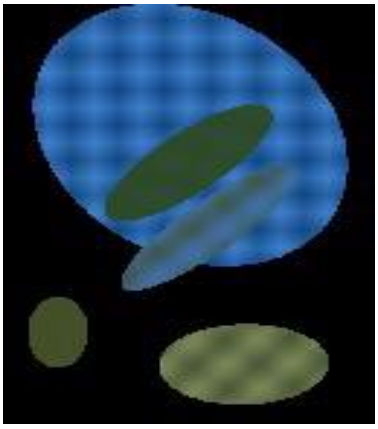
final report for

Application of Forest Image Analysis to Monitoring and Modeling of Psychological, Silvicultural, and Wildlife Habitat Attributes

submitted to:

Mr. Victor A. Rudis
USDA Forest Service
Southern Research Station
P. O. Box 928
Starkville, MS 39760-0928
Tel: 662-338-3109, Fax: 662-338-3101
Email: vrudis@fs.fed.us

August 15, 2000



submitted by:

Z. Long and J. Picone
Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571
413 Simrall, Hardy Road
Mississippi State, Mississippi 39762
Tel: 662-325-3149
Fax: 662-325-3149
Email: {long, picone}@isip.msstate.edu

EXECUTIVE SUMMARY

The objective of this continuing project is to extract features from digitized forest images and develop specific models for monitoring psychological, silvicultural, and wildlife habitat attributes. Previously, we had developed algorithms to extract diverse features such as color histograms, densities of long lines and short lines, entropy, etc., from forest images. We had also built an automatic image segmentation system based on these features. Our work this year has focused on improving the prototype system. We completed optimization of the edge and line detectors, designed frequency domain features based on the Discrete Cosine Transform (DCT) and Gabor filters, and incorporated syntactic information into the system.

First, we optimized the edge and line detectors embedded in our system. These detectors are used to extract the densities of long and short lines, which are very important features for identifying trees, bushes and grasses. To generate the line features reliably, parameters associated with the detectors needed to be optimized. We designed an objective metric to evaluate the performance of edge and line detectors, and then optimized the performance of these detectors. With this metric, our best system resulted in a detection error rate of 29%, and had an acceptable insertion rate. As a comparison, the detection error rate before optimization was 42%. In the course of this optimization, we discovered and corrected a bug in the existing code, which was resulted from a misunderstanding of the Canny's algorithm for edge detection.

Second, we investigated the discrete cosine transform (DCT) and integrated this technique into the feature extraction process. The DCT is the most widely used spectral analysis technique in the area of image processing because of its efficiency in packing energy into a small number of DCT coefficients. We analyzed the amplitude distribution of the DCT coefficients on our forest image database. Based on this analysis, we designed features which were generated by filtering DCT coefficients. We integrated these DCT-based features into the software, and evaluated their effectiveness in segmenting images. On a small pilot evaluation, the DCT-based features produced a segmentation error rate of 65.7%. This error rate is comparable to the best error performance we achieved with the system, which was 61.4%, with "rgb+entropy" as the feature vector.

We also investigated another type of frequency domain technique for image analysis — Gabor filters. Gabor filters have been successfully applied to texture image segmentation problems. We implemented a Gabor feature design proposed by Jain and Farrokhnia, and adapted it to our image analysis problem. These features displayed an outstanding capability in distinguishing between typical forestry regions — the pixel classification error rate was as low as 7.3%. However, the associated computational efficiency needed to be improved for practical applications. We modified the design using various subsampling and filtering approaches, and reduced the computational complexity by an order of magnitude. Unfortunately, that still placed the computational requirements beyond reach for the amount of data we process in a typical evaluation. More work on optimization of this algorithm is needed.

Finally, we anticipated that a system incorporating syntactic information would be a logical next step. Our goal was to explore a promising new technology known as Bayesian Networks. As a first step, we explored the correlation between adjacent blocks and designed a syntactic algorithm which classifies an image block by generating a hypotheses based these adjacent blocks. Results thus far have been inconclusive.

TABLE OF CONTENTS

ABSTRACT

In this project, we wish to extract useful features from forest images and build statistical models for forest structure analysis. In our prior efforts, we had developed a prototype image segmentation system. This year, we have been working on improving the system. First, we optimized the edge and line detectors which were used to generate important line features. In the process, we designed an objective metric for evaluating the performance of the detectors. With the optimization, we reduced the detection error rate from 42% to 29%. Second, we designed features based on the Discrete Cosine Transform (DCT), and evaluated their effectiveness in segmenting images. These DCT-based features gave a 65.7% block classification error rate on our standard evaluation task. Third, we investigated the possibility of extracting features with Gabor filter bank. We proved that Gabor filter-based features are good at distinguishing between forestry regions with different textures. We improved the computational efficiency of an existing Gabor filter bank-based feature design. Finally, we conducted a pilot study on a segmentation algorithm which incorporated syntactic information into the segmentation system.

1. INTRODUCTION

Image analysis is an important approach to understanding human perception. It plays a key role in building automated systems for monitoring natural resources. The goal of this project is to extract meaningful features from digitized forest images using image analysis techniques and to develop specific models for monitoring psychological, silvicultural, and wildlife habitat attributes. In our previous research, we had extracted diverse features such as color histograms, densities of long and short lines, entropy, etc., from forest images [1]. We had also developed an automatic image segmentation system based on these features [2]. However, the performance of the segmentation system was not as good as we had expected. An analysis of its deficiencies indicated that better features were required to improve discrimination between different categories of images. Consequently, our top priority in this year was to develop better features.

First, we chose to improve the line features, i.e., the densities of long and short lines in a forest image. These line features are important because long lines identify trees, while short lines are indications of the presence of bushes and grasses. Previously, we had incorporated edge and line detectors into our system to extract the line features, but we had not done any comprehensive research to elaborate the performance of those detectors. Therefore, we optimized the edge and line detectors by tuning the key parameters involved. In the course of this optimization, we designed an objective metric for measuring performance of the detectors.

Second, we investigated frequency domain features based on the discrete cosine transform (DCT). Although our current system offers a wide range of features, we had not taken a serious look at frequency domain features. We chose the DCT because it has many advantages over other frequency analysis techniques, such as a high efficiency in packing energy into DCT coefficients and the availability of fast computation algorithms. We analyzed the amplitude distribution of the DCT coefficients on all the forestry region categories defined in this study, designed various feature vectors based on these DCT coefficients, and evaluated their effectiveness in image segmentation. The best combination of DCT features we have developed thus far delivered a block classification error rate of 65.7% on a pilot data set.

Third, we investigated the use of a Gabor filter bank in feature extraction. Gabor filters have been shown to emulate the early stages of the human visual system, and hence are very promising in computer vision applications. Jain and Farrokhnia [3] proposed a feature design for texture segmentation on the basis of a bank of Gabor filters. Their results illustrated the effectiveness of Gabor features on the segmentation problem. We applied their feature design to typical images from our database and verified their ability to distinguish between different types of forestry. However, the computation speed with this design was prohibitively slow. We revised the feature extraction algorithm and reduced the computational complexity significantly. However, the algorithm still needs significant improvements in speed before it is practical.

In addition to feature extraction, we also began investigating ways to incorporate syntactic information into the system. In our prototype segmentation system, the classification of a block was done independently of all other blocks in the same image. Undoubtedly, this approach is flawed in disconnecting a block from its surrounding counterparts. For forest images, which present natural scenes, obviously there is causal relationship between the category of one block and those of its neighbors. To make up for this shortcoming, we analyzed the correlation between neighboring blocks from the same image and designed an algorithm which incorporated such syntactic information using Principal Component Analysis (PCA).

2. OPTIMIZATION OF EDGE AND LINE DETECTORS

The densities of long and short lines in a forest image are important features for automated forest image classification and segmentation [1]. Previously, we incorporated edge and line detectors into our feature extraction algorithms. However, there are many parameters involved in these detectors, and we had not tuned them for this specific application. The optimization was difficult for two reasons. First, we need to quantify the performance of edge and line detectors. Subjective evaluations may be reliable, but are inefficient because they usually involve the use of human subjects and require tremendous amounts of time. Second, there is no universally accepted objective metric. Although some evaluation metrics have been widely used [4], they all have shortcomings with respect to our application. For example, the metric proposed by Kitchen and Rosenfeld [5] only measures the continuity and thinness of detected edges, while we are also interested in the mismatch rate and the false alarms introduced by the detectors.

Our optimization is an evaluation-driven process. The procedure is illustrated in Figure 1. The key components involved are generation of a large reference database using manual transcription, development of an evaluation metric that compares detector output to the reference data for that image, and optimization of the parameters of each algorithm to minimize the error rate associated with the evaluation metric. This is not as trivial as it might seem since determining whether a detected line matches a reference line requires some amount of intelligence, since the lines never match on a pixel-by-pixel basis.

2.1. Manual Transcription

The first step in our evaluation methodology is to manually transcribe lines. From this transcription, we create a set of reference line data which is used to evaluate detector performance. We can transcribe lines either from a virtual perspective or from a physical perspective. Examples of both perspectives are shown in Figure 2. In a physical perspective,

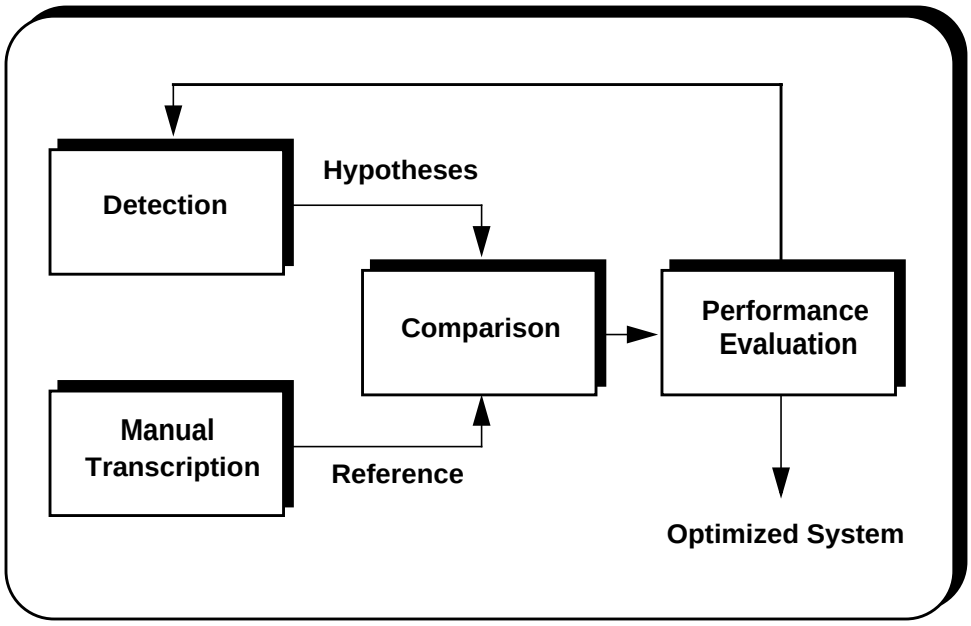


Figure 1. This diagram illustrates our objective evaluation paradigm for detector optimization.

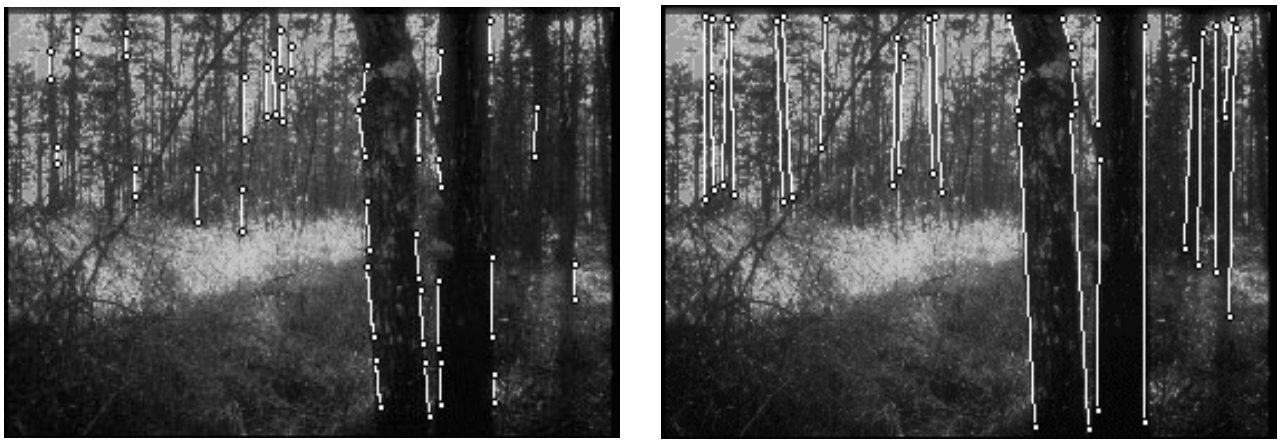


Figure 2. Example transcriptions using two different transcription perspectives. The image on the left was transcribed using a physical perspective — only existing lines were labeled. The image on the right was transcribed from the virtual perspective, where obstructed lines, such as outlines of tree trunks behind shrubs, were also labeled.

shown on the left in Figure 2, we make no such assumptions about the continuity of obstructed lines. We simply trace lines as they appear in the image. The drawback of this approach is that trees obstructed by bushes are not easily discernible from their line segments.

In a virtual perspective, shown on the right in Figure 2, we use perception of lines as a criterion to determine their continuation when obstructed. For example, we label outlines of tree trunks behind shrubs as well as those visible parts. Although a virtual perspective is more consistent with human perception, it requires a much more intelligent line detection algorithm, since line continuity is based on human perception rather than signal processing. The physical perspective fits the existing state of the art better, and hence was chosen as our criterion for this study.

2.2. Metric Design

To measure the performance of an algorithm, we need to compare lines detected by the algorithm to the reference data. This is not trivial since line orientation and length can differ by small amounts due to computational issues, and yet be indistinguishable from a visual perspective (at normal viewing resolution). Hence, some form of an evaluation metric is required that is forgiving of such small errors. In our metric, we evaluated such properties as location, length, and slope.

We determine the location of a line by finding the coordinates of its middle point. Suppose the coordinates of the two endpoints of a line are (x_0, y_0) and (x_1, y_1), respectively. Then the length and slope are computed as

$$Length = \sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2} \text{ and } Slope = \text{atan}\left(\frac{y_1 - y_0}{x_1 - x_0}\right), \quad (1)$$

where $x_1 \neq x_0$ for the slope computation.

The evaluation process can be described as a two-stage procedure. In the first stage, we process all reference lines and find the best match for each reference line in the set of line segments hypothesized for a given image. In the second stage of the evaluation, we tabulate errors by considering the similarity between the reference line and its best match in the hypothesized data.

To find the best matches (i.e., the first stage), we search through all detected lines and compute the distance from the middle point of each detected line to a reference line. The minimum of this distance is the best match for that reference line. We count all detected lines without matches in the reference data as **insertion errors**.

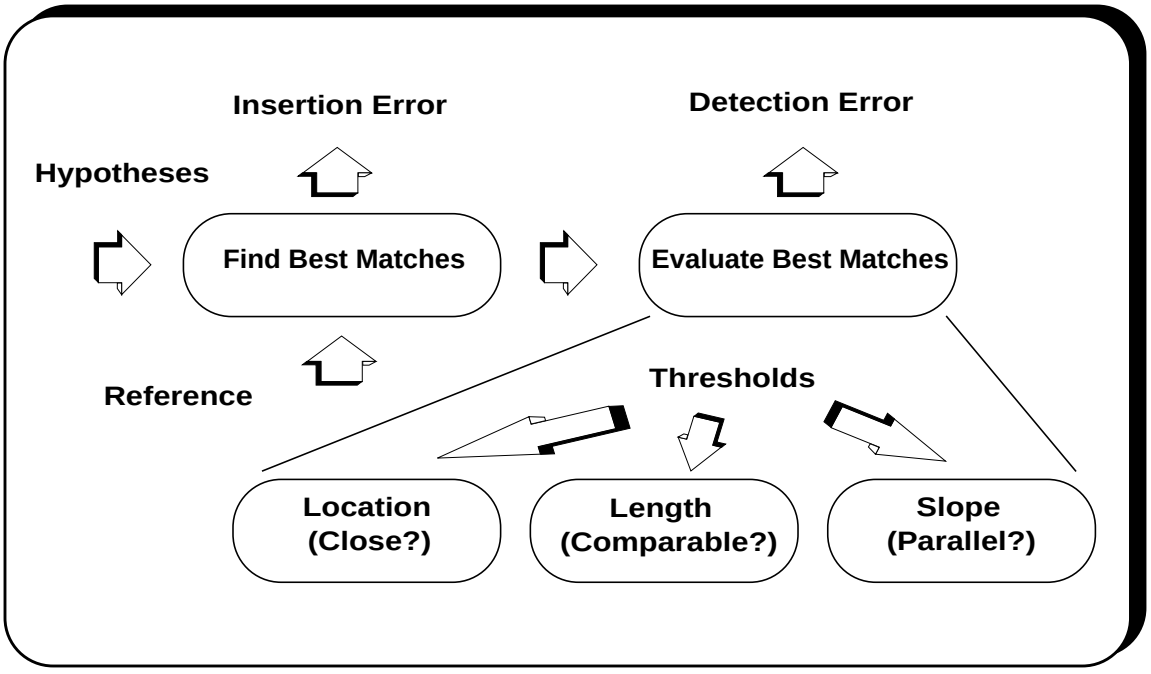


Figure 3. An overview of the basic concepts incorporated in our evaluation metric.

To evaluate the best matches (i.e., the second stage), we need to handle four situations. First, the best match and the associated reference line are close parallel lines of approximately the same length. This is considered a correct detection. Second, we can have close parallel lines of unequal lengths. It is a correct detection if the difference in lengths is within 25% of the length of the reference line. Third, we can have close non-parallel lines of approximately the same length. These are considered a correct detection if the angle between the two lines is less than 20 degrees. Finally, non-parallel lines of unequal lengths are considered a correct detection if the angle between the two lines is less than 20 degrees and the length difference is within 25% of the length of the reference line.

The notion of proximity must be determined empirically using a threshold. To determine if a detected line is close to the associated reference line, we compute the distance from the middle point of the former to the latter. If that distance is within 5 pixels, then we consider these lines as close. We then apply the criteria described above to determine the nature of the match. Using these criteria, we will have two types of errors: insertion and detection. Insertion errors were described previously. Detection errors represent all errors for which the reference line and the best match do not pass the criteria described above (location, length and slope).

2.3. Algorithms

We now describe the specific edge and line detection algorithms used in this study. Canny proposed several criteria for edge detector design and derived corresponding optimal operators by numerical methods [6]. He also introduced a novel edge detection scheme on the basis of these optimal operators.

2.3.1. Edge Detection

An overview of Canny's edge detection algorithm is given in Figure 4. First, a symmetric two-dimensional Gaussian mask is convolved with the original image to smooth out noise present in the source. We used a standard 3x3 Gaussian mask:

$$\frac{1}{\sqrt{2\pi\sigma^2}}\begin{bmatrix} \exp\left(-\frac{1}{\sigma^2}\right) & \exp\left(-\frac{1}{2\sigma^2}\right) & \exp\left(-\frac{1}{\sigma^2}\right) \\ \exp\left(-\frac{1}{2\sigma^2}\right) & 1 & \exp\left(-\frac{1}{2\sigma^2}\right) \\ \exp\left(-\frac{1}{\sigma^2}\right) & \exp\left(-\frac{1}{2\sigma^2}\right) & \exp\left(-\frac{1}{\sigma^2}\right) \end{bmatrix} \tag{2}$$

Second, differentiations in both horizontal and vertical directions are performed. The corresponding masks used to implement this are:

Horizontal: Vertical:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

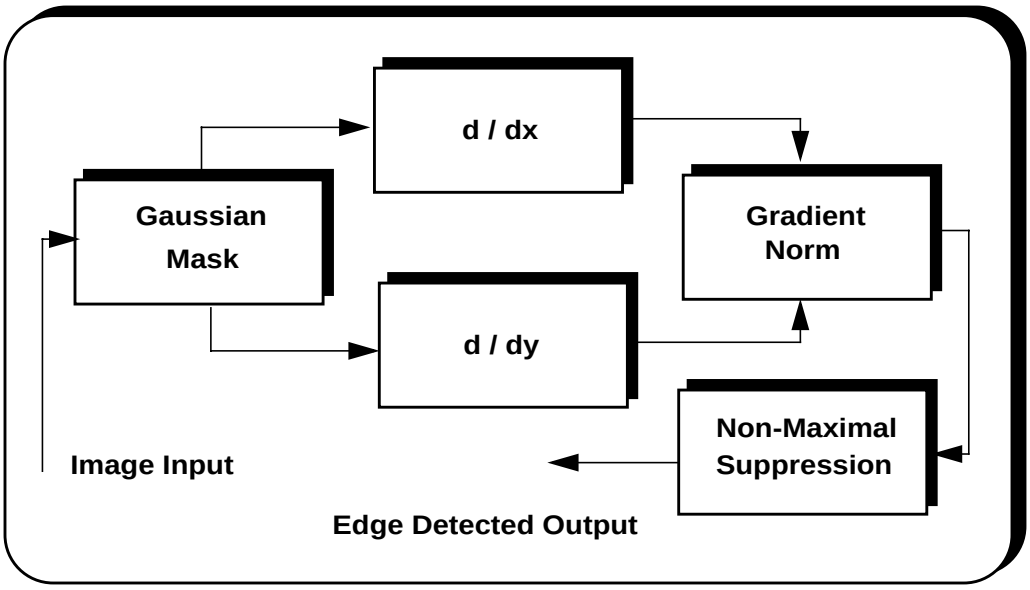


Figure 4. An overview of the Canny edge detection algorithm.

Third, gradient values are calculated using the results of both differentiations:

$$Gradient = \sqrt{\left(\frac{d}{dx}\right)^2 + \left(\frac{d}{dy}\right)^2} \ . \tag{4}$$

An edge pixel is defined to be a local maximum of the gradient values.

The final step in edge detection is to suppress non-maximal gradient values. This is summarized in Figure 5. To accomplish this last step, an adaptive thresholding technique known as hysteresis is used. First, two threshold values are set. If the gradient value of a pixel is greater than the higher threshold, then the pixel is treated as an edge pixel. If a pixel is less than the higher threshold, yet greater than the lower threshold, and connected to a previously identified edge, then the pixel is subsequently treated as an edge pixel. Otherwise, it is a non-edge pixel. Hysteresis greatly reduces the number of broken edge contours generated.

2.3.2. Line Detection

Our approach to line detection is quite simple. We postprocess the output of the edge detector and compare edge lengths with a threshold parameter. For an edge to be a line, its length must be above the line threshold. For this specific project, we are only interested in those vertical lines which represent tree stems, shrubs and grasses. Therefore, we do not detect lines in the horizontal direction.

2.4. Experiments And Results

We prepared two data sets for experimentation. Data set 1 contained 165 images randomly chosen from training set 01 of USFS Pre-Phase 01 image data. Data set 2 consisted of 159 images randomly chosen from test set 01 of Pre-Phase 01 [7]. We experimented with key parameters for both the edge and the line detectors. These parameters included the Gaussian standard deviation

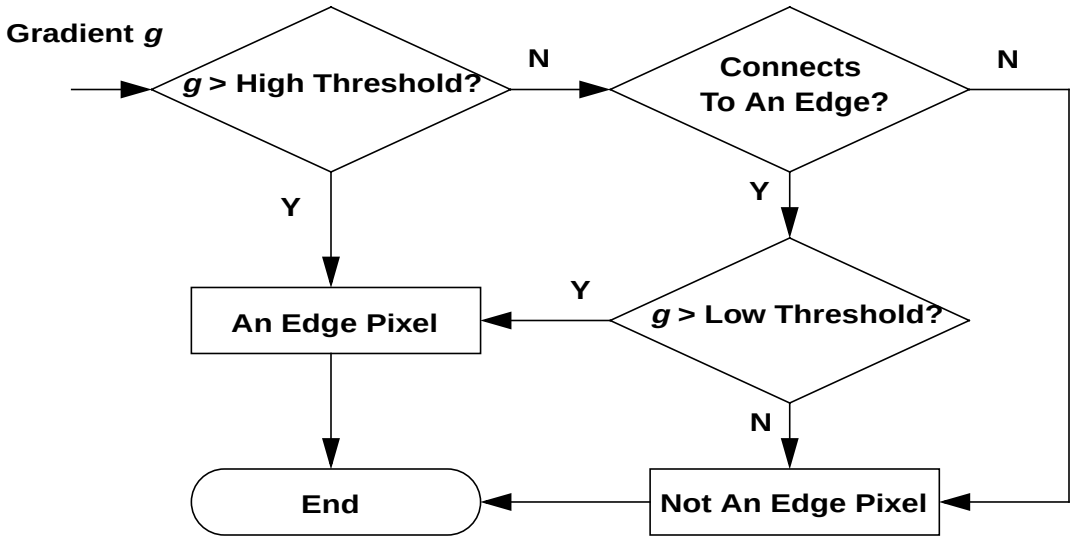


Figure 5. A flow chart that explains the hysteresis procedure involved in implementing the non-maximal suppression block in the Canny algorithm.

(σ in the Gaussian mask function), the low and high edge thresholds for edge detection [6], and the line threshold for line detection.

Detector performance was assessed using a combination of two resulting values. First, we considered the error rate, which is the ratio of detection errors to the total number of reference lines. Second, we considered the insertion rate, which is the total number of insertion errors. Both errors have been described previously in the metric design. A system which achieves a low error rate and a low insertion rate simultaneously is desirable. However, one interesting phenomenon we noticed from the experiments was that, when we lowered the thresholds and the Gaussian standard deviation, the error rate tended to decrease, and the insertion rate increased. The reason for this trade-off is that lower thresholds result in more lines, which simultaneously increases the chance for both correct matches and undesirable insertions. Figure 6 illustrates the results for experiments with the high edge threshold. More plots with various experimental conditions can be found in the Appendix.

An optimal parameter set is one that balances the error rate and the insertion rate. By visual inspection of the error rate curves and the insertion rate curves, we found that with the following settings of parameters, we achieved a good balance between both the error rate and the insertion rate: 2.0 for the Gaussian standard deviation, 60 pixels for the high edge threshold, 30 pixels for the low edge threshold, and 40 pixels for the line threshold. The error rate achieved with these settings was 29% on data set 2.

A comparison of parameters and performance between the optimized system and the previous system is shown in Table 1. The corresponding insertion rate was 272,073 lines for all 159 images. Given that we had transcribed only significant lines (not all existing lines) as reference data, some of the inserted lines might actually correspond to a correct detection. Hence such an insertion rate seemed acceptable.

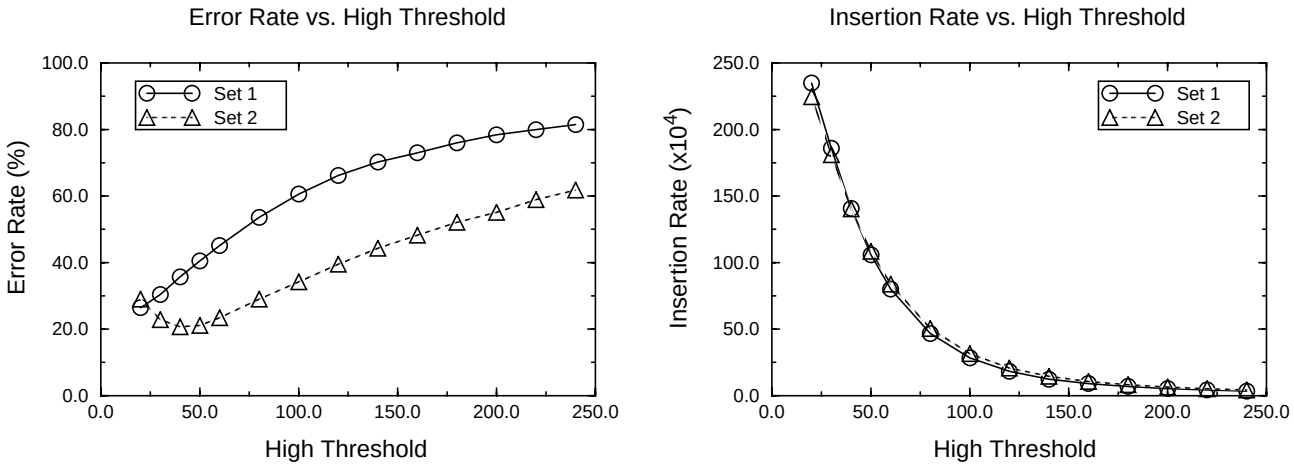


Figure 6. Results from optimization experiments with the high edge threshold involved in Canny edge detection algorithm. Data set 1 contained 165 images randomly chosen from training set 01 of USFS Pre-Phase 01 image data. Data set 2 consisted of 159 images randomly chosen from the test set 01 of Pre-Phase 01. The Gaussian standard deviation was 2.0; the low edge threshold was set to half of the high edge threshold, and the line threshold was 25.

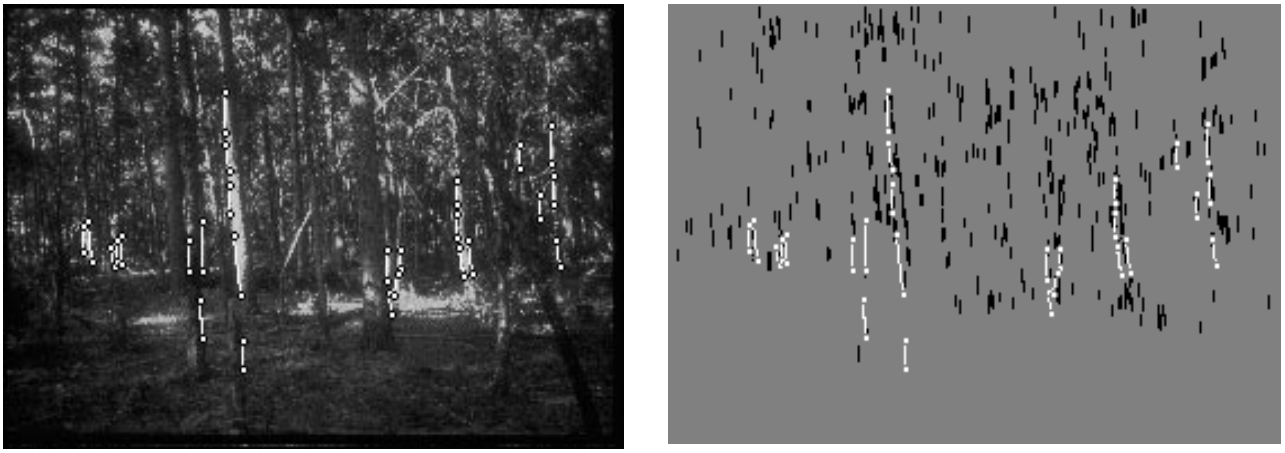


Figure 7. An example output from the optimized detector. The image on the left is a manually transcribed image from data set 2 (the white lines indicate the reference transcriptions). The image on the right is the corresponding detection image in which the back lines represent the detected lines.

Optimization	Parameters				Performance	
	σ	High Edge Threshold	Low Edge Threshold	Line Threshold	Error Rate	Insertion Rate
Before	2.0	180	60	15	42%	701,877
After	2.0	60	30	40	29%	272,073

Table 1. A comparison of parameters and performance for the optimized system and the original system.

In the process of the optimization, we discovered an error with the previous implementation of the Canny algorithm. The non-maximal suppression procedure, which is supposed to suppress the lower gradient values, had been implemented as: a pixel is an edge pixel only if its gradient value falls between the two thresholds. In this case, pixels with gradient values greater than the higher threshold will be masked out. Obviously, there had been a misunderstanding of the algorithm. We fixed this error before we optimized the system.

3. DEVELOPMENT OF FREQUENCY DOMAIN FEATURES

As we know, good features are of significant importance to pattern recognition problems. Previously, we had extracted many features from the forest images, but none of them presents spectral information of the images. Given the fact that most forestry regions display remarkable variations in the spatial domain, and that the patterns of those variations change with each specific kind of region, we believe that there should exist some frequency features helpful for distinguishing between those regions. Therefore, we investigated two common frequency domain analysis techniques: the discrete cosine transform (DCT) and the Gabor filter bank.

3.1. DCT-based Features

First, we investigated the use of the discrete cosine transform. We applied the DCT to the forest images under study using our standard evaluation paradigm, and then analyzed the characteristics of the transform coefficients. Based on this initial analysis, we designed and tested several feature extraction schemes.

3.1.1. DCT Analysis

The discrete cosine transform is a widely used frequency domain analysis technique in image processing. It has many advantages over other transformation techniques. For example, only real numbers are involved in the computation. Also, no spurious spectral components will result as is the case with the discrete Fourier transform (DFT). Further, fast DCT implementations are available. Finally, and most importantly, with this method, energy is packed efficiently into a small number of DCT coefficients [8].

The definition of the two-dimensional forward DCT of an $n \times n$ data block is:

$$F(u,v) = \frac{4C(u)C(v)}{n^2} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} f(j,k) \cos\left[\frac{(2j+1)u\pi}{2n}\right] \cos\left[\frac{(2k+1)v\pi}{2n}\right], \quad (5)$$

where the constants are computed as

$$C(w) = \begin{cases} \frac{1}{\sqrt{2}} & w = 0 \\ 1 & otherwise \end{cases}. \quad (6)$$

On the other hand, the inverse 2-D DCT (IDCT) is defined as:

$$f(j,k) = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} C(u)C(v)F(u,v)\cos\left[\frac{(2j+1)u\pi}{2n}\right]\cos\left[\frac{(2k+1)v\pi}{2n}\right] \quad (7)$$

where $C(w)$ is calculated the same way as it is in the forward DCT.

To better understand the idea of a DCT analysis, we first carried out a forward DCT on the forest image data. We chose one typical region from our image database for each of the six categories under study, i.e., tree, foliage, bush, grass, background sky and sky. Then we generated the DCT coefficients for the selected data with the block size ranging from 8 to 128. There appeared to be a unique pattern in the coefficients' amplitude distribution for each kind of region. To illustrate this more intuitively, we plotted all the original data and the resulting DCT coefficients in MATLAB. Some example plots are shown in Figure 8 and 9. The plots shown in Figure 8 describe amplitude distributions of the DCT coefficients calculated on a typical tree region and a foliage region with the image block size set as 64x64, while the plots in Figure 9 show the DCT coefficients computed on a foliage region with the block size set as 8 x 8 and 32 x 32, respectively. Plots for other categories and block sizes are located in the Appendix.

From these plots, we note that the DCT coefficients' amplitude distribution varies significantly between the regions. This variation may indicate a potential of DCT coefficients to distinguish between the forestry regions. Another observation is that the block size for the DCT computation has a noticeable impact on the pattern of the amplitude distribution. The reason for this sensitivity to the block size may be that small block sizes are not sufficient for the data to contain enough texture information, resulting in patterns inconsistent with those generated on larger blocks. Also, we notice an interesting phenomenon that whatever the block size is, the energy tends to be packed into the lower frequency components by the transformation. Therefore, the lower frequency components are much more important than the higher components, since they collect the majority of the energy.

Based on these observations, we believe that if we choose a suitable block size and use the lower DCT coefficients to generate features, we would be able to achieve good segmentation performance.

3.1.2. DCT-based Feature Design And Evaluation

We designed a few features based on what we had observed in the DCT analysis. The basic idea behind the design is: first compute DCT coefficients on an image block, then filter them in the frequency domain. After that choose the lower frequency components as the feature vector. To be more specific, we designed our baseline system in the following way: perform DCT on the green pixels, average every four of them for the filtering stage, then choose the first 16 filtered outputs to create the feature vector.

With this scheme, we are building the feature vector on the first 64 DCT coefficients. This choice is reasonable since we have already verified that the energy is packed into the lower frequency components, and for those typical regions, the first 64 DCT coefficients concentrate the majority of the energy. If we include all DCT coefficients in the feature computation, we would have problems with large DCT block size. Suppose we are doing DCT on a 64x64 block, and we are

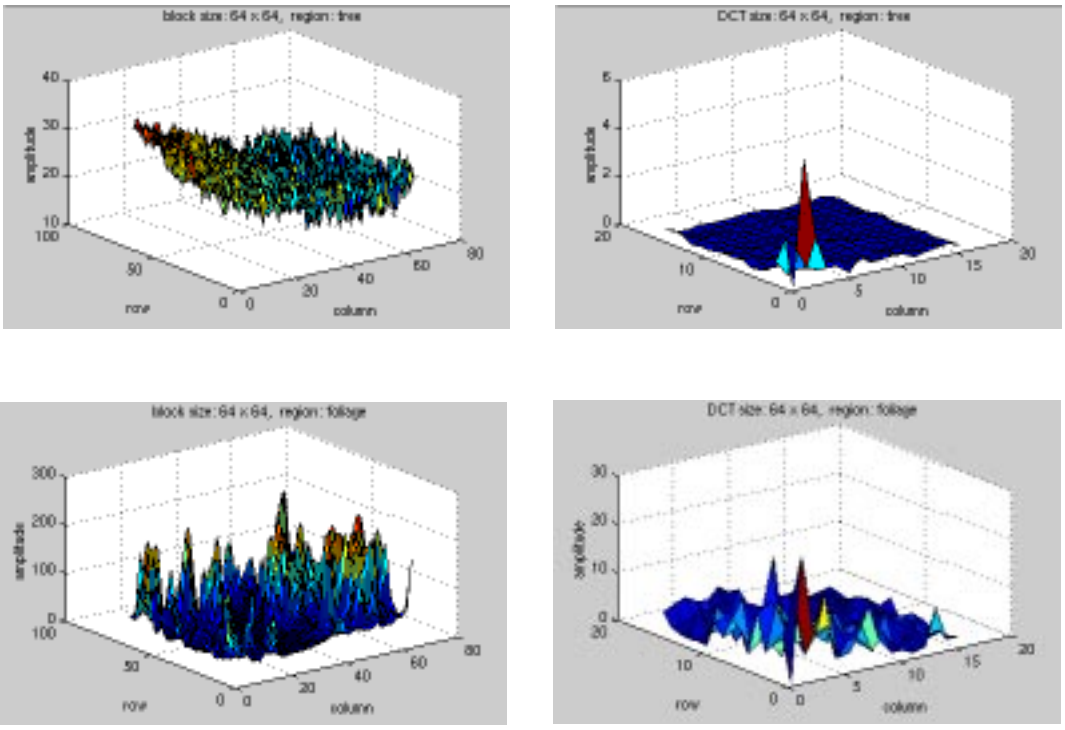


Figure 8. The amplitude distribution plots of DCT coefficients computed on some typical forestry regions (trees on the top; foliage on the bottom). Note the left column plots are for the original data, and the right ones are for the DCT coefficients.

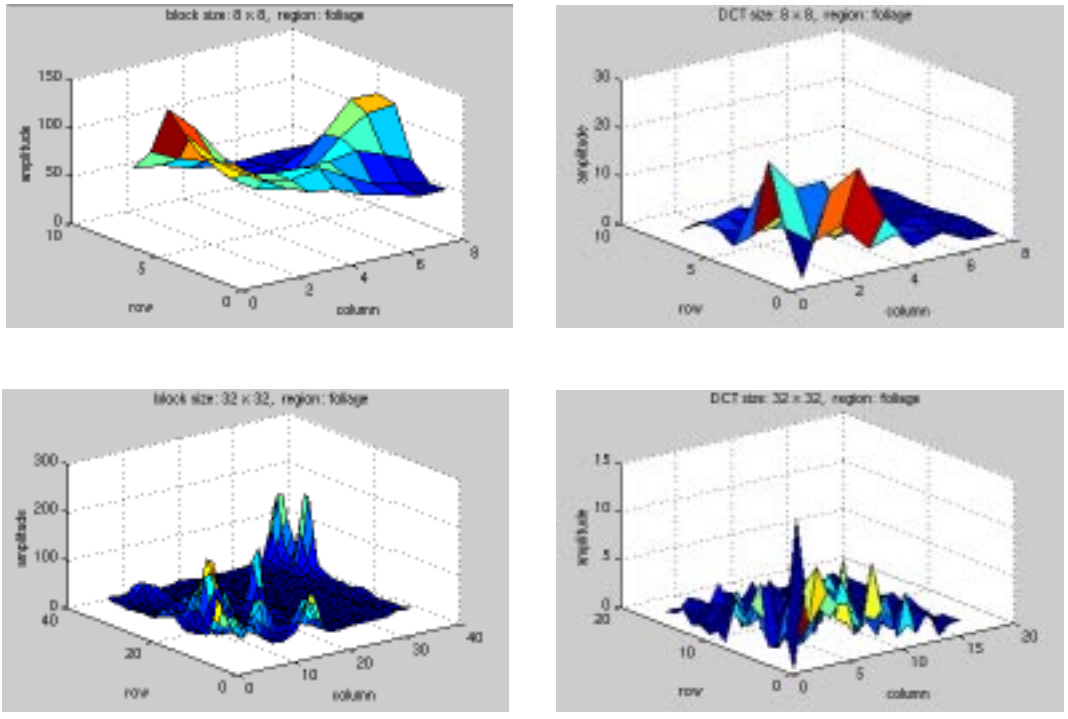


Figure 9. The amplitude distribution plots of DCT coefficients computed on a typical foliage region with the block size set as 8 x 8 and 32 x 32, respectively. Note the left column images are for the original data and the right ones are for the DCT coefficients.

averaging every four of all the DCT coefficients to generate a feature vector. Then the dimension of the outcome feature vector would be 64 x 64 / 4 = 1024. This large size would unnecessarily increase the computational complexity of training and test processes.

To examine how well our feature design fits for the segmentation problem, we ran several evaluation experiments on a small data set. The training data set is shown in Table 2. Note that we chose different images to train the models. The reason for this was to make the training process more efficient. Since the six categories of regions are distributed unevenly across the database, if we had used only one training set for all the models, the training set would have had to be increased to a much larger size to guarantee a sufficient amount of training data (image blocks) for each model. The test data set consisted of 5 random images with a total of 1406 64x64 image blocks.

As a first step, we evaluated the effects of the windowing scheme on this baseline system. The results indicate that the system performance is sensitive to the windowing parameters. As we see from Table 3, when the frame and window sizes were set to be 64x64 and 96x96 respectively, the system performed best on green pixels. Next, we investigated performance with different color features, i.e., we computed DCT coefficients on the red, green and blue pixels respectively. For these evaluations, we set the frame size to 64x64, and the window size to 96x96. As shown in Table 4, generally the error rates with all three colors were comparable. This similarity is reasonable because DCT coefficients are supposed to describe the spatial variation, or the texture pattern the image block displays. This texture characteristics should not vary remarkably with different color components.

In the schemes discussed so far, we chose the frequency filter size to be 4. However, that filter parameter may not necessarily be the optimal choice. Therefore, we tested a smaller filter size of 2. In this scenario, all other conditions for feature generation were kept the same as the baseline system. The windowing parameters also remained unchanged, that is, we used 64 x 64 frames and 96 x 96 windows. We observe in Table 5 that the system with a larger filter worked slightly better. However, the difference in the error performance was not significant.

Finally, we evaluated the impact of the number of DCT coefficients. The results are shown in Table 6. We used the first 64 and 128 coefficients to generate feature vectors respectively. For the first experiment, we had a 16-dimensional feature vector, while for the second experiment, we used a 32-dimensional vector. The results verified the idea that using more DCT coefficients is not necessarily better — with 128 DCT coefficients, the system performance was 5% worse than that of the system with the features based on only 64 coefficients.

3.2. Gabor Filter Bank-based Features

Another frequency analysis technique we investigated was Gabor filters. This filtering technique was justified by a physiological study of human vision system [9]. An unsupervised texture image segmentation algorithm based on this technique was developed by Jain and Farrokhnia and proved very effective in distinguishing between a small amounts of texture categories [10]. We first duplicated experiments described in the original paper and verified the performance of this algorithm. Then we applied it to images representing different forestry regions of interest in our study, and identified problems with this feature design for our application.

Region	Number Of Images	Number Of Blocks	Block Size
tree	5	523	64 x 64
foliage	6	632	64 x 64
bush	5	520	64 x 64
grass	6	523	64 x 64
background sky	10	675	64 x 64
sky	10	61	64 x 64

Table 2. The training data set for DCT-based feature evaluation.

Experiment No.	Frame Size	Window Size	Percentage Error Rate
1	32 x 32	32 x 32	72.6
2	32 x 32	64 x 64	67.6
3	64 x 64	64 x 64	67.9
4	64 x 64	96 x 96	**66.6**
5	64 x 64	128 x 128	67.6

Table 3. Performance for a variety of windowing schemes.

Experiment No.	Color Component	Percentage Error Rate
1	red	**65.7**
2	green	66.6
3	blue	67.3

Table 4. Performance across different color components.

Experiment No.	Filter Size	Percentage Error Rate
1	4	66.6
2	2	69.2

Table 5. Performance as a function of the frequency filter sizes.

Experiment No.	Amount Of DCT Coefficients Involved	Percentage Error Rate
1	64	66.6
2	128	72.5

Table 6. The performance evaluation results with different amounts of DCT coefficients involved in the feature computation.

3.2.1. Gabor Filter Bank

Gabor filters are important in visual image analysis. They have been found to fit very well for receptive field profiles of simple cells in a striate cortex [3]. The impulse response of an even-symmetric Gabor filter is given by:

$$h(x, y) = \exp\left\{-\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right\}\cos(2\pi u_0 x) \ . \tag{8}$$

Here, u_0 is the frequency of a sinusoidal plane wave along the x axis (or the 0^o orientation), σ_x is the space constant of the Gaussian envelope along the x axis, and σ_y is the other space constant along the y axis. To obtain a Gabor filter with an arbitrary orientation, we need to rotate the $x - y$ coordinate system accordingly.

The Fourier domain representation of (8) is given by

$$H(u, v) = A\exp\left\{-\frac{1}{2}\left[\frac{(u-u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\} + A\exp\left\{-\frac{1}{2}\left[\frac{(u+u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\} \tag{9}$$

where $\sigma_u = \frac{1}{2\pi\sigma_x}$, $\sigma_v = \frac{1}{2\pi\sigma_y}$, and $A = 2\pi\sigma_x\sigma_y$. This equation is also referred to as a *modulation transfer function* (MTF) since it specifies the amount by which the filter modifies each frequency component of the input image.

Gabor filters are able to keep an optimal balance between both the resolution in the spatial domain and that in the frequency domain [3]. This is a significant property for the texture segmentation problem, where high resolution in the spatial domain is necessary for locating texture boundaries, and smaller bandwidth in the frequency domain is desirable for distinguishing between different kinds of textures. The usefulness of Gabor filters in texture segmentation has been demonstrated by the research work described in [3].

3.2.2. Gabor Features Design And Evaluation

We designed our Gabor features on the basis of the design described in [3]. First, a bank of Gabor filters are convolved with an input texture image. For these Gabor filters, the orientation θ is set to

be 0°, 45°, 90°, and 135°, respectively, and the following values are used for the radial frequency u_0: $1\sqrt{2}$, $2\sqrt{2}$, $4\sqrt{2}$, ..., and $(N_c/4)\sqrt{2}$. Here N_c is the number of columns of the input image. This set of Gabor filters is sufficient for detecting textures with both high frequency components and low spectral contents, of any arbitrary orientation.

Second, the filtered outputs, denoted as $r_k(x, y)$, are transformed nonlinearly using the equation:

$$\psi(t) = \tanh(\alpha t) = \frac{1 - e^{-2\alpha t}}{1 + e^{-2\alpha t}} . \tag{10}$$

The curve for this function is shown in Figure 10. From the curve, we can see that the function actually stands for a rapidly saturating, threshold-like transformation, which would transform the sinusoidal modulations in the filtered images to square modulations. The constant in the equation, α, determines how fast the function saturates.

After the nonlinear transformation, feature images are generated by averaging over windowed regions centered at each pixel of the transformed images. The feature computation for a filtered image $r_k(x, y)$ is given as

$$e_k(x, y) = \frac{1}{M^2} \sum_{(a, b) \in W_{xy}} \left| \psi(r_k(a, b)) \right| , \tag{11}$$

where W_{xy} is an $M \times M$ window centered at the position of (x, y). Actually, the nonlinear transformation is incorporated into this equation.

In equation (11), the features are calculated with rectangular windows. To keep a balance between both a reliable measurement of texture features and an accurate localization of region boundaries, a Gaussian window, instead of the rectangular one, has been adopted in Jain's algorithm. Therefore, equation (11) is modified to:

$$e_k(x, y) = \frac{1}{M^2} \sum_{(a, b) \in W_{xy}} g(a - x, b - y) \left| \psi(r_k(a, b)) \right| \tag{12}$$

where

$$g(x, y) = \exp\left\{ -\frac{1}{2} \cdot \frac{x^2 + y^2}{\sigma^2} \right\} \tag{13}$$

The constant σ in (13) is proportional to the average size of the intensity variations in the image, which is denoted as T. That is, $\sigma = 0.5T$, where $T = N_c/u_0$. The scale was empirically determined to be 0.5. However, it was not mentioned in [3] as how to determine the window size M. It is supposed to be positively proportional to the value of σ, but the exact scale was not revealed in the paper.

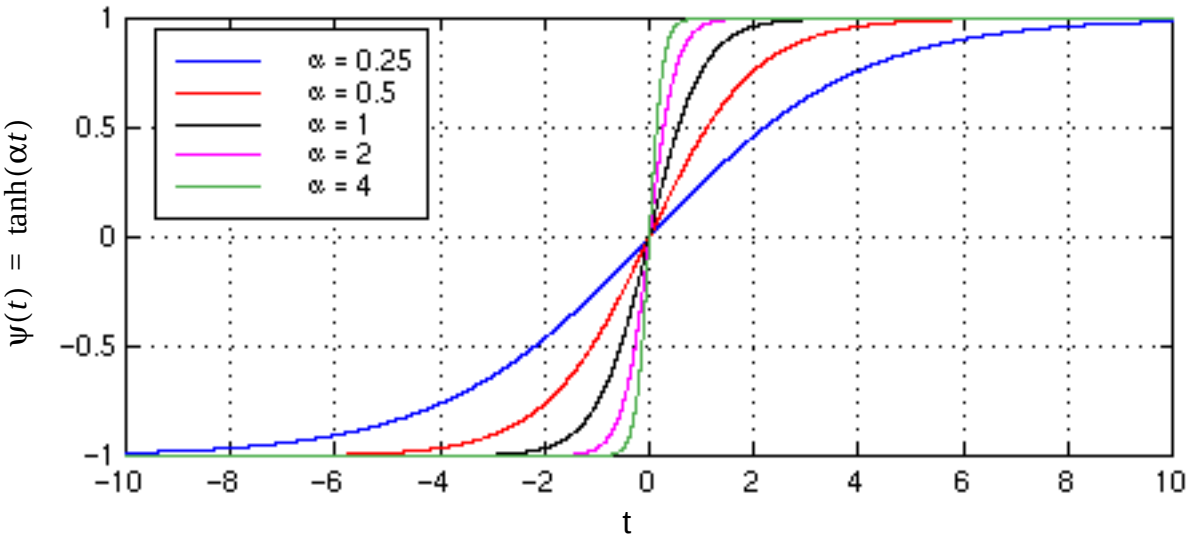


Figure 10. The nonlinear transformation used in the Gabor filter approach.

When feature computation is performed for all filtered images, feature values of the same position are assembled into a feature vector, which is then used for classifying the pixel at that position by *c-means* classification. In our first attempt, we tried to replicate the experiments presented in the paper. Classification results on an example texture image are shown in Figure 11. We did not get results exactly identical to the reported numbers, since we did not have access to the original texture images used in the reference paper, and some parameters such as the window size M were described ambiguously. However, our results were fairly comparable to the original results. We believe this is sufficient for verification purposes.

Therefore, we proceeded to evaluate this approach on our forest images. We selected several typical forestry regions from the Pre-Phase 01 image data set, and carried out pair-wise discrimination experiments on these regions. An example image and the corresponding segmentation are shown in Figure 12. More results can be found in the Appendix. In general, with these Gabor features, discrimination between regions was very encouraging. Most misclassification cases occurred in shaded areas — textures there were difficult to distinguish because of poor lighting conditions. It is interesting to note that in the experiment with the "bgsky-bush" pair, although the error rate was 22.0%, the segmenter was able to figure out the exact contour of the background sky area. The surrounding branches and leaves in the transcribed background sky region look more like the bushes on the right half of the image, resulting in their classification as bushes.

Unfortunately, the computational requirements for the algorithm are prohibitively time-consuming. With this algorithm, computing feature vectors for all pixels within a 128x128 image block took around 120 minutes on a 333 MHz processor with 512 Mbytes of memory. In our application, the dimension of an image is of 1536x1024. Consequently, if we apply the Gabor features introduced in the paper directly to the USFS image data, the feature computation on only one image would be approximately 192 hours. Hence, we investigated ways to speed up the algorithm without sacrificing performance.

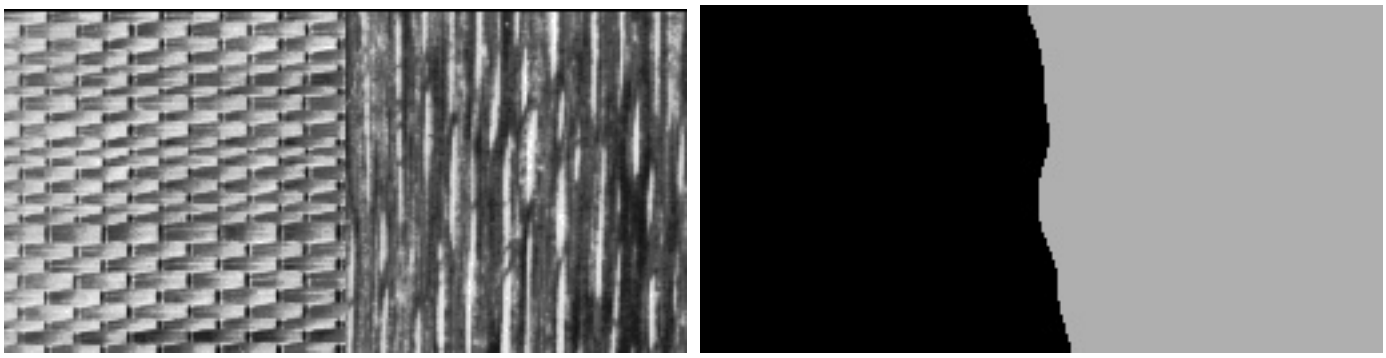


Figure 11. Results from an experiment to replicate what was reported in [3]. The original image on the left is the D55-D68 texture pair from the widely used Brodatz texture album. The segmentation is on the right. The pixel misclassification rate for this texture image was 1.3% with our experiment, while the error rate reported in the paper was 0.61%.

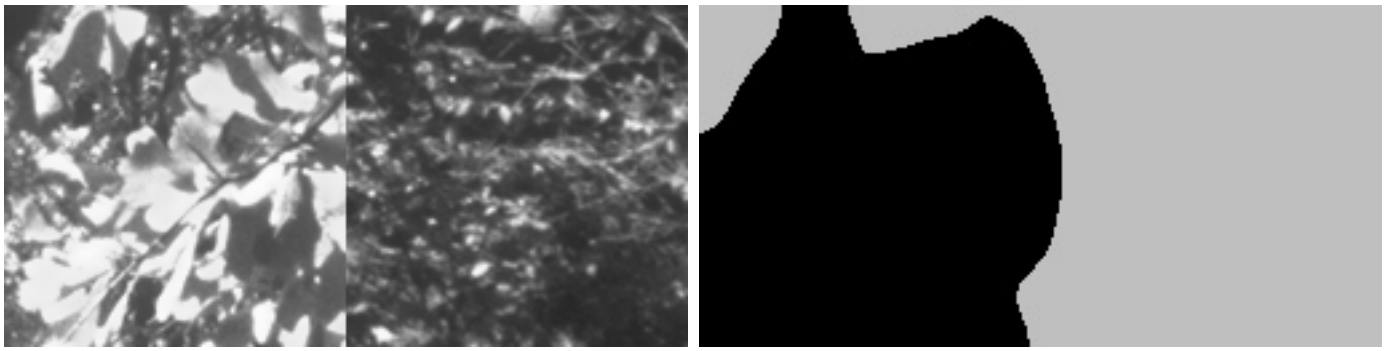


Figure 12. An example segmentation generated by applying Jain's algorithm to typical forestry regions. The image consists of two categories. The left half is a typical "foliage" block, and the right half is a "bush" block. In the segmentation image, the black region stands for hypotheses for the "foliage" category, while the grey color is used for "bush". The pixel classification error rate was 7.3%.

Since we are actually doing block-level classification, we only need one feature vector for an image block. As we observed in the previous exploration, the Gabor feature values computed for pixels of the same category did not vary dramatically. Hence it would be feasible to compute a feature vector for an image block by averaging over a small amount of neighboring pixels. In our experiments, we assumed that a feature vector computed at the central pixel of an image block represented all characteristics necessary to distinguish this block from blocks of other categories. With this assumption, the feature vector for a block center was used as the vector for the whole block. Hence the feature computation time for one 128x128 block was reduced from 120 minutes to 12 minutes. However, this time was still prohibitive. To run a training job on 1000 image blocks with a size of 128x128, the time needed is estimated to be 200 hours, or 8.3 days.

Further improvement on the computation speed can be achieved by adjusting the parameters used in generating the Gabor features. For example, we can reduce the number of Gabor filters. We could also decrease the size of the Gaussian windows applied to the filtered images. However, evaluation of such changes would require significant experimentation. Instead of pursuing this time-consuming engineering activity, we decided instead to being investigation of a new generation of segmentation algorithms based on syntactic constraints.

4. INCORPORATION OF SYNTACTIC INFORMATION

Thus far, image segmentation has been performed with each block being processed separately. We train models on typical image blocks, then classify a block by comparing its feature vector with the reference models. We always treat an image block independently, without relating it to any other blocks from the same image, as if there were no relationship between them.

However, since the images under study are photos of natural scenes, this assumption of independency should not be correct. As we can see, in a natural image, regions nearby usually share similar properties such as intensities and colors. Specifically, for forest images, a block belonging to the category of foliage is most probably surrounded by blocks of the same class, given that the block size is not too large. On the other hand, it is very unlikely that a bush block has no peers in its immediate neighbors. Therefore, if we could identify the connections between neighboring blocks, and incorporate such syntactic (or contextual) information into our algorithm, the accuracy of block classification should be improved.

4.1. Algorithm Design

First of all, we examined the relationship between neighboring pixels of an image. We investigated the correlation between intensity values of two neighboring pixels. Here, various neighborhood relationship, such as two adjacent points, two points that are two pixels apart, and two points that are four pixels apart, has been studied. By plotting the intensity value of one pixel against that of the other, we could see clearly how correlated they were. Such scatter plots are shown in Figure 13.

From these plots, it is clear that correlation between neighboring pixels is indeed very strong. This high correlation is reasonably explained by the fact that objects in a natural scene tend to be spatially continuous in their reflectance. The correlation justifies the idea of classifying a block with reference to categories of its neighbors. We designed our syntactic segmentation algorithm on the basis of adjacent blocks only. The basic idea behind the algorithm is: when we classify an image block, we not only examine the feature vector computed on this block, but also refer to the classification results for its adjacent neighbors. The algorithm is similar to the current PCA block classification scheme, with the only difference being that Euclidean distances are modified with weights representing extent of correlation between adjacent blocks.

Details of the syntactic algorithm are as follows. For training, in addition to creating statistical models for each class as we did for the PCA training, we also need to learn how each block is related to the adjacent blocks. We use a conditional probability to quantify the correlation relationship. That is, we find out the probability for a block to be of a specific class given category information of all its immediate neighbors. Suppose we have a block X with the neighborhood relationship as shown in Figure 14, and the total number of categories is n. then the probabilities may be held in a table illustrated in Figure 15.

For testing, to classify a block, we extract a feature vector from the block, then compute Euclidean distances from the feature vector to reference vectors from each trained model. Afterwards, we weight those Euclidean distances with corresponding conditional probabilities we

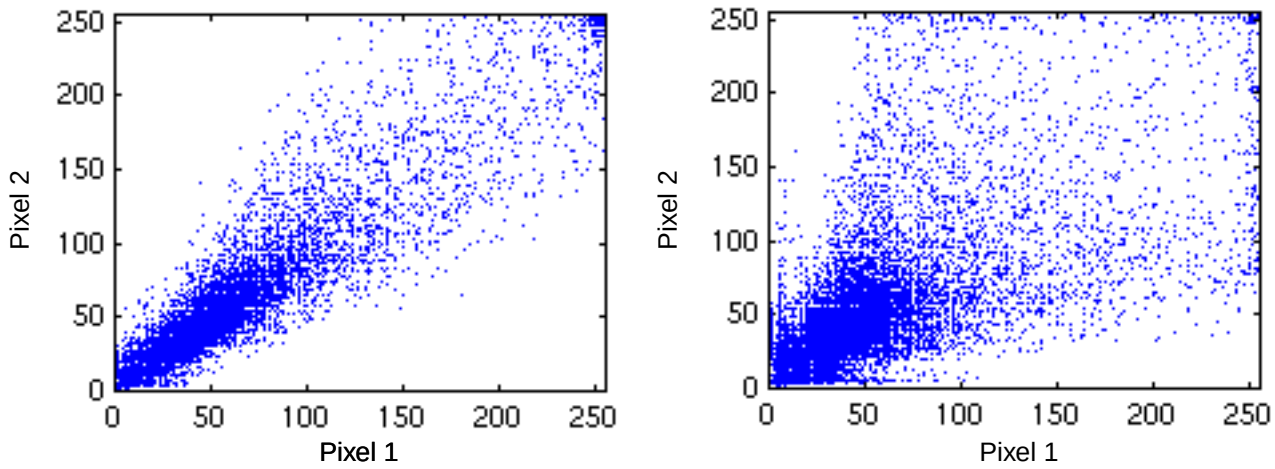


Figure 13. Scatter plots demonstrating correlation between neighboring pixels. For the left plot, pixel 1 and pixel 2 are adjacent to each other. For the right one, pixel 1 and pixel 2 are four pixels apart.

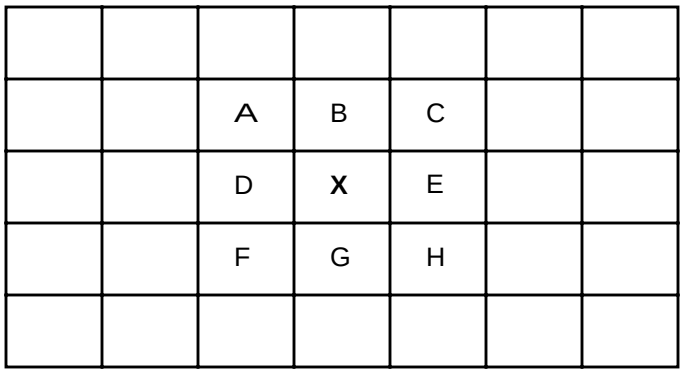


Figure 14. An example of the neighborhood relationship used for demonstrating the syntactic algorithm.

Central Block	Surrounding Blocks			
	A = 0, B = 0, C = 0, D = 0, E = 0, F = 0, G = 0, H = 0.	A = 1, B = 0, C = 0, D = 0, E = 0, F = 0, G = 0, H = 0.	...	A = n-1, B = n-1, C = n-1, D = n-1, E = n-1, F = n-1, G = n-1, H = n-1.
X = 0	P(X = 0|A = 0, ..., H = 0)	P(X = 0|A = 1, ..., H = 0)	...	P(X = 0|A = n-1, ..., H = n-1)
X = 1	P(X = 1|A = 0, ..., H = 0)	P(X = 1|A = 1, ..., H = 0)	...	P(X = 1|A = n-1, ..., H = n-1)
...	...	...	...	...
X = n-1	P(X = n-1|A = 0, ..., H = 0)	P(X = n-1|A = 1, ..., H = 0)	...	P(X = n-1|A = n-1, ..., H = n-1)

Figure 15. An illustration of the table used to store conditional probabilities. Here, the blocks are located as shown in Figure 14. The total number of categories is n.

acquired in training. The weighted distances are compared with each other to determine an appropriate category for the block under test. We compute a weighted Euclidean distance, or a likelihood, in the following way:

$$L(X = i) = d_i / P_r(X = i | A = a, B = b, C = c, D = d, E = e, F = f, G = g, H = h) , \quad (14)$$

where i stands for the i^{th} category $(0 \le i < n)$, d_i is the Euclidean distance between the test and reference vector for the i^{th} class, and the numerator is the conditional probability obtained from the training process. With all n likelihood values computed, the classification is done by finding the category corresponding to the smallest likelihood value. In this syntactic algorithm, we assume that all adjacent blocks are of equal importance to the central one.

4.2. Implementation Issues

The most important implementation issue is related to the table used to store conditional probabilities. Since we have n categories, if we compute probabilities conditioned on all eight adjacent blocks, the size of the table would be $n \times n^8 = n^9$. In the current segmentation problem, where $n = 6$, we would have to compute $6^9 = 10,077,696$ entries for the table. Suppose each probability value is represented by four bytes (a data type of *float* in C/C++), then the table alone would take about 38 Mbytes of memory. However, the table is expected to have many entries with values close to zero. Hence, sparse matrix representations of this data will be an important implementation issue.

To utilize the proposed syntactic information, we need to know category information from all adjacent neighbors of a block. It would be efficient if we implement the syntactic algorithm in a "post-processing" manner. That is, we first run the non-syntactic algorithm on a complete image, output information (category, feature vector, etc.) for all blocks into a data file, then apply the syntactic algorithm to the data file, through which we may search for information pertaining to all neighboring blocks we are interested in. This syntactic scheme could also be carried out in an iterative way. We did not have time to explore these alternatives.

5. SUMMARY AND FUTURE WORK

In this report, we reviewed progress for the past year. We have focused on feature and algorithm development. For feature development, first we optimized the edge and line detectors by tuning key parameters. We designed an evaluation-driven optimization scheme, and proposed an objective metric for performance evaluation. Second, we investigated the use of frequency domain features based on the discrete cosine transform. We analyzed the DCT coefficients' amplitude distribution on typical forestry regions and discovered a few basic rules which would be helpful for segmentation. Based on this analysis, we designed several feature extraction schemes and evaluated their effectiveness. The best system performance with these DCT-based feature designs was 65.7% block classification errors on a small pilot data set. Third, we explored the use of a texture classification scheme based on Gabor filters for image segmentation.

We also initiated investigations into incorporating syntactic information into the segmentation system. We explored correlation between neighboring image pixels. We found out that closely located pixels are strongly correlated to each other in properties such as intensity. Based on this, we proposed a syntactic segmentation algorithm which incorporates category information of all adjacent blocks into the classification of a specific block. We also discussed some important issues related to the implementation of this approach.

There are still many interesting issues to be explored in forestry image analysis. We need to improve our preliminary feature design using DCT coefficients. As discussed earlier, there are still parameters, such as the frequency filter size, and the number of DCT coefficients involved in the computation, which need to be optimized. Moreover, to combine the DCT coefficients more effectively, we will have to understand thoroughly the physics behind the DCT. The Gabor filter bank-based features are very promising for segmenting forest images, mainly because it reveals the nature of human vision system. At this point, what we need to do is to solve the problem related to the computation speed.

Other features worthy of investigation include contrast and correlation [11]. Contrast possibly describes the lighting condition, which is a very important factor in subjective scenic beauty evaluation of a forest scene. Correlation has already been discussed in the syntactic algorithm design, and is one of a number of issues relating to the use of syntactic information that need further exploration.

6. ACKNOWLEDGMENT

This work was supported by the United States Department of Agriculture (USDA) through the Ecosystem Management research for the Ouachita-Ozark National Forests at the United States Forest Service (USFS) Southern Research Station. This project has been ISIP's longest running research project. Our long association with Vic Rudis of the Southern Research Station has been enjoyable and educational. The work presented on this project over the years has impacted many other projects in ISIP, and has been an important proving ground for basic signal processing technology.

7. REFERENCES

[1] N. Kalidindi, V. Ramani and J. Picone, "Scenic Beauty Estimation of Forestry Images," *Final Project Report for the Southern Forest Experiment Station, United States Forest Service*, Institute for Signal and Information Processing, Mississippi State University, December 1998.

[2] Z. Long and J. Picone, "Applications of High Performance Statistical Modeling to Image Analysis of Forest Structure," *Final Project Report for the Southern Research Station, United States Forest Service*, Institute for Signal and Information Processing, Mississippi State University, August 1999.

[3] A. K. Jain and F. Farrokhnia, "Unsupervised Texture Segmentation Using Gabor Filters," *Pattern Recognition*, vol. 24, no. 12, pp. 1167-1186, December 1991.

[4] R. N. Strickland and D. Chang, "An Adaptable Edge Quality Metric," *Proceedings of SPIE*, vol. 1360, pp. 982-995, Bellingham, Washington, USA, October 1990.

[5] L. Kitchen and A. Rosenfeld, "Edge Evaluation Using Local Edge Coherence," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-11, no. 9, pp. 597-605, September 1981.

[6] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679-698, November 1986.

[7] Z. Long and J. Picone, "USFS: DATA," *http://www.isip.msstate.edu/projects/usfs/data/index.html*, Institute for Signal and Information Processing, Mississippi State University, January 2000.

[8] M. Rabani and P. W. Jones, *Digital Image Compression Techniques*, SPIE Optical Engineering Press, Bellingham, Washington, USA, 1991.

[9] B. A. Olshausen and D. J. Field, "Vision and the Coding of Natural Images," *American Scientist*, vol. 88, no. 3, pp. 238-245, May 2000.

[10] Y. Huang and R. Chang, "Texture Features for DCT-Coded Image Retrieval and Classification," *ICASSP Proceedings*, vol. 6, pp. 3013-3016, Phoenix, Arizona, USA, March 1999.

[11] H. Mo, S. Satoh, and M. Sakauchi, "Video Scene Annotation by Classification Based on Typical Scene Images," *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, vol. 5, pp. 223-228, Orlando, Florida, USA, July 2000.

APPENDIX

A. Optimization Of The Edge And Line Detectors

Exp.	σ	Set 1				Set 2			
		Reference	Detection	Error(%)	Insertion	Reference	Detection	Error(%)	Insertion
1	1.0	3176	681636	67.6	680608	3376	746269	41.7	744302
2	2.0	3176	640911	67.5	639880	3376	703848	41.6	701877
3	3.0	3176	371683	73.1	370830	3376	421944	48.6	420208
4	4.0	3176	197896	78.6	197216	3376	232348	56.5	230879

Table 1. Optimization experiments with the Gaussian variance. Here the high and the low thresholds for the edge detector were 180 and 60 respectively, and the line threshold was 15. Data set 1 contained 165 images randomly chosen from the training set 01 of USFS Pre-phase 01 image data. Data set 2 consisted of 159 images randomly chosen from the test set 01 of Pre-Phase 01.

Exp.	High Threshold	Low Threshold	Set 1				Set 2			
			Reference	Detection	Error(%)	Insertion	Reference	Detection	Error(%)	Insertion
1	240	120	3176	34555	81.5	33969	3376	42993	61.8	41705
2	220	110	3176	42697	80.0	42063	3376	52557	58.9	51168
3	200	100	3176	53522	78.4	52837	3376	65214	55.1	63698
4	180	90	3176	68390	76.0	67627	3376	82599	52.1	80981
5	160	80	3176	90358	73.0	89501	3376	107854	48.2	106106
6	140	70	3176	125070	70.2	124125	3376	146506	44.3	144626
7	120	60	3176	182904	66.2	181831	3376	209430	39.5	207389
8	100	50	3176	284695	60.6	283444	3376	317499	34.2	315279
9	80	40	3176	468147	53.6	466674	3376	507845	29.0	505449
10	60	30	3176	801505	45.1	799761	3376	840532	23.4	837947
11	50	25	3176	1060511	40.5	1058622	3376	1086917	21.1	1084253
12	40	20	3176	1408473	35.7	1406430	3376	1406020	20.7	1403342
13	30	15	3176	1860484	30.4	1858273	3376	1812818	22.8	1810212
14	20	10	3176	2350541	26.4	2348203	3376	2248794	29.0	2246397

Table 2. Optimization experiments with the high and low thresholds of the edge detector. Here the standard deviation was 2.0, the line threshold was 25, and the low thresholds were set to be half of the high thresholds as recommended in Canny's paper.

Exp.	Low Threshold	Set 1				Set 2			
		Reference	Detection	Error(%)	Insertion	Reference	Detection	Error(%)	Insertion
1	80	3176	95603	72.4	94728	3376	113599	47.0	111810
2	60	3176	186558	65.8	185471	3376	213187	39.0	211128
3	40	3176	460542	53.9	459077	3376	500603	29.6	498226
4	20	3176	1370829	36.0	1368795	3376	1370686	21.4	1368032
5	10	3176	2308121	26.3	2305780	3376	2211064	29.2	2208674

Table 3. Optimization experiments with the low threshold of the edge detector. Here the high threshold was 100, the standard deviation was 2.0, and the line threshold was 25.

Exp.	Line Threshold	Set 1				Set 2			
		Reference	Detection	Error(%)	Insertion	Reference	Detection	Error(%)	Insertion
1	45	3176	178213	48.8	176586	3376	198994	34.2	196773
2	40	3176	248794	47.2	247118	3376	274461	29.3	272073
3	35	3176	355927	45.9	354210	3376	387083	25.7	384575
4	30	3176	524477	45.3	522741	3376	560715	24.0	558148
5	25	3176	801505	45.1	799761	3376	840532	23.4	837947
6	20	3176	1278402	45.0	1276656	3376	1309756	23.4	1307169
7	15	3176	2163597	45.0	2161851	3376	2160635	23.4	2158048

Table 4. Optimization experiments with the line threshold. Here the high and low thresholds for the edge detector were 60 and 30 respectively, and the standard deviation was 2.0.

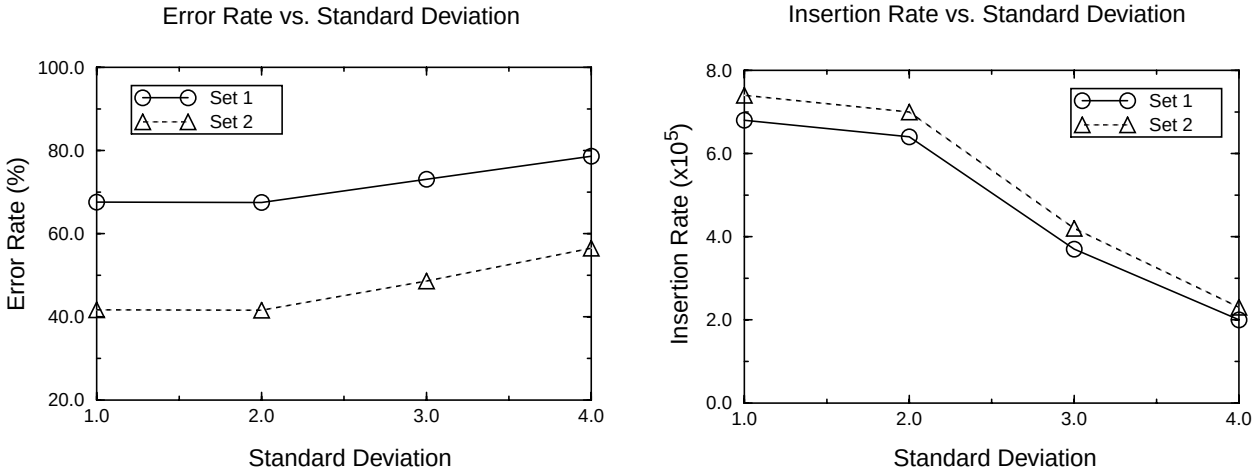


Figure 1. Results from optimization experiments with the Gaussian standard deviation involved in Canny edge detection algorithm. Data set 1 contained 165 images randomly chosen from the training set 01 of USFS Pre-phase 01 image data. Data set 2 consisted of 159 images randomly chosen from the test set 01 of Pre-Phase 01. The high edge threshold was 180; the low edge threshold was 60; and the line threshold was 15.

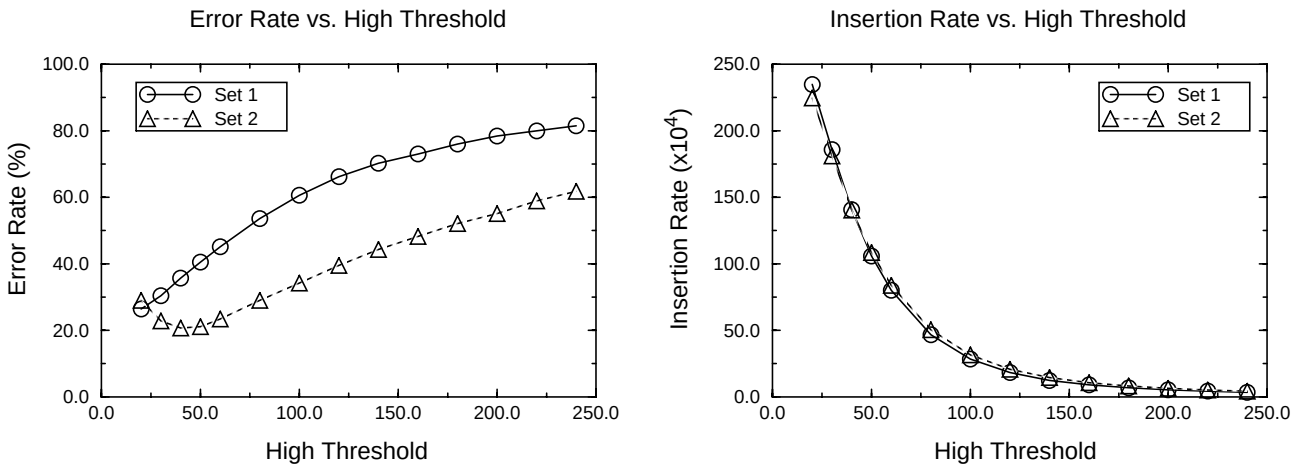


Figure 2. Results from optimization experiments with the high edge threshold involved in Canny edge detection algorithm. Data sets were the same as those in Figure 1. The Gaussian standard deviation was 2.0; the low edge threshold was set half of the high edge threshold as suggested in Canny's paper; and the line threshold was 25.

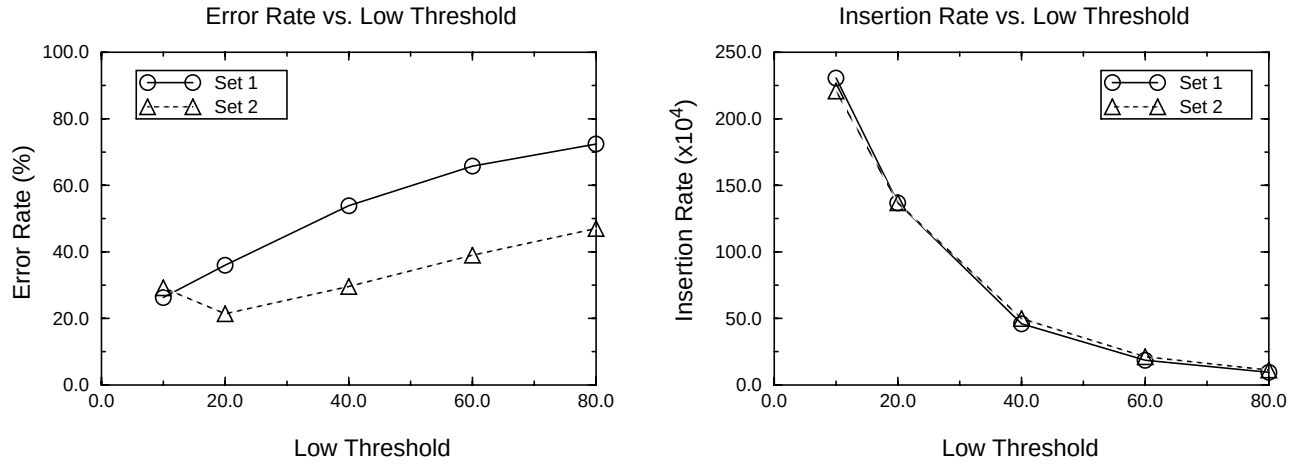


Figure 3. Results from optimization experiments with the low edge threshold involved in Canny edge detection algorithm. Data sets were the same as those in Figure 1. The Gaussian standard deviation was 2.0; the high edge threshold was 100; and the line threshold was 25.

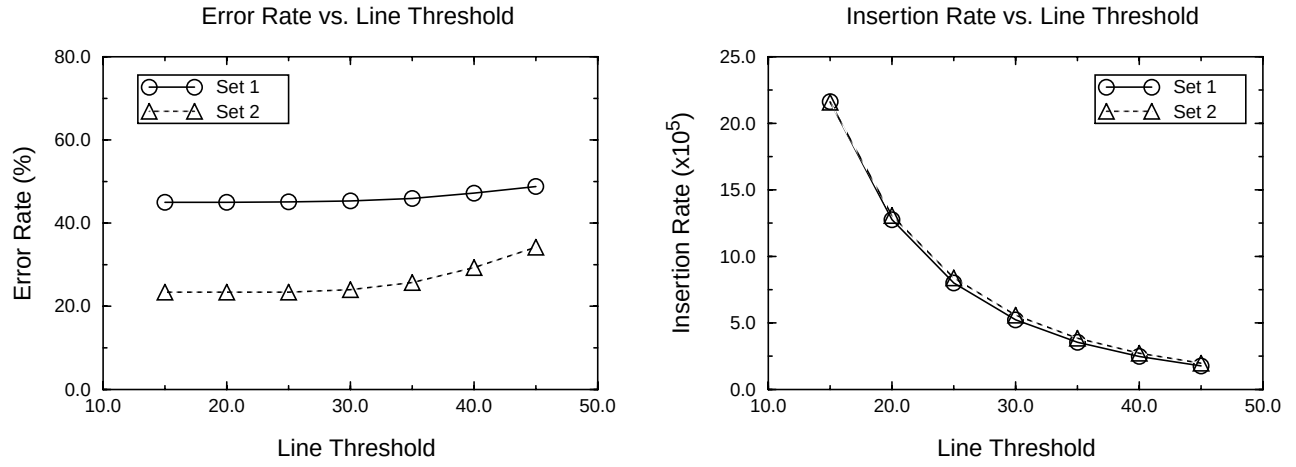


Figure 4. Results from optimization experiments with the line threshold involved in the line detection algorithm. Data sets were the same as those in Figure 1. The Gaussian standard deviation was 2.0; the high edge threshold was 60; and the low edge threshold was 30.

B. DCT Amplitude Plots

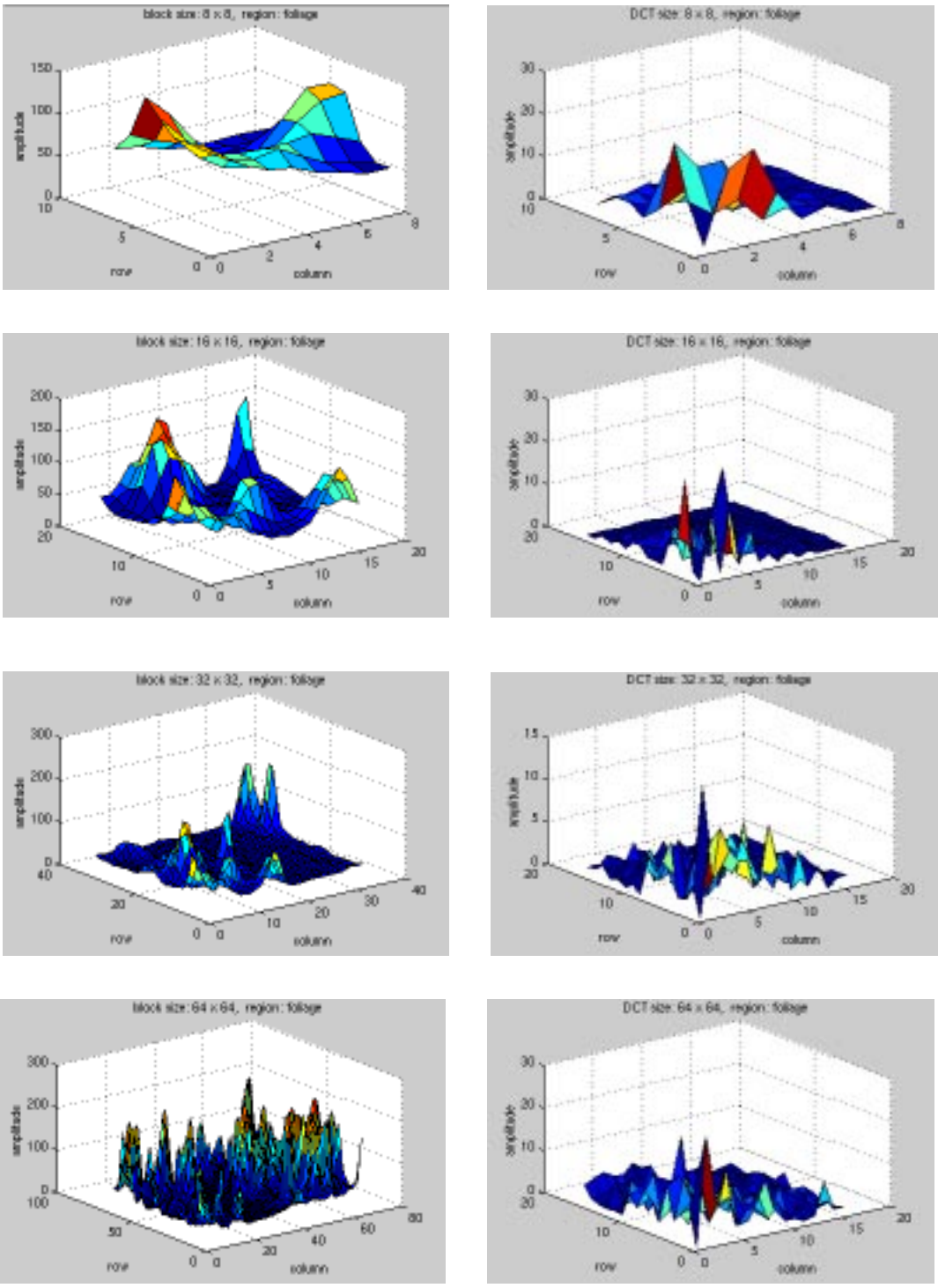


Figure 5. The amplitude distribution plots of DCT coefficients computed on a typical foliage region with different block sizes. Note the left column images are for the original data and the right ones are for the DCT coefficients.

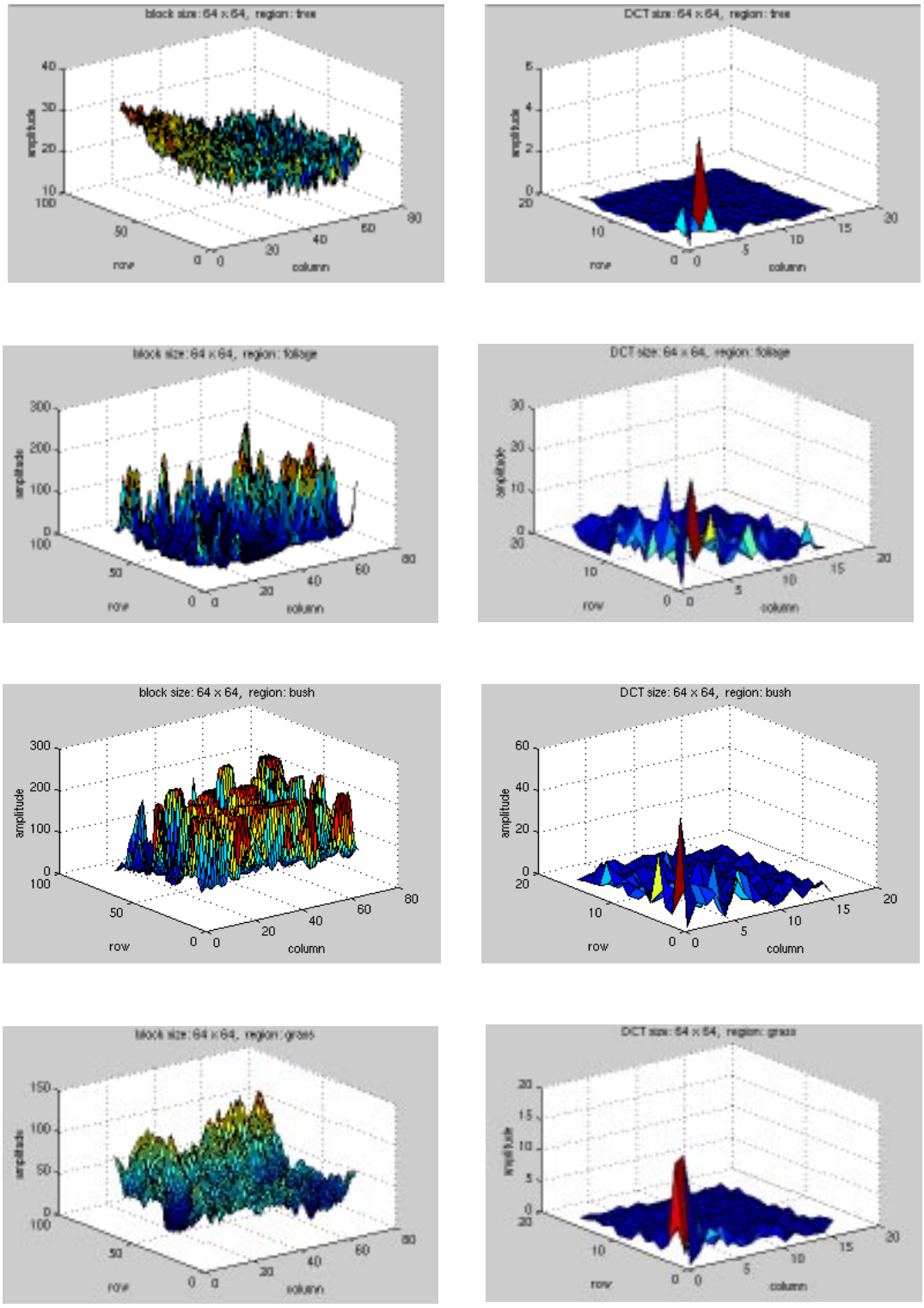


Figure 6. The amplitude distribution plots of DCT coefficients computed on typical forestry regions. Note the left column plots are for the original data, and the right ones are for the DCT coefficients.

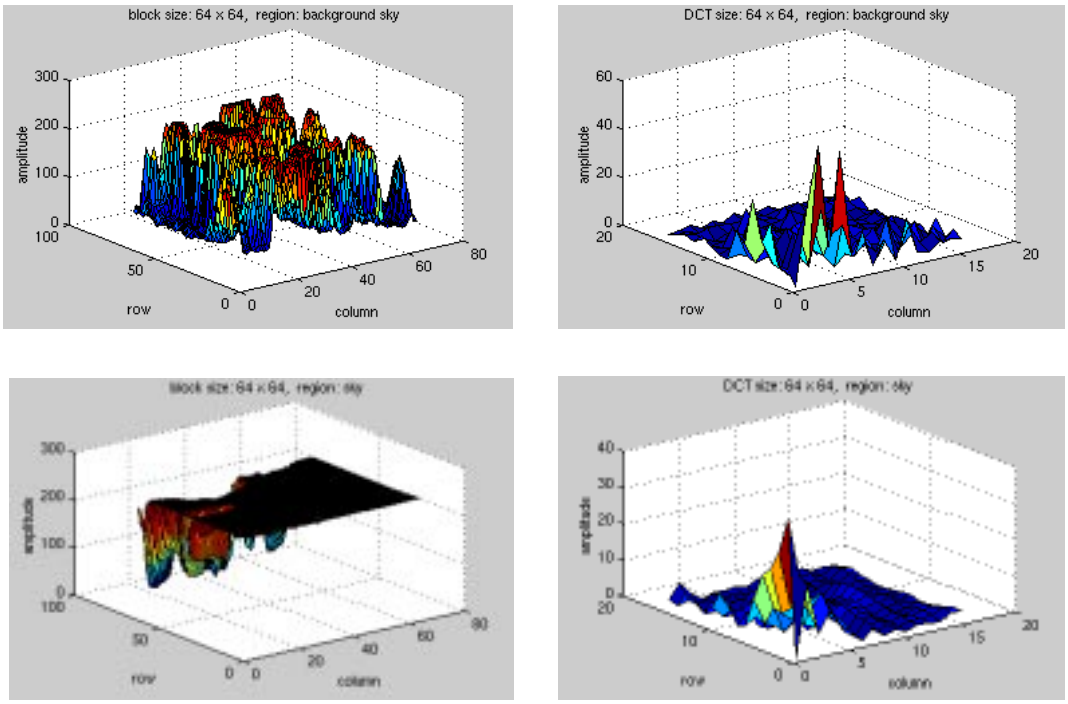


Figure 7. The amplitude distribution plots of DCT coefficients computed on typical forestry regions (continued).

C. Segmentation Results

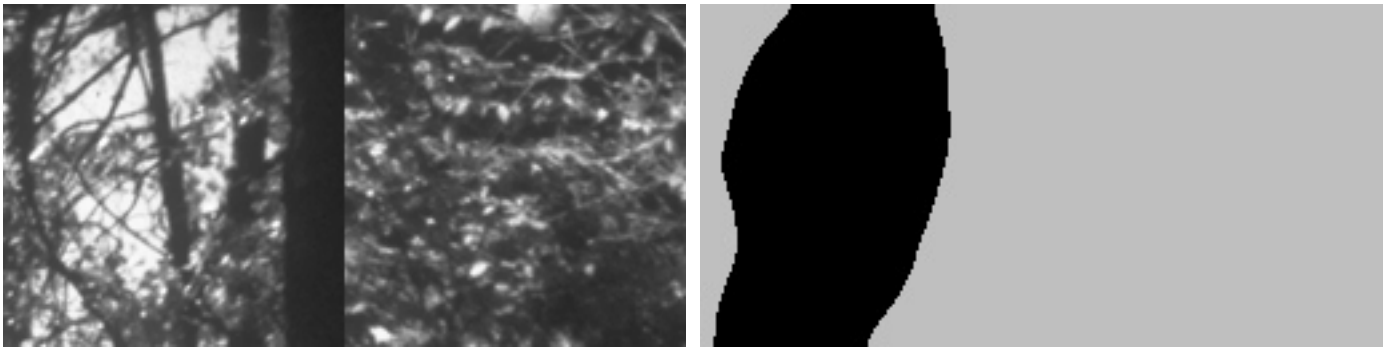


Figure 8. The image consists of two forestry regions. The left half is a typical “bgsky” (background sky) block, and the right half is a “bush” block. In the segmentation image, the black region stands for hypotheses for the “bgsky” category, while the grey color is used for “bush”. The pixel classification error rate was 22.0%.

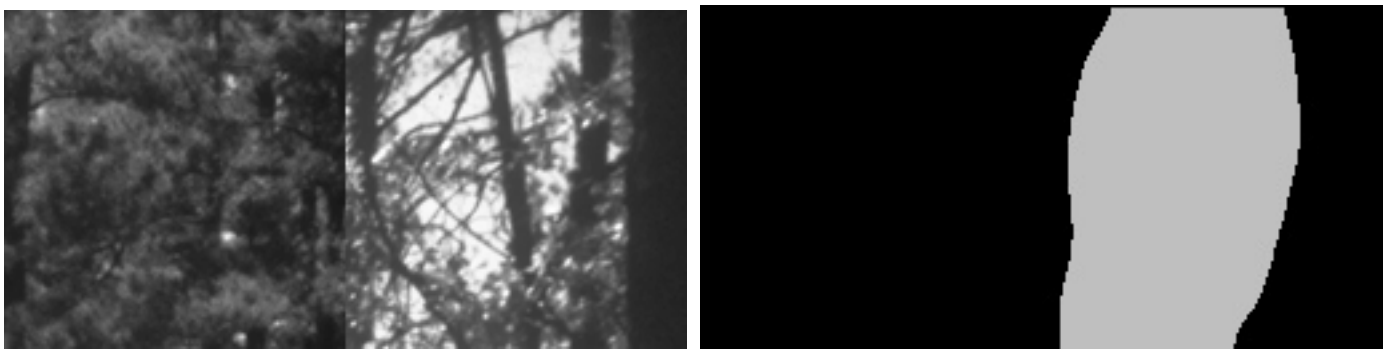


Figure 9. The image consists of two forestry regions. The left half is a typical “foliage” block, and the right half is a “bgsky” block. In the segmentation image, the black region stands for hypotheses for the “foliage” category, while the grey color is used for “bgsky”. The pixel classification error rate was 19.6%

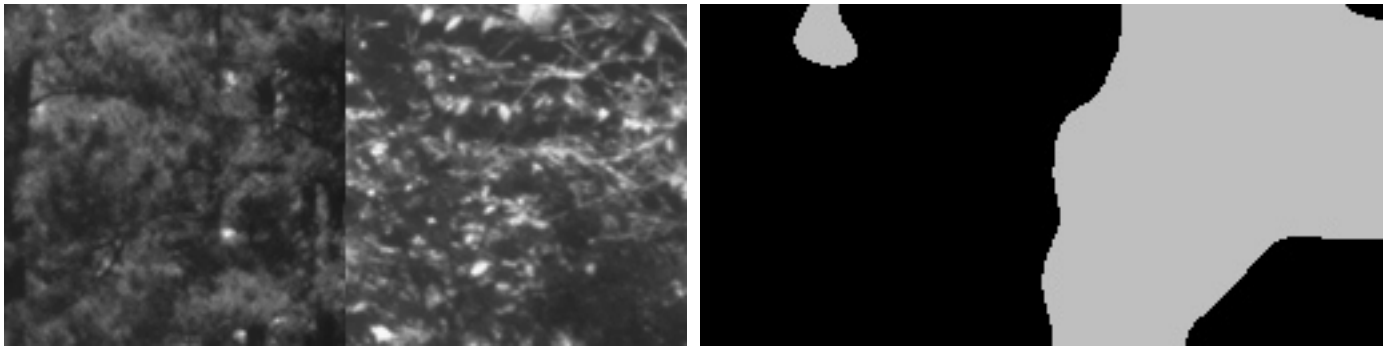


Figure 10. The image consists of two forestry regions. The left half is a typical “foliage” block, and the right half is a “bush” block. In the segmentation image, the black region stands for hypotheses for the “foliage” category, while the grey color is used for “bush”. The pixel classification error rate was 12.9%.

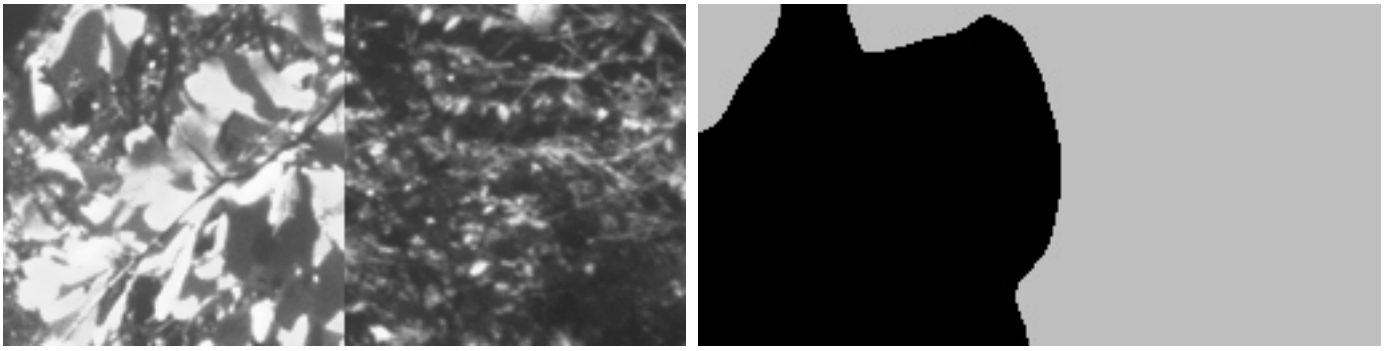


Figure 11. The image consists of two forestry regions. The left half is another typical “foliage” block, and the right half is a “bush” block. In the segmentation image, the black region stands for hypotheses for the “foliage” category, while the grey color is used for “bush”. The pixel classification error rate was 7.3%.

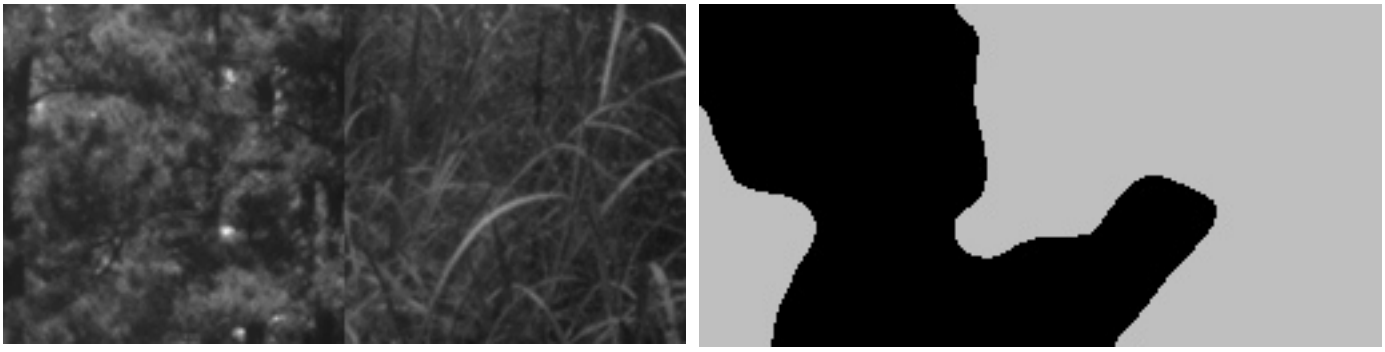


Figure 12. The image consists of two forestry regions. The left half is a typical “foliage” block, and the right half is a “grass” block. In the segmentation image, the black region stands for hypotheses for the “foliage” category, while the grey color is used for “grass”. The pixel classification error rate was 21.9%.

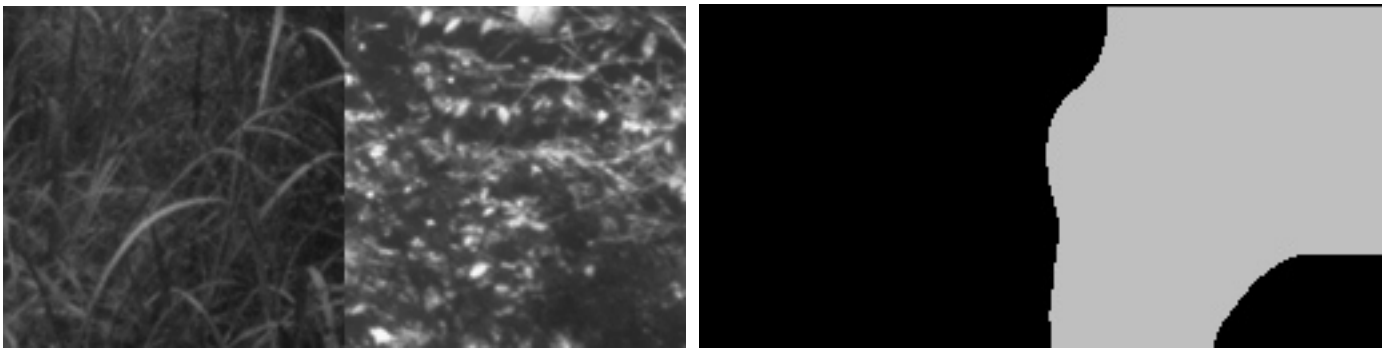


Figure 13. The image consists of two forestry regions. The left half is a typical “grass” block, and the right half is a “bush” block. In the segmentation image, the black region stands for hypotheses for the “grass” category, while the grey color is used for “bush”. The pixel classification error rate was 9.1%.

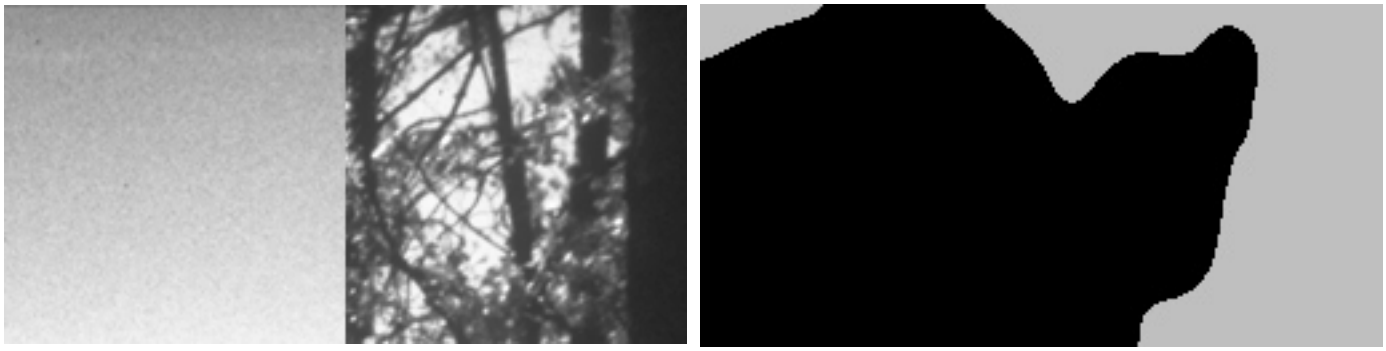


Figure 14. The image consists of two forestry regions. The left half is a typical “sky” block, and the right half is a “bgsky” block. In the segmentation image, the black region stands for hypotheses for the “sky” category, and the grey color is used for “bgsky”. The pixel classification error rate was 23.7%.