Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 52561, Pages 1–11
DOI 10.1155/ASP/2006/52561

Texture Classification Using Sparse Frame-Based Representations

Karl Skretting and John Håkon Husøy

Department of Electrical and Computer Engineering, University of Stavanger, 4036 Stavanger, Norway

Received 31 August 2004; Revised 20 April 2005; Accepted 2 June 2005

A new method for supervised texture classification, denoted by frame texture classification method (FTCM), is proposed. The method is based on a deterministic texture model in which a small image block, taken from a texture region, is modeled as a sparse linear combination of frame elements. FTCM has two phases. In the design phase a frame is trained for each texture class based on given texture example images. The design method is an iterative procedure in which the representation error, given a sparseness constraint, is minimized. In the classification phase each pixel in a test image is labeled by analyzing its spatial neighborhood. This block is represented by each of the frames designed for the texture classes under consideration, and the frame giving the best representation gives the class. The FTCM is applied to nine test images of natural textures commonly used in other texture classification work, yielding excellent overall performance.

1. INTRODUCTION

Most surfaces exhibit texture. For human beings it is quite easy to recognize different textures, but it is more difficult to precisely define a texture. Under all circumstances, a texture may be regarded as a region where some elements or primitives are repeated and arranged according to a placement rule. Tuceryan and Jain [1] list more possible definitions and give a comprehensive overview of texture classification. Possible applications can be grouped into (1) texture analysis, that is, finding some appropriate properties for a texture, (2) texture classification, that is, identifying the texture class in a homogeneous region, and (3) texture segmentation, that is, finding a boundary map between different texture regions of an image. The boundary map may be used for object recognition and scene interpretation in areas such as medical diagnostics, geophysical interpretation, industrial automation, and image indexing. Finally, (4) texture synthesis, that is, generating artificial textures to be used for example in computer graphics or image compression. Some examples of applications are presented in [2–6].

Typically, texture classification algorithms have two main parts: a local feature vector is found, which is subsequently used for texture classification or segmentation. The methods for feature extraction may be loosely grouped as statistical, geometrical, model-based, and signal processing (filtering) methods [1]. For the filtering methods the feature vectors are often built as variance estimates, local energy measures, for each of the subbands of a filter bank. Also, there are numerous classification or pattern recognition methods available. The Bayes classifier is probably the most common one [7, 8]. The min- or max-selector is a simple one that can be used if each entry in the feature vector measures the similarity to, or corresponds to, a texture class. Nearest-neighbor classification, vector quantization (codebook vectors representing each class) [9] and learning vector quantization (LVQ) (codebook vectors defining the decision borders) [10–12], neural networks, watershed-based algorithm [13], and support vector machines (SVM) [14] are other methods.

One approach to texture classification may be to focus on the feature extraction part and make it easy to decide the texture class from the feature vector [15]. The opposite approach is to make the feature extraction as simple as possible, for example by feeding the gray-level values for the pixels in image blocks directly to the classifier [16]. The FTCM belongs to the first approach, as the overall classification scheme is quite similar to the scheme used in [11], the main distinction being that we have replaced the filter part by a *sparse representation part*. On the other hand we also recognize relationships to the opposite approach. The SVM scheme, as used in [16], finds a set of support vectors for each texture and this set identifies a hyperplane which separates the given texture from the rest of the textures, while FTCM finds a set of frame vectors for each texture and this set is trained to efficiently represent the given texture by a sparse linear combination, thus identifying the texture. Also,

FTCM has much in common with texture classification using vector quantization [9]. Actually, FTCM may be regarded as a generalization of the vector quantization approach.

This paper is organized as follows. Sparse frame-based representations are briefly explained in Section 2. Section 3 presents the texture model and gives a motivation for the frame texture classification method. FTCM is a *supervised* texture classification method and it has two main parts. Firstly, training is done to build the frames based on some example images for each texture class, see Section 4. Secondly, in Section 5, we describe the classification or segmentation using these frames to label the pixels of a test image. Finally, in Section 6, the experimental results are presented both for synthetic textures based on the texture model and for natural textures.

2. SPARSE FRAME-BASED REPRESENTATIONS

A set of N-dimensional vectors, spanning the space $\mathbb{R}^N$, $\{\mathbf{f}_k\}_{k=1}^K$, where $K \geq N$, is a *frame*. In this paper frames are represented as follows. A frame is given by a matrix $\mathbf{F}$ of size $N \times K$, $K \geq N$, where the columns are the frame vectors, $\mathbf{f}_k$. A column vector of N signal samples is formed from a 2-dimensional image block (size $N_1 \times N_2$) that is simply rearranged into a column vector (length $N = N_1 N_2$). The column vector is denoted by $\mathbf{x}_l$ to indicate that it is one out of L available signal blocks, such signal (image) blocks can be represented by a weighted sum of frame vectors

$$\tilde{\mathbf{x}}_l = \sum_{k=1}^{K} w_l(k)\mathbf{f}_k = \mathbf{F}\mathbf{w}_l. \tag{1}$$

This is a *signal expansion* that, depending on the selection of weights, $w_l(k)$, may be an exact or an approximate representation of the signal block. The weights, $w_l(k)$, can be represented by a column vector, $\mathbf{w}_l$, of length K. It is convenient to collect the L signal vectors and the corresponding weight vectors into matrices,

$$\begin{aligned} \mathbf{X} &= [\mathbf{x}_1 \;\; \mathbf{x}_2 \;\; \cdots \;\; \mathbf{x}_L], \\ \mathbf{W} &= [\mathbf{w}_1 \;\; \mathbf{w}_2 \;\; \cdots \;\; \mathbf{w}_L]. \end{aligned} \tag{2}$$

The synthesis equation (1) may now be written as

$$\tilde{\mathbf{X}} = \mathbf{F}\,\mathbf{W}. \tag{3}$$

In a *sparse representation* many of the weights in the signal expansion (1) are zero. To quantify the degree of sparseness we use the number s, which is the number of nonzero weights allowed in the sparse representation of each signal block, $\mathbf{x}_l$. s is the same for all signal blocks.

A frame can be designed or trained to give a good sparse representation of a set of L training vectors. A linear combination of basis vectors from an arbitrary basis of $\mathbb{R}^N$ can be used to represent each vector in the training set. Such representations will in general be dense, that is, they have N nonzero coefficients. A large frame, using all the L training vectors as frame vectors, can be used to give the ultimate sparse representation, each of the training vectors can be represented by only one frame vector. In this work we use rather small frames where $N < K \ll L$, typically $2N \leq K \leq 4N$. Each of the training vectors can now be well *approximated* by a sparse linear combination of the frame vectors, allowing only s nonzero weights to be used in the expansion. The K frame vectors can be designed to minimize the sum of representation errors for a given sparseness.

The problem of finding the sparse weight vector, for a given sparseness, such that the 2-norm[1] of the residual is minimized, is an NP-hard problem [17]. Many practical solutions employ greedy *vector selection* algorithms, such as matching pursuit (MP), orthogonal matching pursuit (OMP), and order recursive matching pursuit (ORMP). When reading this paper, it is not necessary to know (the details of) these methods. They are thoroughly described elsewhere, [18–24]. All we need to know is that the vector selection algorithm used here, which by the way is ORMP, finds the weights in a sparse representation.

3. THE TEXTURE MODEL

Textures are often described by random models and statistical properties, [25–27]. Random models often seem to capture the essential properties of the textures quite well, as can be seen from the textures synthesized by these models [28], and obviously most natural textures have a random element. We will here present a deterministic texture model which will fit many periodic textures quite well. Based on this model the frame texture classification method (FTCM) emerges as a natural method for texture classification. The main result of this section is that it is reasonable to model a small texture image block as a sparse linear combination of frame elements. The results-oriented reader may wish to jump to Section 4.

The idea behind the proposed texture model is quite simple. A texture is modeled as a tiled floor, where all tiles are identical. The color, or gray-level, at a given position on the floor is given by an underlying continuous periodic two-dimensional function which we denote by $c(x, y)$, an image is a regular sampling of this function. In this section we will show that all image blocks can be represented as a linear combination of only four elements, where the four elements are taken from a set, that is, a frame, with a finite number of elements. The FTCM directly uses this model. In the training phase it finds a frame for each texture and in the classification phase representations, or approximations, of blocks from a test image are found as linear combinations of four elements. Because of this close connection we may say that the model explains the good performance of FTCM, or alternatively, the good performance of FTCM validates the model.

One period of the periodic function $c(x, y)$ defines a quadratic tile where each side has unit length, that is, $c(x, y) = c(x - \lfloor x \rfloor, y - \lfloor y \rfloor)$. In this model the function is

[1] In this paper we use the 2-norm, $\|\mathbf{x}\|^2 = \sum_{n=1}^{N} x(n)^2$, for vectors and the trace or Frobenius norm, $\|\mathbf{A}\|^2 = \sum_i \sum_j A(i, j)^2$, for matrices.

defined by a finite number of control points placed on the tile. This is illustrated in Figure 1 where two complete tiles and parts of their neighboring tiles are shown. The 16 control points on each tile are regularly distributed on a 4×4 grid, the control points can be labeled c_{ij} where only the indexes are shown in the figure. Generally, in this model, the $M = M_1 M_2$ control points are placed on a rectangular $M_1 \times M_2$ grid. The color of any point on a tile (on the floor) is given as a *bilinear interpolation* of the closest control points, that is, $c(x, y) = a_1 c_{i_1 j_1} + a_2 c_{i_2 j_2} + a_3 c_{i_3 j_3} + a_4 c_{i_4 j_4}$. The bilinear interpolation is actually a *convex combination*, with $a_1 + a_2 + a_3 + a_4 = 1$ and $0 \leq a_k \leq 1$. For example, the color value for the center of a tile in Figure 1 is $c(x, y) = (1/4)c_{22} + (1/4)c_{23} + (1/4)c_{32} + (1/4)c_{33}$. We also note that some parts of $c(x, y)$ within a tile need control points from neighboring tiles in forming the interpolation. We let the coordinate system be aligned to match a tile, such that the center of the first tile is given by $(x, y) = (1/2, 1/2)$, and the corners are $(0, 0), (1, 0), (0, 1)$, and $(1, 1)$.

Samples of $c(x, y)$ on a rectangular sampling grid, not necessarily aligned with the coordinate system implied by the first tile, constitute the digital texture image. By choosing

(i) the number and positions of *control points* in a tile,
(ii) the gray-level value (color) of each of the control points,
(iii) the orientation of the sampling grid relative to the coordinate system aligned with the tiles, denoted by angle α, and finally,
(iv) the distance between neighboring sampling points, denoted by δ, in the sampling grid,

we obtain a digital texture image. Figure 2 illustrates sampling. In this example we have $\delta = 0.187$ and $\alpha = 15$ degrees.

The texture model described above has the capability of generating a wide variety of textured images, some examples are shown in Figure 6. We will now look closer on a small block of pixels from the texture image. In Figure 2 a 3×3 block ($N = 9$ pixels) is marked. This block forms a size-N vector, $\mathbf{x} = [x(1), x(2), \ldots, x(9)]^T$. How the numbering is done is not important, but we may assume that $x(1)$ is the upper left pixel and the rest are numbered columnwise. We note that the location of pixel $x(1)$ may be anywhere on the floor, but since translations by unit lengths up and down will give exactly the same value for $x(1)$, and also the vector $\mathbf{x}$ will be unchanged by such translations, the location of $x(1)$ can be restricted to be on the first tile.

Having the texture image specified as above, that is, by control points and by a sampling grid, we realize that all possible vectors $\mathbf{x}$ can be formed by translating the position of $x(1)$ within a tile. An infinite number of different vectors $\mathbf{x}$ can be formed. For gray-level images this set of vectors is a subset of the space $\mathbb{R}^N$. We may say that this set defines the texture. The challenge now is to make an efficient description of this set in a way that makes it easy to decide whether a test vector belongs to this set or not. In the following we argue that all vectors from this infinite set, corresponding to a specific texture, can be represented as a linear (convex) combination of four frame vectors taken from a finite subset

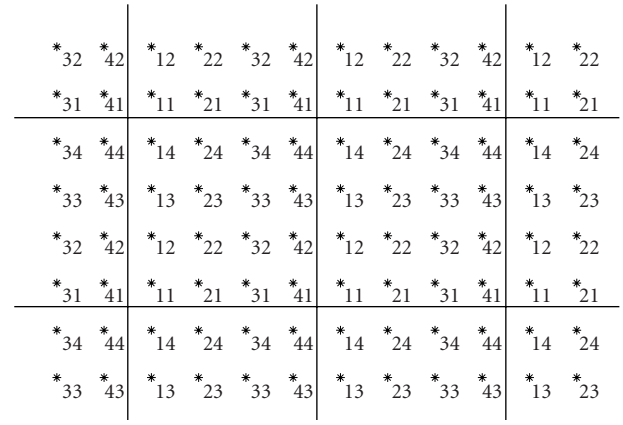


Figure 1: Two complete tiles of a tiled floor. The control points are marked and labeled.

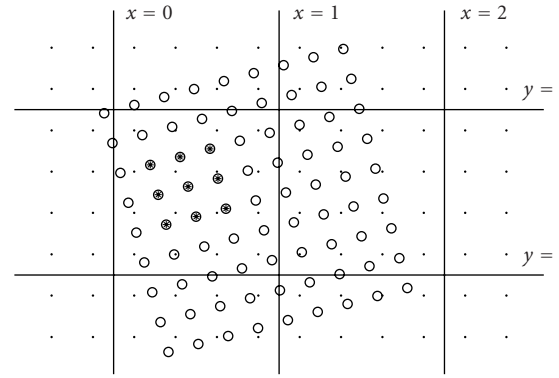


Figure 2: A part of a tiled floor with sample points. The control points are marked as dots, and the sample points (center of the image pixels) as small circles.

of vectors containing at most MN^2 vectors, where again M denotes the number of control points in each tile. This finite set is a frame and its elements are frame vectors. Note that the frame vectors span the space $\mathbb{R}^N$, but adding a sparseness constraint during representation makes them "span" only a subspace, which contains all the $\mathbf{x}$ vectors. This subspace is the union of a finite number of s-dimensional spaces, where s is the number of frame vectors allowed in the sparse representation, here $s = 4$.

In Figure 2 the marked upper left pixel, $x(1)$, is above and to the right of control point c_{13}. Its value is a linear combination of the values in the four neighboring control points c_{13}, c_{23}, c_{14}, and c_{24}. If $x(1)$ is translated anywhere within the small box with these control points as corners, it is still a linear combination of the same control points. At a corner $x(1)$ will take the value of the control point. This observation can also be stated as follows: Within a small rectangular box of the tile, the value $x(1)$ will be a linear combination of its values at the corner points. This is true as long as no horizontal or vertical line through any control point passes through the

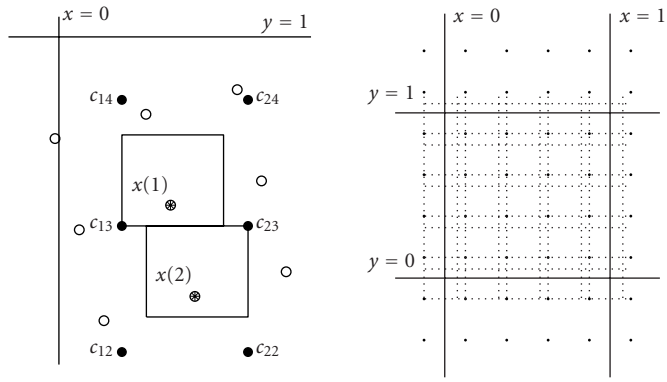


FIGURE 3: The left part shows a smaller part of a tiled floor, six control points and some nearby sample points are plotted. The right part shows the tile divided into small boxes such that when $x(1)$ is within one box the vector $\mathbf{x} = [x(1), x(2)]^T$ is a convex combination of its value at the corner points.

small box. The same statement is obviously also valid for another pixel, for example $x(2)$ below $x(1)$.

The left part of Figure 3 illustrates the situation when we consider two points simultaneously. The points are entries in the vector $\mathbf{x} = [x(1), x(2)]^T$, in this example $N = 2$. Translating this vector means that we translate both its entries the same distance vertically and horizontally. The position of $x(2)$ is given by the position of $x(1)$ and their relative distance is given by the sampling grid. This implies that the positions of all entries in $\mathbf{x}$, and thus the value of $\mathbf{x}$, are given by the position of $x(1)$ within the tile. In the figure a box is plotted around $x(1)$, such that when $x(1)$ moves within this box $x(2)$ moves within the box plotted around $x(2)$. The neighboring control points will not change for either of the pixels. This can also be stated as follows: placing $x(1)$ within a small rectangular box of the tile, the value of vector $\mathbf{x}$ will be a linear combination of its values at the corner points. This is true as long as the box around $x(1)$ is so small that all of the entries of the vector do not involve new control points. The dotted lines in the right part of Figure 3 divide the tile into such boxes. Placing $x(1)$ on an intersection between the dotted lines, the corresponding vector $\mathbf{x}$ can be stored as a frame vector $\mathbf{f}_k$. Collecting all these frame vectors into a frame, we observe that any $\mathbf{x}$ generated by this texture model can be represented as a linear combination of four frame vectors.

This reasoning can easily be extended to a larger vector $\mathbf{x}$ of length N. We will now find how many small boxes the tile should be divided into for this case. First we move $x(1)$, and the sampling grid to which $x(1)$ is attached, vertically within the tile. Everywhere when the position of an entry of vector $\mathbf{x}$ crosses one of the horizontal lines that can be drawn through a control point, we draw a horizontal line through $x(1)$. This will give at most M_2N horizontal lines. Then we move $x(1)$ horizontally within the tile. Everywhere when the position of an element of vector $\mathbf{x}$ crosses one of the vertical lines that can be drawn through a control point, we draw a vertical line through $x(1)$. This will give at most M_1N vertical lines. Placing $x(1)$ at one of the $M_1NM_2N = MN^2$ intersections between a horizontal and vertical line, we will have a corresponding vector $\mathbf{x}$. These vectors constitute the elements of a finite frame. All vectors $\mathbf{x}$, with $x(1)$ anywhere on the tile, and which are the elements of the set that defines this specific texture image, can be represented as a linear (convex) combination of four frame vectors taken from the frame containing at most MN^2 vectors.

To take advantage of this model in a practical way some shortcuts are taken. First, we note that finding the correct frame for an example texture is not possible unless we have available the model parameters and even then the number of frame vectors will often be quite large. By using fewer frame vectors, $K \ll MN^2$, we accept that the test vector will only be approximated by the sparse representation. Secondly, only a limited number of combinations of the frame vectors should be used in the sparse representation. In this model the frame vectors are the $\mathbf{x}$ vectors taken when $x(1)$ is placed on the corners of the many small boxes that a tile can be divided into. The four frame vectors used in a sparse representation should belong together; they should be the four corners of one of these small boxes. By allowing *any* combination of the frame vectors to be used, we do not have to consider a relative position of the frame vectors. Thirdly, the representation (approximation) according to the model should strictly be a bilinear interpolation between four points. It would be just as reasonable to define the periodic function $c(x, y)$ by a linear interpolation between *three* control points (in a triangular grid).

Taking these three shortcuts, we can use the frame design method, first presented in [29] and used for texture images in [30], to design a frame that represents a texture class. The method is briefly described in the next section.

4. FRAME DESIGN

The task of designing, or training, a frame is to find its frame vectors such that they can be used to efficiently represent the texture class. The frames are designed based on available sets of texture example images corresponding to the texture classes under consideration, not on the usually unknown parameters in the texture model.

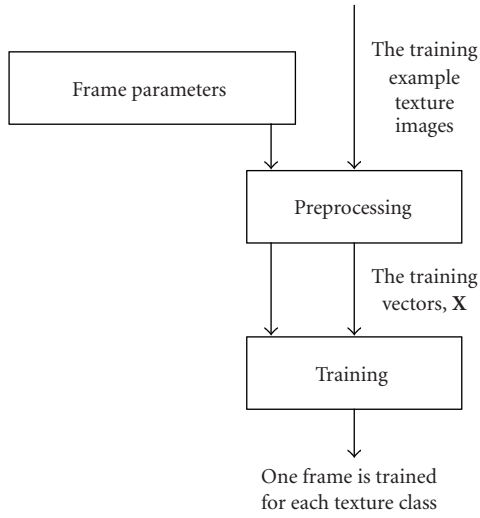


FIGURE 4: The setup for training of frames in FTCM is very similar to the general frame design setup, [30].

If the number of different texture classes is C, we design C frames, which are denoted $\mathbf{F}^{(i)}$ for texture class $i = 1, 2, \ldots, C$. A frame is designed to achieve the best possible sparse representation of the training vectors for a particular texture, that is, the example image(s) of the texture. Training is a computationally demanding process, but it is done before classification and only once for each texture class. The process has three main steps as shown in Figure 4.

The very first step in the FTCM training phase is to decide the frame parameters. These parameters can be chosen quite freely.

(i) The shape, usually rectangular, and the size of the block around each pixel. The pixels within this block are organized as a column vector of length N.
(ii) The number of vectors in the frame, K. As a rule of thumb, found from the comprehensive experiments done, we may use $N \leq K \leq 5N$.
(iii) The sparseness to use, represented by the number of frame vectors used in the sparse representation, s. The main objective is to choose a value of s that provides a good discrimination of the different textures. The experiment part of this paper confirms that the model suggested values $s = 3$ and $s = 4$ are suitable values.

Having set the frame parameters, the next step is to build the training vectors from the texture example images. As suggested before, this can be as simple as rearranging the pixels from small image blocks, which may partly overlap each other, into column vectors, or it can be more involved. The sets of training vectors are arranged into $N \times L$ matrices, as in (2), and denoted by $\mathbf{X}^{(i)}$ for texture class $i = 1, 2, \ldots, C$. Later, during classification, the test vectors should of course be formed by the same procedure as for the training vectors.

In the training the parameter set, N, K, and s, is fixed. For each frame to design, $\mathbf{F}^{(i)}$, we use the corresponding set of training vectors, $\mathbf{X}^{(i)}$, generated from the example images. For notational convenience we skip the superscript indexes below. As explained in Section 2 the synthesis equation can be written as $\widetilde{\mathbf{X}} = \mathbf{FW}$. We want to find the frame, $\mathbf{F}$, of size $N \times K$, and the sparse coefficient vectors, $\mathbf{w}_l$, that minimize the sum of the squared errors. The objective function to be minimized is

$$J = J(\mathbf{F}, \mathbf{W}) = \|\mathbf{X} - \widetilde{\mathbf{X}}\|^2 = \|\mathbf{X} - \mathbf{FW}\|^2. \tag{4}$$

Finding the optimal solution to this problem is difficult if not impossible. We split the problem into two parts to make it more tractable, similar to what is done in the GLA design algorithm for VQ codebooks [31]. The iterative solution strategy presented below results in good, but in general suboptimal, solutions to the problem.

The algorithm starts with a user-supplied initial frame $\mathbf{F}_0$, usually K arbitrary vectors from the set of training vectors, and then improves it by iteratively repeating two main steps.

(1) $\mathbf{W}_t$ is found by vector selection using frame $\mathbf{F}_t$. The objective function is $J(\mathbf{W}) = \|\mathbf{X} - \mathbf{F}_t\mathbf{W}\|^2$, and a sparseness constraint is imposed on $\mathbf{W}$.
(2) $\mathbf{F}_{t+1}$ is found from $\mathbf{X}$ and $\mathbf{W}_t$, where the objective function is $J(\mathbf{F}) = \|\mathbf{X} - \mathbf{FW}_t\|^2$. This gives

$$\mathbf{F}_{t+1} = \mathbf{X}\mathbf{W}_t^T\left(\mathbf{W}_t\mathbf{W}_t^T\right)^{-1}. \tag{5}$$

Then we increment t and go to Step 1.

t is the iteration number. The first step is suboptimal due to the use of practical vector selection algorithms, while the second step finds the $\mathbf{F}$ that minimizes the objective function.

In a texture classification context the frame concept has been used together with the discrete wavelet transform, see [7, 14, 32, 33]. We must point out that the frame in FTCM has a different role. In the discrete wavelet frame transform context the frame is used as the analysis filter bank, the frame arises when the wavelet subbands are not down sampled. If a perfect reconstruction synthesis filter bank exists, many can exist [34], the outputs of the analysis filter bank can be regarded as an alternative representation of the image. In FTCM the analysis filter bank is replaced by a matching pursuit algorithm, and the frame is used to synthesize the signal as in (1). Also, the FTCM uses several frames, each giving one element of the feature vector, as opposed to the filter bank approach where each subband gives one element of the feature vector.

5. CLASSIFICATION

Texture classification of a test image, containing regions of different textures, is the task of classifying each pixel of the test image to belong to a certain texture. This is done by generating test vectors from the test image. The classifying process for the FTCM is illustrated in Figure 5.

A test vector is represented in a sparse way using each of the different frames that were trained for the textures under consideration, the set of C frames $\{\mathbf{F}^{(i)}\}$. Each sparse representation of each test vector $\mathbf{x}_l$ gives a representation error,

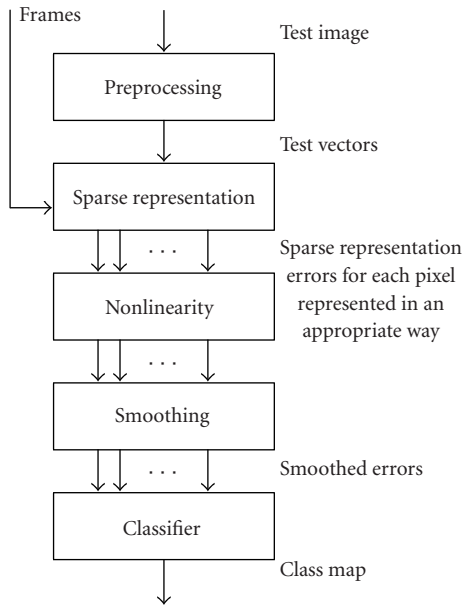


Figure 5: The setup for the classification approach in FTCM. This setup is similar to a common setup in texture classification used in [11].

$\mathbf{r}_l^{(i)} = \mathbf{x}_l - \mathbf{F}^{(i)}\mathbf{w}_l^{(i)}$. Each test vector $\mathbf{x}_l$ corresponds to a pixel of the test image. Classification consists of selecting the index i for which the norm squared of the representation error, $\|\mathbf{r}_l^{(i)}\|^2 = \mathbf{r}_l^{(i)T}\mathbf{r}_l^{(i)}$, is minimized.

Direct classification based on the norm squared of the representation error for each test vector (pixel) gives quite large classification errors, but the results can be substantially improved by smoothing the error images. Smoothing is reasonable since it is likely that neighboring pixels belong to the same texture. For smoothing Randen and Husøy [11] concluded that the separable Gaussian lowpass filter is the better choice, and this is also the filter used here. The unit pulse response for the 1D kernel of this filter is

$$h_G(n) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(1/2)(n^2/\sigma^2)}. \tag{6}$$

The parameter σ gives the bandwidth of the smoothing filter. The effect of smoothing is mainly that more smoothing gives lower resolution and better classification within the texture regions. The cost is often more classification errors along the borders between different texture regions.

To improve texture segmentation a nonlinearity may be included before the smoothing filter is applied, [35]. The nonlinearity is applied on $\|\mathbf{r}_l^{(i)}\|^2$, that is, a scalar property is calculated by a nonlinear function $f(\|\mathbf{r}_l^{(i)}\|^2)$. The function may be the square root to get the magnitude of the error, or the inverse sine of the magnitude which gives the angle between signal vector and its sparse approximation, or a logarithmic operation. Experiments we have done [30] indicate that usually the logarithmic nonlinearity is the better choice.

6. EXPERIMENTS

6.1. Synthesized textures

The experiments presented here demonstrate the close connection between the texture model and the FTCM. Let us define two tiles that both give braided textures, tile A defined by a 4×4 ($M = 16$) grid of control points and tile B defined by a 6×6 ($M = 36$) grid of control points. The intensity values for the control points are

$$A = \begin{bmatrix} 0.5 & 0 & 0.5 & 0 \\ 1 & 0 & 1 & 1 \\ 0.5 & 0 & 0.5 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{7}$$

From Figure 6 we see that the black and white bands are wider on tile A than on tile B, tile B will have more of the gray background. Based on these tiles we define six textures using different values for the sample distance δ and the rotation angle α. We generate example images of each texture, which are used for training of the frames. We also make a test image, Figure 6, consisting of segments from all the six texture classes. Visually the textures seem quite similar and are quite difficult to distinguish from each other just by looking at them.

Many frames were designed, using different sets of frame parameters, for each of the six textures. We always used image blocks of size 5×5 to form the training vectors of length $N = 25$, while the number of frame vectors K and sparseness s varied. We used these frames to classify the test image; the results are shown in Figure 7. Here we have used a quite narrow lowpass filter, $\sigma = 2$, and the classification results are almost perfect. For most cases the number of wrongly classified pixels is less than 1%, often less than 0.5%, which means that only some few pixels along the texture borders are wrongly classified. Even the vector quantization case, $s = 1$, does quite well when the number of frame (codebook) vectors, K, is large. We observe that the smaller frames, $K \leq 50$, do quite well for sparseness choices $s = 3$ and $s = 4$, which is the sparseness suggested by the model of Section 3. Also without filtering (results not shown here) more than 90% of the pixels were correctly classified for $s > 1$ and $K \geq 150$, while for $s = 1$ and $K = 200$, 70% of the pixels were correctly classified. Without filtering we clearly saw that as the number of frame vectors increased the results improved, as we would expect from the model.

The conclusion so far is not surprising: when the textures are generated in accordance with the model, texture classification using FTCM, motivated by the model, achieves excellent results.

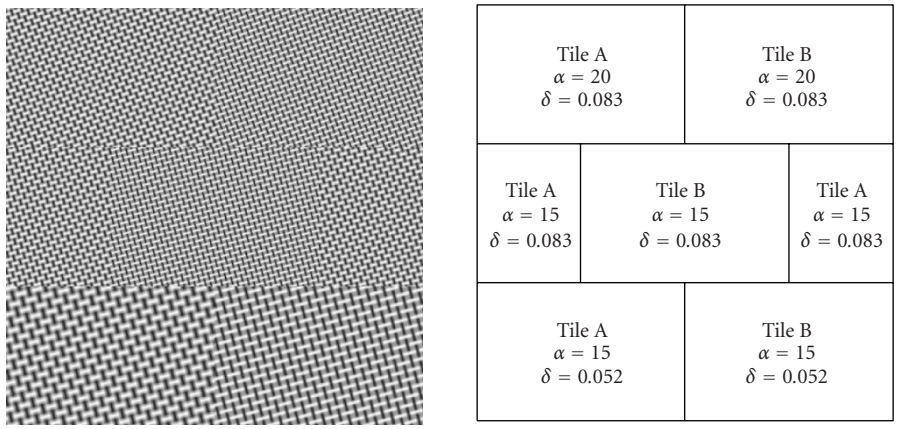


Figure 6: The synthesized test image on the top and its reference below. The reference tells how the different regions of synthesized test image are built.

6.2. Natural textures

We also test the FTCM on some real data, and we choose to use the nine test images of Randen and Husøy [11]. These consist of 77 different natural textures, taken from three different and commonly used texture sources: the Brodatz album, the MIT Vision Texture Database, and the MeasTex Image Texture Database. The test images are denoted by (a) to (i) and are shown in [11, Figure 11], where also a more detailed description of the test images can be found.[2] Due to space considerations only test image (c) is shown in this paper, Figure 10(a). The same test images were also used in other papers [8, 13, 16, 36, 37].

The procedures of Sections 4 and 5 were used. The first step is to design the $C = 77$ class-specific frames from the example images of all the texture classes under consideration. Many different frame parameter sets were used in our experiments. This was done to find which parameter sets perform best on natural textures. We used 5×5 and 7×7 pixel blocks, giving training and test vectors of lengths $N = 25$ and $N = 49$. The number of frame vectors in each frame were $K = \{25, 50, 100, 200\}$ for $N = 25$ and $K = \{50, 100\}$ for $N = 49$. This gives six different sizes for the frames. The numbers of frame vectors in the sparse representation were from $s = 1$ to $s = 6$. For each parameter set a frame was designed for all the texture classes of interest, the number of training vectors was $L = 10000$. The design of all the frames needed several days of computer time, one to five minutes for each frame, but this task must be done only once.

The texture classification capabilities of the FTCM were tested using the procedure from Section 5. The nonlinearity was logarithmic and Gaussian smoothing filters were used. The bandwidths used were in the range from $\sigma = 2$ to $\sigma = 16$. To find the best parameter sets we performed experiments whose results are summarized in Figure 8, where

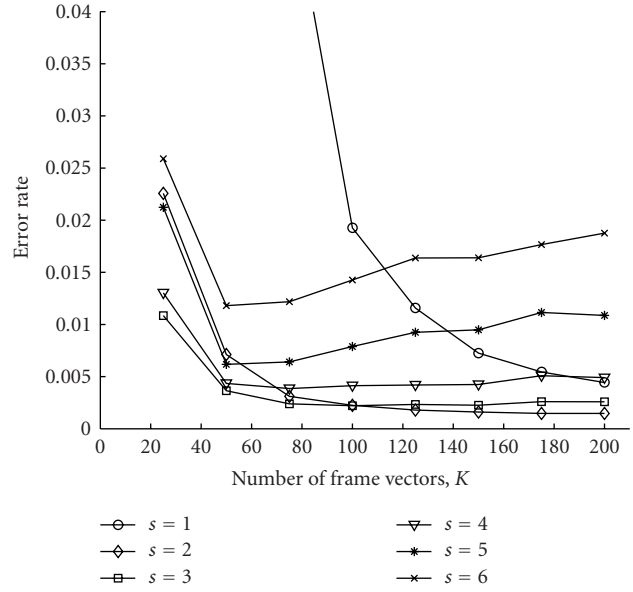


Figure 7: Error rate, that is, number of mislabeled pixels divided by total number of pixels, in classification of the test image in Figure 6. Here we have lowpass filtering with a quite narrow filter, $\sigma = 2$.

the mean classification error rate of the nine test images are shown for all the 36 different frame parameter sets, and in Figure 9 where 6 parameter sets are used with varying degrees of smoothing. We see that having $s = 3$ or $s = 4$ gives the smallest classification error rate for all the frame sizes investigated. This is in line with the results on synthetic textures and the model presented in Section 3. For the tests with the FTCM and $s = 3$ or $s = 4$ the number of wrongly classified pixels is almost halved compared to the cases when $s = 1$ and compared to the results of [11]. We also note that the frame size in FTCM is important, especially for the cases where $s > 1$. The model suggests that the number of frame vectors to use should be quite large, and these results show that the

[2] The training images and the test images are available at http://www.ux.his.no/~tranden/.

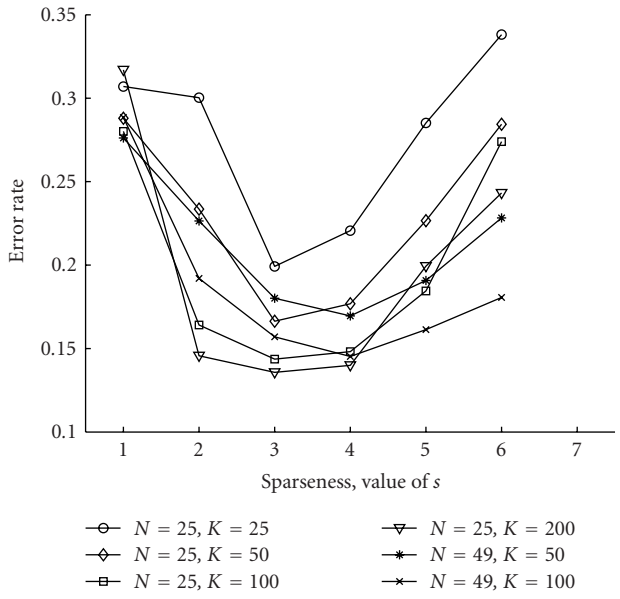


Figure 8: Average error rate, that is, number of mislabeled pixels divided by total number of pixels, in classification of the natural texture test images (a) to (i). Each point represents a unique frame parameter set, (N, K, s). The number of vectors to use in the sparse approximation, s, is along the x-axis. Here, the width of the lowpass filter is given by $\sigma = 8$.

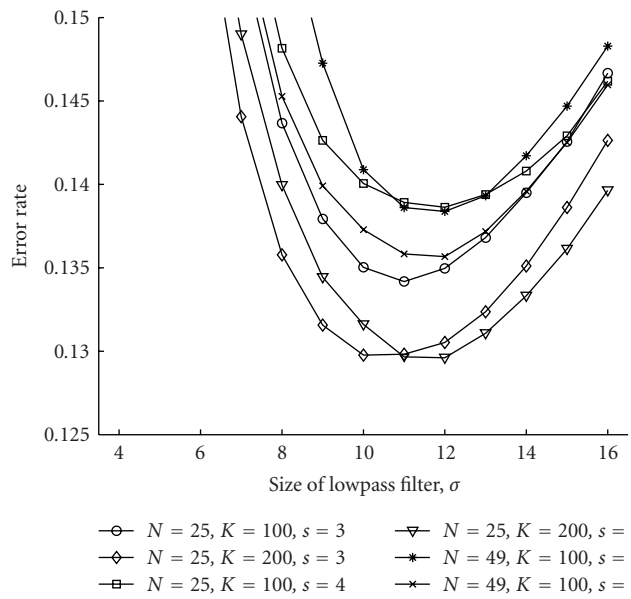


Figure 9: Average error rate in classification of the natural texture test images (a) to (i). Each line represents a unique frame parameter set, (N, K, s). Note the small range for the y-axis. The bandwidth of the smoothing filter, σ, is along the x-axis.

classification result gets better as the number of frame vectors, K, increases. Practical reasons stop us from using larger values of K.

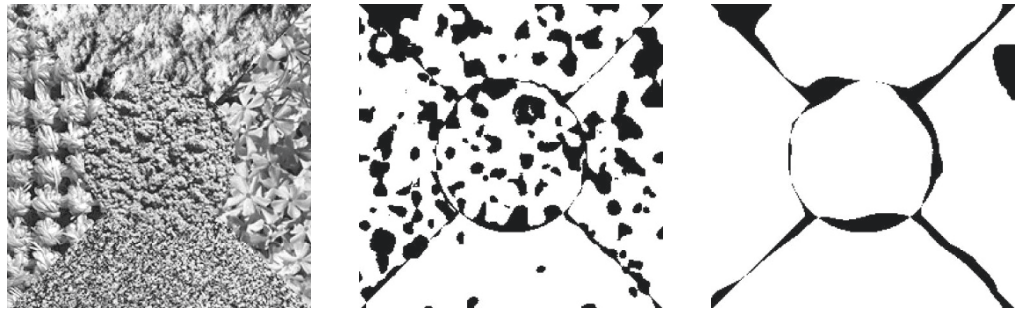
Another interesting observation is that the number of vectors used in the representation, s, should be increased when the parameter N is increased. For $N = 25$ the frames where $s = 3$ perform best, while for $N = 49$ the frames where $s = 4$ perform best. This observation can be explained by the fact that when N is larger the number of vectors to select must be larger to have the same sparseness ratio, s/N, or to have a reasonably good representation of the test vectors.

The effect of the smoothing filter is illustrated in Figure 10. Little smoothing, $\sigma = 4$, gives many error regions scattered in the test image, while more smoothing, $\sigma = 12$ gives better classification within the texture regions, but the cost is often more classification errors along the borders between texture regions. Figure 10 also shows that the fine texture in the lower region is easier to identify than the coarser textures in the rest of the test image.

As a last step we compare the results of FTCM with those of other methods. Table 1 shows the classification errors, given as percentage of wrongly classified pixels, for different methods (rows) and the nine test images (a) to (i). Some of the best classification results from [11] are shown in the upper part of Table 1. The same test images were also used in other papers [8, 13, 16, 36, 37], and results from these are shown in the next part of the table. It should be noted, however, that these latter results are not necessarily directly comparable since we do not know the exact experiment setup used. The lower part of Table 1 shows the results for some of the parameter sets used in the FTCM.

The methods from [11] listed in Table 1 are now briefly explained: "f8a" and "f16b" use subband energies of textures filtered through a tree-structured bank of quadrature mirror filters (QMF). The filters are finite impulse response (FIR) filters of lengths 8 and 16, respectively. The method denoted "Daub-4" uses the Daubechies filters of length 4, and the same structure as that used for the QMF filters. The referred results use the nondyadic subband decomposition illustrated in [11, Figure 6d]. The methods denoted by "J_{MS}" and "J_U" are FIR filters optimized for maximal energy separation, [15]. The last two methods use co-occurrence and autoregressive features. For more details of the classification methods referred and results of more methods we recommend [11]. For the methods in the middle part of Table 1 please consult the given references.

The results for the vector quantization case, FTCM with $s = 1$, give an average error rate of approximately 30 percent, Figure 8, which is comparable to the best results of [11]. The mean for the method "f16b" was 25.9 percent wrongly classified pixels, while the parameter set 49×50 for $N \times K$ and $\sigma = 12$ gave 25.4 percent wrongly classified pixels, see Table 1. Even though the means are comparable, the results for the individual test images vary significantly. For the test image (h) the result is 39.8 for the "f16b" filtering method, and 29.6 for FTCM with frame size 49×50 and $\sigma = 12$, while for the test image (i) the results are 28.5 and 37.1, respectively. Generally, we note that the different filtering methods and the autoregressive method perform better on test image (i) than on test image (h), and that the co-occurrence method and the FTCM (two exceptions in Table 1) perform better on test image (h) than on test image (i).



(a) (b) (c)

Figure 10: (a) Test image "(c)" and the wrongly classified pixels for little ((b) $\sigma = 4$, 25.8% errors) and much ((c) $\sigma = 12$, 9.4% errors) smoothing. The frame parameters are $N = 25$, $K = 50$, and $s = 3$.

Table 1: Classification errors, given as the percentage of wrongly classified pixels, for different methods and natural test images. The results in the middle part are not necessarily directly comparable to the rest.

Method (parameters: $N \times K$, s, σ)	a	b	c	d	e	f	g	h	i	Mean
f8a [11]	7.2	21.1	23.7	18.6	18.6	37.5	43.2	40.1	29.7	26.6
f16b	8.7	18.9	23.3	18.4	17.2	36.4	41.7	39.8	28.5	25.9
Daub-4	8.7	22.8	25.0	23.5	21.8	38.2	45.2	40.9	30.1	28.5
J_{MS}	16.9	36.3	32.7	41.1	43.0	47.3	51.1	59.7	49.9	42.0
J_U	12.7	33.0	26.5	34.3	43.4	45.6	46.5	35.9	30.5	34.3
Co-occurrence	9.9	27.0	26.1	51.1	35.7	49.6	55.4	35.3	49.1	37.7
Autoregressive	19.6	19.4	23.0	23.9	24.0	58.0	46.4	56.7	28.7	33.3
M-band wavelet [36]	2.1	15.1	—	—	—	21.0	—	20.5	—	—
SMLFM [8]	5.8	5.4	8.8	8.3	4.9	8.3	10.0	6.5	5.6	7.1
UMLFM [8]	5.7	5.5	9.1	5.7	5.4	—	—	—	—	—
ICM [8]	9.4	9.3	10.6	13.7	6.5	—	—	—	—	—
Method in [13, Table 2]	7.1	10.7	12.4	11.6	14.9	20.0	18.6	12.0	15.3	13.6
Method in [16, Figure 4b.]	—	—	—	18.5	—	—	—	—	—	—
Local binary pattern (LBP) in [37]	6.0	18.0	12.1	9.7	11.4	17.0	20.7	22.7	19.4	15.2
Gray-level difference (p_8) in [37]	7.4	12.8	15.9	18.4	16.6	27.7	33.3	17.6	18.2	18.7
FTCM (25×50, 3, 4)	2.1	18.1	25.8	27.0	20.0	29.9	38.2	35.2	43.9	26.7
FTCM (25×50, 3, 8)	3.5	11.4	10.6	12.5	8.7	21.8	23.1	24.3	26.8	15.9
FTCM (25×50, 3, 12)	5.4	10.3	9.4	10.4	6.6	20.2	20.8	22.6	21.4	14.1
FTCM (25×100, 3, 12)	5.5	9.1	14.6	7.9	6.1	19.3	18.0	18.6	22.4	13.5
FTCM (25×100, 4, 12)	5.5	6.9	16.5	7.4	7.0	21.6	17.9	26.0	15.8	13.9
FTCM (25×200, 3, 10)	4.3	9.8	11.6	7.6	6.4	20.8	15.9	18.2	22.3	13.0
FTCM (25×200, 3, 12)	5.2	9.8	12.1	7.6	6.9	20.5	16.3	17.2	21.7	13.1
FTCM (25×200, 4, 12)	5.5	7.3	13.2	5.6	10.5	17.1	17.2	18.9	21.4	13.0
FTCM (49×50, 1, 12)	8.9	17.5	14.6	30.8	24.6	25.2	39.9	29.6	37.1	25.4
FTCM (49×100, 4, 12)	5.7	10.7	9.0	6.5	9.6	17.5	19.7	19.4	24.1	13.6

The conclusion of the experiments can be summarized as follows. For the nine test images used, the FTCM performs very well. There is little improvement achieved when increasing the block size from 5×5 to 7×7 pixels. It is better to increase the number of frame vectors; $K = 200$ is marginally better than $K = 100$ as can be seen from Table 1. The number of frame vectors to use in the sparse representation should be $s = 3$ or $s = 4$ according to the model, and this is confirmed by the experiments both on synthetic and natural textures. The optimal width of the lowpass filter, given by σ, is more dependent on the texture characteristics and boundaries between texture patches in the test image than on the frame

parameters; for example, the fine textures in test image (a) are best classified using a small value of σ. The average result for these test images is the best for $10 \leq \sigma \leq 12$. The experiments here indicate that a frame size of 25×200, $s = 3$, and $\sigma = 10$ is a good choice.

7. CONCLUSION

In this paper we have presented the frame texture classification method for supervised texture segmentation of images. Both methods for training based on texture example images and for classification of test images were described, together with a theoretical model motivating the method. The method is conceptually simple and straightforward, but it is computationally demanding, especially the training part. The classification results are excellent. The FTCM provides superior classification performance, for many test images the number of wrongly classified pixels is more than halved, compared to the many methods presented in the large comparative study of Randen and Husøy [11]. The results presented also compare favorably with those presented in several other recent contributions.

REFERENCES

[1] M. Tuceryan and A. K. Jain, "Texture analysis," in *Handbook of Pattern Recognition and Computer Vision*, C. H. Chen, L. F. Pau, and P. S. P. Wang, Eds., chapter 2.1, pp. 207–248, World Scientific, Singapore, 2nd edition, 1998.

[2] R. J. Dekker, "Texture analysis and classification of ERS SAR images for map updating of urban areas in the Netherlands," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 9, pp. 1950–1958, 2003.

[3] M. K. Kundu and M. Acharyya, "M-band wavelets: application to texture segmentation for real life image analysis," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 1, no. 1, pp. 115–149, 2003.

[4] F. Mendoza and J. M. Aguilera, "Application of image analysis for classification of ripening bananas," *Journal of Food Science*, vol. 69, no. 9, pp. 471–477, 2004.

[5] S. Arivazhagan and L. Ganesan, "Automatic target detection using wavelet transform," *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 17, pp. 2663–2674, 2004.

[6] S. Singh and M. Singh, "A dynamic classifier selection and combination approach to image region labelling," *Signal Processing Image Communication*, vol. 20, no. 3, pp. 219–231, 2005.

[7] M. Unser, "Texture classification and segmentation using wavelet frames," *IEEE Transactions on Image Processing*, vol. 4, no. 11, pp. 1549–1560, 1995.

[8] S. Liapis, E. Sifakis, and G. Tziritas, "Colour and texture segmentation using wavelet frame analysis, deterministic relaxation, and fast marching algorithms," *Journal of Visual Communication and Image Representation*, vol. 15, no. 1, pp. 1–26, 2004.

[9] G. F. McLean, "Vector quantization for texture classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 637–649, 1993.

[10] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[11] T. Randen and J. H. Husøy, "Filtering for texture classification: a comparative study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 4, pp. 291–310, 1999.

[12] C. Diamantini and A. Spalvieri, "Quantizing for minimum average misclassification risk," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 174–182, 1998.

[13] N. Malpica, J. E. Ortuño, and A. Santos, "A multichannel watershed-based algorithm for supervised texture segmentation," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1545–1554, 2003.

[14] S Li, J. T. Kwok, H. Zhu, and Y. Wang, "Texture classification using the support vector machines," *Pattern Recognition*, vol. 36, no. 12, pp. 2883–2893, 2003.

[15] T. Randen and J. H. Husøy, "Texture segmentation using filters with optimized energy separation," *IEEE Transactions on Image Processing*, vol. 8, no. 4, pp. 571–582, 1999.

[16] K. I. Kim, K. Jung, S. H. Park, and H. J. Kim, "Support vector machines for texture classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 11, pp. 1542–1550, 2002.

[17] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 227–234, 1995.

[18] G. Davis, "Adaptive nonlinear approximations," Ph.D. dissertation, New York University, New York, NY, USA, 1994.

[19] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[20] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in *Proceedings of 27th IEEE Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 40–44, Pacific Grove, Calif, USA, November 1993.

[21] S. Chen and J. Wigger, "Fast orthogonal least squares algorithm for efficient subset model selection," *IEEE Transactions on Signal Processing*, vol. 43, no. 7, pp. 1713–1715, 1995.

[22] M. Gharavi-Alkhansari and T. S. Huang, "A fast orthogonal matching pursuit algorithm," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, vol. 3, pp. 1389–1392, Seattle, Wash, USA, May 1998.

[23] S. F. Cotter, R. Adler, R. D. Rao, and K. Kreutz-Delgado, "Forward sequential algorithms for best basis selection," *IEE Proceedings—Vision, Image and Signal Processing*, vol. 146, no. 5, pp. 235–244, 1999.

[24] K. Skretting and J. H. Husøy, "Partial search vector selection for sparse signal representation," in *Proceedings of IEEE Norwegian Symposium on Signal Processing (NORSIG '03)*, Bergen, Norway, October 2003.

[25] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *Proceedings of IEEE International Conference on Image Processing (ICIP '95)*, vol. 3, pp. 648–651, Washington, DC, USA, October 1995.

[26] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *International Journal of Computer Vision*, vol. 40, no. 1, pp. 49–71, 2000.

[27] R. Paget, "Strong Markov random field model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 3, pp. 408–413, 2004.

[28] R. Paget, "Nonparametric Markov random field models for natural texture images," Ph.D. dissertation, University of Queensland, Queensland, Australia, 1999, available at http://www.vision.ee.ethz.ch/~rpaget/publications.htm.

[29] K. Engan, S. O. Aase, and J. H. Husøy, "Method of optimal directions for frame design," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '99)*, vol. 5, pp. 2443–2446, Phoenix, Ariz, USA, March 1999.
[30] K. Skretting, "Sparse signal representation using overlapping frames," Ph.D. dissertation, Norwegian University of Science and Technology, Trondheim, Norway, October 2002, available at http://www.ux.his.no/~karlsk/.
[31] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Norwell, Mass, USA, 1992.
[32] A. Laine and J. Fan, "Frame representations for texture segmentation," *IEEE Transactions on Image Processing*, vol. 5, no. 5, pp. 771–780, 1996.
[33] A. Van Nevel, " Texture classification using wavelet frame decompositions," in *Proceedings of 31st IEEE Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 311–314, Pacific Grove, Calif, USA, November 1997.
[34] H. Bolcskei, F. Hlawatsch, and H. G. Feichtinger, "Frame-theoretic analysis of oversampled filter banks," *IEEE Transactions on Signal Processing*, vol. 46, no. 12, pp. 3256–3268, 1998.
[35] M. Unser and M. Eden, "Nonlinear operators for improving texture segmentation based on features extracted by spatial filtering," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 4, pp. 804–815, 1990.
[36] M. Acharyya, R. K. De, and M. K. Kundu, "Extraction of features using M-band wavelet packet frame and their neuro-fuzzy evaluation for multitexture segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1639–1644, 2003.
[37] T. Ojala, K. Valkealahti, E. Oja, and M. Pietikäinen, "Texture discrimination with multidimensional distributions of signed gray-level differences," *Pattern Recognition*, vol. 34, no. 3, pp. 727–739, 2001.

Karl Skretting was born in Naerbo, Norway, in 1962. He received the B.S., the M.S., and the Ph.D. degrees in electrical engineering from the Stavanger University College, Norway, in 1985, 1998, and 2002, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Stavanger, Norway. His research interests include signal representations and compression, texture classification, and computer graphics. He serves as Chairman of the Norwegian Signal Processing Society.

John Håkon Husøy was born in Toronto, Ontario, Canada, in 1956. He received the M.S. and Ph.D. degrees in electrical engineering in 1981 and 1991, respectively, from the Norwegian Institute of Technology, University of Trondheim, Trondheim, Norway. He has been involved in hardware and software development in various positions in several companies. Since 1992 he has been a Professor with the Department of Electrical and Computer Engineering, University of Stavanger, Norway. His research interests include adaptive algorithms, digital filtering, signal representations, image compression, bioelectrical signal processing, and image analysis.