Feature Sets for Texture Classification

Paul E. Utgoff, Gang Ding, & Edward M. Riseman

CMPSCI TR 03-35
November 16, 2003

Feature Sets for Texture Classification

Paul E. Utgoff
Gang Ding
Edward M. Riseman

Technical Report 03-35
November 16, 2003

Department of Computer Science
140 Governor's Drive
University of Massachusetts
Amherst, MA 01003

This report was completed September 26, 2002.

Keywords

Texture classification, feature set, artificial neural network, forest classification, brodatz textures, co-occurrence matrix, equalization.

Abstract

Many classification tasks require the ability to discriminate instances by characteristics of their visual texture. A great many approaches have been developed for this purpose, including many based on aggregate features computed as functions of a co-occurrence matrix that is indexed by a pair of gray-level values. We investigate use of the co-occurrence probabilities themselves as a a set of features for use as the basis for training a classifier. Empirical results are presented for a forest-type classification task of interest to environmentalists, and on a set of well-known difficult texture classification problems.

Summary

The paper explores a particular combination of texture representation and classifier inducer that produce high classification accuracies. The texture representation is the set of gray-level co-occurrence frequencies normalized to form a probability distribution. The classifier inducer is the RPROP form of error backpropagation as used to train a two-layered artificial neural network. This combination does very well on a set of Brodatz textures, evidenced by outperforming the best algorithms reported by Randen & Husøy in seven of twelve cases.

The paper describes two applications. The first is a forest-type classification problem, in which each image is an aerial views of a tract of forested land. This is a real application with real data. The second is a set of Brodatz textures, as mentioned above. Various experiments are reported, one in which the effect of paying attention to pixel orientation is varied, one in which the equalization methods is varied, and one in which the effect of applying a smoothing operator is measured.

1 Introduction

The ability to discriminate visually those items that have different characteristic textures is essential for many important applications. For example, one may need to recognize objects for automation purposes in an industrial setting, or detect anomalies in sectional photographs for purposes of medical diagnosis, or measure various categories of vegetation in aerial surveys. When the required discriminations are difficult in terms of features such as shape or color, one may need to rely on textural features. Even when other clues may be strong, texture features may be able to augment them to improve accuracy.

We focus here upon visual texture, in particular a two-dimensional view with each pixel represented by gray-level intensity. By its very nature, no single pixel can exhibit texture. Rather, texture is a function of the pattern of variation in intensity of a local region of pixels. Too small a region prevents perceiving the texture, and too large a region adds no new information because the essence of the texture is already evident. Assuming that one observes a local region of adequate size, how can texture be represented in a manner that enables discrimination by a classification algorithm? When texture differences are subtle, discrimination among classes becomes quite challenging. The problem is even more difficult near the boundaries of texture regions.

It is common practice to build a texture classifier by compiling a set of labeled training examples and presenting them to a classifier inducer (learning program). Each training example is an (image region, texture class label) ordered pair. By including representative examples for all the texture classes, it is left to the classifier inducer to identify an accurate means of mapping an unlabeled image region to its class.

The designer must select a learning algorithm and a representation of the image that provides measurable characteristics (features) of texture that the algorithm can employ to build the classifier. There are many induction algorithms from which to choose, but none is yet powerful enough on its own to invent a good mapping from the pixels of the image to the class label. Instead, the designer must invent one or more intermediate representations, so that the learning algorithm can perform satisfactorily in the designed instance representation. An intermediate representation is computed by one or more transformations of the input pixel image. Each such transformational computation is known as a *feature*, and the result that it computes is a *feature value*. A feature set constitutes an alternative representation of its input. The features are useful to the extent that building a classifier over the alternative representation produces an accurate classifier.

We measure how well a particular feature set combined with a classifier induction algorithm performs on real pixel data for a forest-type classification task. We also examine how well this combination does on several well-known texture classification tasks from the Brodatz album. For the forest-type task, four classifier inducers are measured. For the well-known tasks, just one classifier inducer is measured, but the results are compared to those presented recently by Randen & Husøy (Randen & Husøy, 1999). Our feature set and learning algorithm produces a classifier of noteworthy classification accuracy.

2 Feature Set Construction

Haralick, Shanmugam & Dinstein (Haralick, Shanmugam & Dinstein, 1973) devised an approach to constructing texture features based on their notion of *spatial dependency*. Instead of considering individual pixels, one considers pairs of pixels at various distances and orientations. By tallying the occurrences of each possible pair, and dividing by the total, one obtains a probability distribution over the possible pairs. To the extent that one can discriminate one probability distribution from another, one can thereby discriminate one texture from another, using each probability distribution as a signature. Indeed, this is the fundamental purpose of feature construction, to produce functions that associate a robustly discriminable set of feature values with each class.

This probability distribution provides a basic representation of the texture in the local region from which the pixel pairs were drawn. It is stored in a lower triangular matrix called the *co-occurrence matrix*, indexed by the gray-level value of each pixel of a pixel pair. For convenience, we shall refer to the probability distribution stored in the triangular matrix simply as the *triangular distribution*. A co-occurrence pair is

inherently unordered. When the orientation of the pixels is of interest, then one should not use a triangular matrix, or one should use a separate triangular matrix for each orientation of interest. For example, for four possible pixel-pair orientations (horizontal, vertical, rising diagonal, falling diagonal), one could produce four triangular matrices. Similarly, to be sensitive to the distance of the two pixels of a pixel pair, one can maintain a triangular distribution for each of the selected distances. A collection of triangular distributions might be considered to be wedge-shaped, but we will stick with the simpler notion of triangular shaped distribution.

The two indices of the triangular matrix are the gray-level values of the two pixels of a pixel pair. For this reason, the size of the matrix depends on the number of gray-levels chosen for the normalization step. One needs to choose enough gray levels to allow discrimination of the various textures. Fewer gray levels corresponds to a smaller matrix and a coarser representation of texture. Similarly, more gray levels corresponds to a larger matrix and a finer representation of texture. There is a classic tradeoff between adequacy of representation and task simplicity. Consider a triangular matrix for pairs of adjacent pixels in any orientation. For g gray levels, there will be $g(g+1)/2$ entries in the triangular matrix. For $g = 8$, a common choice, there would be 36 entries. For $g = 20$, there would be 210 entries, which would be quite unwieldy.

For a given number of gray levels g, one obtains a corresponding triangular matrix. Because the triangular distribution is constructed (estimated) from a sample of pixel pairs, one must take steps to ensure an adequately-sized sample. For our simple example, the sample of pixel pairs is created by taking all adjacent pairs of pixels in a local region of the image. To obtain a sample of adequate size, one must choose the size of the local region to be considered, which determines how many (and which) pixel pairs will be in the sample. Given a local region, the process of producing the sample of pixel pairs and computing the triangular distribution is entirely determined, as is described below.

Having chosen the number of gray levels g, one then needs to choose a sample size of pixel pairs that will lead to a well-estimated probability distribution. For example, suppose that we would like to have $50 \cdot g(g+1)/2$ pairs counted. Then for $g = 8$, we would want 1800 pixel pairs. For a square pixel region of size kxk, there are $2(k-1)(2k-1)$ pixel pairs, under the assumption of wanting all pairs of adjacent pixels in any orientation. Hence, we would need a pixel region of size 22x22. So, as g is raised, the size of the necessary pixel region rises quadratically with it.

Similarly, if one wishes to pick a region of a particular size and have a sufficiently large sample of pixel pairs, then one can afford only so many gray levels. Under our pixel pair sampling assumption (adjacent in any orientation), the number of gray levels g, the length of the side of a square pixel region k, and the average number of samples per triangular matrix cell s are related by $s \cdot g(g+1)/2 \leq 2(k-1)(2k-1)$. There are three degrees of freedom. For usefully small region sizes, say 32x32, and adequate sampling, say 100 pairs per cell on average, the number of possible gray levels will also be small, in this case 8.

We refer to all the elements of the triangular distribution as the *triangular distribution features*, indicated by *tdf* in the various tables below. Recall that each element indicates the probability of seeing that pair of gray values in the region on which the distribution is based. It is customary, due to Haralick and others, to think of computing the triangular distribution as a stepping stone to computing another layer of feature values, based on various properties of the triangular distribution features. This idea of mapping the triangular distribution, which is a basic representation of the texture of a region of an image, to a higher level representation is a classical approach to simplifying the task of inducing an accurate classifier (Duda & Hart, 1973; Mitchell, 1997). Mapping from the triangular distribution to a feature vector loses information in an attempt to gain regularity. One needs to determine the extent to which mapping from one representation to another helps classification accuracy.

Haralick *et al* implemented fourteen (14) features, each of which computes a single real value from the triangular distribution features. Weszka et al (Weszka, Dyer & Rosenfeld, 1976) studied using just four of these as a feature set. These are *Energy*, computed as: $\sum_{i,j} P(i,j)^2$, *Inertia*, computed as: $\sum_{i,j}(i-j)^2 P(i,j)$,

Correlation, computed as: $\frac{\sum_{i,j}(ij)P(i,j)-\mu_x\mu_y}{\sigma_x\sigma_y}$, and *Entropy*, computed as: $-\sum_{i,j}P(i,j)\log P(i,j)$. We refer to this set of four features simply as the co-occurrence features *csf*, noting that it is drawn from the strictly larger set of original co-occurrence features. This is a feature set that was used in the study by Randen & Husøy (Randen & Husøy, 1999), and we have included it here for comparison purposes. Other choices are of course possible.

By choosing pairs of pixels, the set of possible pixel relationships is fundamentally two-dimensional. Alternatively, it is possible to compute distributions over larger patterns. For example, Wang & He (Wang & He, 1990) developed the notion of texture units, which are based on more pixels, but just three gray levels. They take the eight pixels that surround a center pixel, and then treat this is an 8-position base-3 number. This is an 8-dimensional co-occurrence, and they use the distribution of these patterns across the 6,561 entry co-occurrence matrix as a signature probability distribution. Oja & Valkealahti (Oja & Valkealahti, 1996) also use higher-dimensional pixel patterns, coupled with a tree-structured self-organizing map that drives nearest-neighbor classification. Ojala et al (Ojala, Valkealahti, Oja & Pietikäinen, 2001) also use high-dimensional patterns, subtracting the mean gray level from all gray levels in the pattern. Each of these methods for evaluating patterns produces its own feature set. A comprehensive discussion of approaches to texture classification, such as these, has been reported recently by Randen& Husøy (Randen & Husøy, 1999).

3 Classifier Induction Algorithms

For our experiments with the two feature sets defined above, we have selected four different classifier inducers, for a variety of reasons. The first is an artificial feed-forward neural network inducer, and the second is a decision tree inducer. Third is the naive Bayes classifier, which is commonly chosen as a baseline for comparisons. Fourth is the k-nearest-neighbor algorithm. Our purpose in choosing several classifier induction algorithms is to provide additional perspective on the merits of the two feature sets, not to draw any particular conclusions about the relative performances of the algorithms.

3.1 Artificial Neural Network (ANN)

An artificial neural network is a parameterized functional expression that maps a real-valued input vector to a real-valued output vector. The input features are either already numeric, or a numerical encoding of non-numeric values. For classification, there are as many output units as there are classes. For a given input, whichever output unit has the largest value is taken as the indication of the class to which the input has been mapped.

The most common form of parameterized expression is of the layered feed-forward variety. For example, each unit of a hidden layer of computing units generates an output value based on its inputs. These outputs constitute a new vector, which can be mapped by a subsequent layer of units. Typically, there is one layer of hidden units, but two layers of hidden units are sometimes employed. Each unit computes a linear combination of its inputs, which is then passed through a sigmoid function, yielding a value between 0 and 1. The backpropagation algorithm (Rumelhart & McClelland, 1986) adjusts all the weights (parameters) of the functional expression by descending the gradient of the error surface over the dimensions of the weight-space.

We chose Riedmiller & Braun's (Riedmiller & Braun, 1993) variation of this approach called RPROP (resilient backpropagation) because it often provides faster convergence than the standard back propagation algorithm. In contrast to other gradient-descent algorithms, RPROP does not use the magnitudes of the partial derivatives, but instead only the sign of each partial. For each weight, RPROP begins with an initial small weight update-magnitude, which is added to or subtracted from the weight to adjust it. To update the weights, RPROP multiplies each update-magnitude by a factor 1.2 (increasing it) for just those components whose current partial has the same sign as the previous partial. To prevent overflow, no update-magnitude is allowed to exceed 50.

Similarly, RPROP multiplies each update-magnitude by a factor of 0.5 (decreasing it) for just those components whose current partial has the opposite sign from the previous partial. However, when the partial derivative changes sign (overshoots), the weight first reverts to its previous value before adjusting it. The update-magnitude is added to the weight if the gradient is negative, and subtracted from the weight if the gradient is positive.

An important design choice for this algorithm is the network architecture, which for us means how many layers of hidden weights to use, and how many units to include in each layer. Scientists have come to recognize that determining the network architecture is an important part of the data fitting process. Accordingly, one must search for the number of hidden units for each layer, and search for the best connection weights, in a manner that is independent of the final assessment of classification accuracy. We follow the well-accepted procedure of using cross-validation (Breiman, Friedman, Olshen & Stone, 1984; Mitchell, 1997) to determine the number of hidden units to use. A standard procedure for avoiding overfitting is also described below. These steps may seem belabored, but they are necessary in order to avoid optimistic (unbiased) accuracy measurements.

We always use a single layer of hidden weights, and select the number of hidden units for that layer based on cross-validation using training and test sets drawn independently from those used in the subsequent experiments. Recall that cross-validation repeatedly uses a portion of the data for fitting purposes, and the rest for testing purposes. The measures obtained are averaged to produce the cross-validated measure. Details of the procedure are described below in Section 5. The number of hidden units that produces the best cross-validation result is the number that is used thereafter for all classification induction experiments for that given task. It is a necessary step that sets an important parameter (number of hidden units) without knowing the specific training and testing sets that will be used in subsequent experiments. This procedure reduces experimental bias (Steel & Torrie, 1980; Mitchell, 1997).

It is also essential to stop the weight adjustment process before the network comes to overfit the training data. We do this in a standard way, by putting aside a portion of the training data, called the *validation set*. Training proceeds on the remaining training data, and stops when the change in the error rate, as measured repeatedly on the validation set, indicates that overfitting has commenced. This is a well-known phenomenon of overtraining; the artificial neural network comes to conform too closely to the data on which it is basing its fit. More specifically, when this validation error measure rises for a specified number of training epochs, we halt the search for the best connection weights. Those weights that gave the minimum error on the validation set are restored. It is only at this point, that the training phase is complete. Thereafter, one can measure error on the testing set (different from the training and validation sets). It is essential that all fitting steps be accomplished without any reference to the testing data, and this procedure does that.

These two steps, determining the number of hidden units and determining weights that do not overfit the training data, are standard, and are done in a way that minimizes experimental bias. They are essential to inducing a classifier, and they operate without accessing the testing data. One must always first induce the classifier without any knowledge of the testing data, and only then measure its accuracy on the independent testing data.

3.2 Classification Tree (DMTI)

To build a classification (decision) tree, we used the DMTI classification tree inducer (Utgoff, Berkman & Clouse, 1997), which is a variant of the ITI algorithm. DMTI (direct metric tree inducer) is a classification tree inducer that conducts a more extensive search of tree space than the classical top-down approach (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1986; Quinlan, 1993). The algorithm installs different possible univariate tests at the root, restructures the subtrees, and then measures the quality of the resulting tree. The best test found in this manner is then reinstalled at the root, and the process is applied recursively to the subtrees. To reduce the effects of overfitting, the algorithm can post-prune a tree using the minimum description length (MDL) principle (Rissanen & Langdon, 1979; Quinlan & Rivest, 1989). This approach

to classification tree induction typically produces smaller and significantly more accurate trees than those that evaluate possible tests based solely on statistics that are computable from the distribution of instances at a leaf node.

There are no parameters of the algorithm to determine ahead of time. For this reason, none of the mechanisms that we employed for RPROP are needed. There is no need to determine a number of hidden units, and the DMTI algorithm has its own tree-pruning method to avoid overfitting of the data.

3.3 Naive Bayes (NB)

The Naive-Bayes classifier (Duda & Hart, 1973) operates by computing the probability of each class, given the instance to be classified, and selecting the most probable class as the answer (Mitchell, 1997). By assuming conditional independence among all the features, given a class, the computation of the posterior probability of a class, given the instance, is reduced to the task of computing the product of each of the independent probabilities of a class given the feature value. These independent probabilities can be tabulated for a discrete distribution simply by measuring each frequency of occurrence in the training data. Although this assumption of conditional independence is likely to be wrong, practice has shown that Naive Bayes often produces good results nevertheless (Domingos & Pazzani, 1997).

When the features are real-valued, as is the case for the triangular probability distribution over pixel pairs, one must take extra steps to produce a probability density function for each real-valued variable, assuming conditional independence as in the discrete case above. To approximate each independent density function, one could resort to a variety of approaches based on forming discrete intervals of interest. We simply assume that for each feature, its class-conditional distribution is normal. For each feature, its class-conditional mean and variance can be calculated easily from the training data. Any feature with a variance of 0 is ignored.

3.4 K-Nearest Neighbor (KNN)

The k-nearest neighbor algorithm (Duda & Hart, 1973) has a straightforward implementation. To train, simply store all of the training instances. When one must classify an unlabeled instance, search the set of saved training instances for the k of them that are closest in distance to the unlabeled instance to be classified, and respond with the most frequently occurring class of those k instances. Of course, one must choose a k, and we did this automatically by considering each value of k from 1 to 10. For each value of k, we did a 10-fold cross-validation to obtain an estimate of classification accuracy for that k. The best performing k was selected to be used for that feature set in subsequent experiments. This is the same procedure as described above for selecting the number of hidden units for an artificial neural network, but instead, we selected k.

A second standard processing step of normalization is needed here for the *csf* features, in which the units of the various features differ. A feature should not be treated as more important for classification because its feature values are of higher magnitude (different units of measure). To prevent this bias, each variable is mapped to standard normal form. This is done by subtracting the mean value of that variable and dividing by the variance of that variable. This normalization is not needed for the *tdf* features because all the units of measure are identical. An alternative normalization method would be to use Mahalanobis distance (Duda & Hart, 1973).

4 Comparisons for Forest Classification by Texture

This section and the next describe two sets of experiments in which the two feature sets are compared. The experiments of this section are based on an environmental project in which it is necessary to determine which of six classes an area of forest represents. The Nature Conservancy and the Winrock Corporation wish to be able to assess biomass from tree type and density, as estimated from aerial views of vast tracts of forest. These variables are critical for making choices with respect to purchase and preservation. Such functions are extraordinarily expensive when trained environmentalists or foresters must make these estimates manually.

The six (6) forest types are illustrated in Figure 1, where each square image is 160x160 pixels, and

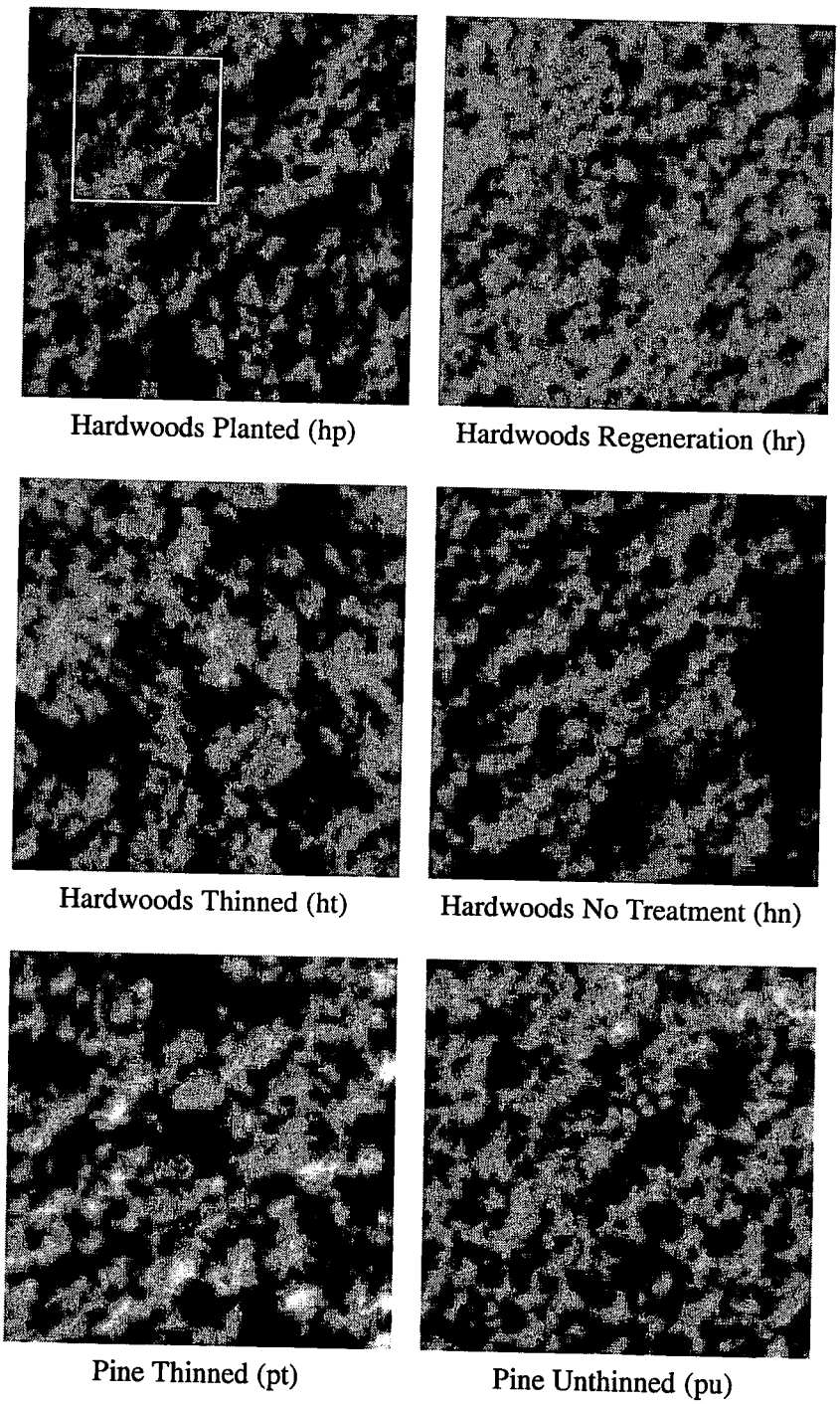

Figure 1. The Six Forest Types

the ground sample distance (gsd) is 0.4 meters, which means that each pixel in the image corresponds to a 0.4x0.4 meter area of the ground. The types vary by hardwood versus conifer, and by different stages of regrowth or harvesting that are critical to forest management. The six-class problem considered here is quite challenging. Trained foresters are able to discriminate these six classes, so we know that there is a sufficient basis for classifying them.

We performed a perfectly crossed experiment by measuring the classification accuracy of the classifier built for each combination of feature set and classifier induction algorithm. The two feature sets and four

Table 1. Class Distribution of Instances

Forest Type	Number of Instances
Hardwoods Planted (hp)	268
Hardwoods Regeneration (hr)	69
Hardwoods Thinned (ht)	49
Hardwoods No Treatment (hn)	246
Pine Thinned (pt)	62
Pine Unthinned (pu)	88

algorithms are described above in the previous two sections.

To obtain training and testing data, we started with sixteen polygons of ground truth defined by an expert forester. Each polygon circumscribes a forest region that is purely of one forest type. Although the images are in color, we converted each image from RGB to 256 gray levels by averaging the three RGB values at each pixel. Other non-uniform mixtures of these values are possible. Color information can also be informative with respect to classification, but our objective here is to study gray-level texture features. For many vegetation classification tasks in which color and shape provide little information, as is the case here, the most discriminating information is in the monochrome texture.

Each monochrome image, of as many as 256 gray levels, is then converted to an image of just nine (9) gray levels using MatLab's *hist_eq* histogram equalization operator. Equalization tends to accentuate detail and reduce effects of variation in brightness. We discuss issues related to equalization below in Section 6.2. Nine gray levels results in a triangular distribution of 45 elements, sampled well enough to capture the signature of the texture present in the region. Each image was then broken into a set of 60x60 non-overlapping pixel regions, each called an *instance*. Figure 1 includes a white outline that overlays the image to indicate the size of a 60x60 pixel region. Extracting as many instances (regions) as possible in this manner yields 782 instances, distributed across the classes as shown in Figure 1. The class label for each instance is the class label for the image polygon from which it was extracted. These 782 instances are computed just once, to provide a database for all the experiments.

Given that we have chosen a 60x60 region size ($k = 60$) and nine gray levels ($g = 9$), one can compute the average number of gray-level pixel pairs per cell (s), by solving $s \cdot g(g+1)/2 \leq 2(k-1)(2k-1)$ for s, giving $s \geq 312$. This could be considered generous. Note however that for $g = 11$, we have $s \geq 212$, so we have just a relatively small number of gray levels that could be used. Alternatively, one could lower k somewhat, but we wanted to ensure that the region was large enough for the texture to be perceivable, and for it to be sampled well.

In classifier induction tasks, one needs to ensure that the training and testing instances are drawn from the same distribution of possible instances, and that the instances are independent of one another. We consider these 782 instances to be independent because they do not overlap. One might be concerned about proximity of instances. Those that are close to each other may be deemed less independent than those more distant, but this leads to undue complication. We assume simply that these instances are representative of those that the resulting classifier will be called upon to classify.

We would like to know whether and how the choice of feature set affects accuracy of the induced classifier. To a lesser extent, we would also like to know whether the choice of the type of classifier inducer affects accuracy. We applied a two-way analysis-of-variance (ANOVA) (Freund, Livermore & Miller, 1962; Steel & Torrie, 1980) to test for significant effects. Consider the application of a feature set as one treatment, and the application of a classifier induction program as a second treatment. One can then measure the classification accuracy that results from every feature-set and classifier-inducer pair. Indeed one can obtain multiple measurements per pair by selecting different sets of training-testing data for each run. Given

Table 2. Mean accuracy for each treatment pair

	ANN	DMTI	KNN	NB
prim	94.1	90.4	88.5	85.3
sec_1	48.9	49.1	49.7	43.7

Table 3. Two-Way Analysis of Variance

	Sum Squares	Degrees	Mean Square	F	p
Features	34793.648	1	34793.648	1694.800	0.0000000000
Algorithms	531.064	3	177.021	8.623	0.0000704561
Interaction	105.782	3	35.261	1.718	0.2516741516
Replicates	258.311	9	28.701	1.398	0.1035288092
Error	1293.367	63	20.530		
Total	36982.171	79			

this perfectly crossed experimental setup, one can proceed to apply two-way analysis of variance to detect whether the treatment choices have any significant effect on classification accuracy.

There are two feature-sets from which to select, as described above in Section 5, and four possible induction algorithms from which to select, as described above in Section 3. Our main interest in this study is to compare feature sets, not to consider all possible means of obtaining the highest possible accuracy. Therefore, for simplicity here, pixel-pair orientation was ignored in all cases.

For each feature-set and classifier-inducer pair, we obtained ten accuracy measurements via stratified cross-validation. All of the 782 labeled instances were randomly partitioned into ten blocks, each with its class distribution as close as possible to that of the whole set. For each of ten runs, nine of the blocks collectively constituted the training instances, and the leftover block constituted the testing instances. The classifier was constructed based on the training instances, and finally its accuracy was measured on the testing instances. These ten runs provide the ten accuracy measurements, which were then averaged, for the particular pair of treatments. To eliminate variability due to instance partitioning, the same partition of the instances was used for all treatment pairs.

The mean accuracy for each pair of treatments is shown in Table 2. The comparatively high accuracies obtained when using the *tdf* feature set stands out for all four of the classifiers. In absolute terms, the accuracies obtained with *csf* feature set is in the vicinity of 50%, whereas with the *tdf* feature set the accuracies are in the vicinity of 90%. The highest accuracy of 94% is good enough to make the classifier a useful piece of automation in our environmental application domain.

The standard tableau for the two-way analysis-of-variance (ANOVA) is shown in Table 3. The analysis indicates that the choice of feature set has a highly significant effect on accuracy (low probability p of differences being due to chance), as does the choice of classifier induction algorithm. Effects due to interaction of feature set and algorithm are not significant. Finally, an insignificant amount of variation in accuracy can be attributed to obtaining ten measurements for each algorithm/feature-set pair.

Why does the *csf* feature set lead to dramatically lower classification accuracies? One explanation is that its features were designed for tasks quite different from forest-type classification. Another is that too much information is lost when using just these features.

As for the choice of classifier, one sees that when using the *tdf* features an artificial neural network outperforms a classification tree, which outperforms the k-nearest neighbor classifier, which outperforms the base-line Naive-Bayes classifier.

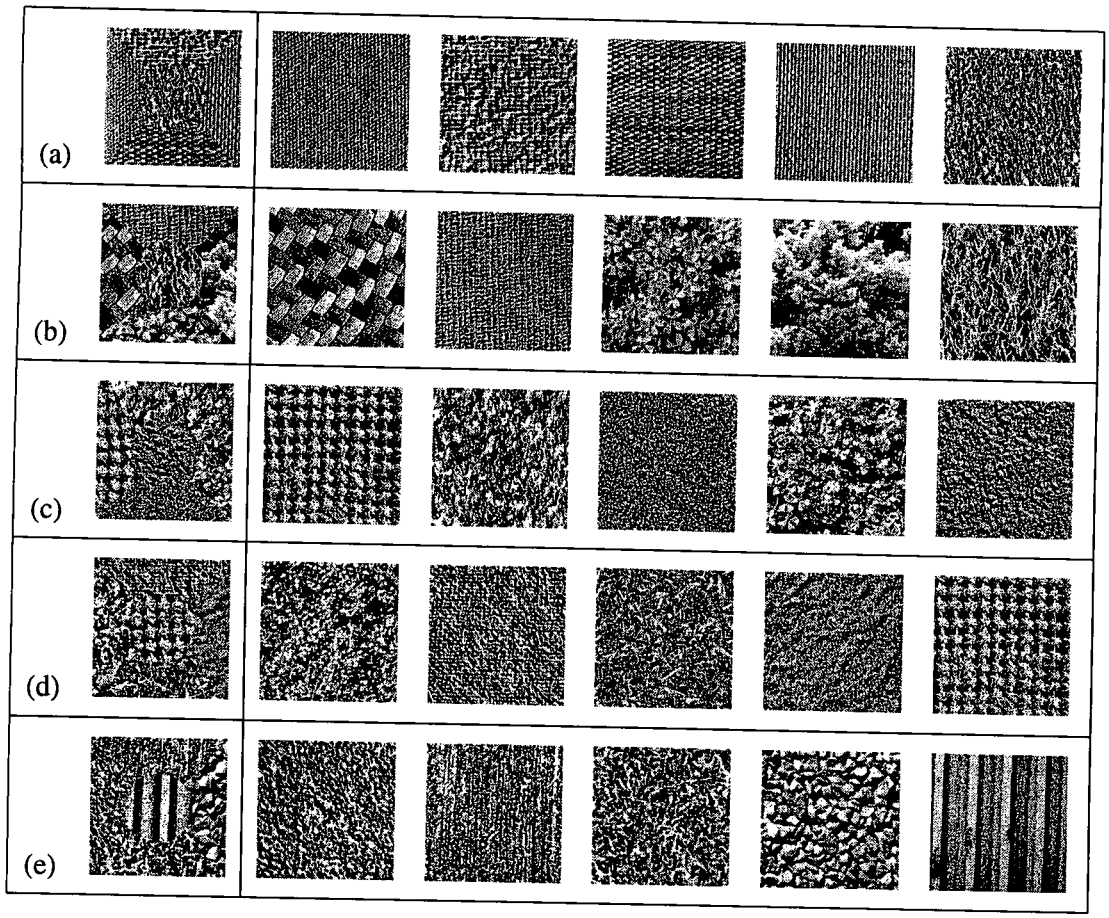

Figure 2. RH Experiments for Five (5) Textures

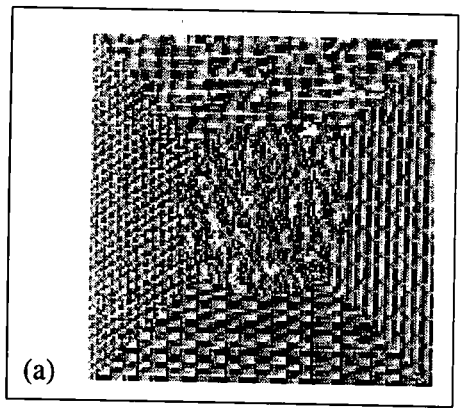

Figure 3. Enlargement of RH Five (5) Texture Test Image (a)

5 Comparisons for Segmentation of Synthetically Constructed Texture Regions

Having observed the high classification accuracies when using the *tdf* feature set, we were naturally curious about the generality of this observation. This led us to a data set of diverse textures that has been greatly studied, and which was used extensively in a recent study on texture classification by Randen & Husøy (Randen & Husøy, 1999). However, for this set of experiments, we used just the RPROP algorithm because it performed best in the forestry task. For brevity, we shall use 'RH' in place of 'Randen & Husøy'.

In a comprehensive study, RH compared a variety of feature sets across a large set of classification

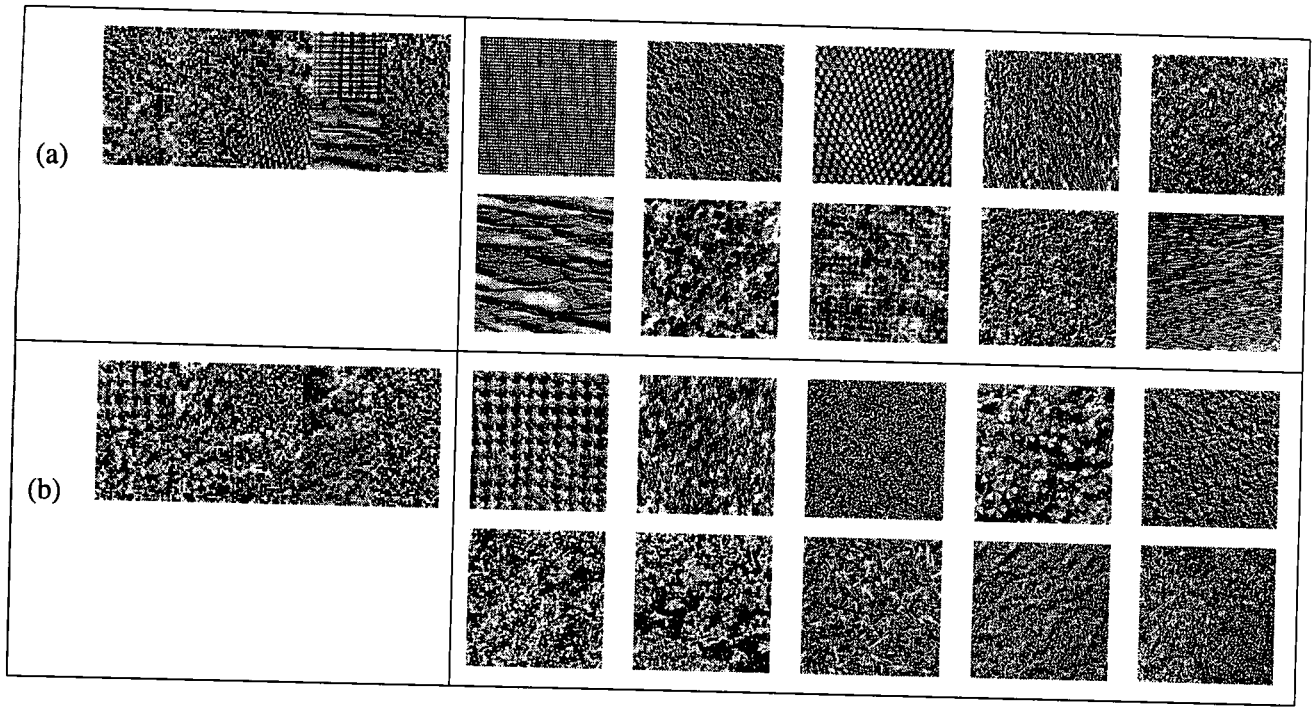

Figure 4. RH Experiments for Ten (10) Textures

tasks. These feature sets included filtering approaches, co-occurrence features, and autoregressive (model-based) features. Their experimental method was to train on example images, each purely of one texture, and then to test on a separate image that is a composite image formed from individual texture segments. For example, in Figure 2(a), the rightmost five images are the training images for the five individual textures, and the leftmost single image is the testing image. Each training image is 256x256 pixels, and in this case the testing image is also 256x256 pixels. The testing image is composed of segments of textures taken from images similar to the training images, and we show a larger version (same pixel resolution) of the testing image in Figure 3. One can see that although each training image is purely of one texture, the testing image composed of them is contrived by their creators. We have taken the images as they are, and note any processing steps explicitly.

Similarly, in Figure 6, the rightmost two images are the training images for the two individual textures, and the leftmost single image is the 256x512 pixel testing image. In Figure 4(a), the rightmost ten images are the training images for the ten individual textures, and the leftmost single image is the 256x640 pixel testing image. The testing image is composed of segments of textures taken from images similar to the training images. Figure 5 follows the same form, using sixteen textures with a 512x512 pixel testing image.

All the images for the synthetic data of the RH experiments are available on http://www.ux.his.no/ tranden. They are used here by permission of authors Randen & Husøy (Randen & Husøy, 1999), and IEEE. The basic images are publicly available as described by RH, and the composite images are available directly from them. The basic images come from the Brodatz Album (BA) (Brodatz, 1996), the MIT Vision Texture Database (MV) (MIT Vision and Modelling Group, 1998), and the MeasTex Image Texture Database (IT) (MeasTex Image Texture Database, 1998).

The correspondence of images in the figures of this article to image names in the databases is as follows: Figure 6(a), from (BA), images D4 and D84. Figure 6(b), from (BA), images D12 and D17. Figure 6(c), from (BA), images D5 and D92. Figure 2(a), from (BA), images D77, D84, D55, D53, and D24. Figure 2(b), from (MV), images Fabric.0000, Fabric.0017, Flowers.0002, Leaves.0006, and Leaves.0013. Figure 2(c), from (MV), images Fabric.0009, Fabric.0016, Fabric.0019, Flowers.0005, and Food.0005. Figure 2(d), from

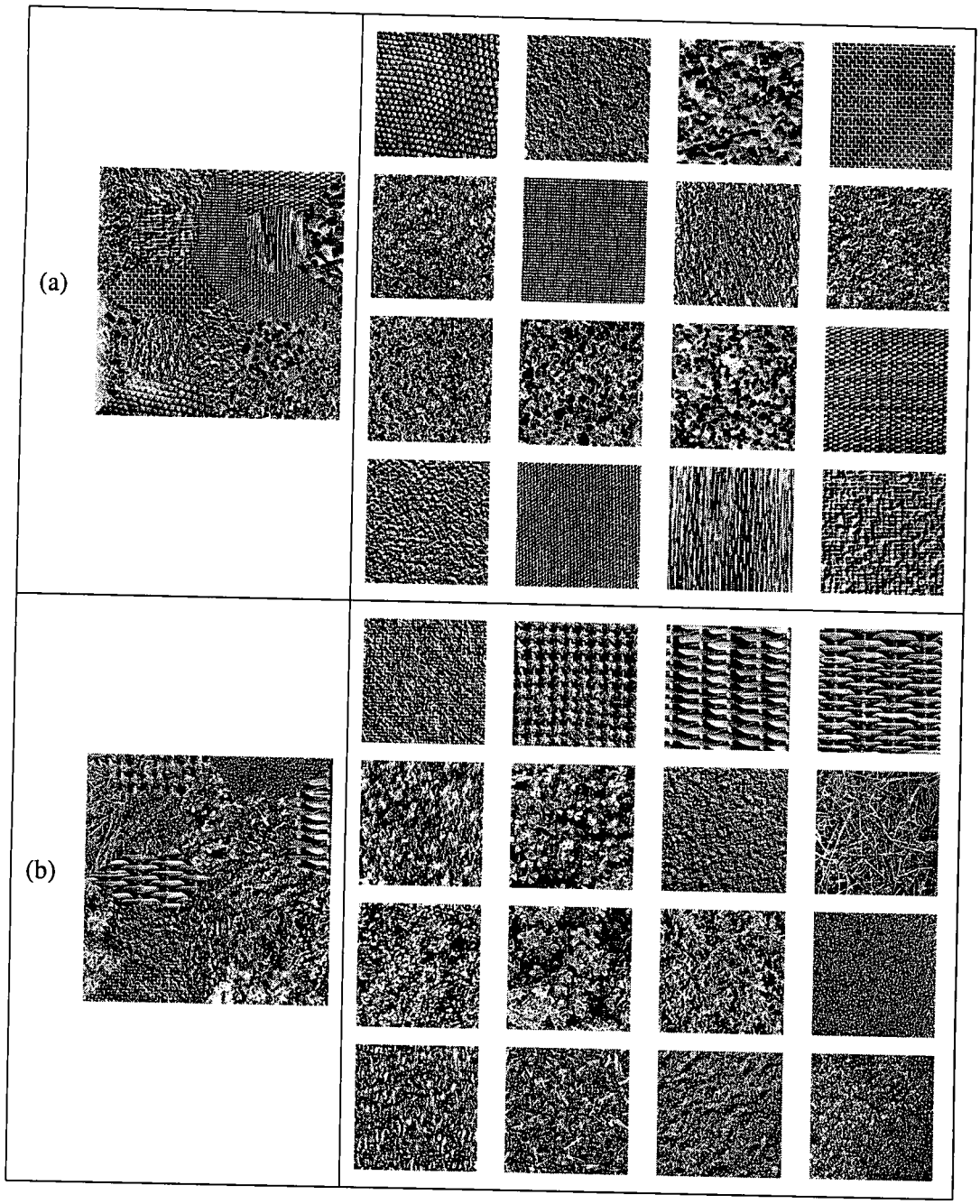

Figure 5. RH Experiments for Sixteen (16) Textures

(MV), images Fabric.0007, Fabric.0009, Leaves.0003, Misc.0002, and Sand.0000. Figure 2(e), from (IT), images Asphalt.0000, Concrete.0001, Grass.0002, Misc.0002, and Rock.0005. Figure 4(a), from (BA), images D4, D9, D19, D21, D24, D28, D29, D36, D37, and D38. Figure 4(b), from (MV), images Fabric.0009, Fabric.0016, Fabric.0019, Flowers.0005, Food.0005, Leaves.0003, Misc.0000, Misc.0002, Sand.0000, and Stone.0004. Figure 5(a), from (BA), images D3, D4, D5, D6, D9, D21, D24, D29, D32, D33, D54, D55, D57, D68, D77, and D84. Figure 5(b), from (MV), images Fabric.0009, Fabric.0013, Fabric.0014, Fabric.0016, Flowers.0005, Food.0005, Grass.0001, Leaves.0003, Leaves.0008, Leaves.0012, Metal.0000, Metal.0002, Misc.0002, Sand.0000, and Stone.0004.

The fundamental classification task is slightly different here from what it was in the forest classification

task described above. For forest classification, each region was classified as a single entity, but for the synthetic classification tasks of this section, each individual pixel is to be classified. Of course no single pixel can exhibit texture, so the texture label of a pixel must be determined by its context. Specifically, the 32x32 pixel region with that pixel closest to its center is classified. The classifier is used in this manner to compute a class label for each pixel in the test image, and the error rate is measured as the percentage of test pixels misclassified.

Note that pixels near texture boundaries are inherently difficult to classify because the enclosing region spans more than one texture. One can see that this task is difficult even for a human. Indeed, a human soon notices the geometric shape of the contrived pattern, here of a circular region inscribed into four triangular regions, biasing the task in favor of the human immediately. Even knowing this, some of the boundaries are difficult to locate exactly. One may expect to make many errors near the texture boundaries. The testing images composed of more textures often have more texture types and more boundaries within each 32x32 pixel region, making those tasks more error-prone.

From these images, one needs to extract regions for the purpose of preparing training and testing data. We have largely followed the choices of RH so that we can reasonably compare our accuracy measures to theirs. Differences are noted where they may apply. So, in addition to comparing performance when using the two feature sets we have discussed above, we wish also to compare to results published by RH for different approaches that they measured. This is an additional point of perspective, which is separate from the discussion of how we obtained our own measures for own algorithm and feature-set combinations.

For the data preparation step, we have used the exact same (as RH) training and test images, and the same (as RH) number of gray levels (eight). For eight gray levels, the triangular distribution has 36 elements, which means that there are 36 *tdf* features. The *csf* feature set is calculated from the triangular distribution as described above. Recall that the triangular distribution is computed by converting the counts in the co-occurrence matrix to probabilities. The *tdf* and *csf* features are based on this distribution. We have also adopted the same (as RH) local region size of 32x32 pixels. This is the set of pixels from which all adjacent pixel pairs are tallied to compute the triangular distribution. In a 32x32 region, there are 3,906 pairs of adjacent pixels, giving a large sample of pixel pairs for the 36 elements of the triangular distribution. Finally, we have employed the same (as RH) four-pixel sliding step for extracting training instances at spaced intervals. One starts with a 32x32 pixel region in the upper left corner of each training image, and then slides it four pixels at a time horizontally or vertically. Recall that each training image is purely of one texture, and therefore each placement of the 32x32 region extractor produces a training instance of the same texture (class). In the case of a training 256x256 image, this produces 3,249 training regions. Each 32x32 training region is labeled with the class of the entire training image.

There are several differences in our experimental method from that of RH, as discussed in detail further below. Briefly, we use a different classifier induction algorithm. Secondly, and importantly, we equalize each 32x32 pixel region (using MatLab's *hist_eq* operator), not each entire training image. Because the data images have already been equalized globally, this amounts to two equalization steps. However, local equalization of each region provides benefits locally because contrasts within a local region can be enhanced. Finally, we ignore orientation in the first set of comparisons, but include it subsequently in a follow-up set of comparisons.

For the RH tasks, we have produced error measurements when using the RPROP algorithm with each of the feature sets defined above. The network has as its inputs the feature values of the feature set used for the experiment. These would be either the *tdf* features or the *csf* features. The network has an output unit for each possible class (texture) for the task at hand.

We determine the number of hidden units anew in each experiment. To do this, we first sequester $\frac{1}{11}$ of the training data as the validation set, and then run a 10-fold cross-validation experiment within the remaining $\frac{10}{11}$ of the training data. This procedure was described above, but here we provide additional detail. Such a measurement is performed for each of the candidate number of hidden units, which ranges

Table 4: Classification Error Rates, RPROP, Adjacent Pixel Pairs, Ignoring Orientation, Local Equalization. Lowest error rates among RPROP columns shown in bold. Column '*RPROPcf*' is without orientation, but column '*RHcf*' is with orientation.

Fig	*RPROP*		*Randen and Husøy*		
	tdf	*csf*	Lowest Error & Method		*RHcf*
6(a)	8.30	32.96	0.7	($J_{MS,U,F}$)	1.9
6(b)	6.35	6.39	0.2	(f32d(a))	4.8
6(c)	3.44	3.62	2.5	(DCT)	3.3
2(a)	7.65	17.04	7.2	(f8a(d))	9.9
2(b)	17.31	22.22	18.9	(f16b(d))	27.0
2(c)	16.01	32.98	20.6	(F_2_1_smpl(d))	26.1
2(d)	30.16	38.46	16.8	(f32d(d))	51.1
2(e)	20.47	21.17	17.2	(f16b(d))	35.7
4(a)	49.77	62.06	32.3	(Dyadic Gabor bk)	35.3
4(b)	28.97	42.20	27.8	(F_2_1_smpl(d))	49.1
5(a)	45.24	62.01	34.7	(Pred. err. filt.)	49.6
5(b)	35.64	54.49	41.7	(f16b(d))	55.4

from three (3) to ten (10). The number of hidden units that produces the lowest cross-validated error on the training data is taken to be the best number of hidden units to use. Note that for simplicity, the single validation set is used to prevent overfitting in each of the folds, although one could select a new validation set each time. Upon determining the desired number of hidden units, the validation set is rejoined with the training set to reconstitute the original set of training data. Now the fitting phase commences, for which $\frac{1}{10}$ of the training data is set aside as the validation set. RPROP is run once on the remaining $\frac{9}{10}$ of the data, using the validation set to prevent overfitting. This is laborious, but is essential for finding a fit of the data without knowledge of the independent testing set.

When the weights have been tuned, the last step is to test the classifier on the test image. The class attributed to each pixel is the class that is attributed to the 32x32 region with that pixel at its center. For pixels so close to the image boundary that the 32x32 region would not fit entirely, a smaller region is used, effectively by clipping the 32x32 region to the area within the test image. This means that for such clipped regions, the triangular distribution is based on fewer total pixel pairs. Error was then tabulated as the percentage of pixels misclassified.

We present three experiments that show the accuracy achieved with the classifier built by RPROP and the two feature sets. In the first experiment, the triangular co-occurrence matrix is tallied from pairs of adjacent pixels without regard to orientation. In the second experiment, orientation is considered. This is done by calculating four triangular distributions, one for each of the 45 degree orientations. The third experiment is like the second, but equalization is applied once to the entire image, not individually to each pixel region that comprises an instance. Following these experiments, we include a simple illustration of how smoothing the classification image can improve accuracy further.

5.1 Experiment #1: Ignoring Orientation, Local Equalization

Table 4 shows the error rates for RPROP on the various tasks using each of the feature sets. For all twelve classification tasks, the *tdf* features (using RPROP) produced more accurate classifiers than the *csf* features. In three of the twelve cases, the *tdf* features produced more accurate classifiers than the best of all of those evaluated by RH. The f16b(d) corresponds to a bank of quadrature mirror filters (QMF), including

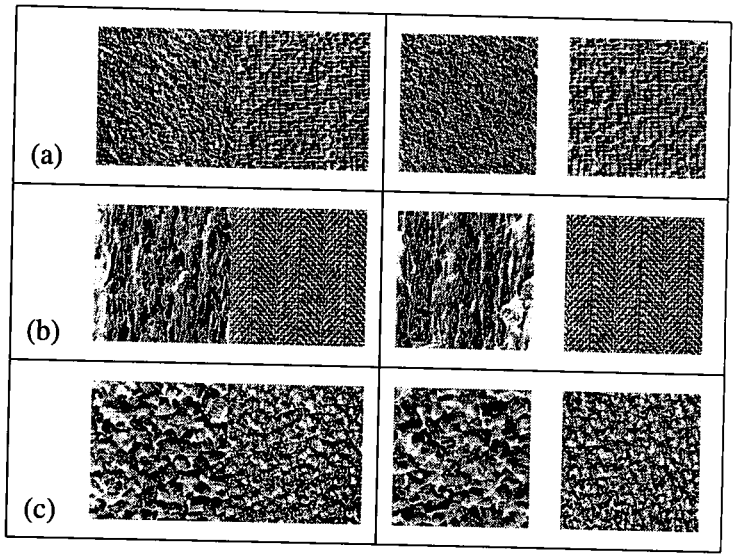

Figure 6. RH Experiments for Two (2) Textures

both infinite impulse response (IIR) and finite impulse (FIR) filters. In particular, "f16b(d)" indicates a 16-tap FIR filter with particular subband decomposition. The F_2_1_smpl(d) is also a form of QMF filter. The J_{MS}, J_U, and J_F are each forms of a Gabor filter.

The '*RHcf*' column shows the error rate reported by RH when using the feature set *csf*, but taking into account the four possible directional relationships for pixel pairs. This can be done by computing four versions of the *csf* features and then using all four sets of features collectively as though one feature set. The RH experiments use only pairs of adjacent pixels, as do we. For the RPROP results, we have ignored orientation, so the *csf* column is not directly comparable. Furthermore, the classification induction algorithm differs too; we have used RPROP, and the RH results are for the Type-One-Learning-Vector-Quantizing (LVQ) supervised classifier (Kohonen, 1990). As mentioned above, we also apply equalization more finely, to each 32x32 region, not just once to the entire image from which the instances were extracted.

Comparing the *csf* column to the *RHcf* column, one sees that our *csf* has a lower error rate in only five of the twelve cases. Some of the accuracy differences are large. For example, for the task shown in Figure 6(a), RPROP with feature set *csf* has a dramatically worse error rate of 32.96% than does '*RHcf* with error rate 1.9%. Inspection of the textures in the figure does suggest strongly that one of the textures has striations at 45 while degrees while the other has striations predominantly in the vertical and horizontal orientations. These results strongly imply that the orientation information is helpful for classification. To this end we reran our suite, this time accounting for orientation.

5.2 Experiment #2: Four Orientations, Local Equalization

When using four co-occurrence matrices, one for each orientation, the total number of entries in the four triangular distributions is $4 \cdot 36 = 144$. In this setting, an excessive amount of time is needed to run the 10-cross validation to determine the number of hidden units. We observed that the best results are always achieved for six, eight or ten hidden units, and that the best among these is only slightly better than the others. Moreover, the network with the best number of hidden units for training instances did not always give the best result on the test image. Taking these conditions into account, we elected simply to estimate the number of hidden units to use. No other optimization of the number of hidden units was undertaken. The number of hidden units used for the 2-texture, 5-texture, 10-texture and 16-texture tasks were 6, 8, 6 and 8 respectively. These values were chosen almost arbitrarily (they are considered to be lucky numbers in China).

Table 5: Classification Error Rates, RPROP, Adjacent Pixel Pairs, Orientation, Local Equalization. Lowest error rates among RPROP columns shown in bold.

Fig	*RPROP*		*Randen and Husøy*		
	tdf	*csf*	Lowest Error & Method		*RHcf*
6(a)	1.59	0.73	0.7	($J_{MS,U,F}$)	1.9
6(b)	2.03	2.00	0.2	(f32d(a))	4.8
6(c)	3.19	4.29	2.5	(DCT)	3.3
2(a)	5.07	10.30	7.2	(f8a(d))	9.9
2(b)	12.67	17.21	18.9	(f16b(d))	27.0
2(c)	11.38	16.82	20.6	(F_2_1_smpl(d))	26.1
2(d)	18.49	19.63	16.8	(f32d(d))	51.1
2(e)	19.01	12.89	17.2	(f16b(d))	35.7
4(a)	25.09	27.38	32.3	(Dyadic Gabor filt. bk)	35.3
4(b)	19.37	28.28	27.8	(F_2_1_smpl(d))	49.1
5(a)	33.45	37.98	34.7	(Pred. err. filt.)	49.6
5(b)	27.57	37.73	41.7	(f16b(d))	55.4

Table 6. Classification Error Rates, RPROP, Adjacent Pixel Pairs, Orientation, Global Equalization

Fig	*RPROP*		*Randen and Husøy*		
	tdf	*csf*	Lowest Error & Method		*RHcf*
6(a)	5.04	2.06	0.7	($J_{MS,U,F}$)	1.9
6(b)	3.53	5.23	0.2	(f32d(a))	4.8
6(c)	1.08	1.35	2.5	(DCT)	3.3
2(a)	11.03	18.96	7.2	(f8a(d))	9.9
2(b)	30.26	17.98	18.9	(f16b(d))	27.0
2(c)	10.75	20.77	20.6	(F_2_1_smpl(d))	26.1
2(d)	31.06	38.88	16.8	(f32d(d))	51.1
2(e)	30.74	18.14	17.2	(f16b(d))	35.7
4(a)	33.55	35.81	32.3	(Dyadic Gabor filt. bk)	35.3
4(b)	37.08	37.48	27.8	(F_2_1_smpl(d))	49.1
5(a)	48.68	49.48	34.7	(Pred. err. filt.)	49.6
5(b)	28.54	40.05	41.7	(f16b(d))	55.4

Table 5 shows the error measurements when accounting for orientation. Notice that the error rates for the *tdf* features are uniformly lower when accounting for orientation (compare the *tdf* column of Figure 4 to the *tdf* column of Figure 5. Similarly, error rates for the *csf* features are all lower with orientation than without, except in one case. Including the RH results, the *tdf* features now produce the best error rate in seven of the twelve problems (up from three of twelve when ignoring orientation). We conclude that accounting for orientation of the pixels is worthwhile. Comparing the orientation-sensitive *csf* measurements to those of the '*RHcf*' column, one sees that *csf* has lower error in ten of twelve cases. RPROP using the *tdf* feature set produces lower error than *RHcf* in all cases.

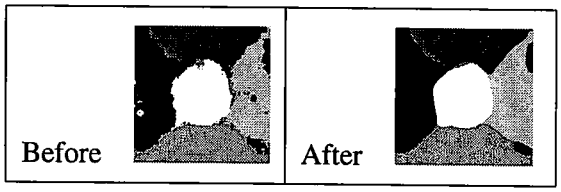

Figure 7. Classification Image for Testing Image of Figure 2(c)

5.3 Experiment #3: Four Orientations, Global Equalization

We have been eliminating differences between our *csf* setup and that of $RHcf$. The only remaining differences are the classifier induction algorithm itself, and the equalization method. Table 6 shows the measurements that one obtains when applying equalization to an entire image at a time, instead of a 32x32 subimage. Although the images have already been equalized in the original data to 256 gray levels, we equalized again nevertheless in order to obtain just eight (8) gray levels. Looking at the results in the table, one can see that classification accuracy suffers in ten cases and improves in two cases. This strongly indicates the importance to the scope of application of equalization. We return to this point below in Section 6.2.

When accounting for orientation and applying equalization finely (locally), one can observe in the previous Table 5 that the *tdf* feature set produces lower error than our *csf* feature set in nine of twelve cases, under identical conditions. The *tdf* feature set also produces lower error than the best of the RH results on seven of the twelve tasks. This leads to our main conclusion, that one should consider using the *tdf* feature set when classifying images by texture.

5.4 Classification Image Smoothing: Ignoring Orientation, Local Equalization

One can lower the error rates further by applying a smoothing operator, which is appropriate under the assumption that texture areas are generally contiguous and coherent. It is informative to redisplay an image according to the texture class indicated by the classifier for each pixel. For each pixel's indicated class, map it to a numeric value associated uniquely with that class, forming a new image that depicts texture class. Finally, apply the 25x25 median filter to the synthetic image, thereby potentially revising the inferred texture class of each pixel.

Figure 7 shows a class-label image for five textures based on the texture class before and after smoothing. Smoothing is a separate post-processing step that can be applied independently of the classifier that was used. This was not done in the above experiments because it would obscure the performance of the classifiers. In Table 7 we show results after applying a 25x25 median filter to the texture class image. We mention this merely to be encouraging about error rates that can be achieved with additional processing. Notice that classification error drops in all tasks for both feature sets. Note that these error rates should not be compared to the others that we have discussed elsewhere because smoothing was not done for them.

6 Discussion

One should consider the *tdf* features of the triangular distribution when building a classifier for a task in which monochrome texture features are an important basis for discrimination. One should also consider orientation because it may provide useful information, even though this was not done for the forest classification task.

6.1 Inspecting the Classifiers

It is sometimes informative to try to examine an induced classifier. One would like to know how the classifier determines a class label. In this section, we discuss one of the artificial neural network classifiers built by RPROP for the forest classification task, and a classification tree built by DMTI using the exact same training data.

Table 7: Classification Error Rates After 25x25 Median Filter, Ignoring Orientation versus Heeding Orientation

Fig	Ignoring		Heeding	
	tdf	*csf*	*tdf*	*csf*
2(a)	5.35	11.31	4.20	7.48
2(b)	12.05	13.46	11.26	14.77
2(c)	11.03	27.58	10.28	9.61
2(d)	26.97	27.13	14.58	14.07
2(e)	11.77	14.30	15.12	9.05
4(a)	44.85	55.88	19.86	19.98
4(b)	20.82	34.62	13.47	23.27
5(a)	39.58	54.18	28.56	33.14
5(b)	29.20	44.42	22.11	30.17
6(a)	5.14	35.51	0.78	0.55
6(b)	3.82	3.63	1.86	1.79
6(c)	1.83	1.66	1.43	1.76

Figure 8 shows the hidden layer of eight units for the artificial neural network induced by RPROP. This is a Hinton diagram, in which the value of a weight is displayed in white if it is negative and black if it is positive. The magnitude of a weight is indicated by the size of the box. For example, hidden unit 6 has a large positive weight for input unit P07. The hidden unit numbers are shown along the discrete horizontal axis, and the input unit numbers are shown along the discrete vertical axis. Each input unit number is shown as a base 9 value, making it easy to determine the cell of the co-occurrence matrix P from which it came. For example, input unit P07 corresponds to the probability of gray level 0 and gray level 7 co-occurring in two adjacent pixels.

As one would expect, it is difficult to make perfect sense of the weights. The learning task is to associate characteristics of the input feature representation with the correct target class labels. In the diagram, one can see generally that the high-contrast features have larger weights, and are presumably more informative, as is often the case in for visual perception by human or machine. Input units such as P08, P07, P06, P18, P17, and P28 have the largest magnitudes, while input units such as P00, P11, P88, P01, P12, and P78 have the smallest magnitudes. It is difficult to attach meaning to the function that each hidden unit computes. A hidden unit represents a kind of secondary feature, one that is learned by the network, and that is typically distinctive from the other hidden units. For example, Unit 7 responds to high values of P06 and P07, but with strong inhibition from P08. Evidently this is a useful feature that has been induced by RPROP, on top of the *tdf* input features. Combinations of these features are used by the output layer to discriminate the various forest types.

Figure 9 shows a classification tree induced by DMTI for the same training fold as above for RPROP. This is a pruned tree, so that it is smaller and more readable here, but as mentioned above in Section 3.2, the unpruned version was used for the experiment. Each test node compares the entry in the indicated cell of the co-occurrence matrix P to a specific constant threshold. This vocabulary of possible univariate tests constrains the decision boundaries to be axis-parallel or axis-orthogonal, excluding oblique decision boundaries that are available in an artificial neural network. One branches left if the cell value is less than the threshold, and right otherwise. This process repeats until a leaf node is reached. Each leaf is shown as a vertical list of class labels. To the right of each class label, in parentheses, is the number of training instances of that class that arrived at that leaf. The class labels are sorted from most frequent to least frequent. For

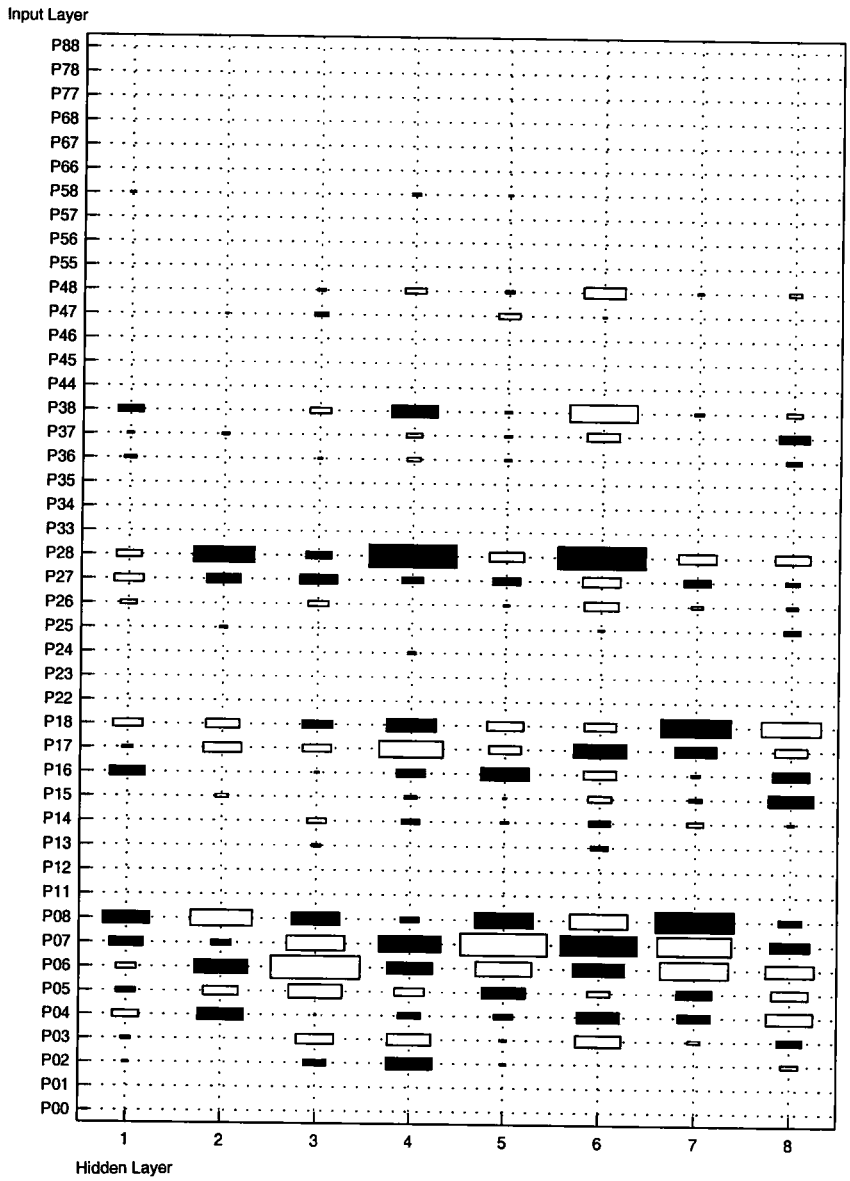

Figure 8. Hinton Graph of RPROP Weights for *tdf* Features

classification purposes, the most frequent class at the proper leaf is the answer returned. Ties are broken consistently but otherwise arbitrarily. For the unpruned version every leaf has instances of just one class.

Interpreting the tree is somewhat difficult, but the principle is the same as for RPROP. Various combinations of input feature values are predictive of the forest-type label, and the classification tree inducer identifies them and uses them when forming the classification tree. The tree shown in the figure does not indicate the same utility for high-contrast features as the neural net. Because the tree is a recursive structure, and is built recursively, it tends to separate one *group* of classes from another. For example, notice that all 44 Hardwood Thinned (ht) training instances are in the left subtree of the root. The subtree that one reaches by branching right then left contains 53 of the 56 Pine Thinned (pt) training instances. The Hardwoods Planted (hp) class has training instances scattered throughout the tree. These totals are for the instances used in training, which are a subset of the total available instances.

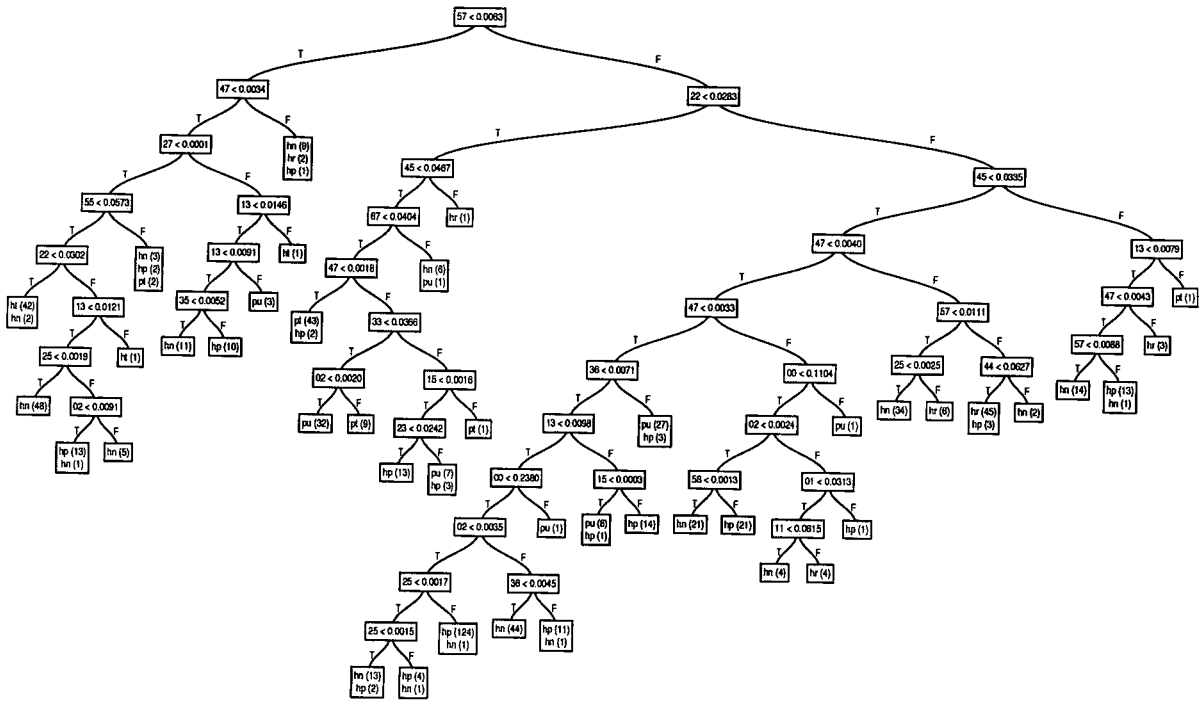

Figure 9. DMTI Tree for *tdf*

6.2 Image Equalization

An ever-present issue is the size of the region to which histogram equalization should be applied. The purpose of this operation is to reduce the variability due to lighting conditions, and to enhance contrast. How large a region should one encompass? For the forest classification task, we tried four different region sizes for equalization. First, we applied the operator once to the entire image that contained all 16 polygons. Second, we applied the operator instead to each of the 16 polygons individually. Third, we tried applying the operator to an 80x80 region around each 60x60 region (an extra 10 pixels beyond each border of the 60x60 region). Fourth, we applied the operator to each 60x60 region.

For the forest classification task, the best results came from applying the operator to each polygon. Thus, to use an induced classifier in practice, one would need to have the expert user circumscribe a polygon of homogeneous forest type (as we did), and then have the classifier proceed to equalize the polygonal region, and then automatically break it into 60x60 regions to be classified individually. One would ascribe to the polygon the class returned most frequently by the classifier. This is a workable approach, but it would be preferable to be able to turn the classifier loose on an entire aerial image data set. It may be possible to do this by eliminating equalization, and training the classifier on instances from a large variety of lighting conditions.

For the RH experiments, they equalized each image globally and then extracted subregions as needed. We found that applying histogram equalization to each extracted subregion individually improves overall accuracy noticeably.

6.3 Summary

The main lesson is that the *tdf* features, consisting of the triangular distribution based on co-occurrence frequencies, compare favorably to a variety of commonly used state-of-the-art feature sets. This was observed both for a forest-type texture classification task and for a set of well known synthetic composite texture classification tasks. It is quite possible that higher-dimensional co-occurrence frequencies, such as those based on texture units, would also work well. Allowing RPROP to find useful patterns over a co-

occurrence distribution appears to be worthwhile, and bears more study. All of these texture classification tasks are extremely challenging, even for humans, due to subtle differences in forest classes and boundary conditions for the composite synthetic texture images. These experiments suggest that the *tdf* feature set may be of value for many other texture classification problems.

A second observation is that the scope of application of histogram equalization can be critical to the quality of the classifier induced. We found that application of equalization to the pixel region of an instance gave the best results.

Acknowledgments

This work was supported by National Science Foundation Grants EIA-9726401 and IRI-9711239. We are grateful to Trygve Randen for answering several technical questions about his experimental method. Permission to include images for the RH experiments was kindly granted by T. Randen and IEEE. The forest-class data was collected over Crooksville Ohio, in collaboration with the Winrock Company, the Nature Conservancy, and American Electric Power. We thank Howard Schultz for his ongoing help in refining the aerial instrumentation for the aerial data collection. Richard Cochran and David Stracuzzi provided helpful comments.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

Brodatz, P. (1996). *Textures: A photographic album for artists and designers*, New York: Dover.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning, 29*, 103-130.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: Wiley & Sons.

Freund, J. E., Livermore, P. E., & Miller, I. (1962). *Manual of experimental statistics.* Prentice-Hall.

Haralick, R. M., Shanmugam, K., & Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-3*, 610-621.

Kohonen, T. (1990). The self-organizing map. *Proc. IEEE, 78*, 1464-1480.

MeasTex Image Texture Database (1998). *Http://www.cssip.elec.uq.edu.au/guy/meastex/meastex.html*, CSSIP.

Mitchell, T., M. (1997). *Machine learning.* McGraw-Hill.

MIT Vision and Modelling Group (1998). *Http://www.media.mit.edu/vismod/*, MIT.

Oja, E., & Valkealahti, K. (1996). Co-occurrence map: Quantizing multidimensional texture histograms. *Pattern Recognition Letters, 17*, 723-730.

Ojala, T., Valkealahti, K., Oja, E., & Pietikäinen, M. (2001). Texture discrimination with multidimensional distributions of signed gray-level difference. *Pattern Recognition, 34*, 727-739.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227-248.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Randen, T., & Husøy, H. (1999). Filtering for texture classification: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 21*, 291-310 .

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of IEEE International Conference on Neural Networks* (pp. 586-591).

Rissanen, J., & Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of Research and Development, 23*, 149-162.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Steel, R. G. D., & Torrie, J. H. (1980). *Principles and procedures of statistics, 2nd edition*. New York, NY: McGraw-Hill.

Utgoff, P. E., Berkman, N. C., & Clouse, J. A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning, 29*, 5-44.

Wang, L., & He, D. C. (1990). Texture classification using texture spectrum. *Pattern Recognition, 23*, 905-910.

Weszka, J. S., Dyer, C. R., & Rosenfeld, A. (1976). A comparative study of texture measures for terrain classification. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-6*, 269-285.