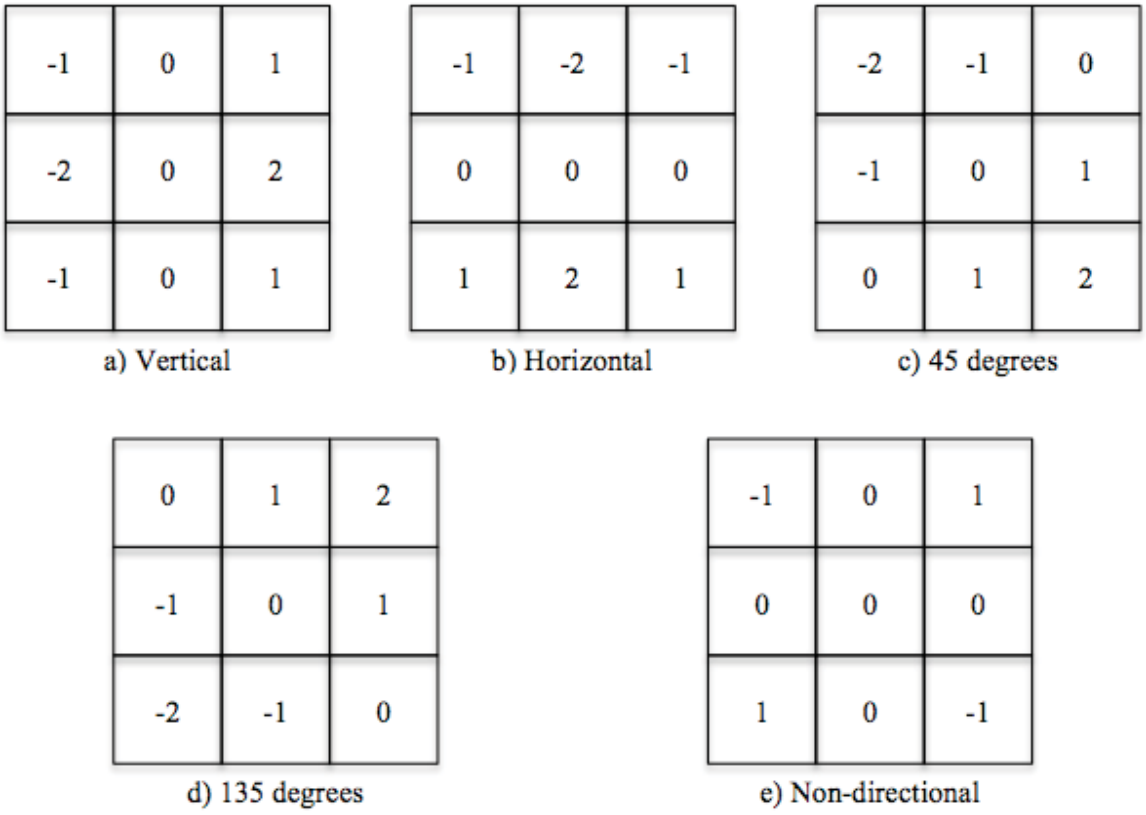

Figure 3-1: Five directional Sobel masks used to create edge maps. When convolved with an image they produce edge maps which are images with edges as principal features in the directions described in a) through e).

When these traditional edge detectors are applied to the less structured types of texture they treat the textured region as noise. However, boundaries between objects or texels may be successfully determined. Counting the number of edges, quantized to specific orientations, which are found within a small image patch, are uniquely identifying features and can be used for image retrieval purposes. This is the idea behind

the Non-Homogeneous Texture Descriptor, also known as the edge histogram descriptor (EHD), used in MPEG-7. The EHD is scale invariant and provides a compact representation [21].

The edge histogram, as described by [21], quantizes edges into 5 directions: vertical, horizontal, 45°, 135°, and non-directional using the masks shown in Figure 3-3. The feature extraction is carried out by partitioning the image into 16 contiguous, square, sub-images of equal size on which each of the five masks are convolved. The sub-images are ordered left to right, top to bottom, (0,0) through (3,3) as shown in Figure 3-2, and a histogram of 80 bins is constructed. Each bin therefore describes the frequency of specific edge types contained in a particular sub-image. The sub-images themselves are further divided into a fixed number of blocks. Each block is then tested for all 5 edge-orientations and the one with the greatest magnitude, above a cutoff threshold, is chosen and the associated bin is updated. Each bin is normalized by dividing it with the number of image-blocks that had edge magnitudes above the cutoff threshold. The bin value is then non-uniformly quantized from [0,1] to a 3-bit value using the Lloyd-Max algorithm. This gives a 240-bit representation of the edge histogram for a particular image.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Figure 3-2: Example of Edge Histogram Descriptor partition of an image into 16 sub-images.

While the number of image blocks is kept constant, due to different sizes of available images in any given database, the size of the individual image-blocks adapts according to,

$$x = \sqrt{\frac{(\text{Image width}) \times (\text{Image height})}{\text{Fixed \# of image blocks}}}, \text{ block size} = \left\lfloor \frac{x}{2} \right\rfloor \times 2. \quad (3.6)$$

However, all images tested in chapter 4 of this thesis are of the same size, so this was not a factor.

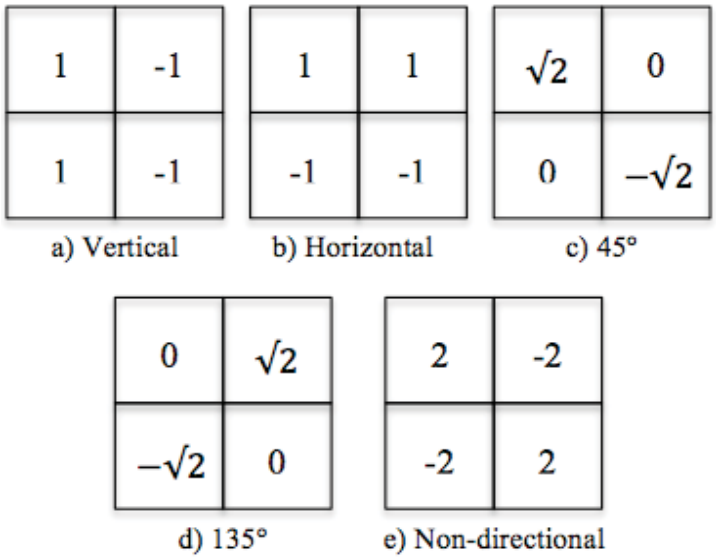

Figure 3-3: MPEG 7 edge detection masks.

Many variants to the MPEG-7 edge histogram have been proposed to improve retrieval rate including using both local and global edge histograms [26] and applying the edge histogram method in the discrete cosine transform (DCT) domain or discrete wavelet transform (DWT) domain for JPEG and JPEG2000 images respectively [10], [11]. The method proposed by [26] argues that since the MPEG-7 EHD only takes into account edge distribution locally in individual sub-images, the global feature information is lost. Their method applied both local edge histograms as well as edge histograms of

overlapping sub-images, which increases the total bin count to 150. Using 3-bits per bin they achieve an Average Normalized Modified Retrieval Rank (ANMRR), defined in chapter 5 equation (5.8), of 0.296, or 0.04 better than using local edge detection alone. In order to apply feature extraction methods for CBIR directly on JPEG or JPEG2000 encoded images [10] and [11] propose methods that calculates the EHD directly on the DCT and DWT coefficients, respectively. The two methods give a slight ANMRR improvement over the proposed MPEG-7 EHD but computationally boast many hundreds of times faster for multiplications and several times faster for additions.

3.2 Gabor Filter Bank

The Gabor filter is named after Denis Gabor, the Nobel Prize winning physicist/electrical engineer. In the spatial domain this filter is represented by a complex sinusoid modulated by a Gaussian envelope, with impulse response given in one dimension by,

$$h_e(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \cos(2\pi u_0 x),$$
$$h_o(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \sin(2\pi u_0 x), \tag{3.7}$$
$$h(x) = h_e(x) + jh_o(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} e^{j(2\pi\omega_0 x)}.$$

In [7], Daugman extends the Gabor filter to two dimensions and shows the 2D filter to provide optimal Heisenberg joint resolution in space and spatial-frequency and that these filters conform well to mammalian visual receptive field profile. In two dimensions we get,

$$h_e(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} \cos(2\pi u_0 x + 2\pi v_0 y), \tag{3.8}$$

and a similar equation for the odd/sine term. In this equation, (u_o, v_o) defines the center frequency and σ_x, σ_y defines the spread of the Gaussian. The Gabor filter is a bandpass filter allowing the location, spread, and orientation of the Gaussian to be tuned to a particular frequency band. It has been speculated that the human visual system decomposes visual spectra into bands in spatial frequency in a similar manner [7]. This concept is particularly useful in extracting textures features that can be modeled as homogeneous patches of a specific narrow frequency band.

Many applications have made use of the tunable, multi-scale, bandpass nature of the Gabor filter including texture segmentation [39], handwriting identification [3], and iris recognition [18]. Applying Gabor filters to texture feature extraction for CBIR has also been a popular research area and its proven effectiveness in this area has caused it to be included in MPEG-7. The Homogeneous Texture Descriptor (HTD), as described in [21, 31], is one of MPEG-7's visual descriptors and utilizes a 30 channel Gabor filter bank to extract texture features. The HTD provides texture-to-texture similarity matching by comparing the feature vectors composed of means and standard deviations of the energy from the 30-channel Gabor filter bank outputs and is both rotation and scale invariant.

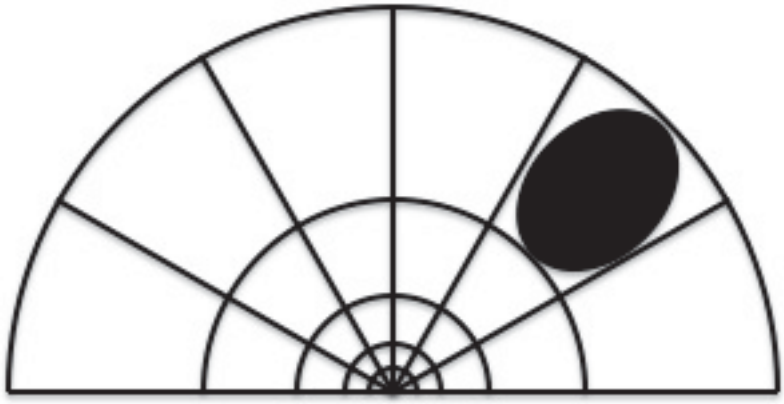

Figure 3-4: Partition of the spatial frequency plane into 30 channels for the Homogeneous Texture Descriptor.

The HTD 30-channel filter bank is constructed with equal divisions in the angular direction and octave divisions radially spanning the normalized spatial frequency half-plane. The 30 feature channels consist of 6 angular bands, with 30° bandwidth, and 5 radial octaves as shown in Figure 3-4. The radial octaves are calculated starting with the highest center frequency at $\omega_0 = \frac{3}{4}$ and the others by $\omega_s = \omega_0 2^{-s}$ with $s \in \{0,1,2,3,4\}$.

As the Gabor functions form a complete but non-orthogonal basis there is redundancy or overlap between these channels. In [20] a strategy of selecting the filter parameters is explained that ensures the adjacent channels touch at their half-peak magnitude, thus reducing overlap. Using this filter bank, the energy mean and energy deviation are computed for each channel. A 62-component feature vector is constructed comprised of the mean and standard deviation of image intensity, as well as the 30 energy mean and 30 energy deviation components. The feature vectors are then used for similarity matching and retrieval.

3.3 Sparse Texture Representation

A sparse representation of a signal is one that uses a small number of coefficients in its description. This is one of the main attractive features of transforms such as the DCT in that it allows faster and simpler processing and less storage space of the same signal. A sparse representation can also reveal the underlying structure of a signal, which can provide insight into developing signal processing techniques such as wavelet denoising. Many such transforms exist that can decompose a signal into a linear combination of elementary waveforms called atoms. A family of such atoms is called a dictionary. However, no single dictionary can provide maximal sparsity to all classes of signals, so adaptive techniques are necessary for dealing with the set of signals for a particular application [1]. One option is to learn the dictionary on a training set that well represents the signals of interest.

In the last ten years the area of sparse signal representation has become very popular [1, 9, 28, 36, 40]. Many new algorithms have been developed which make this model attractive for many image-processing applications. Sparse representation can be generalized into two major areas, sparse coding and dictionary learning, which will be defined in the following sections. The idea of sparse texture representation for image retrieval applies sparse coding and dictionary learning to a texture image database in order to search and perform similarity matching with the sparse approximation of the texture images. This method is dubbed the Sparse Texture Descriptor (STD) in this thesis. The following sections explain some key ideas in sparse approximation as well as the algorithms employed in this thesis.

3.3.1 Sparse Coding

The main objective in sparse coding is, given a signal, we would like to find a sparse representation or approximation which is a linear combination of a few atoms of an overcomplete dictionary. An overcomplete dictionary is one such that the number of atoms exceeds the dimensions of the signal space. This means that any signal in this space can be represented by more than one linear combination of atoms.

Suppose we have the following linear system $D\underline{a} = \underline{x}$, with $D_{N\times K}, \underline{a}_{K\times 1}$ and $\underline{x}_{N\times 1}$, where $\underline{x}$ is the signal of interest, D is an overcomplete dictionary and $\underline{a}$ is the sparse representation of our signal $\underline{x}$. If $K > N$, then this system is underdetermined and there exist infinitely many solutions. In this context, however, we want to find the sparsest solution. Intuitively, we could order the solution space by some measure of sparsity and choose the sparsest solution. This is accomplished through regularization by a sparsity inducing cost function J as shown in equation (3.9),

$$\hat{\underline{a}} = \arg\min_{a} J(\underline{a}) \text{ s.t. } D\underline{a} = \underline{x}. \tag{3.9}$$

In order to admit only sparse solutions we should have $J(\underline{a}) = \left|\left|\underline{a}\right|\right|_p$, where $p = 0$ or 1. There are advantages and drawbacks for either choice of p. While ℓ_0 leads to greater sparsity, ℓ_1 is convex.

If we consider the ℓ_p norm in place of J in equation (3.9), it can be shown that as p goes to zero, the solutions to (3.9) get sparser [9]. This leads us to the ℓ_0 quasi-norm, which measures the sparsity of a signal by counting the nonzero entries. Our goal in sparse representation is to approximate our given signal by one that has as few non-zero entries as possible.

The two fundamental problems in sparse coding are as follows:

$$\hat{\underline{a}} = \arg\min_{a} \|\underline{a}\|_0^0 \text{ s.t. } D\underline{a} = \underline{x} \tag{3.10}$$

$$\hat{\underline{a}} = \arg\min_{a} \|\underline{a}\|_0^0 \text{ s.t. } \|D\underline{a} - \underline{x}\|_2 \le \varepsilon. \tag{3.11}$$

The first is known as the exact sparse coding problem and the second as the approximation. Another useful problem statement found in many applications is,

$$S - Sparse \quad \hat{\underline{a}} = \arg\min_{a} \left\|D\underline{a} - \underline{x}\right\|_2^2 \text{ subject to } \left\|\underline{a}\right\|_0 \le S \tag{3.12}$$

where S is a known and user defined constant. Equation (3.12) expresses that we wish to minimize the squared Euclidean distance between $\underline{a}$ and $\hat{\underline{a}}$, while using a maximum of S non-zero coefficients to form the approximation signal $\hat{\underline{a}}$.

Equations (3.10) through (3.12) are deceivingly complex. This is due to the discrete, discontinuous, and non-convex nature of ℓ_0. An exhaustive search solution of equation (3.10) is not computationally feasible. For example, assume D is of size 256x1024 and suppose we know the sparsest solution has only 10 non-zero entries. An exhaustive combinatorial search will be on the order of $\binom{1024}{10} = \frac{1024!}{10!1014!} \approx 1 \times 10^{30}$ sec, or about 3×10^{22} years assuming each calculation takes 1×10^{-6} sec. Equation (3.10) and its variants, (3.11) and (3.12), have been proven to be NP-Hard [27]. Although an exhaustive search is impossible, there exist sub-optimal algorithms that can approximate the solution in a reasonable amount of time, even with large data sets such as images.

3.3.2 Pursuits Algorithms

In the previous section it was demonstrated that a brute force method for solving equation (3.10) or its variants is hopeless. However, many algorithms have been developed in the last two decades that can give sub-optimal solutions to these problems. The two main avenues of success are greedy algorithms and convex relaxation of equation (3.10). Many algorithms are available today for solving large systems such as in image processing problems, however, we must consider a tradeoff of runtime versus approximation error. In general, greedy algorithms run faster, but have inferior approximation error as compared to convex relaxation avenue, which use linear programming algorithms. The next two sections give some detail on the greedy algorithm named orthogonal matching pursuit (OMP) [36], and briefly mention the Basis Pursuit (BP) algorithm [4], respectively.

3.3.2.1 Greedy Algorithms

From equation (3.10) we observe that the unknown vector $\underline{a}$ is composed of the support of the solution and the non-zero values over this support. If we can find the support, the non-zero values can be calculated one at a time by least squares. The support is discrete so attacking equation (3.10) in this fashion leads us to a greedy algorithm.

Matching pursuit (MP) and its many variants are greedy algorithms that have been successfully applied to approximating problems such equation (3.10). The basic idea of matching pursuit is that it finds the one atom in D which best matches our given signal $\underline{x}$, then, using the atoms found in previous iterations, it finds the best atom for the current residual. The algorithm halts when $\|D\underline{a} - \underline{x}\|_2$ is below a given threshold. An improved

variant of MP, known as OMP [36], evaluates the coefficients of $\underline{a}$ each iteration via least squares.

OMP Algorithm

PURPOSE: To approximate the solution of Equation (3.11).
INPUT: dictionary D, signal $\underline{x}$, threshold ε_0
INITIALIZE: $k = 0$

- Initial solution $\underline{a}_0 = \underline{0}$
- Initial residual $\underline{r}_0 = \underline{x}$
- Initial solution support $S_0 = \varnothing$

MAIN LOOP:
$k = k + 1$
Sweep: For each column $\underline{d}_j$ in D, compute errors $\varepsilon(j) = \min_{z_j} \left\| \underline{d}_j z_j - \underline{r}_{k-1} \right\|_2^2$ using the optimal $z_j^* = \underline{d}_j^T \underline{r}_{k-1} / \left\| \underline{d}_j \right\|_2^2$.

Update support: Find a minimizer, j_0 of $\varepsilon(j) : \forall j \notin S_{k-1},\ \varepsilon(j_0) \leq \varepsilon(j)$ and update $S_k = S_{k-1} \cup \{j_0\}$.

Least Squares: Find $\underline{a}_k = \min_a \left\| D\underline{a} - \underline{x} \right\|$ s.t. supp{$\underline{a}$}=S_k.

Update Residual: Set $r_k = \underline{x} - D\underline{a}_k$.

Stop: If $\left\| r_k \right\|_2 < \varepsilon_0$ stop, otherwise next iteration.

3.3.2.2 Convex Relaxation

As mentioned earlier, we can relax equation (3.10) by replacing ℓ_0 with ℓ_1. The ℓ_1 norm is smooth and, moreover, convex, which makes this a linear programming problem. In the sparse approximation domain, the algorithm used to solve such a problem is known as Basis Pursuit (BP) [4]. It is well known that existing linear programming algorithms do not have strong polynomially bounded runtimes. Thus, compared to OMP, the BP

algorithm runs much slower, but is known to give stronger guarantees with respect to error. Until the last few years BP has mostly been used in the compressive sampling domain due to the low error guarantee and not in image processing since it is restrictively slow for large data sets. For this reason it is not implemented in this thesis.

It should be mentioned that recent greedy algorithms such as Regularized OMP (ROMP) [28] have bridged the error gap and are also being investigated for compressive sampling.

3.3.3 Dictionary Learning

In applying OMP to equation (3.10), or its variants, it has been assumed so far that the dictionary D is given. This section focuses on how to obtain the dictionary. A dictionary can be predesigned; for example, the homogeneous texture descriptor (HTD) in section 3.2 could be described and constructed as a Gabor wavelet dictionary. Another example is the bi-orthogonal wavelet dictionary used in JPEG2000 [13]. Such dictionaries are designed to work well on any type of image and at great computational savings, as the discrete wavelet transform (DWT) transform is on the order $O(N)$ [9]. However, such designed dictionaries do not provide the sparsest representation since they are not adapted to a specific image or set of images [9].

A dictionary with more basis vectors than the dimensionality of the signal space is called overcomplete and allows for a richer set of building blocks with which to represent the signal. In general an overcomplete dictionary that is learned on a specific set of signals can better represent each signal, with respect to sparsity, than a predesigned orthogonal dictionary [9]. How to extract atoms from the signal training set and which ones to choose is the main idea behind dictionary learning algorithms.

We consider the problem where we are given T training signals $X = \{x_i\}_{i=1}^{T}$, and we want to find the dictionary $D_{N\times K}$ that represents each signal in this set with a S-sparse vector $\underline{a}_i$ (i.e. with at most S non-zero entries). The vectors $\underline{a}_i$ form the representation matrix A. We can state this as a matrix factorization problem with a sparseness constraint,

$$\min_{D,A} \|X - DA\|_F^2 \text{ subject to } \forall i, \|\underline{a}_i\|_0 \le S, \tag{3.13}$$

where $||A||_F$ denotes the Frobenius norm of the matrix A.

Many techniques have been developed to tackle the dictionary learning problem including probabilistic methods such as maximum likelihood and maximum *a-posteriori* [9]. Several years ago an iterative algorithm was proposed that uses sparse coding to build a dictionary using a signal training set. This algorithm is called K-SVD and has been successfully applied to many image processing inverse problems such as de-noising, de-blurring, and in-painting with better results than previously proposed methods [1].

K-SVD is a generalization of K-Means clustering algorithm and there is a clear relationship between clustering and sparse representation. In clustering we have, say, K clusters of N data points and we wish to be able to decide which data belong to which cluster. Very generally, the K-Means algorithm solves the clustering problem by alternating between two steps. First, it guesses the center of a cluster, then for each data point K-Means determines the center a particular datum is closest to in ℓ_2. Second, a new center is calculated based on the new set of data members; then rinse and repeat.

Once K-Means completes we can approximate any data point by its associated cluster mean or centroid; this is known as vector quantization (VC). We can consider the cluster

mean as the one-element sparse approximation for all the members of the cluster. On the other hand, sparse coding represents a signal as a linear combination of k dictionary atoms analogous to the cluster centroids, and is therefore a generalization of K-Means.

The K-SVD algorithm generalizes K-Means by iteratively applying and updating two steps:

- Sparse coding stage: The dictionary is fixed and we find the sparse A in equation (3.13), column by column. This is analogous to using fixed cluster centers and finding the data points belonging to them.
- Dictionary update stage: All dictionary columns are fixed with the exception of the k^{th} which is updated along with the corresponding coefficients that multiply it in A. The new column $\widetilde{d_k}$ and corresponding coefficients in A are found such that the mean square error (MSE) is minimized. Minimizing the mean square error one column at a time is accomplished using singular value decomposition (SVD). This is analogous to updating the cluster centroid in K-Means.

The sparse coding stage is rather straight forward as equation (3.13) can be decoupled into T distinct problems of the type in equation (3.12), where any pursuit algorithm may be employed.

In the dictionary update stage, assume that D and A are fixed and we consider a particular column in D, $\underline{d}_k$, and the coefficients in the corresponding *row* of A, denoted $\underline{a}_T^k$. The penalty function in (3.13) can be expanded as,

$$\left\|X - DA\right\|_F^2 = \left\|X - \sum_{j=1}^{K} \underline{d}_j \underline{a}_T^j\right\|_F^2 = \left\|(X - \sum_{j \neq k} \underline{d}_j \underline{a}_T^j) - \underline{d}_k \underline{a}_T^k\right\|_F^2 = \left\|E_k - \underline{d}_k \underline{a}_T^k\right\|_F^2 \qquad (3.14)$$

where $X_{N\times T}$, $D_{N\times K}$ and $A_{K\times T}$.

In equation (3.14) we have decomposed *DA* into the sum of *K* rank-1 matrices, where $K-1$ are fixed and we update the remaining one. At this point we would like to decompose the k^{th} error term, E_k, using SVD in order to minimize the MSE, that is $E_k = U\Sigma V^T$ and set $\underline{d}_k = \underline{u}_1$ and $\underline{a}_T^k = \Sigma_{(1,1)}\underline{v}_1$, where $\underline{u}_1, \underline{v}_1$ are the first columns of *U* and *V* and $\Sigma_{(1,1)}$ denotes the first entry of Σ. We assume that the singular values in Σ are ordered with the largest value in the $(1,1)^{th}$ entry. However, we need to construct a masking matrix to use on the penalty function so as to only involve the non-zero entries in *A*, otherwise we risk losing the sparsity gained in the sparse coding stage.

To prevent this problem a set of indices ω_i is constructed that indicates which training signals $\underline{x}_i$ use the column $\underline{d}_k$, that is, the training signals for which $\underline{a}_T^k \neq 0$. More precisely, $\omega_k = \{i \mid 1 \le i \le K,\ a_T^k(i) \neq 0\}$. We further define the masking matrix Ω_k, of size $T \times |\omega_k|$, where $|\omega_k|$ denotes cardinality of the set, constructed with 'ones' in the $(\underline{\omega}_k(i), i)$ entries and zeros elsewhere, where $\underline{\omega}_k$ is a vector formed with the elements of the set ω_k and $\underline{\omega}_k(i)$ denotes the i^{th} entry. Thus, multiplying $\underline{a}_T^k\Omega_k$ shrinks the row vector $\underline{a}_T^k$ by discarding any zero entries and multiplying $X\Omega_k$ creates a matrix of size $N \times |\omega_k|$ that includes the training signals that use $\underline{d}_k$. For example, suppose $T = 4$, and $\underline{a}_T^k = [0, a_2, 0, a_4]$, then $\omega_k = \{2,4\}$, $\underline{\omega}_k = [2,4]$, and Ω_k is of size 4×2 and has ones in the $(2,1)$ and $(4,2)$ entries. Thus, $\Omega_k = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$, and $\underline{a}_T^k\Omega_k = [a_2, a_4]$. Post-multiply the masking matrix Ω_k with the right hand side of equation (3.14) gives,

$$\left\| E_k \Omega_k - \underline{d}_k \underline{a}_T^k \Omega_k \right\|_F^2 = \left\| E_k^R - \underline{d}_k \underline{a}_R^k \right\|_F^2 \tag{3.15}$$

Using equation (3.15) it is now possible to properly decompose $E_k^R = U\Sigma V^T$ and set

$$\tilde{\underline{d}}_k = \underline{u}_1 \text{ and } \underline{a}_R^k = \Sigma_{(1,1)} \underline{v}_1 \tag{3.16}$$

which minimizes the penalty function by approximating E_k^R with its closest rank-1 matrix.

K-SVD Algorithm

PURPOSE: Learn an overcomplete dictionary *D* given a training set of *T* signals.
INPUT: Training set $X = \{\underline{x}_i\}_{i=1}^T$, initial dictionary $D^{(0)} \in \mathbb{R}^{N \times K}$, stopping criteria ($\varepsilon_0$ or a maximum of *M* iterations) and desired sparsity *S*.
INITIALIZE:

- Normalize columns of $D^{(0)}$
- Set $J = 1$ (for $D^{(J)}$)

MAIN LOOP:
Sparse Coding Stage: Use any pursuit algorithm to compute the representation vectors $\underline{a}_i$ in $A^{(J-1)}$ for all training vectors $\underline{x}_i$ by approximating the solution of

$$\hat{\underline{a}}_i = \arg\min_{\underline{a}} \left\| \underline{x}_i - D^{(J-1)} \underline{a}_i \right\|_2^2 \text{ subject to } \left\| \underline{a}_i \right\|_0 \le S, \text{ for } i = 1,...,T .$$

The vectors $\hat{\underline{a}}_i$ form the matrix $A^{(J)}$.

Dictionary Update Stage: Update all of the columns of the dictionary to obtain $D^{(J)}$ by the following: for each column $\underline{d}_k$ of $D^{(J-1)}$ where $k = 1,...,K$

- Define the index set of training vectors that use column $\underline{d}_k$ by,
  $$\omega_k = \left\{ i \mid 1 \le i \le T,\ \underline{a}_T^k(i) \ne 0 \right\},$$
  where $\underline{a}_T^k(i)$ is the i^{th} entry in the k^{th} row of matrix $A^{(J)}$.

- Construct the mask Ω_k as previously described.

- Compute the residual matrix by $E_k = X - \sum_{i \ne k} \underline{d}_i \underline{a}_T^i$, for $i = 1,...,K$,
  where $\underline{a}_T^i$ is the i^{th} row of $A^{(J)}$.

- Post-multiply the mask Ω_k such that $E_k\Omega_k - \underline{d}_k \underline{a}_T^k \Omega_k = E_k^R - \underline{d}_k \underline{a}_R^k$.
- Decompose $E_k^R = U\Sigma V^T$ by SVD.
- Update $\underline{d}_k = \underline{u}_1$ and $\underline{a}_R^k = \Sigma_{(1,1)} \underline{v}_1$.
- With the updated columns $\underline{d}_k$ form $D^{(J)}$ and rows $\underline{a}_R^k$ update $A^{(J)}$.

Stop: If $\left\|X - D^{(J)}A^{(J)}\right\|_2^2 < \varepsilon_0$ (or if maximum iteration halting criteria is used, stop when $J = M$)
Otherwise $J = J + 1$.

3.3.4 Sparse Texture Descriptor

With the knowledge from the previous sections in hand the sparse texture descriptor (STD) can be constructed. The K-SVD algorithm can be applied to learn a single dictionary on a set of texture images. With this dictionary and the OMP algorithm the sparse approximation of the texture images can be found. These sparse approximation images can then be used to perform similarity matching where similarity is based on proximity of the sparse images. The details are found in section 4.3.

Chapter 4: Implementation

4.1 Texture Database

In order to assess and compare the texture feature extraction methods described in chapter three a small image database is constructed. The database consists of textures from two collections: USC's Signal and Image Processing Institute (SIPI) image database [34], which includes some Brodatz images, as well as MIT's Media Lab VisTex database [29]. In all, 25 8-bit images were cropped into quadrants forming 100, 64 by 64 pixel, non-overlapping sub-images. The majority of the original images were of size 128 by 128, and the few images that were not, were scaled down in order to be cropped into 64 by 64 pixel quadrants. Out of the 100 images, 75 comprise the texture database and 25 are query images. The texture database was constructed with the idea that the images should span the texture spectrum (regular, near regular, irregular, near stochastic and stochastic) with about 80% or 60/75 images falling into the near regular, irregular and near stochastic range. The image database is shown in Figure 4-1 and the list of images and collections from which they were obtained is shown in Table 4-1.

The edge histogram descriptor (EHD), as well as the homogeneous texture descriptor (HTD) and the sparse texture descriptor (STD), were implemented in Matlab ver. 7.10, on a MacBook Pro with a 2.53 GHz Intel Core 2 Duo processor. The details of their implementation are found in the following sections.

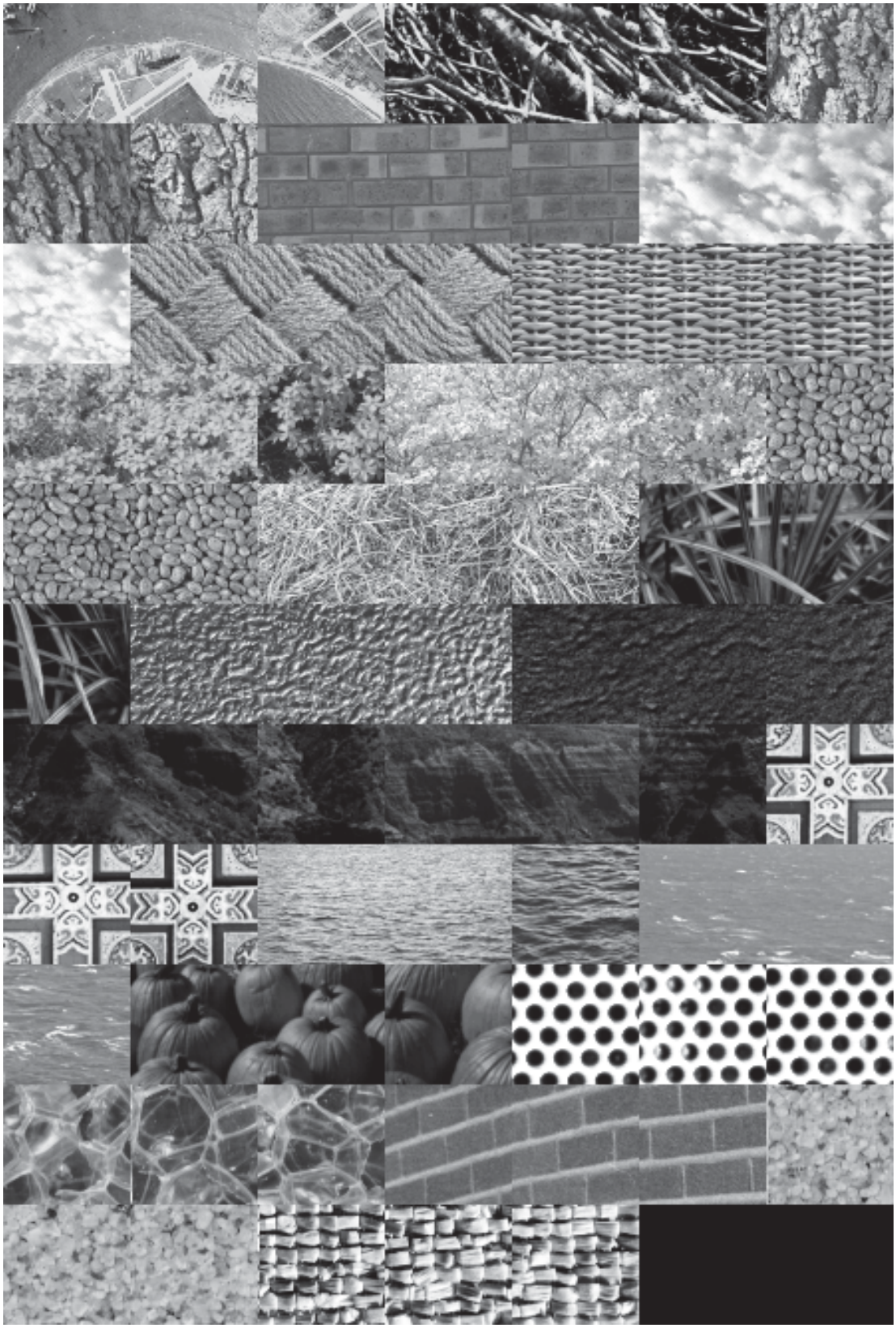

Figure 4-1: Texture database of 75 images cropped from USC SIPI and MIT VisTex collections.

Table 4-1: List of database and query images tested in chapter 5.

Database and Query List	Images
DB: 1-3/Query: 1	SIPI USC Aerial San Diego (North Island NAS)
DB: 4-6/Query: 2	MIT VisTex Bark.0003
DB: 7-9/Query: 3	MIT VisTex Bark.0009
DB: 10-12/Query: 4	MIT VisTex Brick.0000
DB: 13-15/Query: 5	MIT VisTex Clouds.0000
DB: 16-18/Query: 6	MIT VisTex Fabric.0011
DB: 19-21/Query: 7	MIT VisTex Fabric.0014
DB: 22-24/Query: 8	MIT VisTex Flowers.0002
DB: 25-27/Query: 9	MIT VisTex Flowers.0004
DB: 28-30/Query: 10	MIT VisTex Food.0000
DB: 31-33/Query: 11	MIT VisTex Grass.0001
DB: 34-36/Query: 12	MIT VisTex Leaves.0000
DB: 37-39/Query: 13	MIT VisTex Metal.0004
DB: 40-42/Query: 14	MIT VisTex Sand.0002
DB: 43-45/Query: 15	MIT VisTex Terrain.0000
DB: 46-48/Query: 16	MIT VisTex Terrain.0003
DB: 49-51/Query: 17	MIT VisTex Tile.0004
DB: 52-54/Query: 18	MIT VisTex Water.0000
DB: 55-57/Query: 19	MIT VisTex Water.0002
DB: 58-60/Query: 20	MIT VisTex Food.0010
DB: 61-63/Query: 21	SIPI USC Texture 1.5.02 Hexagonal hole array
DB: 64-66/Query: 22	SIPI USC Texture 1.1.13 Brodatz Plastic bubbles (D112)
DB: 67-69/Query: 23	SIPI USC Texture 1.4.01 Brick wall
DB: 70-72/Query: 24	SIPI USC Texture 1.1.07 Brodatz Beach sand (D29)
DB: 73-75/Query: 25	SIPI USC Texture 1.2.10 Brodatz Raffia (D84 H.E.)

4.2 Edge Histogram Descriptor Implementation

The edge histogram descriptor (EHD), as described in [21] and section 3.1, partitions an image into 16 contiguous sub-images, shown in Figure 4-2. All images used in the implementation of the texture database as well as query images are of size 64 by 64 pixels, therefore, each of the 16 sub-images is of size 16 by 16 pixels.

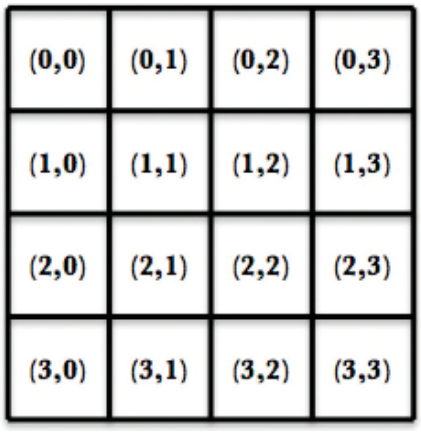

Figure 4-2: Example of Edge Histogram Descriptor partition of an image into 16 sub-images.

The specific details in implementing the EDH are as follows:

1. Create 5 copies of the image and perform convolution with the 5 Sobel masks from Figure 3-1 to produce 5 edge maps.
2. Calculate the maximum intensity value for each pixel given the 5 edge maps. Form an index matrix called MaxIndex as the index for the maximum of the 5 edge maps for all pixels, i.e. MaxIndex is a 64 by 64 matrix with integer values in the range [1,5] indicating which edge map had the maximum value for each pixel.

3. Partition the MaxIndex per Figure 4-2 and calculate a 5-element histogram for each sub-image.
4. Create the 80-element EHD feature vector by concatenating the 16, 5-element histograms into one vector.

The preceding can be more compactly described in Figure 4-3, which is followed by an example of the process in Figure 4-4.

The EHD algorithm implementation was nearly identical to that described in [21] and section 3.1, with two notable exceptions. The edge detectors implemented are Sobel masks in Figure 3-1, not the four element masks implemented in MPEG-7 and shown in Figure 3-3. As described in chapter 3, the Sobel masks have many desirable properties and are easier to implement than 4 element masks. The second change was not quantizing the values in the 80-element feature vector. As mentioned in section 3.1, the MPEG-7 implementation of the EHD quantizes the values in the 80-element feature vector to 3 bits each. Since memory considerations are not an issue for such a small image database, quantization was considered unnecessary. However, not quantizing could lead to slightly better performance than the MPEG-7 implementation. Similarity matching of the feature vectors is then accomplished using a Euclidean K-Nearest Neighbor (K-NN) search.

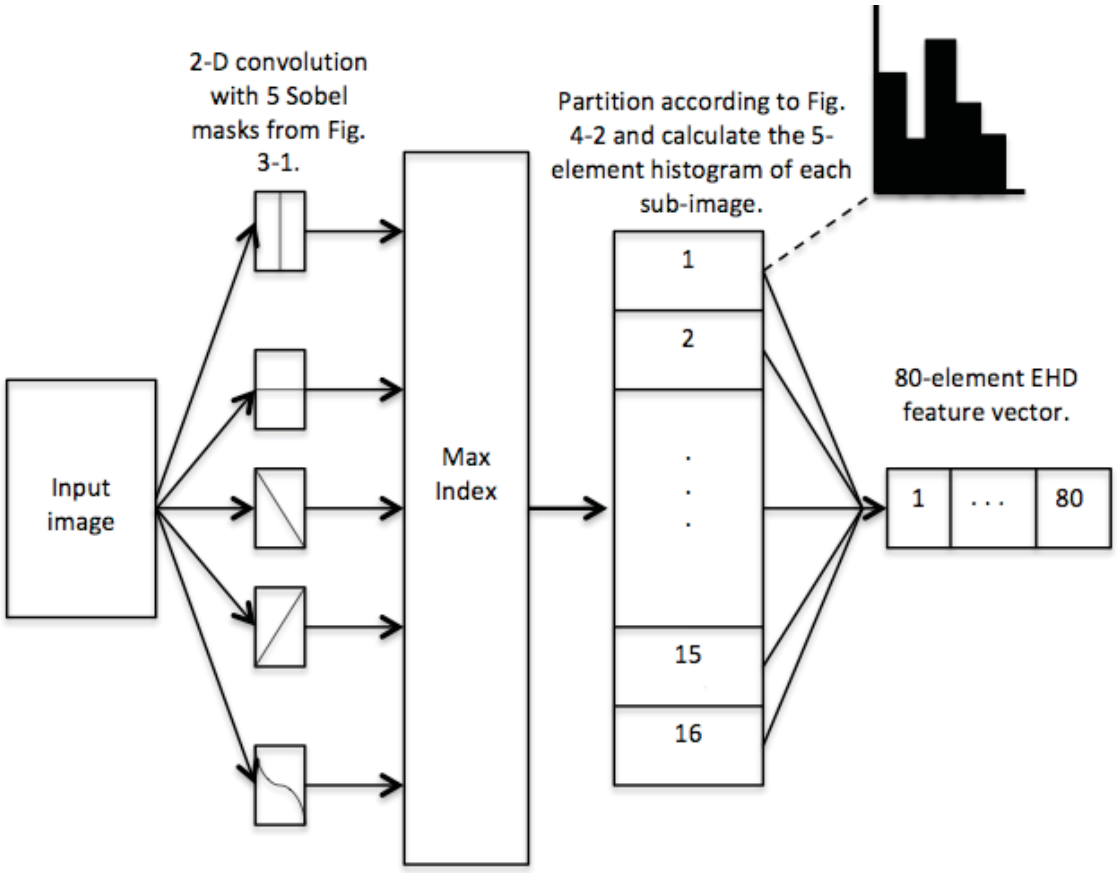

Figure 4-3: Flow diagram of the Edge Histogram Descriptor implementation.
The input image is filtered with the 5 Sobel masks from Figure 3-1 producing 5 edge maps. For each pixel across the 5 edge maps, the maximum value is obtained and the index is stored in MaxIndex (i.e. which of the 5 filters the maximum value belongs to). The MaxIndex image is then partitioned into 16 sub-images described by Figure 4-2 and a 5-element histogram is calculated for each. The 80-element feature vector is then formed.

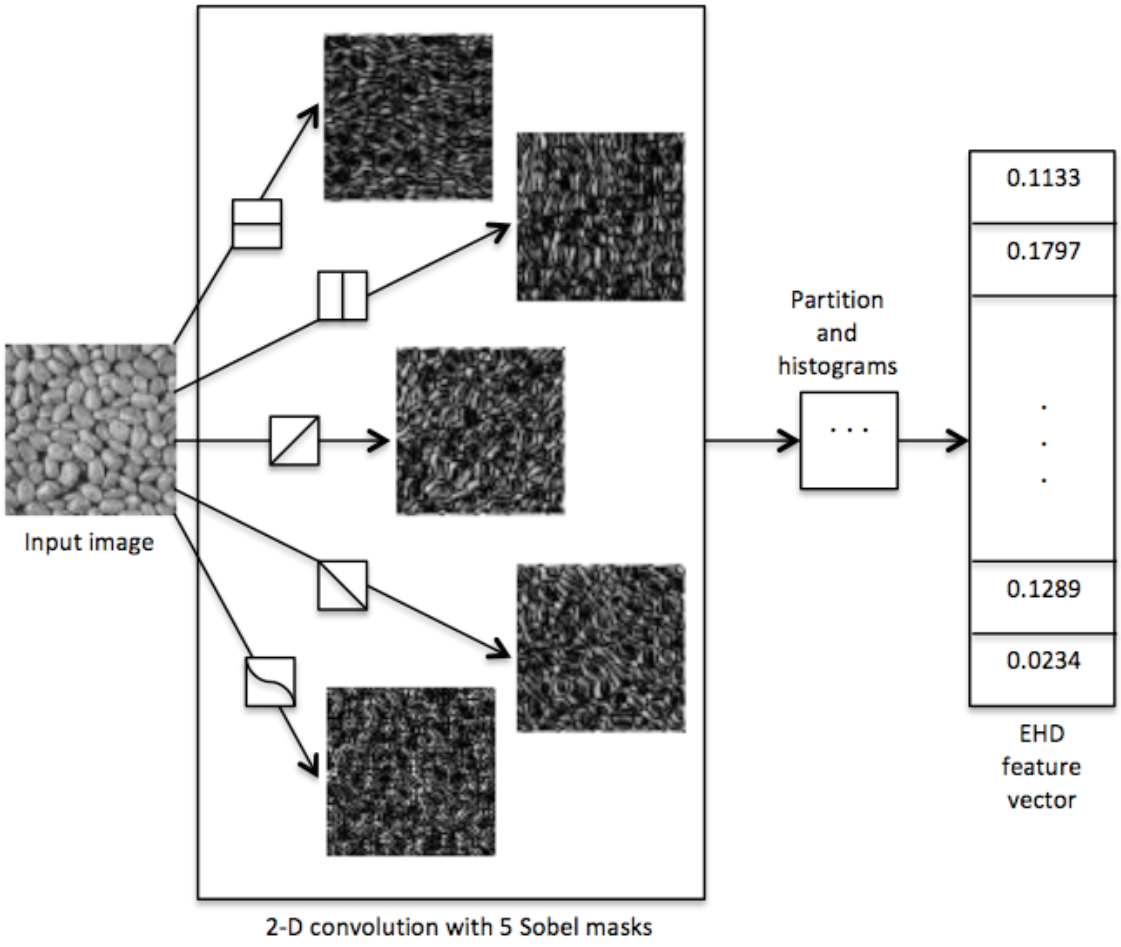

Figure 4-4: Example of the Edge Histogram Descriptor implementation with MIT VisTex "Food.0000" as the input image.
The figure shows the 5 edge maps created by filtering the input with the Sobel masks and the structure of the feature vector.

4.3 Homogeneous Texture Descriptor Implementation

The homogeneous texture descriptor (HTD) was implemented as described in [31, 20]. The implementation given in [31, 20] describes a 2D Gabor filter in polar coordinates in the spatial frequency plane given by equation (4.1),

$$G_{P_{s,r}}(\omega,\theta) = \exp\left[\frac{-(\omega-\omega_s)^2}{2\sigma_{\omega_s}^2}\right]\cdot\exp\left[\frac{-(\theta-\theta_r)^2}{2\sigma_{\theta_r}^2}\right], \tag{4.1}$$

where $G_{P_{s,r}}(\omega,\theta)$ is the Gabor filter at the s^{th} radial and r^{th} angular index. The standard deviation, or spread, in the radial and angular directions is given by σ_{ω_s} and σ_{θ_r} respectively. With equation (4.1) in mind, the spatial frequency plane is partitioned into 30 channels comprised of 6 angular and 5 radial bands. Each angular band has 30° bandwidth. The radial bands are calculated starting with the highest center frequency at $\omega_0 = \frac{3}{4}$, followed by $\omega_s = \omega_0 2^{-s}$ with $s \in \{0,1,2,3,4\}$. An example of this partition with one channel shaded is shown in Figure 3-4. As there is overlap between adjacent filters it is necessary to devise a method to reduce this redundancy. The strategy implemented is found in [31] and partitions the normalized spatial frequency plane into 30 channels, such that the overlap of adjacent filters occurs at the one half maximum magnitude in both radial and angular directions. The results of the partitioning given as the polar coordinate centers, bandwidths, and standard deviations for each of the 30 channels are found in Tables 4-2 and 4-3.

Table 4-2: Gabor filter bank parameters in the radial direction.

Radial Index (s)	0	1	2	3	4
Center Frequency (ω_s)	3/64	3/32	3/16	3/8	3/4
Spatial bandwidth	1/32	1/16	1/8	1/4	1/2
σ_{ω_s}	$\frac{1}{64\sqrt{2\ln 2}}$	$\frac{1}{32\sqrt{2\ln 2}}$	$\frac{1}{16\sqrt{2\ln 2}}$	$\frac{1}{8\sqrt{2\ln 2}}$	$\frac{1}{4\sqrt{2\ln 2}}$

Table 4-3: Gabor filter bank parameters in the angular direction.

Angular Index (r)	0	1	2	3	4	5
Center Frequency (θ_r)	0°	30°	60°	90°	120°	150°
Angular bandwidth	30°	30°	30°	30°	30°	30°
σ_{θ_r}	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$	$\frac{30^\circ}{2\sqrt{2\ln 2}}$

Using the parameters found in Tables 4-2 and 4-3 and equation (4-1), the Gabor filter bank can be implemented. The feature vector is then extracted from a given texture image as follows:

1. Calculate the intensity mean and variance of the input image.
2. Take the 2D DFT of the input image using an FFT algorithm.
3. Multiply the Fourier transformed input image with each of the 30 Gabor filters to produce the filtered output.
4. Calculate the mean and variance of each of the 30 filtered outputs.
5. Form the 62-element feature vector comprised of: the intensity mean and variance, followed by the 30 energy means and 30 energy variances.

This procedure is shown in Figure 4-5 along with an example shown in Figure 4-6. As with the EHD, the HTD feature vector values are not quantized and similarity matching is accomplished using a Euclidean K-NN search.

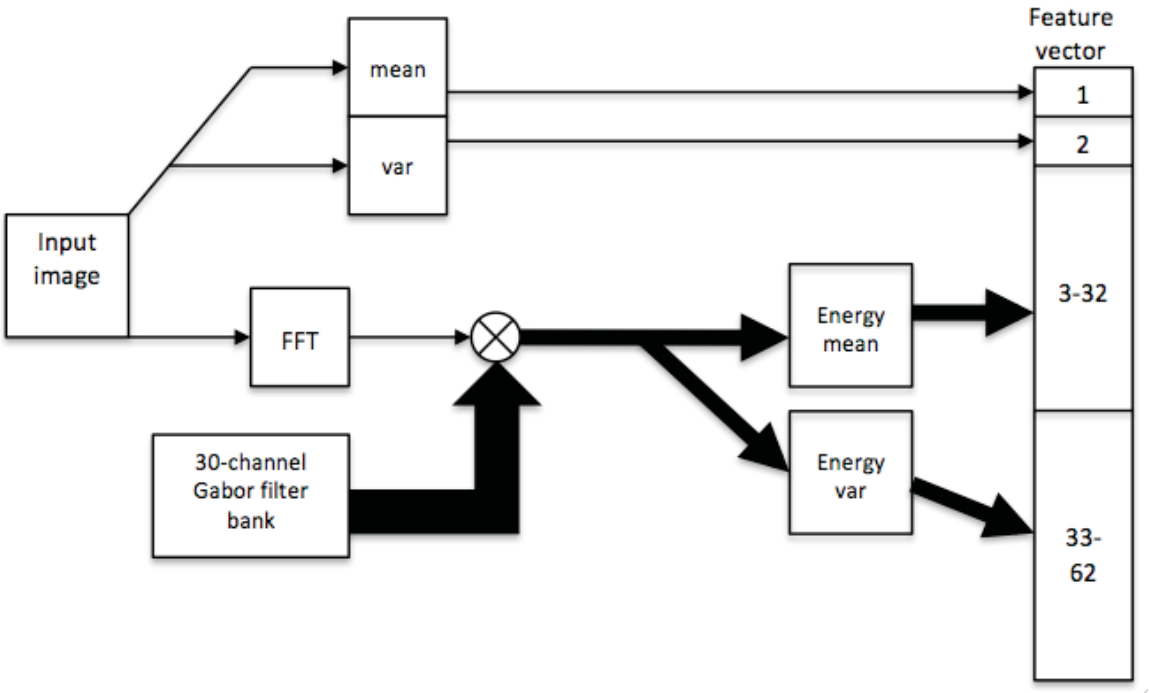

Figure 4-5: Flow diagram of the Homogeneous Texture Descriptor implementation. The 2D Fourier transformed input image is filtered with each of the 30 Gabor filters and the mean and variance is extracted from each filtered output. The input image intensity mean and variance along with the 30 energy means and 30 energy variances form the HTD feature vector.

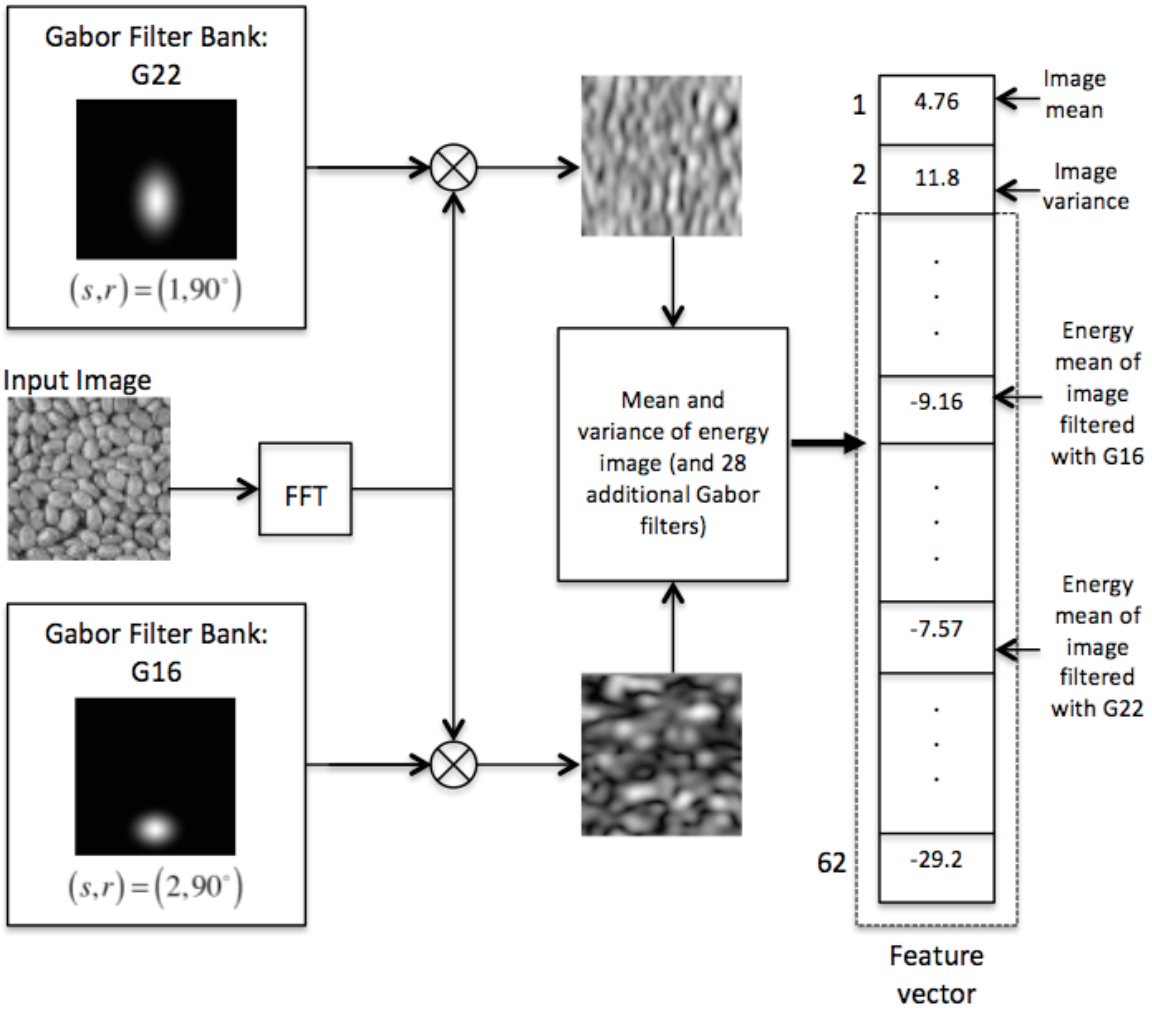

Figure 4-6: Example of feature extraction using the Homogeneous Texture Descriptor. This figure shows the input image MIT VisTex Food.0000 and two of 30 Gabor filters in the spatial frequency domain. Next, the filtered outputs are displayed (in the spatial domain) and the extraction of values to form the feature vector.

4.3 Sparse Texture Descriptor Implementation

The sparse texture descriptor (STD) is implemented according to the *Sparseland* Model [9] that assumes that an overcomplete dictionary exists in which an atom is a learned image element and that a patch of an image can be approximated as a linear combination of a few atoms. In this implementation, an atom is an 8 by 8 pixel image learned by sampling all possible 8 by 8 sub-images of the entire texture database comprised of the 75 images found in Figure 4-1. In order to speed up the runtime, only $1/10^{th}$ of all possible *298,289*-8 by 8 sub-images are considered to create each atom (the entire texture database as a single image is of size 320x960, therefore the total number of 8x8 overlapping sub-images is $(320-8+1)(960-8+1)=298289$). This is carried out using the K-SVD algorithm described in section 3.3.3. The K-SVD algorithm is initialized using an overcomplete 2D separable discrete cosine transform (DCT) dictionary of size 64 by 256 shown in Figure 4-7. While there are many options for initializing the dictionary in K-SVD, the 2D DCT is simple and fast to construct as well as provides a very low initial mean square error (MSE) [9]. This allows for a small number of iterations of the K-SVD algorithm to run in order to reach the desired tolerance. The pursuit algorithm implemented is orthogonal matching pursuit (OMP) due to its simplicity and speed. The K-SVD algorithm was run for 20 iterations, taking 19.8 minutes to build the dictionary.

The image training set consists of the 75 database images, shown in Figure 4-1. The K-SVD algorithm is run on this set to produce the database dictionary shown in Figure 4-8. Using this dictionary and the OMP algorithm, the sparse approximations for each database and query image are calculated and consist of at most 8 atoms (each 8 by 8

pixels) or 512 pixels. The space spanned by the sparse database images is searched to find the closest neighbor to a given query image. The fact that at most 8 atoms are used in OMP to create the sparse images implies that up to 512 pixels (8 atoms x 64 pixels each) of 4096, or 12.5% of the pixels are used to approximate each image.

Due to implementation problems with Matlab it was not possible to directly apply a Euclidean K-NN search when conducting a query, since these variables were stored as *sparse* structures. In order to circumvent this issue, while still searching with the sparse data, the matrices are made 'full', from 8 atom-sparse structures into 256x64 matrices in which only 512 entries are non-zero. The 64-singular values of these full matrices are obtained and form the vectors used to represent the database and query images. Using these 64-singular value vectors, Euclidean K-NN is then applied when performing a query. This procedure is more compactly described in Figure 4-9.

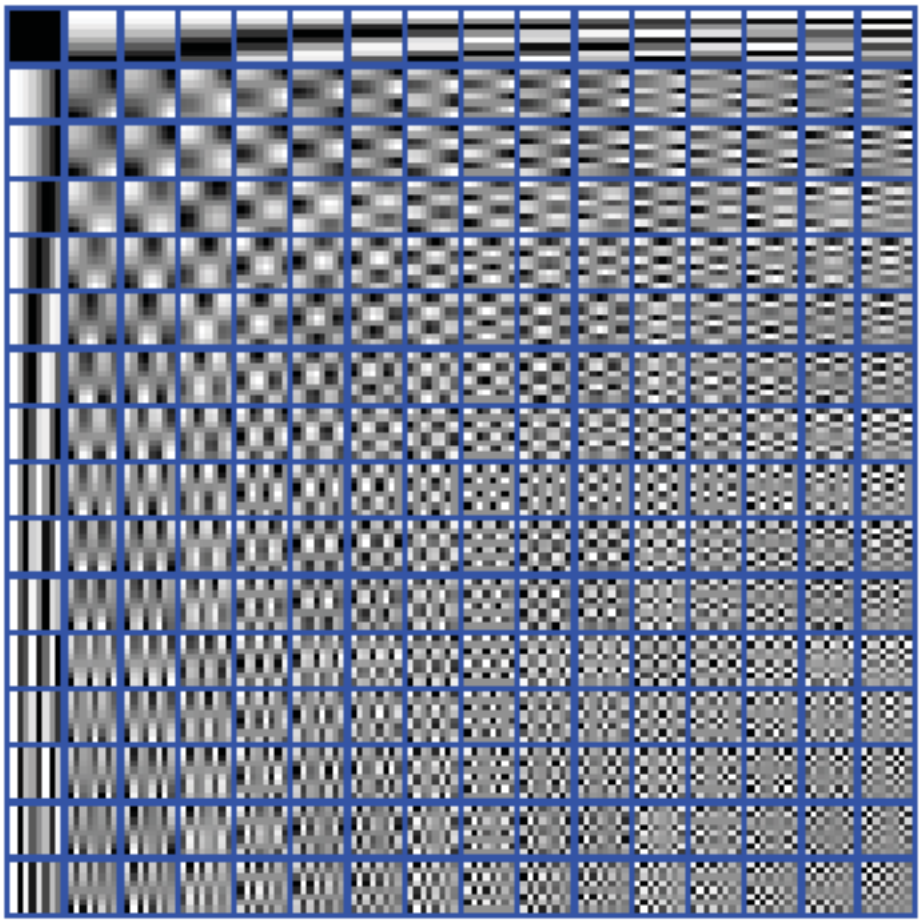

Figure 4-7: Image of the separable 2D-DCT dictionary used to initialize the K-SVD algorithm. The dictionary is displayed as 16x16 atoms, each one 8x8 pixels. In fact, the matrix is implemented and stored with dimensions 64x256. The 256 columns or atoms are formed into square patches for display only.

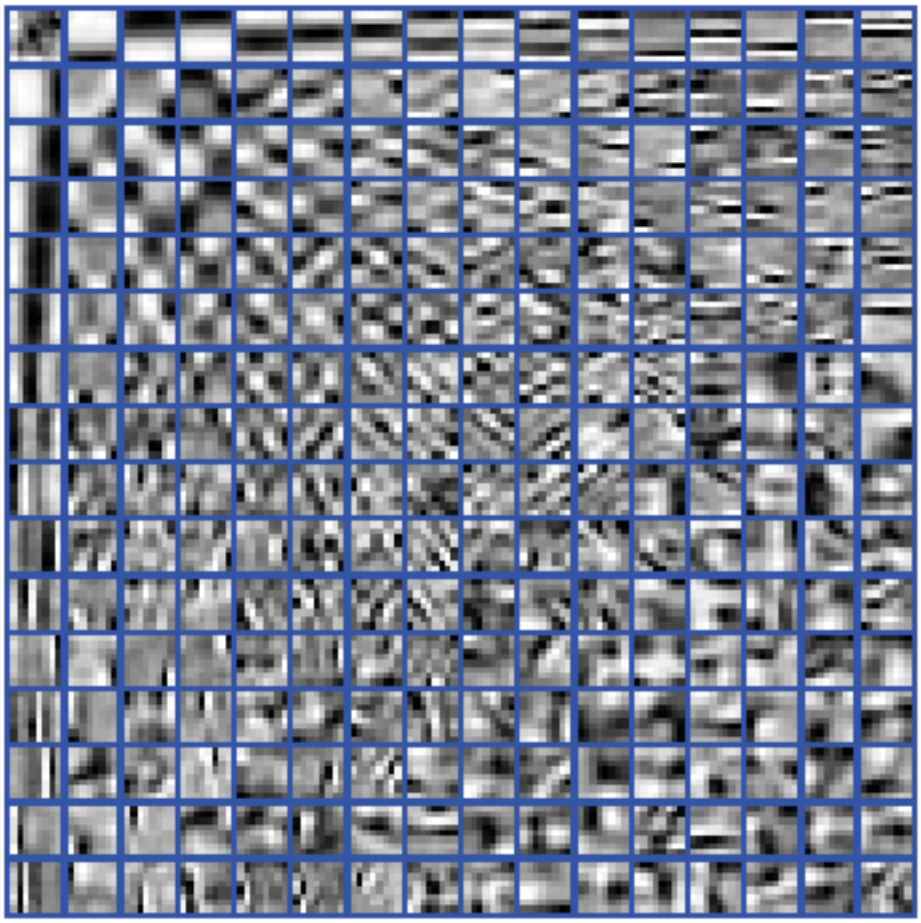

Figure 4-8: Image of the learned dictionary implemented in the Sparse Texture Descriptor. The dictionary is obtained by applying the K-SVD algorithm on the 75 texture database images. The dictionary is displayed as 16x16 atoms, each one 8x8 pixels. In fact, the matrix is implemented and stored with dimensions 64x256. The 256 columns or atoms are formed into square patches for display only.

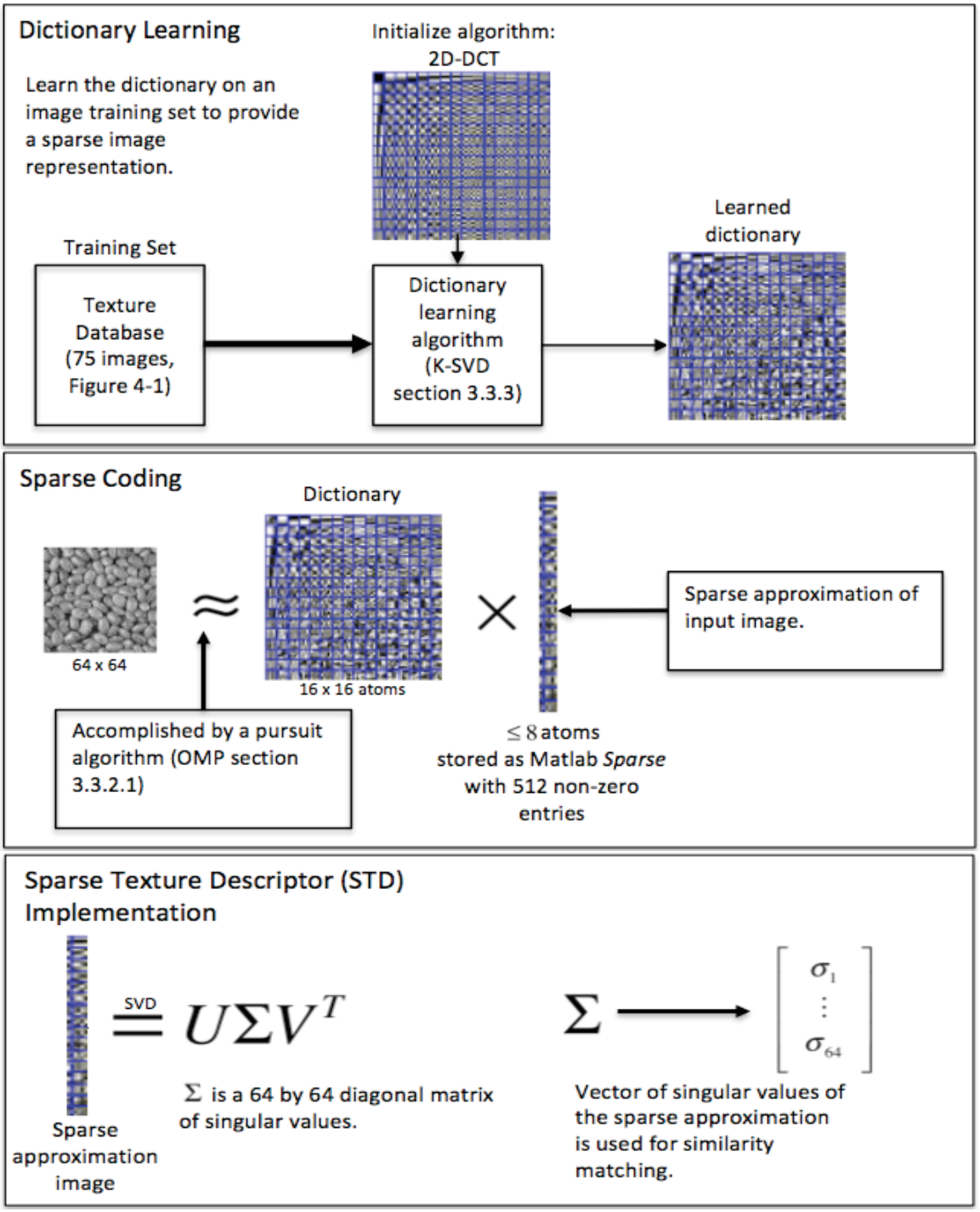

Figure 4-9: Flow diagram of the Sparse Texture Descriptor implementation. (Top) Description of how to obtain the dictionary used in the pursuit algorithm. (Middle) Description of the implementation used to obtain the sparse representation of an image through the use of the orthogonal matching pursuit algorithm. (Bottom) The vector used for similarity matching in the Sparse Texture Descriptor.

Chapter 5: Results

To test each of the three feature extraction methods, 25 of the same queries were performed on each method. The testing followed the *query by example* paradigm for evaluating and comparing the performance of each method. In query by example, feature vectors are extracted for each image in the database as well as the set of test images with which queries are made. The query feature vector is then matched to its closest vectors in the database feature space. Often the similarity measure of choice is the l_1 or l_2 norm but specifically designed statistical measures may be used and can result in higher performance. The results obtained in this thesis use a Euclidean K-nearest neighbor search (K-NN).

A ground truth (GT) set must be established for each query image for comparisons to remain objective. Each of the 25 query images has 3 corresponding database images all of which were cropped from the same larger image. These three images comprise the GT set for a particular query. Slight variations exist among the images in the GT set and between GT and query due to non-uniform illumination, the image not being flat, or simply differences due to where the sub-images were cropped. To a human observer this type of matching would be simple but it is non-trivial for a computer. In order to find and rank the most similar images to the query, a search is conducted of the closest feature vectors to that of the query and the results are ranked using l_2 distance.

The results in this thesis are obtained primarily to assess whether or not a sparse model is applicable for the purpose of image retrieval, since the MPEG-7 homogeneous texture descriptor (HTD) and edge histogram descriptor (EHD) are well documented [8, 10, 20, 21, 26, 31]. In order to compare the sparse texture descriptor (STD) with EHD

and HTD and assess how well each performs in image retrieval a few metrics are developed. The first metric, called *Top 6*, simply counts a successful retrieval as having at least one GT match in the top six nearest neighbor results. This is tallied for each query and normalized by the total number of queries. *Top N* can be useful metric when one is quickly browsing the results of a query and does not wish to look any further than the first page of results.

The second metric, called *retrieval rate* (RR), is very popular and defined by,

$$RR(q) = \frac{NF(\alpha, q)}{GT(q)}, \tag{5.1}$$

where *NF* is the number of GT images found within the first $\alpha = N \cdot GT(q)$ results for query *q*, and $N > 0$ (and in most cases an integer) [21]. The factor α determines the tolerance of the metric. Retrieval rate takes on values between 0 and 1, where a 0 indicates the query found no GT images in the first α results and 1 indicates that all GT images were obtained in α results. Retrieval rate can be averaged over the number of all queries (NQ) giving the *average retrieval rate* (ARR) defined by,

$$ARR = \frac{1}{NQ} \sum_{q=1}^{NQ} RR(q). \tag{5.2}$$

One disadvantage of ARR is that when implementing large image databases the size of the ground truth set will most likely not be the same value for every query. According to [21], varying *GT(q)* introduces a bias for certain queries. However, in the image retrieval experiments in this thesis, the GT is fixed at 3 for all queries.

The metric used by the MPEG group to evaluate the visual descriptors in MPEG-7 is called the *average normalized modified retrieval rank* (ANMRR). First, a rank function is defined which penalizes missing members of the GT set from a query and is given by,

$$Rank^*(k) = \begin{cases} Rank(k), & \text{if } Rank(k) \le K(q) \\ 1.25K, & \text{if } Rank(k) > K(q) \end{cases} \tag{5.3}$$

where *Rank(k)* is the position in which image *k* is retrieved according to, in this case, Euclidean K-NN search. The number $K(q)$ acts as a tolerance similar to α in RR and is given by,

$$K(q) = \min\{4 \cdot GT(q),\ 2 \cdot \max[GT(q), \forall q]\}. \tag{5.4}$$

With this we define *average rank* (AVR) by,

$$AVR(q) = \frac{1}{GT(q)} \sum_{k=1}^{GT(q)} Rank^*(k). \tag{5.5}$$

Then to minimize the influence of varying *GT(q)* (not an issue in this thesis), the *modified retrieval rank* (MRR) is defined by,

$$MRR(q) = AVR(q) - 0.5[1 + GT(q)]. \tag{5.6}$$

This is then normalized with respect to *GT(q)* giving us the *normalized modified retrieval rank* (NMRR),

$$NMRR(q) = \frac{MRR(q)}{1.25K - 0.5[1 + GT(q)]}. \tag{5.7}$$

Finally, this is averaged over all queries giving us the *average normalized modified retrieval rank* (ANMRR) expressed by,

$$ANMRR = \frac{1}{NQ} \sum_{q=1}^{NQ} NMRR(q). \tag{5.8}$$

NMRR takes on values between 0 and 1, where 0 indicates that all GT images, for a particular query, have been retrieved with rank less than *K*, and a 1 indicates all GT images have retrieval rank greater than *K*.

Table 5-1 displays the performance of each feature extraction method under the three metrics *Top 6*, *ARR* and *ANMRR*. Keep in mind that *Top 6* has retrieval rate values between 0 and 1, where 1 implies all queries have at least one GT image in the top six ranked results. Thus, a value closer to 1 indicates better performance, similarly with *ARR*. Conversely, with ANMRR, a value closer to 0 indicates better performance.

Table 5-1: Retrieval results of the three texture feature extraction methods based on 25 queries.

Texture Descriptor	Top 6	ARR	ANMRR
EHD	0.760	0.533	0.541
HTD	1.000	0.853	0.159
STD	0.960	0.720	0.277

The results for ARR were obtained using a tolerance of $\alpha = 2 \cdot GT(q) = 6$. Similarly, for ANMRR the tolerance value chosen is $K(q) = 6 \ \forall q$. The results in table 3 clearly indicate, in all three metrics, that the HTD performs best, followed closely by STD and then EHD. While the data set in this experiment is on the small side it is doubtful that EHD would surpass STD or either surpass HTD in performance should a much larger data set be used. The results reported in [21] were 0.77 (ARR) for HTD and 0.34 (ANMRR) for EHD. The database used in [21] had 10000 images for both descriptors and the authors used the normalized ℓ_1 norm for similarity matching. An ℓ_1 norm was also tested in this thesis but the results were nearly identical to those in Table 5-1.

An analysis of the results was performed but no definitive conclusions could be made as to why EHD or STD showed poor performance on certain textures. Out of the 25 queries, EHD had 8 in which no GT images were found in *Top 6*. An attempt was made to pre-filter the images with a Gaussian mask to smooth the less dominant edges with the

idea that the stronger edges would be extracted and make for better results. However, the EHD performance actually decreased.

The query images for which STD failed to provide a single GT in Top 6 do not seem to have anything apparent in common. All such images would appear to be from the irregular to the stochastic end of the texture spectrum. However just as many near stochastic query images found all 3 GT images in the top 6 results.

Chapter 6: Conclusion

This thesis presented texture analysis and specifically feature extraction methods for content-based image retrieval (CBIR). The texture feature extraction methods included in MPEG-7 as well as an application of sparse representation to image retrieval were explored. The three methods, edge histogram descriptor (EHD), homogeneous texture descriptor (HTD), and the sparse texture descriptor (STD) were implemented in Matlab and their performance was analyzed with a set of metrics.

As was stated previously, a major motivation for this thesis is to attempt to answer whether or not sparse representation could be used in image retrieval and how well it performed as compared to the well-established MPEG-7 texture descriptors. To do this, a dictionary is learned on the database images that are then sparse coded using a pursuit algorithm and said dictionary. A query image is likewise sparse coded and the space of sparse images is searched and ranked using a distance measure. The results in chapter 5 show that, at least on the small set of images used in this experiment, the STD works relatively well in comparison to the MPEG-7 texture descriptors. The fact that HTD outperformed STD should not be unexpected as the MPEG group of experts worked for many years on texture feature extraction and Gabor wavelets culminating in the HTD. That being said, this thesis should prove, at the minimum, that image retrieval in the sparse representation domain deserves a closer look.

An issue not touched on by this thesis, with respect to sparse image retrieval, is the computational requirement of learning the dictionary on very large data sets. For the small database and small images (grayscale, 64 by 64 pixels) used in this thesis, building the dictionary, searching the database, and storing the sparse vectors presented no

computational or storage issues. However, for image databases of any practical size, the dictionary would need to be learned initially, which could take a considerable amount of time. Then, as images are added to the database, the dictionary would need to be periodically updated to reflect these changes, otherwise the bounds guaranteeing sparsity using a pursuit algorithm would no longer hold. Also, modifying K-SVD or any other dictionary learning algorithm would be necessary to allow for images of varying size. The dictionary itself could become unreasonably large to store as a single structure. Thus, some balance of error and dictionary management would need to be investigated to make this feasible.

Many more issues may come to light with further research in sparse image retrieval. The computational issues described could prohibit a practical implementation. However, the recent application of sparse representation to inverse image processing problems and the field of compressed sensing would not have been computationally feasible even a decade ago. It should also be mentioned that while this thesis presented sparse image retrieval for textures the same implementation could be used for retrieval of natural images with no modifications. In future work it would be interesting to compare the retrieval performance of the STD against groups of features, such as color and texture descriptors, on natural images.

References

[1] Aharon, M., Elad, M. and Bruckstein, A., "K-SVD : An Algorithm for Designing of Overcomplete Dictionaries for Sparse Representation." *Signals* **54**, pp. 4311-4322 (2006).

[2] Bouman, C.A., "Markov Random Fields and Stochastic Image Models." *IEEE International Conference on Image Processing,* October 95, Washington, D.C., (1995).

[3] Chen, J., Cao, H., Prasad, R., Bhardwaj, A. and Natarajan, P., "Gabor features for offline Arabic handwriting recognition." *Proceedings of the 8th IAPR International Workshop on Document Analysis Systems DAS 10,* pp. 53-58 (2010).

[4] Chen, S.S., Donoho, D.L. and Saunders, M.A., "Atomic Decomposition by Basis Pursuit." *SIAM Journal on Scientific Computing* **20**, pp. 33-61 (1998).

[5] Cross, G.R. and Jain, A.K., "Markov random field texture models." *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**, pp. 25-39 (1983).

[6] Datta, R., Joshi, D., Li, J. and Wang, J.Z., "Image retrieval: Ideas, Influences, and Trends of the New Age." *ACM Computing Surveys* **40**, pp. 1-60 (2008).

[7] Daugman, J.G., "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters." *Journal of the Optical Society of America A Optics and image science* **2**, pp. 1160-1169 (1985).

[8] Deselaers, T., Keysers, D. and Ney, H., "Features for image retrieval: an experimental comparison." *Information Retrieval* **11**, pp. 77-107 (2007).

[9] Elad, M., *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, 1st Edition, Springer, New York, NY, (2010).

[10] Eom, M. and Choe, Y., "Fast Extraction of Edge Histogram in DCT Domain based on MPEG7." *Engineering and Technology* **9**, pp. 209-212 (2005).

[11] Eom, M. and Choe, Y., "Edge Histogram Descriptor in Wavelet Domain Based on JPEG2000." *IEICE Transactions on Communications E90-B*, pp. 3745-3747 (2007).

[12] Field, D.J., "Relations between the statistics of natural images and the response properties of cortical cells." *Journal of the Optical Society of America A Optics and image science* **4**, pp. 2379-2394 (1987).

[13] Gonzalez, R.C. and Woods, R.E., *Digital Image Processing*, 3rd Edition, Prentice Hall, Upper Saddle River, N.J., (2008).

[14] Haindl, M. "Texture segmentation using recursive Markov random field parameter estimation." *Proceedings of the 11th Scandinavian Conference on Image Analysis*, (1999).

[15] Haralick, R.M., Dinstein, I. and Shanmugam, K., "Textural features for image classification." *IEEE Transactions On Systems Man And Cybernetics* **3**, pp. 610-621 (1973).

[16] Haykin, S, *Adaptive Filter Theory,* 4th Edition, Prentice Hall, Upper Saddle River, N.J., 2002, pp. 213-229.

[17] Liu, Y., Lin, W.C. and Hays, J.H., "Near Regular Texture Analysis and Manipulation." *Proceedings of SIGGRAPH 04* (2004).

[18] Ma, L., Wang, Y. and Tan, T., "Iris Recognition Based on Multichannel Gabor Filtering." *ACCV2002: The 5th Asian Conference on Computer Vision*, 23-25 January, Melbourne, Australia, pp. 1-5 (2002).

[19] Mallat, S. and Zhang, Z., "Matching pursuit with time-frequency dictionaries." *IEEE Transactions on Signal Processing* **41**, pp. 3397-3415 (1993).

[20] Manjunath, B.S. and Ma, W.Y., "Texture features for browsing and retrieval of image data." *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**, pp. 837-842 (1996).

[21] Manjunath, B.S., Ohm, J.R., Vasudevan, V.V. and Yamada, A., "Color and texture descriptors." *IEEE Transactions on Circuits and Systems for Video Technology* **11**, pp. 703-715 (2001).

[22] Martins, A.D.M., Torres, W. and Filho, D.A., "A new method for multi-texture segmentation using neural networks." *Neuron* **1**, (2002).

[23] Mishra, A., Aloimonos, Y. and Fah, C.L., "Active Segmentation with Fixation." *Computer* **06**, pp. 468-475 (2009).

[24] Moore, A.W., "K-means and Hierarchical Clustering." *Science* **8**, pp. 1-24 (2004).

[25] The Moving Picture Experts Group (MPEG), 2012, http://mpeg.chiariglione.org/, "site accessed on 5 January 2012".

[26] Park, D.K., Jeon, Y.S. and Won, C.S., "Efficient use of local edge histogram descriptor." *Proceedings of the 2000 ACM workshops on Multimedia MULTIMEDIA 00* pp. 51-54 (2000).

[27] Natarajan, B.K., "Sparse Approximate Solutions to Linear Systems." *SIAM Journal on Computing* **24**, pp. 227 (1995).

[28] Needell, D. and Vershynin, R., "Signal Recovery From Incomplete and Inaccurate Measurements Via Regularized Orthogonal Matching Pursuit." *IEEE Journal of Selected Topics in Signal Processing* **4**, pp. 310-316 (2010).

[29] Picard, R., Graczyk, C., Mann, S., Wachman, J., Picard, L., and Campbell, L., Vision Texture Database (Vis Tex), MIT Media Lab, Massachusetts Institute of Technology, http://vismod.media.mit.edu/vismod/imagery/VisionTexture/, 2002, "site accessed on 12 November 2011".

[30] Renninger, L.W. and Malik, J., "When is scene identification just texture recognition?" *Vision Research* **44**, pp. 2301-2311 (2004).

[31] Ro, Y.M.R., Kim, M.K., Kang, H.K.K., Manjunath, B.S.M. and Kim, J.K., "MPEG-7 Homogeneous Texture Descriptor." *ETRI Journal* **23**, pp. 41-51 (2001).

[32] Rogowitz, B.E., Frese, T., Smith, J.R., Bouman, C.A., and Kalin, E., "Perceptual image similarity experiments." *Proceedings of SPIE* **3299**, pp. 576-590 (1998).

[33] Rui, Y., Huang, T.S. and Chang, S.F., "Image Retrieval: Current Techniques, Promising Directions, and Open Issues." *Journal of Visual Communication and Image Representation* **10**, pp. 39-62 (1999).

[34] The USC-SIPI Image Database, Signal and Image Processing Institute (SIPI), University of Southern California, http://sipi.usc.edu/database/database.php, "site accessed on 12 November 2011".

[35] Taponecco, F. and Alexa, M., "Vector Field Visualization using Markov Random Field Texture Synthesis." *Proceedings of the Joint Eurographics IEEE TCVG Symposium on Visualization VisSym 03*, pp. 195-202 (2003).

[36] Tropp, J.A., "Greed is Good: Algorithmic Results for Sparse Approximation." *IEEE Transactions on Information Theory* **50**, pp. 2231-2242 (2004).

[37] Tropp, J.A. & Gilbert, A.C., "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit." *IEEE Transactions on Information Theory* **53**, pp. 4655-4666 (2007).

[38] Unser, M., "Texture classification and segmentation using wavelet frames." *IEEE Transactions on Image Processing* **4**, pp. 1549-1560 (1995).

[39] Weldon, T., Higgins, W. and Dunn, D., "Efficient Gabor-Filter design for Texture Segmentation." *Pattern Recognition* **29**, pp. 2005-2006 (1996).

[40] Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T. S., and Yan, S., "Sparse Representation for Computer Vision and Pattern Recognition." *Proceedings of the IEEE* **98**, pp. 1031-1044 (2010).