TEXTURE CLASSIFICATION USING SPARSE FRAME BASED REPRESENTATIONS

Karl Skretting and John Håkon Husøy

Stavanger University College, Department of Electrical and Computer Engineering
P. O. Box 2557 Ullandhaug, N-4091 Stavanger, Norway
Phone: +47 51 83 20 16, Fax: +47 51 83 17 50, E-mail: karl.skretting@tn.his.no

ABSTRACT

In this paper a new method for texture classification, denoted Frame Texture Classification Method (FTCM), is presented. The main idea is that a *frame* trained to make a *sparse* representation of a certain class of signals is a model for this signal class. The signal class is given by many representative image blocks of the class. Frames are trained for several textures, one frame for each texture class. A pixel of an image is classified by processing a block around the pixel, the block size is the same as the one used in the training set. Many sparse representations of this test block are found, using each of the frames trained for the texture classes under consideration. Since the frames were trained to minimize the representation error, the tested pixel is assumed to belong to the texture for which the corresponding frame has the smallest representation error.

The FTCM is applied to nine test images, yielding excellent overall performance, for many test images the number of wrongly classified pixels is more than halved, in comparison to state of the art texture classification methods presented in [1].

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. A simple definition could be: 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 [2] list more possible definitions, and give a more complete overview of texture classification. Texture classification using vector quantization [3] is a quite interesting approach since the method we propose here, FTCM, may be regarded as a generalization of the vector quantization approach.

Texture is a local property of an image, but it is not confined to a single point, this means that to decide the texture of a point a small area around it must be included. When we say that a pixel belong to a certain texture we mean that a block around that pixel has some properties that define it as belonging to this texture. If the block must be large to properly identify the texture, for example 100×100 pixels, the texture is coarse, while if the texture is well defined (recognizable) for a small block, for example 7×7 pixels, the texture is said to be fine. In FTCM the block size should be rather small. We used 5×5, 7×7, or 9×9, but since a later step in the classification method includes smoothing of the features, a larger neighborhood contributes to the final classification of a pixel.

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 signal block, denoted $\mathbf{x}_l$ to indicate that it is one out of L available signal blocks, can be represented by a weighted sum of these 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 approximate representation of the signal block. In this paper we use approximate *sparse* representations in which a small number of the weights $w_l(k)$ are non-zero. 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,

$$\begin{aligned} \mathbf{X} &= [\ \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \cdots \quad \mathbf{x}_L\ ], \\ \mathbf{W} &= [\ \mathbf{w}_1 \quad \mathbf{w}_2 \quad \mathbf{w}_3 \quad \cdots \quad \mathbf{w}_L\ ]. \end{aligned} \tag{2}$$

The synthesis equation, Equation 1, may now be written as $\tilde{\mathbf{X}} = \mathbf{F}\,\mathbf{W}$. It is also convenient to let the frame be *normalized*, i.e. each frame vector is scaled such that its 2-norm is one.

Matching Pursuit algorithm:

1. Initialize: $\mathbf{r} := \mathbf{x}$, $\mathbf{w} := \mathbf{0}$
2. Find the inner products: $\mathbf{u} := \mathbf{F}^T \cdot \mathbf{r}$
3. Find k such that $|u(k)| = \max_i |u(i)|$
4. Update weight k: $w(k) := w(k) + u(k)$
5. Update residual: $\mathbf{r} := \mathbf{r} - u(k) \cdot \mathbf{f}_k$
6. Repeat step 2-5 until $\mathbf{w}$ has s non-zero entries.

Figure 1: The Matching Pursuit algorithm when the frame is normalized. The stop criterium in step 6 could also be that a predefined number of iterations, for example s, is done or that the norm of the error is smaller than a given limit.

In a *sparse representation* many of the weights in the signal expansion, Equation 1, are zero. To quantify the degree of sparseness the *sparseness factor* is defined as

$$S = \frac{\text{number of non-zero weights in } \mathbf{w}_l}{\text{number of signal samples in } \mathbf{x}_l} = \frac{s}{N}. \quad (3)$$

To find the sparse weight vectors practical solutions employ greedy *vector selection* algorithms such as Matching Pursuit (MP) [4], Orthogonal Matching Pursuit (OMP) [5], and Fast Orthogonal Matching Pursuit (FOMP) [6]. In the MP algorithm, Figure 1, the residual, $\mathbf{r}$, is only orthogonal to the most recently selected frame vector, but for OMP and FOMP the residual will be orthogonal to all the previously selected frame vectors.

This paper is organized as follows: In Section 2 we describe the process of designing frames for the FTCM, and in Section 3 we describe the classification part of FTCM. Finally, in in Section 4 the experimental results are presented.

2. TRAINING FRAMES FOR THE TEXTURES

Training the frames for each of the available texture example images is an important part of FTCM. It is a computationally demanding process, but is only done once for each frame. The process has three main steps as shown in Figure 2. The very first step in the FTCM is to decide the frame parameters. These parameters can be chosen quite freely, they are:
a) The shape, usually rectangular, and size of the block made around each pixel. The pixels within this block are organized as a column vector of length N.

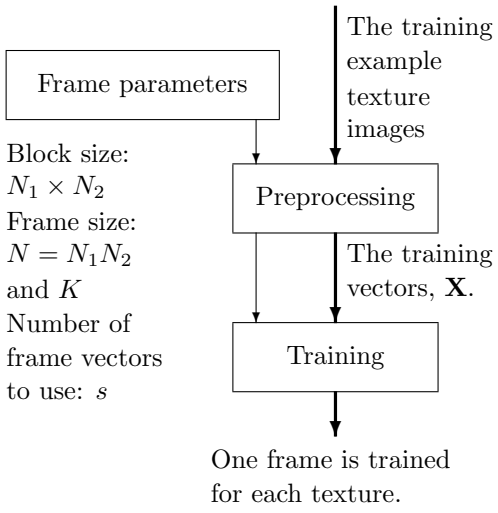


Figure 2: The setup for training of frames in FTCM is very similar to the general frame training setup.

b) The number of vectors in the frame, K, may be chosen quite freely. As a rule of thumb, found from the comprehensive experiments done, we may use $N \leq K \leq 5N$.
c) The number of frame vectors that are used to approximate the signal vector, denoted by a lowercase letter s. Since vector selection is more difficult, i.e. computationally demanding, the larger s is, a small value of s is usually preferred. But the main objective is to choose a value of s that provides a good discrimination of the different textures.

Preprocessing is the task of generating the sets of training vectors for the signal classes, given by some texture example images. The many possible steps of preprocessing can be grouped into two classes: (1) The texture image may be processed to "improve" the texture quality, i.e. make the texture clear and uniform throughout the relevant image region. Then L blocks, which may partly overlap each other, are (randomly) picked from the texture example image and reshaped into training vectors. (2) The individual training vectors may be scaled, translated and/or normalized, i.e. scaled (after a possible translation) such that each has norm one. The same preprocessing steps should also be done on the test image and the test vectors during classification in FTCM, Figure 3.

The last box in Figure 2 is for the actual training of the frames. For each frame the parameter set, N, K and s, and the training vectors, $\mathbf{X}$ of size $N \times L$, are given. The frame design procedure [7] [8] is similar to Generalized Lloyd Algorithm (GLA) [9] and has the following steps

1. If $s = 1$, we start by choosing K arbitrary vec-

tors from the set of training vectors as the initial frame.

2. If $s > 1$, the frame designed with $s = 1$ is used as the initial frame.

3. For each training vector, $\mathbf{x}_l$, we find a sparse representation, i.e. $\mathbf{w}_l$ with s non-zero values is found, typically applying an MP algorithm. If $s = 1$ this is the same as finding the frame vector closest to the training vector, i.e. nearest neighbor. The training vectors associated to a certain frame vector form a cluster. In standard GLA there is only one non-zero value in each column of $\mathbf{W}$, $\mathbf{w}_l$, and this value is 1.

4. The new frame is found by the equation $\mathbf{F} = \mathbf{X}\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$ derived in [8]. This $\mathbf{F}$ matrix is the frame that minimize the norm of the representation error, $\|\mathbf{X} - \mathbf{F}\mathbf{W}\|$, when $\mathbf{X}$ and $\mathbf{W}$ are fixed. If $s = 1$ this gives each column vector of $\mathbf{F}$ as the mean or centroid (weighted mean if not all non-zero values of $\mathbf{W}$ are 1) of the corresponding cluster.

5. The frame is normalized, i.e. scaled such that each frame vector has norm one. This step is not done in standard GLA.

6. Step 3 to 5 are repeated for a predefined number of times or until the method has converged, i.e. no change, or only a minimal change, in the frame since last iteration.

3. CLASSIFYING A TEST IMAGE

Texture classification of a test image is the task of classifying each pixel of the test image to belong to a certain texture. To do this, a small block around each pixel to be tested must be made into a test vector. The classifying process for the FTCM is illustrated in Figure 3. A test image (of size $M_1 \times M_2$) is used to generate $M = M_1 M_2$ test vectors, one for each pixel. For pixels near the edge of the image the reflection of the edge pixel is used, i.e. when $x(m)$ is defined for $m = 1, 2, \ldots, M_1$ then $x(1 - m) = x(1 + m)$ and $x(M_1 + m) = x(M_1 - m)$ for $m = 1, 2, 3, \ldots$.

A test vector, $\mathbf{x}$, is represented in a sparse way using an MP algorithm and each of the different frames that were trained for the textures under consideration, $\mathbf{F}^{(i)}$ for texture class i and $i = 1, 2, \ldots, C$. C is the number of texture classes that are tested for this image. Each sparse representation gives a representation error, $\mathbf{r}^{(i)} = \mathbf{x} - \mathbf{F}^{(i)}\mathbf{w}^{(i)}$. The norm squared of each error, $\mathbf{r}^{(i)T}\mathbf{r}^{(i)}$, is stored in a three dimensional matrix $\mathbf{R}$ of size $M_1 \times M_2 \times C$.

Direct classification based on the norm squared of the representation error for each pixel (the values stored in the $\mathbf{R}$ matrix) gives quite large classification errors, but the results can be substantially improved by smoothing, i.e. low-pass filtering each of the C error images (layers of $\mathbf{R}$). For smoothing Randen and Husøy [1] concluded that the separable Gaussian low-pass filter is the better choice, and this is also the filter used here. The unit pulse response for this filter is

$$h_G(n) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{n^2}{\sigma^2}}. \tag{4}$$

The parameter σ gives the bandwidth of the smoothing filter. The effect of smoothing is illustrated in Figure 4: Little smoothing, $\sigma = 4$, gives a lot of 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 4 also shows that the fine texture in the upper region is easy to identify and the coarse texture in the left region is more difficult to identify.

A nonlinearity may be included before the smoothing filter is applied. This 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. The effect of different nonlinearities is

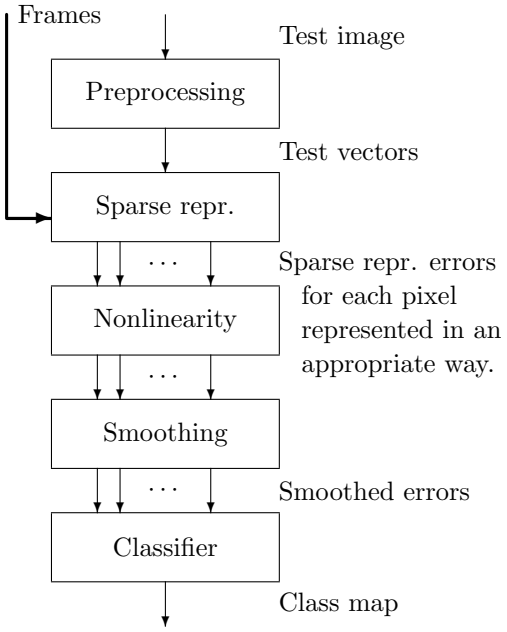


Figure 3: The setup for the classification approach in FTCM, this setup is similar to a common setup in texture classification used in [1].

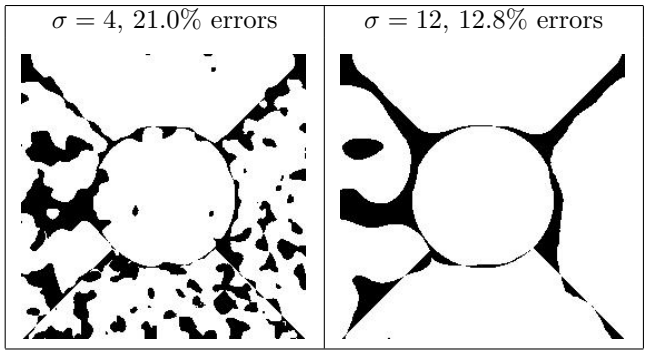


Figure 4: The wrongly classified pixels for the test image (b) in Figure 6. The logarithmic nonlinearity is used. The frame parameters are $N = 25$, $K = 49$, and $s = 3$.

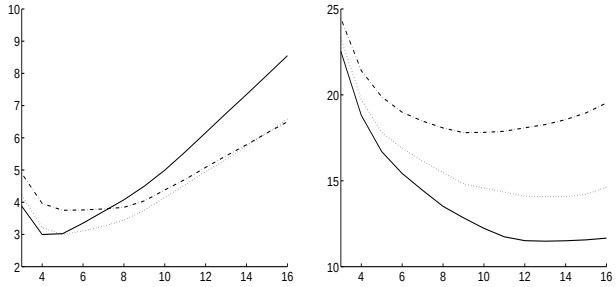


Figure 5: The percentage wrongly classified pixels along the y-axis for two of the test images, note that the scale varies. Along the x-axis is σ used in the smoothing filter. The nonlinearities are: logarithmic as solid lines, magnitude as dotted lines, and energy (the output after sparse representation) as dash-dot lines. The frame parameters are $N = 49$, $K = 98$, and $s = 3$.

shown in Figure 5. The results varied for the different test images, and also for different sets of frame parameters, but in overall the logarithmic nonlinearity is the better choice.

The final classification is very simple in the FTCM. The frame, – remember the training resulted in one frame for each texture class, that gives the best approximation to the test vector gives the class of the test vector.

4. EXPERIMENT

In our experiments, we evaluate different parameter sets for the FTCM by performing supervised segmentation on nine test images of varying complexity. They are denoted (a) to (i) in Figure 11 in [1], where also a more detailed description of the test images can be found[1]. Test images (a) and (b) are shown in Figure 6. Ten different frame parameter sets were used in our experiments. The classification results, given as percentage wrongly classified pixels, are presented in Table 1, one column for each test image, the parameter sets are separated by horizontal lines.

Square blocks with side lengths of 5, 7 and 9 were used to make the blocks around the pixels, this gave the length of the training and test vectors to $N = 25$, $N = 49$, and $N = 81$. The number of frame vectors in each frame were $K = 49$ (and $K = 100$), $K = 98$ and $K = 144$ respectively. The numbers of frame vectors to use in the sparse representation were set to $s = 1$, $s = 3$ and $s = 5$. For each parameter set a frame was designed for all the textures of interest. Note that the training vectors were made from separate example images of each texture, not from blocks of the test images. The number of textures used in the nine test images is 77, and the number of parameter sets used here is 10, thus it was necessary to design 770 frames. The design of all the frames needed many weeks of computer time (in average one hour for each frame), but this task must only be done once.

Some of the best results from the large comparative study presented in [1] are shown in Table 2. The table shows the classification errors, given as percentage wrongly classified pixels, for different methods (rows) and the same nine test images as in Table 1 (columns).

The methods compared in [1] are now briefly explained: "f8a" and "f16b" use a tree structured bank of quadrature mirror filters (QMF), the filters are finite input response (FIR) filters of length 8 and 16, respectively. The method denoted "Daub-4" use the Daubechies filters [10] of length 4, and the same structure as used for the QMF filters, the referred results use the non-dyadic subband decomposition illustrated in Figure 6d in [1]. The methods denoted "J_{MS}" and "J_U" are FIR filters optimized for maximal energy separation, [11]. For the J_{MS} method the filters are designed to maximize the ratio between the extracted mean feature values, $J_{MS} = \mu_{v_1}/\mu_{v_2}$ where μ_{v_i} is the mean feature value for texture i. For the J_U method optimization was done with respect to the criterium $J_U = (\mu_{v_1} - \mu_{v_1})^2/\mu_{v_1}\mu_{v_2}$. The last two methods use co-occurrence and autoregressive features. The classification results in Table 2 taken from [1] are directly comparable to the results of the proposed method, FTCM, Table 1.

The texture classification capabilities of the FTCM were tested as described above. Test vectors were made also for pixels near the edge. The nonlinearity was logarithmic and Gaussian smoothing filters were

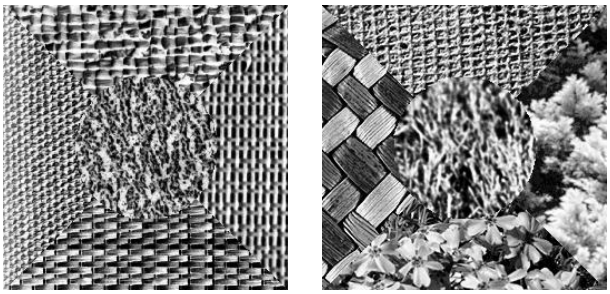
[1]The training images and the test images are available at `http://www.ux.his.no/~tranden/`.

$N \times K$	s	σ	a	b	c	d	e	f	g	h	i	Mean
25×49	1	4	8.9	29.5	28.6	41.3	37.4	35.9	55.3	43.0	53.9	37.1
25×49	1	8	9.1	23.6	18.1	29.5	27.6	30.3	46.4	36.7	39.3	29.0
25×49	1	12	10.8	21.2	19.1	29.2	26.7	29.9	45.9	36.9	35.0	28.3
49×98	1	4	7.9	27.6	30.5	38.2	33.7	34.4	54.1	42.4	56.5	36.1
49×98	1	8	8.0	18.5	21.2	28.0	24.4	28.2	43.7	34.0	45.5	27.9
49×98	1	12	9.4	16.0	22.5	27.0	21.9	27.6	40.5	32.7	43.6	26.8
81×144	1	4	7.3	29.5	29.9	36.0	35.6	34.5	55.2	42.1	58.5	36.5
81×144	1	8	7.1	22.3	21.9	26.8	28.3	27.9	45.8	33.7	49.0	29.2
81×144	1	12	8.0	20.4	22.3	26.5	27.7	26.9	42.8	31.6	46.8	28.1
$N \times K$	s	σ	a	b	c	d	e	f	g	h	i	Mean
25×49	3	4	2.1	21.0	28.4	27.1	18.6	29.2	39.3	35.2	43.1	27.1
25×49	3	8	3.6	14.2	12.1	12.6	8.7	21.0	24.0	24.3	26.7	16.4
25×49	3	12	5.5	12.8	9.3	10.5	7.6	19.2	21.5	21.6	21.2	14.4
25×100	3	4	2.3	13.8	22.7	25.8	20.2	28.7	33.8	33.2	39.3	24.4
25×100	3	8	3.9	7.6	10.1	12.2	11.8	21.8	18.2	21.8	23.9	14.6
25×100	3	12	5.9	6.5	9.8	8.9	10.8	21.1	16.3	18.9	20.1	13.2
25×100	3	16	8.3	6.8	12.0	10.9	11.9	21.7	17.0	18.6	18.6	14.0
49×98	3	4	3.0	18.8	23.6	26.6	22.7	27.6	38.5	33.1	42.0	26.2
49×98	3	8	4.1	13.5	12.0	12.0	11.5	19.2	21.1	22.9	25.8	15.8
49×98	3	12	6.2	11.5	10.0	10.3	10.4	17.1	19.2	20.4	21.6	14.1
81×144	3	4	3.5	19.3	21.6	27.0	22.6	27.8	43.0	32.8	43.7	26.8
81×144	3	8	3.9	14.0	10.4	12.3	13.2	19.2	29.9	21.6	27.1	16.8
81×144	3	12	5.8	12.2	9.8	10.3	12.5	17.3	25.6	18.2	22.9	15.0
$N \times K$	s	σ	a	b	c	d	e	f	g	h	i	Mean
25×49	5	4	2.2	19.2	29.9	31.4	31.0	32.5	44.1	40.3	47.5	30.9
25×49	5	8	3.5	12.7	17.1	20.7	24.8	26.6	32.4	36.4	32.7	23.0
25×49	5	12	5.2	12.9	13.5	20.0	24.2	26.4	30.7	36.8	28.8	22.1
49×98	5	4	2.5	16.1	23.4	22.7	17.4	28.5	35.9	33.0	41.7	24.6
49×98	5	8	3.6	9.7	11.6	9.0	11.4	21.5	24.5	24.3	27.3	15.9
49×98	5	12	5.4	9.1	9.6	7.4	10.1	21.1	23.3	22.6	23.7	14.7
81×144	5	4	3.4	17.7	21.9	23.2	18.7	26.4	37.2	31.1	38.7	24.2
81×144	5	8	3.8	11.8	11.1	9.3	10.6	18.1	24.0	20.6	23.6	14.8
81×144	5	12	5.7	11.2	10.0	7.2	10.9	15.6	22.8	18.1	20.0	13.5

Table 1: Classification errors, given as percentage wrongly classified pixels, for different sets of frame parameters and smoothing filters (rows) and different test images (columns). The logarithmic nonlinearity was used before smoothing. The first three columns are the frame parameters and σ used in the Gaussian smoothing filter. The next nine columns show the results for the test images, Figure 11 in [1]. The last column is the mean for these test images.

Method	a	b	c	d	e	f	g	h	i	Mean
f8a	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

Table 2: Classification errors, given as percentage wrongly classified pixels, for different methods and different test images as presented by Randen and Husøy in [1].



(a) (b)

Figure 6: Two of the test images also used in [1].

used, the bandwidths used were $\sigma = 4$, $\sigma = 8$ and $\sigma = 12$, and for one parameter set also $\sigma = 16$. The classification results are presented in Table 1.

Let us start by looking at the case where $s = 1$. This is the same as vector quantization classification, or nearest neighbor classification. These results are quite similar to the best results of [1]. The mean for the method "f16b" was 25.9 percent wrongly classified pixels, while the parameter setup 49×98 for $N \times K$ and $\sigma = 12$ gave 26.8 percent wrongly classified pixels. Even though the means are comparable, the results for the individual test images may be more different. For the test image (h) the result is 39.8 for the "f16b" filtering method, and 32.7 for the 49×98 and $\sigma = 12$ method, while for the test image (i) the results are 28.5 and 43.6 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 with $s = 1$ perform better on test image (h) than on test image (i).

For the tests with the FTCM and $s > 1$ much better results are achieved. The number of wrongly classified pixels is almost divided by two, and this is very good. We do not have a good explanation for why the sparse representation model using $s > 1$ should be so much better than using $s = 1$, but the many experiments are quite conclusive. One interesting thing is noteworthy: 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 better than the frames where $s = 5$. For $N = 49$ the two different values of s perform almost equal, and for $N = 81$ the frames where $s = 5$ are better that the frames where $s = 3$. 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 factor, Equation 3, or to have a reasonable good representation of the test vectors. Best results were found using three or five frame vectors to represent each test vector, this indicates that "the optimal" number of vectors to use, s, probably is in the N-range explored: for $N = 25$ the best s may be 3 or 4, for $N = 49$ the best s may be 4 and for $N = 81$ the best s may be 5 or 6. The block size should probably be larger for coarse textures than what is necessary for the more fine textures. The case where $N = 25$ is best for the fine textures in the test image (a), while the cases with larger values for N perform better for the test images (c) and (d) which also contain regions with coarser textures.

5. CONCLUSION

In this paper we have presented the Frame Texture Classification Method. For the simple case, using only one frame vector for each test vector, it is like the vector quantizing method, and the classification results is comparable to other state of the art texture classification methods. Utilizing the capabilities of the FTCM, approximating each test vector by a linear combination of three to five frame vectors, the FTCM provides superior classification performance.

6. REFERENCES

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

[2] 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 Publishing Co, Singapore, 1998.

[3] G. F. McLean, "Vector quantization for texture classification," *IEEE Trans. Systems, Man, Cybernetics*, vol. 23, no. 3, pp. 637–649, May/June 1993.

[4] S. G. Mallat and Z. Zhang, "Matching pursuit with time-frequency dictionaries," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[5] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive nonlinear approximations," 1994, Similar to Davis' Ph.D. thesis.

[6] M. Gharavi-Alkhansari and T. S. Huang, "A fast orthogonal matching pursuit algorithm," in *Proc. ICASSP '98*, Seattle, USA, May 1998, pp. 1389–1392.

[7] K. Engan, S. O. Aase, and J. H. Husøy, "Method of optimal directions for frame design," in *Proc. ICASSP '99*, Phoenix, USA, Mar. 1999, pp. 2443–2446.

[8] K. Skretting, J. H. Husøy, and S. O. Aase, "General design algorithm for sparse frame expansions," Submitted for publication, available at `http://www.ux.his.no/~karlsk/`.

[9] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Norwell, Mass., USA, 1992.

[10] I. Daubechies, *Ten Lectures on Wavelets*, Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992, Notes from the 1990 CBMS-NSF Conference on Wavelets and Applications at Lowell, MA.

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