1994

An MS Windows prototype for automatic general purpose image-based flaw detection

Jorn Lyseggen
Iowa State University

An MS Windows prototype system for automatic general purpose image-based flaw detection

by

Jorn Lyseggen

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical Engineering and Computer Engineering
Major: Electrical Engineering

Approved:

Signature redacted for privacy

In Charge of Major Work

Signature redacted for privacy

For the Major Department

Signature redacted for privacy

For the Graduate College

Iowa State University
Ames, Iowa

1994

To Mi-Young Song,
for always being there

TABLE OF CONTENTS

1. FLAW DETECTION

1.1. Introduction

Flaw detection plays a crucial role in many industries to make sure that the products meet the specified quality requirements. When making for example a car it is important that all the parts satisfy certain quality standards to make sure the consumer buys a car that is safe to operate. A crack or another weakness in a crucial part can be catastrophic. To make sure their cars are as safe as possible, car manufacturers are conducting thorough testing of crucial parts. Similar tests are done in a wide variety of industries, and these quality controls are often referred to as *flaw detection.* Any cracks, voids, or other weaknesses that can cause danger are called *flaws.*

Flaw detection is often done, or preferred done, in real time -- in an assembly line fashion. An important constraint, in addition to reliability, is therefore speed.

The techniques used in these tests varies. Common techniques are ultrasonic waves (1-D or 2-D), eddy current imaging, x-ray imaging, thermal imaging, and fluorescent penetrent imaging.

In this thesis I will discuss *automatic general purpose image-based flaw detection.* "Automatic" means that the flaw detection is performed without human supervision, and "general purpose" means that the inspection is not tailored to a specific task (i.e. one particular flaw in one particular type of object), but is ideally applicable to any detection problem.

The thesis is organized in five chapters. Chapter 1 is discussing flaw detection in general and earlier work done in this area. Chapter 2 is an introductory tutorial to pattern recognition with emphasis on neural networks and fuzzy logic. Chapter 3 discusses implemented techniques in the developed prototype system Sherlock. Chapter 4 contains a user manual of Sherlock. Chapter 5 explains inspection strategies. Chapter 6 is reporting on classification results, and chapter 7 has a conclusion and suggestions for future work.

1.2. Review of Earlier Work by John P. Basart's Group

John P. Basart's group, in which I have been doing my work, has been working with image-based flawdetection for a number of years. The development of this group's work describes pretty much how the general trend in this area has developed.

This group's earliest work was focused on image enhancement techniques. The objective was to enhance the flaws by image processes and thereby ease detection by a human inspector. Some of the techniques used for this purpose were edge detection [Wong,1987], trend removal [Doering, 1987], maximum entropy deconvolution [Zheng,1987], correlated noise analyses [Zheng,1987], Adaptive Kalman filtering [Zheng,1987], and template matching [Gabot,1988].

Work was then done in the area of *automatic* flaw detection (see Fig 1.1). At first, automatic *task specific* routines was focused. The most important contribution here was a system developed by J. Xu for Martin Marietta which detected void-like flaws in welds in fuel tanks for the NASA Space Shuttle [Xu J. et al., 1989]. This system was based on a decision tree structure. Their approach was first to identify the weld region, and then to extract information like gradients, mean, and variance. This information was then fed into the tree structure that made the decision to whether there was a flaw, its location, and its size.

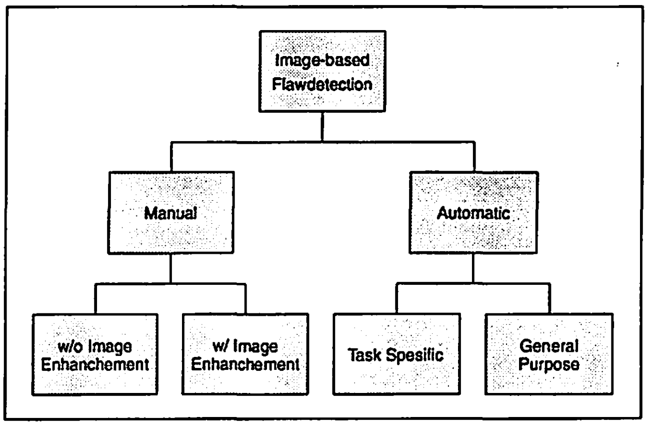


Fig 1.1

Categorization of image based flaw detection

Xu's system was reasonably successful. However, of the initial training images, there was a large overrepresentation of images with flaws, and when the system was tested with images without flaws, a problem of false alarms was encountered. This problem could probably have been solved by modifications of Xu's tree structure, but was never done because of lack of funding.

Other significant work in automatic flaw detection was done by K. W. Ulmer [1992], and E. M. Siwek [1994]. Their work were in the area of *general purpose* flaw detection. Ulmer and Siwek both tried to solve problems related to the geometry of the object being inspected. For example, when x-raying an object with a complicated geometric structure, the resulting image will often have large intensity variations due to the thickness variations of the object. These intensity variations can often complicate the inspection considerably. If intensity fluctuations due to the geometry can be successfully removed, flaws would ideally stand out from a uniform background, and should be easy to detect. Ulmer worked on a method that modeled the surface by growing piece-wise-continues third order polynomial surfaces. Siwek's approach was to subtract a CAD model of the object being inspected. For this purpose she used XRSIM [Gray and Inanc, 1990] a simulation program that generates simulated X-ray images from CAD models.

Both techniques were reasonable successful. The techniques removed complicated geometries, revealing flaws that earlier was difficult to identify. However, both methods introduced artifacts that could be confused with flaws. The artifacts from Ulmer's technique were due to problems in boundary regions between two or more surfaces. These artifacts were typically present at sharp edges. Artifacts from Siwek's technique stemmed mainly from difficulties with gray scale and spatial registration. Gray scale registration is the process of matching grayscale distribution between the simulated and actual image, and spatial registration is scaling and aligning the two images. In spite of the artifacts, the work of Ulmer and Siwek showed encouraging results in the area of automatic general purpose flaw detection.

My work is a continuation of the development of John P. Basart's group in automatic general purpose flaw detection. Instead of trying to remove the geometry, I rely on a pattern recognition scheme that extracts local numerical quantities, *features*, that will be used to discriminate (classify) between flaws and non-flaws. To accommodate generality, a number of feature extraction methods and classifiers are provided. Before the system is ready for classification, an operator aided initialization, *training*, is necessary. The operator would choose method of preprocessing, feature extraction, and classification based upon the training data i.e. known examples (prototypes) of flaws and non-flaws. After the classifier has processed the training data, the system is ready for inspection of unknown data.

2. PATTERN RECOGNITION

2.1. Introduction

Pattern recognition can be considered the umbrella term for all artificial intelligence (AI) techniques. What all AI techniques essentially attempts to do is to automate a decision process which means, on the lowest level, to make a computer discriminate between two or more phenomena, *patterns*. A typical example of pattern recognition would be optical character recognition (OCR) where a page of text is automatically read into a text file by recognizing and discriminating between the different characters and punctuation. An example of abstract pattern classes would be normal or abnormal heart conditions which can be found by analyzing electrocardiograms [Tou and Gonzalez, 1974].

In general, pattern recognition can be broken down into two tasks: feature extraction and classification. *Feature extraction* is the task of gathering information on which to base the classification on, and *classification* is the process where the different pattern classes are discriminated between.

The reliability of the decision made of a pattern recognition system is of course highly dependent of how well the information extracted separates the different pattern classes, and how well the classifier performs the actual discrimination. There is a strong symbiotic relationship between the feature extractor and classifier. The feature extractor needs to produce discriminatory information the classifier can use, and the classifier needs to utilize the discriminatory information provided. Kandel [1982] made an interesting point; the feature extractor and the classifier are two processes that ideally tries to eliminate each other. A perfect feature extractor would extract information that completely discriminates between the pattern classes, and an ideal classifier would be able to distinguish between two or more classes regardless of the information provided. In real life, though, there are no such thing as a perfect feature extractor or a perfect classifier. In fact, it is often very difficult to find an optimal method of feature extraction or an optimal classifier. Choosing what information to extract and which criteria to use for classification are therefor often done heuristically.

The reason for this should be clearer by reading the rest of this chapter in which I will describe the principles of pattern recognition in greater detail and also review some of the most important families of classifiers.

2.2. Features and Feature Space

Features are quantified information produced by the feature extractor. Each feature is a numerical quantity, a measure of a particular characteristic. The actual feature value can be either measurements or calculated quantities. In a climate analyses, a useful feature could be the measured temperature. An alternative temperature feature could be the maximum deviation from the average over seven daily measurements. A binary temperature feature could be 1 for temperatures above the freezing point and -1 for temperatures below.

An important characteristic of features is that often the value of the feature itself is of less importance. What is sought is its capability of discriminating between the pattern classes.

One of the fundamental problems in pattern recognition is associated with feature extraction and is often referred to as the "problem of sensing" [Tou and Gonzalez, 1974]. Perfect representation of a pattern, which is measured in discriminating ability, is often difficult. Feature extraction is in many ways analogous to sampling a signal without knowing the signal's original frequency. Our sampled values represent the signal, but it is difficult to know how well.

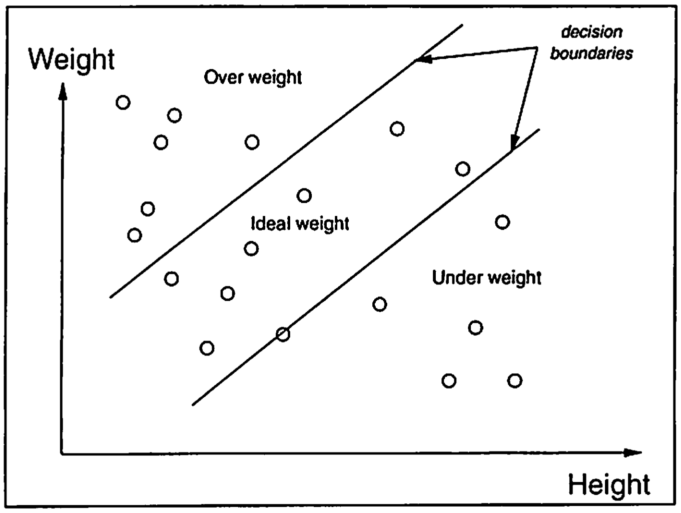


Fig 2.1
Simple example of a two dimensional feature space and two linear decision boundaries

Features are organized in feature vectors. A vector can be plotted in a vector space, and the vector space associated with feature vectors is called *feature space.*

Considering feature vectors as points in feature space, the classification process can be looked at as labeling regions in feature space as belonging to one or another class. Based on this view, the purpose of the feature extraction can be rephrased to: mapping patterns to feature space in such a manner that patterns of same class are grouped together while patterns of different classes are separated. The classification process then becomes one of determining *decision boundaries* between the different class regions.

A simple example should clarify the above principles. Imagine a nutrition study based on the two features *height* and *weight.* This gives a feature vector of dimension two. These feature vectors can be plotted in an X-Y coordinate system, a two dimensional feature space. The objective of the study could for example be to classify individuals as (a) under weight, (b) ideal weight, and (c) over weight. Such a feature space with appropriate decision boundaries between the different classes could look something like the illustration in Fig 2.1. In this illustration patterns are represented by circle points in feature space and they are separated into three classes by two parallel linear decision boundaries.

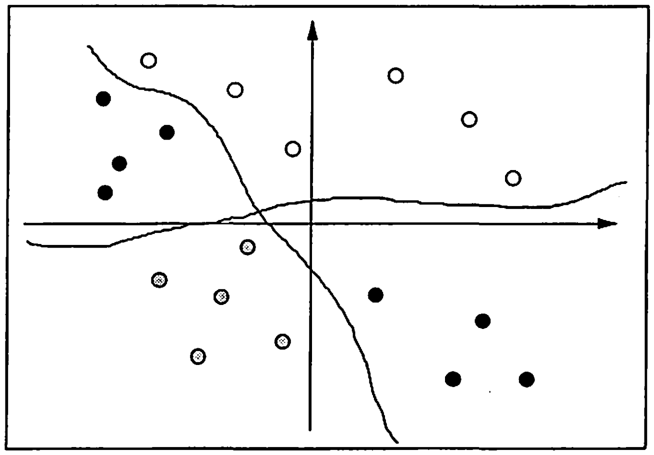


Fig 2.2
Examples of three pattern classes and nonlinear decision boundaries. The class denoted by black points are clustered in two clusters.

In general, decision boundaries are not parallel straight lines, and patterns from one class is not clustered in one cluster. Examples of more generalized feature space, patterns, and decision boundaries are shown in Fig 2.2. In this feature space the patterns are separated by nonlinear decision boundaries. The different pattern classes are labeled with different colors. The pattern class denoted by black points is clustered in two different clusters.

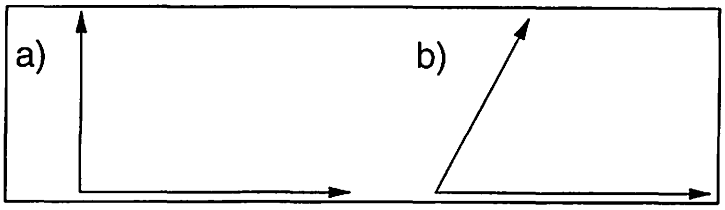


Fig 2.3
a) orthogonal vectors b) unorthogonal vectors

In the above illustrations of feature space we assumed that the features were orthogonal. This is a common assumption made in many classifiers. Orthogonality has several desirable properties. Firstly, a feature vector with orthogonal features does not contain any redundant information (Fig 2.3) Secondly, distances between patterns in feature space can be calculated using the popular Euclidean distance measures. However, sometimes it is difficult or time consuming to calculate orthogonal features. Then unorthogonal features have to do the job. In that case feature space will not be of the Cartesian style we are so familiar with, but rather a skewed version depending on the correlation between the features. (see Fig 2.3b). In some cases this can prove useful if the patterns are scattered in a more appropriate way.

2.3. Feature Optimization

Feature Optimization is the process of remapping the feature vectors to another domain (feature space) to make it easier for the classifier to do its job. One way of doing this would be to use only the features that proved to be most valuable for classification. Features that are not important for the classification process can seriously confuse the classification stage and should therefore be removed. Another way would be to do some kind of transformation that would better group patterns of the same class and separate patterns of different classes.

A lot of theoretical work has been done in this important area, but in my thesis I have concentrated very little of my effort on feature optimization. Instead, I have focused on choosing the right feature set. No matter how good a feature optimization algorithm is, it is not a miracle cure. What is most important is that the necessary information is available in the extracted features; garbage in will give garbage out. However, for fine tuning and further development of my work, feature optimization would be an interesting area to look into.

2.4. Classifiers

2.4.1. Overview

In Section 2.2, I defined the classification process as determining decision boundaries in feature space between different pattern classes. This is what classifiers are practically doing. However, many people like to look at the classification process as a function estimation [Kosko,1992; Bezdek,1982; Kandel,1982], and clearly this can be justified. What the classifiers are doing is mapping data from an input domain to an output domain. Classifier design can then be viewed as finding the transfer function between the input domain and the desired output domain.

Classifiers can be divided into two main categories: supervised and unsupervised. Supervised classifiers are classifiers that before classification have to go through what is often referred to as a *training phase* or *learning*. There is a lot of hype about "intelligent machines" that can "learn from experience". The *learning,* however, is simply an iterative process, an algorithm, that may or may not converge and that often is attempting to minimize an error criterion. Unsupervised classifiers are classifiers that do not need to go through this training process.

Another useful categorization of classifiers is distinguishing between model-based and model-free classifiers. Model-based classifiers often assume some sort of a priori statistical properties and are closely related to statistical estimation and optimization. Model-free classifiers are deterministic algorithms that often attempt to minimize a predefined object function.

The rest of this chapter will give a brief overview of the most important types of classifiers.

2.4.2. Unsupervised Classification, Clustering

Clustering algorithms are an important type of unsupervised classifiers. The object of an unconstrained clustering algorithm is to group pattern classes as far apart in the feature domain as possible. For this reason, many clustering algorithms use some sort of multidimensional distance measure.

Since clustering algorithms are not supervised i.e. given any additional information except the clustering criteria, they are usually not very fit to distinguish between classes that are spatially very close or partially overlap in feature space. Another disadvantage of clustering algorithms is that they can be computationally demanding, especially if the dimension of the feature vector is large and there are many patterns to cluster. This is the case because clustering algorithms have to iterate through all the patterns many times to group them into clusters.

The two most well-known clustering algorithms are the K-mean and the Isodata clustering algorithm. Given the number of desired clusters, K-mean attempts to minimize the squared distance from the patterns in a cluster domain to the cluster center. This algorithm is discussed in greater detail later. The Isodata clustering determines itself the number of clusters to group the patterns into. The algorithm consists of a fairly comprehensive set of heuristic procedures for splitting, merging, and moving cluster centers.

For more information about clustering algorithms, interested readers are encouraged to read [Tou,1974].

2.4.3. Unsupervised Classification, Statistical Model-based

Statistical classifiers are closely related to statistical estimation and optimization. A statistical Model-based classifier assumes certain statistical properties and then either minimizes an error or maximizes a likelihood function.

The most well-known statistical classifier is the Bayes' classifier. It is statistically an optimum classifier. The information it requires are the a priori probabilities and densities of each class as well as the cost of decision [Tou,1974]. For two classes the general form of the Bayes' classifier is:

X is assigned to class w_1 IFF

$$\frac{p(X|w_1)}{p(X|w_2)} > \frac{p(w_2)}{p(w_1)} \frac{(L_{21} - L_{22})}{(L_{12} - L_{11})} \qquad (2.1)$$

L_{ij} is the cost of assigning X to class i when it actually belongs to class j

For classification problems where the a priori probabilities and losses are known or can be reliably estimated, the Bayes' classifier is a very useful classifier and is widely used.

2.4.4. Supervised Classification, Deterministic Model-free

The largest group of supervised classifiers is model-free. These classifiers estimate the mapping function from the input domain to the desired output domain by generalizing a set of *training data* (training). *Training data* are prototypes of the different pattern classes, i.e. known pairs of input and output data. Based on these prototypes, decision boundaries in feature space between the classes are made.

Since the decision boundaries are purely based on the training data, it is extremely important that the training data cover as much of the feature space as possibly or alternatively, that they represent the data to be classified well. If not, the classification result will be unreliable. To find out whether the training data is representative or not is often very difficult. This is especially true for the cases with feature vectors of high dimensions. The problem of finding good training data is one of the largest disadvantages of supervised classifiers.

The simplest and most intuitive supervised classifier is the *Nearest Neighbor.* Unknown data are simply assigned to the same class as the closest prototype in feature space. The most commonly used measure for *closeness* in this classifier is the Euclidean norm.

Other important classes of model-free supervised classifiers are supervised neural networks and supervised classifiers based on fuzzy logic. These classifiers will be described in more detail in the two subsequent chapters.

2.5. Artificial Neural Networks

Artificial neural networks are a type of model-free classifiers, supervised and unsupervised, that have been given a lot of attention the last few years. These algorithms are massively parallel in architecture and are inspired by biological models of human neurons.

2.5.1. Definition of a Neuron

A neuron is often called a processing unit (PU) and is the fundamental building block in an artificial neural network, hereafter simplified to neural network. The basic characteristics of a neuron are the number of input signals, its individual weighting of these, its number of output signals, and its activation function (see Fig. 2.4).

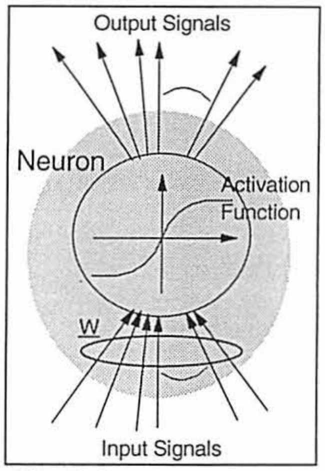


Fig 2.4
An illustration of a neuron, a neural network cell.

The input signals are output signals from other neurons (except for neurons in the input layer), and the output signals, all identical, serve as input signals for other neurons.

Activation functions can take a wide variety of forms, but they usually all have in common that they are nonlinear, like their biological counterparts. The most common activation functions are binary hardlimiter, bilinear hardlimiter, and sigmoid functions (Fig 2.5).

The general transfer function between input and output signals is:

$$output = f(\bigcirc_i x_i \Theta w_i) \qquad (2.2)$$

where f = activation function, $\bigcirc$ and Θ are operators, i=0 .. N-1, N = number of input signals, x_i is input signal i and w_i is the weight associated with x_i.

The most common operators used are summation and multiplication i.e. summing over products between input signals and their respective weights. This gives the transfer function:

$$output = f(\sum_i x_i w_i) \quad (2.3)$$

This type of processing elements are connected in different types of networks called neural networks. In the rest of this chapter the most common networks types are briefly discussed.

2.5.2. The Hopfield Network

The Hopfield network, invented by John Hopfield in 1982 [Hopfield, 1982], is a single layer network using a bilinear hardlimiter activation function (Fig 2.6). This network has versions for both binary and continuous valued inputs and outputs, but the binary version is the most well-known.

The Hopfield network can be configured to solve different types of problems, but here I will discuss the binary Hopfield network as a simple associate memory system. This network can be useful for binary patterns such as binary images or ASCII character code. There are other neural network models that do a better job than the Hopfield network for this purpose, but the Hopfield network is the most well-known and is also the easiest to understand.

Associate memories are memories which are addressable by content. When excited with an input pattern, an associate memory system will search in its stored pattern bank and output the pattern that closest resembles the input pattern. This way distorted or incomplete data can be completely recovered. Common applications of associate memories are noise cleaning and spell checking.

The Hopfield network stores its pattern bank in its weights, w_{ij}. These prototype patterns are located in states of energy minima (not necessary global) where energy is defined as:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} u_i u_j \qquad (2.4)$$

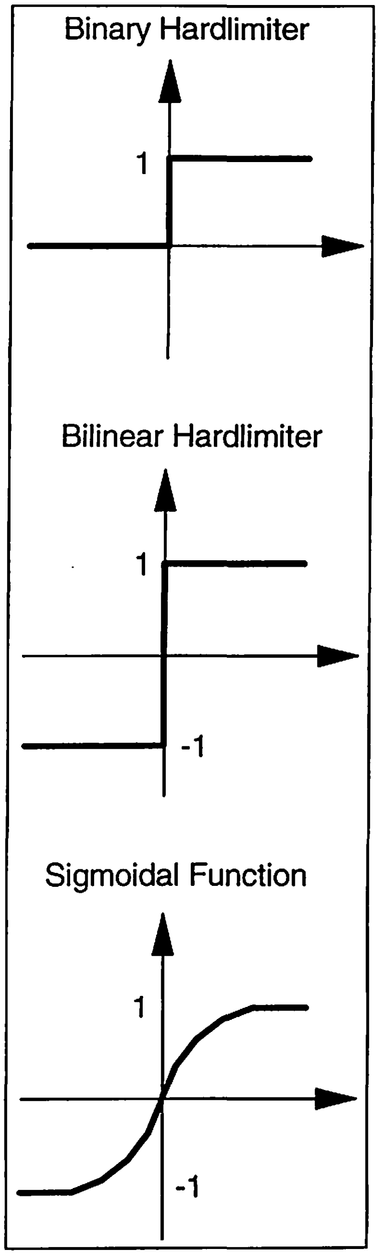


Fig 2.5:
Different types of common activation functions

where u_i is state of node i. After initializing the nodes with the input pattern, the Hopfield network randomly updates the node states (neurons) one by one with the following relation:

$$u_i(k+1) = f\left[\sum_{\substack{i \\ i \neq j}} u_i(k) w_{ij}\right] \quad (2.5)$$

where u_j(k+1) is state of node j at iteration k+1, i is node index, and f is the bilinear hardlimiter. This updating procedure has the property that the new energy state is smaller than or equal to the old. The network therefore eventually converges to a local energy minima.

The disadvantages of the Hopfield network are inefficient pattern storage, and unreliable prototype recall for a large pattern bank. The recall problem is due to generation of energy minima that are not associated with any of the prototype patterns. The Hopfield network is today not the associate memory of choice, but when it was first introduced it sparked new interest in the development of artificial neural networks.

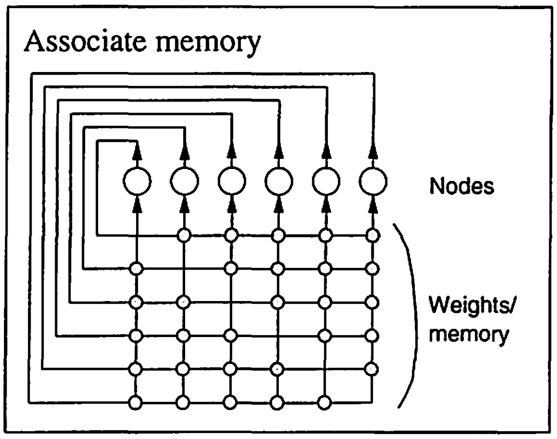


Fig 2.6
Illustration of a Hopfield network. Gray dots symbolizes weights and connection between nodes.

2.5.3. Probabilistic Neural Networks

Probabilistic neural networks are implementations of old statistical algorithms [Meisel,1972] in a neural network architecture. This was first done by Donal Specht in 1990 [Specht, 1990].

The idea behind this network is to model an unknown pdf (probability density function) by using a basis-function [Parzen,1962], for example, a multivariable gaussian. A 1-D example is shown in Fig 2.7. The principle is similar to making a frequency table for discrete data. In the continuous case, the bin-increment is replaced by a continuous function as for example a gaussian as in Fig 2.7.

It is shown that the estimated pdf converges asymptotically to the true density as the sample size increases [Masters,1993], thus we can construct a classifier that is asymptotically Bayes' optimal.

Introducing the prior probability p and cost of misclassification c while denoting the modeled pdf d, the classification can be formulated as follows: Assign $\underline{X}$ to class i IFF

$$p_i c_i d_i(X) > p_j c_j d_j(X) \qquad (2.6)$$

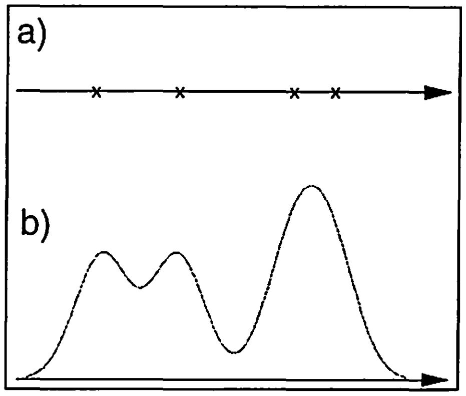


Fig 2.7
Modeling of 1-D pdf using an exponential basis function. a) shows sample data and b) shows modeled pdf.

where

$$d_i(X) = \frac{1}{n}\sum_{i=0}^{n-1} B\left[\frac{X-X_i}{\sigma}\right] \qquad , B[\ ] = \textit{basis function} \quad (2.7)$$

The prior probabilities are often set equal to each other and the same is true for the costs of misclassification if no additional information is known. The classification will in that case reduce to simply comparing the values of the two modeled density functions, d_i and d_j.

The density functions are usually averaged over the number of samples n to normalize the pdf's describing different classes. The pdf's could alternatively be normalized by using the requirement that their integral should be equal to 1, but this is usually not done because of the computational expense. Besides, the actual value of the pdf's or the pdf's integral is of no importance since classification is done by comparing relative pdf magnitudes according to eq. 2.6.

The disadvantages of probabilistic neural networks are slow classification and that there are no good general way of finding the spread of the basis function σ in eq. 2.7. The classification performance is highly dependent on this parameter, so the latter is a serious difficulty associated with probabilistic neural networks. However, once a good basis function and a useful spread is found, the probabilistic neural networks perform well.

2.5.4. The Backpropagation Network

The most popular of all neural networks is the *Backpropagation network.* This network has been successfully implemented in many applications. Among its earliest achievements was speech synthesizing from text in a system called NETalk developed by Sejnowski in the late 1980s [Anderson, 1988]. According to Kosko [1989] a tape recorder replayed NETalk's training experience, from babble to baby talk to articulate speech.

The backpropagation net-work is based on a feed-forward architecture called multi-layer perceptron architecture (Fig 2.8). This type of architecture has been theoretically proven by Hornike and White [White,1989] to have the capability of approximating any Borel-measurable function to any desired accuracy. This holds provided there are enough hidden nodes i.e. neurons between the input and the output layer.

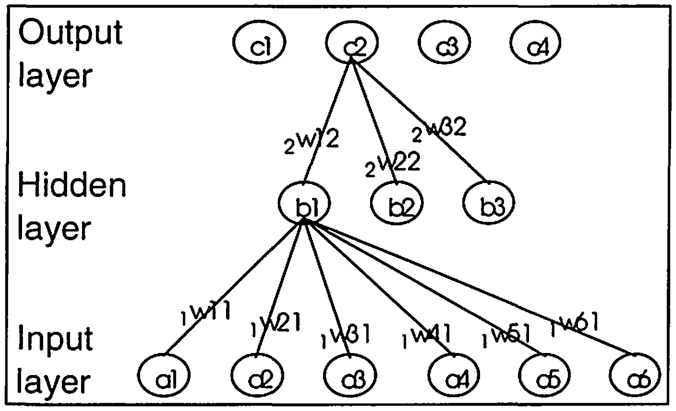


Figure 2.8
A schematic illustration of a feed-forward multi-layer perceptron network with one hidden layer. All signal propagation is directed forward i.e. from the input neurons a_i to the output neurons c_i. All nodes in two successive layers are connected, but only a selected set of node connections are shown to clearer explain the architecture.

This network's learning algorithm is called the backpropagation learning algorithm gave the network its name. It was introduced by Rumelhart et al. in 1986 [Rummelhart,1986], who referred to it as *the generalized delta rule.* This algorithm is a nonlinear extension of stochastic theories for least mean square optimization. It is based on a gradient descent search on a random squared-error surface. Usually gradient descent methods are guaranteed to converge to a local minima, but because this error surface is stochastic in nature rather than deterministic, the backpropagation algorithm does not always converge and can oscillate or even wander chaotically [Kosko, 1989].

Another problem associated with the backpropagation learning is its computational cost. Adequate training can often be extremely computationally intensive. For large systems the training can take as much as days.

In spite of the above mentioned problems, the backpropagation network is the most widely used neural network today. It is used for a wide variety of applications such as process control, speach recognizion, time series analyses, OCR, and approximation of complex or unknown transfer functions.

2.6. Fuzzy Logic

2.6.1. Introduction

Fuzzy logic, introduced by Lofti Zadeh in 1965 [Zadeh, 1965], is a superset of traditional (bivalent) logic. Instead of only allowing only two values: *true* and *false*, fuzzy logic introduces a *degree of truth.* This degree, often referred to as *membership*, usually lies in the open interval 0 to 1.

$$\mu_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases} \qquad (2.8)$$

$$\mu_A(x) \in [0,1] \qquad (2.9)$$

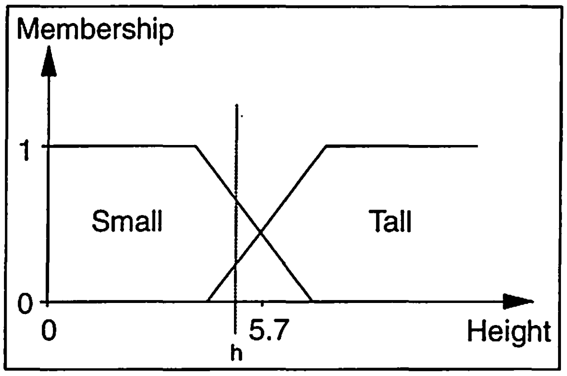


Fig 2.9.
Illustration of two fuzzy memberships: Small, and Tall. The height h would have a membership of about .25 and .75 in the classes Small and Tall respectively

Equation 2.8 and 2.9 defines bivalent logic and fuzzy logic memberships respectively for the event x belonging to the class A. A practical example is shown in Fig 2.9.

An interesting observation made by Kosko [1992] is that bivalent paradoxes stemming from the principle of non-contradiction (X AND not-X = 0) and excluded middle (either X OR not-X = 1), appears as fuzzy midpoints. An example of such a paradox is a card that on one side has printed "The sentence on the other side is true" and on the other side "The sentence on the other side is false". The degree of truth in these two statements turns out to be 0.5 in the fuzzy domain from 0 to 1.

A more practical and intuitive application of fuzziness can be seen from considering the classical *sorites* paradox. According to bivalent induction we can remove graines of sand one by from a sand beach and after each grain argue that there is still a beach of sand. No grain takes us from a beach of sand to not a beach of sand. A fuzzy measure, however, would for each removed grain give a smaller membership value to the class *beach of sand*, and after removing the last grain the membership would be zero.

2.6.2. Fuzziness vs. Probability

At first fuzziness and probability can appear to be very similar if not the same. Consider a flaw-candidate that has a probability of 0.7 being a flaw of a particular class and a fuzzy membership value of 0.7 to this flaw. What is the difference? The answer to this question is that the probability of 0.7 tells us that statistically 7 out of 10 flaw candidates of this *type* are flaws whereas the membership value of 0.7 tells us that *this particular* flaw candidate has characteristics that matches with a degree of 0.7 with the flaw characteristics. This is a fundamental difference that favors the fuzzy measure over probability.

Mendel illustrated the difference between probability and fuzzy membership with two liquids A and B, A having 0.1 probability of being lethal poisonous and B having 0.1 fuzzy membership value of being lethal poisonous. Which liquid is safest to drink? After chemical analyses, it turns out that the liquids were both ordinary beer. This changed the 0.1 probability of A being poisonous to 0, but the fuzzy membership of B remained at 0.1. This is because the alcohol in the beer is a lethal poison, and revealing that B was a beer didn't change liquid B's characteristics. If choosing between liquid A and B, liquid B would be the safest because it

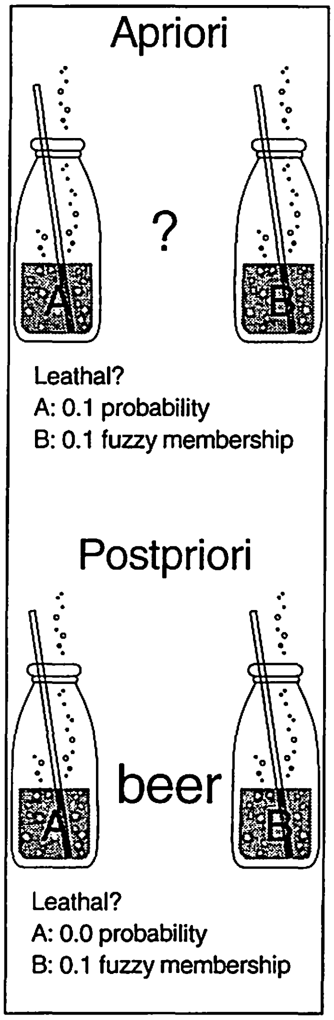


Fig 2.10
The difference between probability and fuzziness.

would never be lethal (only 0.1). Choosing liquid A would be lethal in 1 of 10 instances which dramatically increases the risk.

From the above discussion it should be apparent that probability is a categorization of a type of phenomena whereas fuzziness is an earned score based on characteristics of the particular event in interest. This means that the fuzzy measure is a more well behaved and reliable descriptor than probability. This claim is of course built upon the assumption that the fuzzy membership function correlates with reality. In real life, it can often be difficult to find good fuzzy membership functions. However, this is comparable to statistical analyses that often assumes or approximates necessary distributions.

2.6.3. Membership Functions

The fundamental building block in a fuzzy logic system is fuzzy member functions. Member functions for the two classes Small and Tall was given in Fig 2.9. These were based on piece wise linear functions. Once can ask: How should membership functions be designed? To this question there is no right answer, which is one of the largest disadvantages of fuzzy logic techniques. Creation of membership functions is often quite arbitrary. Membership functions are therefore created heuristically, based on the designers experience and understanding of the problem to solve.

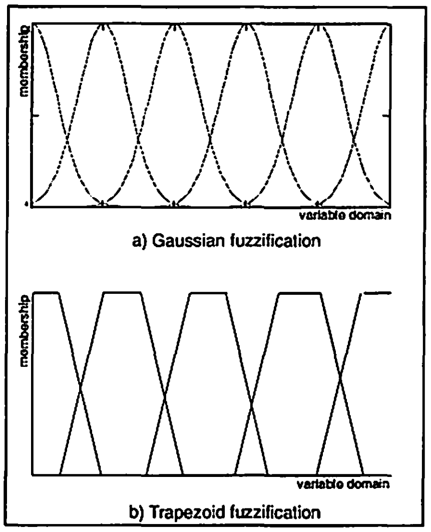


Fig 2.11
Common types of fuzzification of a variable interval. The classes in b) could for example be labeled small, kind of small, medium, kind of large, and large.

Commonly used membership functions are piece-wise linear and gaussian. The reason for this is partly ease of implementation, and partly because these functions have proved to be useful. Examples of fuzzification of a variable domain using these functions is shown in Fig 2.11 a) and b).

2.6.4. Fuzzy Set Operations

A membership function can be looked at as a set with all the possible values between and including 0 and 1. Operations on such a set or between two such sets are called *fuzzy set operations.*

Unitary operations, operations on one set, are often called *hedges.* Hedges are linguistic modifications of a base class. The two most common hedges are associated with the terms *very* and *somewhat.* Examples of such modifications can be seen in figure 2.12. c) Here the base membership "Fast" in b) is modified by point-wise taking the square root and square of the original, respectively, to create the two new hedge derived classes "Somewhat Fast" and "Very Fast".

The most common binary operations, operations on two sets, are *negation, conjunction,* and *disjunction.* They are defined in eq. 2.10, 2.11, and 2.12 respectively. There are also many other binary fuzzy operations, but the mentioned operations are the most common because of their simplicity and robustness to noise.

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \qquad (2.10)$$
$$\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)] \qquad (2.11)$$
$$\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)] \qquad (2.12)$$

To see what these fuzzy operations really means it useful to look at a practical example. Consider an inspection system that is to be described using the two membership functions *inexpensive* and *fast.* These can be seen in Fig 2.12 a) and b) respectively. The input variable in both these classes is price. In Fig 2.12 d) the class "Inexpensive" is negated to create the class "Expensive". In e) the new class "Inexpensive and Fast" has been created by a conjunction between "Inexpensive" and "Fast". In f) the class "Inexpensive OR Fast" has been generated by a disjunction of the same two base classes.

2.6.5. Translating IF-THEN Rules to Fuzzy Domain (inference)

Fuzzy logic systems have the unique feature that they can utilize linguistic rules, rules based on quantitative information which resembles everyday speech. This can allow a doctor to design an expert system using the following type of rules: IF the body temperature is *high* and the color of the face is *pale* THEN the patient is *ill.* Such rules can be fuzzified using for example the strategies illustrated in Fig 2.13.

When processing rules, the fuzzy system uses the basic operations of negation, AND, and OR. A rule is a way of deducing a conclusion based on the given data. Most deduction is based on *modus ponus.* This types of reasoning can be illustrated as follows:

Rule:	*All male humans have a Y-chromosome*
Premise:	*This human is a male*
Conclusion:	*This human has a Y-chromosome*

The two most common methods for applying such a reasoning to fuzzy sets are *correlation minimum* and *correlation product.* In both strategies the basic conclusion membership class is modified with the degree of thruth of the premise. Correlation minimum is simply taking the point-wise minimum of the premise and the basic conclusion membership class as seen in Fig 2.13 c). Correlation product, on the other hand, performs a multiplication between the value of the premise and the basic conclusion membership.

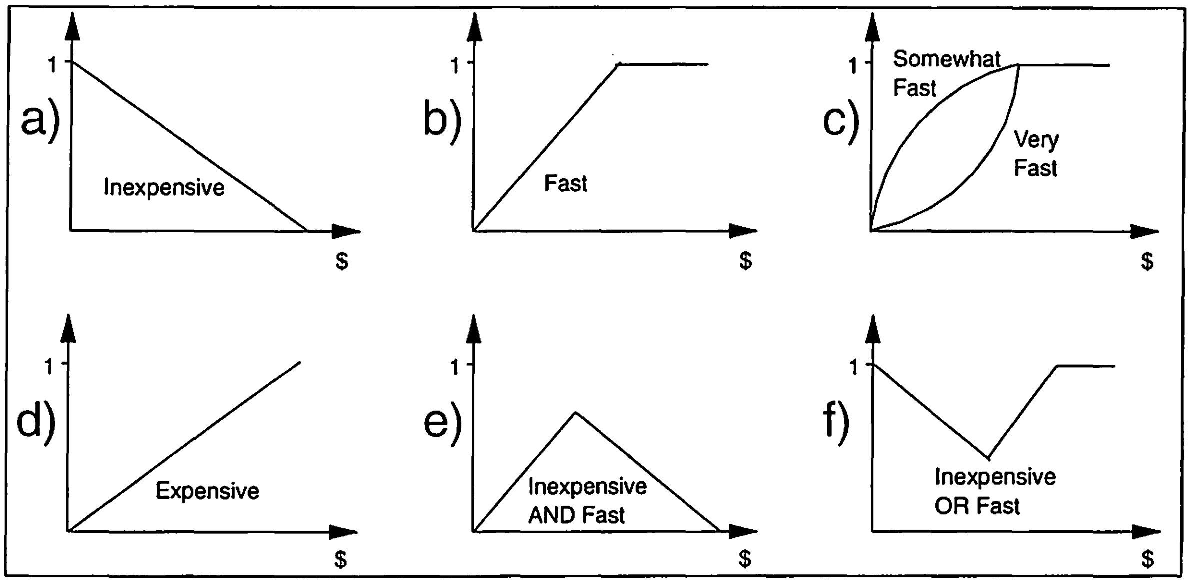


Fig 2.12
Illustrations of fuzzy set operations. a) and b) original fuzzy sets, c) hedges, d) negation of "Inexpensive", e) conjunction between "Inexpensive" and "Fast", f) disjunction between "Inexpensive" and "Fast". The input variable for all classes is price in dollars.

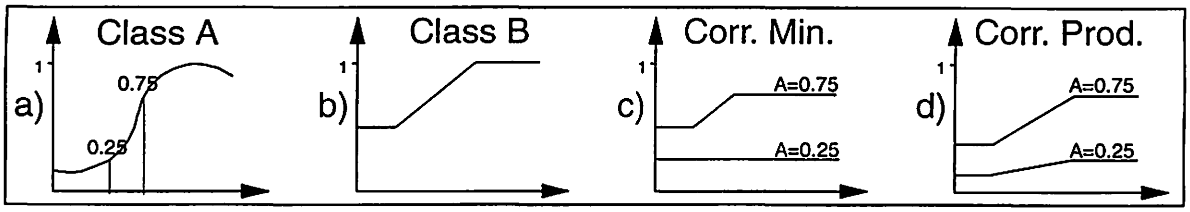


Fig 2.13
Illustration of translating the rule "IF A THEN B" into fuzzy domain. The original membership classes A and B are shown in a) and b), respectively. Marked in A is also the two cases of A=0.25 and A=0.75. Two alternative correlations with B are shown in c) and d). Each correlation strategy is shown for both cases of A as labeled.

A practical example will illustrate how these two strategies differ. Consider the rule: "IF *A* THEN *B*". Assume class A and B has the membership functions in Fig 2.13 a) and b) respectively. With these membership functions and the two cases where A=0.25 and A=0.75, we would get the fuzzy translations of the rule as illustrated in Fig. 2.13 c) and d). From Fig 2.13 c) we can see an apparent disadvantages of the method of correlation minimum when considering the case of A=0.25. For this case all information in the basic conclusion membership function shown in Fig 2.13. b) is lost for function values larger than the degree of truth of A (the premise). The method of correlation product shown in Fig 2.13 d), on the other hand, is scaling the conclusion membership and thereby keeping all the original information.

2.6.6. Combining Fuzzy Rules

The basic idea when combining fuzzy rules is that rules with the same conclusion are ORed and rules with opposite conclusion are ANDed. Let's consider a practical example. Consider a simple flaw detection system that has the fuzzy rules:

Rule 1: *IF steep edge AND low mean THEN flaw detected*
Rule 2: *IF not-steep edge AND round contour THEN flaw detected*
Rule 3: *IF not-critical location, THEN not - flaw detected*

The flaw detection system we have defined uses four features: edginess, mean, contour shape, and location. The necessary membership functions associated with these variables are defined in Fig 2.14 d), e), h), and j) respectively. The base membership function for "Flaw" is defined in 2.14 a).

The three bottom rows in Fig 2.14 correspond to the three rules. The horizontal stippled lines are indicating degrees of truth of the premises. As defined in chapter 2.14 ANDing two premises is the same as choosing the minimum of their degree of truth. This minimum is then used to modify the base membership "Flaw". Comparing this membership with Fig 2.14 f), i), and l), it is apparent that our system is using *correlation product inference* as defined earlier.

Combination of rules can bee understood by looking at Fig 2.14 c). The point-wise maximum of rule 1 and rule 2 are chosen (ORing) because these rules have the same conclusion, namely *flaw detected.* The point-wise minimum of the resulting membership function and the membership function from rule 3 is then calculated (ANDing) because the third rule has an opposite conclusion membership of the first two. The grand result is marked with thick line and represents the fuzzy domain of all rules combined given the particular values of the premises.

However, the result in Fig 2.14 c) would be of little use for an operator who wants to know whether there is a flaw or not. This is taken care of in the *defuzzification* which is described in the next section.

2.6.7. Defuzzification

Defuzzification is the process of translating the combined fuzzy membership of all rules into a scalar. For this purpose there are two commonly used methods. One method is to choose the point whose membership is the largest. This has the disadvantages that the membership function has multiple maxima. A more reliable method is to find the centroid. The centroid can be interpreted as the point at which the membership would balance if it was made of a homogeneous solid and is defined as follows:

$$\bar{x} = \frac{\int_D x\mu(x)dx}{\int_D \mu(x)dx}, \qquad \text{D = domain of } \mu(x) \qquad (2.13)$$

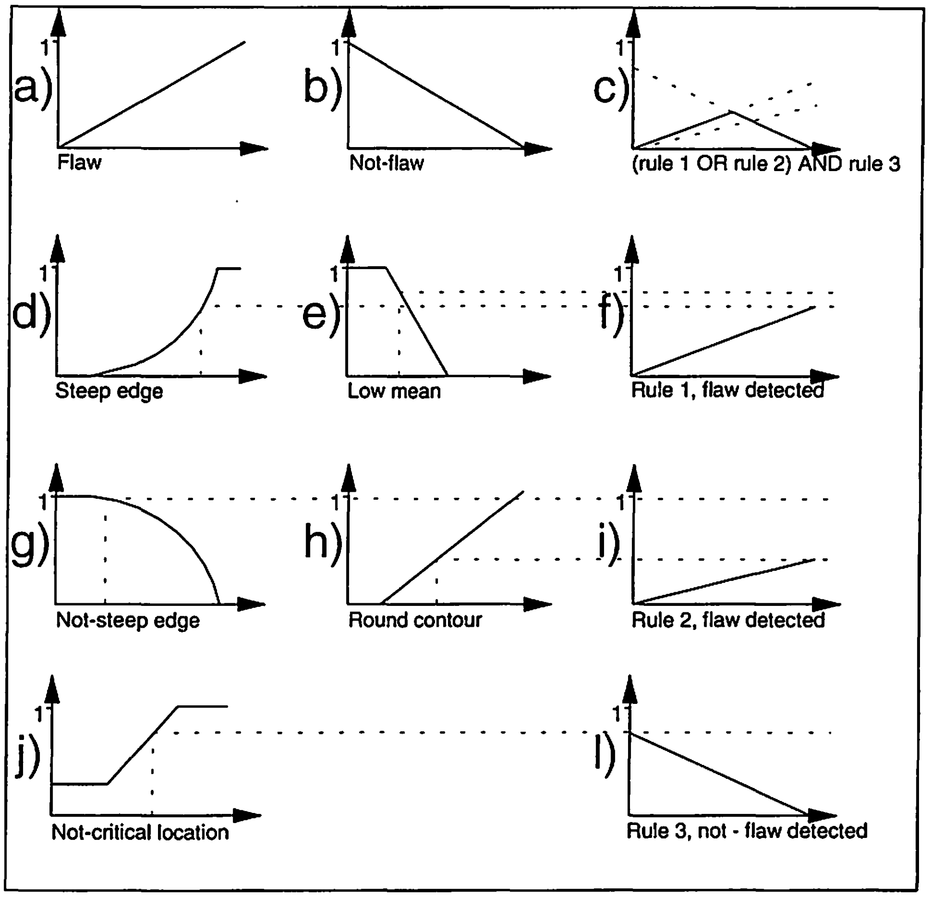


Fig 2.14
The illustration shows how fuzzy rules are combined. The three bottom rows correspond to three fuzzy rules. The three rule outputs are then combined as shown in c). Rules with same conclusion are ORed and rules of opposite conclusions are ANDed.

In practice, a piecewise linear approximation is used. By using a trapezoid model (Fig 2.15) any function can be estimated with arbitrary accuracy. Integration of such an approximation reduces to summing areas of trapezoids. Indexing the trapezoid with the variable i the expression for the centroid in closed form can be written as:

$$\bar{x} = \frac{\sum_i \frac{1}{6}(b_i - a_i)\left[h_1(2a_i + b_i) + h_2(a_i + 2b_i)\right]}{\sum_i \frac{1}{2}(b_i - a_i)(h_1 + h_2)} \qquad (2.14)$$

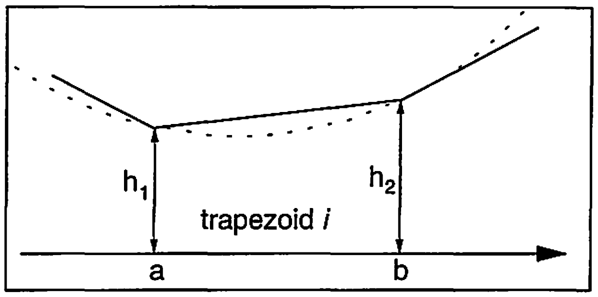


Fig 2.15
Piece wise linear function approximation

3. SHERLOCK

3.1. Motivation

The work with Sherlock was motivated by the large amount of human inspector-based flaw detection in industry. If we could automate the inspection process by developing a system that could reliably find flaws in images without human supervision, a lot of man power, time, and money could be saved. Other advantages of an automatic computer-based system are increased consistency and quantified decision justifications.

Consistency in a computer based-inspection is much higher than in a manual inspection system because manual inspectors perform differently, and performance of one particular inspector fluctuates over time due to drowsiness and external disturbances. A computer, on the other hand, would be absolutely consistent from time to time.

Quantified decision justification are in general preferred for documentation purposes over qualitative judgment which often is quite subjective. If flaws can be described in numerical quantities, it can, for example, also be useful for comparison and categorization of flaws. In negotiation with suppliers or customers, quantified documentation can also play an important role.

A computer based inspection system would also have potentials for automatically providing documentation of the inspection. In addition to being very convenient such documentation could include a confidence measure based on the numerical quantities (features) describing the flaws, which would be extremely useful in evaluating the inspection results. Flaw detection systems based on human inspectors, for comparison, do not have a good objective way of producing a confidence of classification.

With all these highly desirable features of automated computer inspection one can ask oneself why inspection is done manually at all. There are two main reasons why automatic inspection systems are not more widespread today. Firstly, it is difficult to develop a computer system that is as reliable and fast as an experienced human inspector. Secondly, people in the decision making positions are often unfamiliar with the techniques used in the automated computer systems and are therefore less likely to trust inspection of large and expensive productions to such systems.

3.2. Design Objectives

The objective I had for my work was to design a prototype system for automatic general purpose image based flaw detection. My hope was that my work would show the feasibility of such a system in an industrial setting.

When starting out I had five major design objectives: automation, generality, reliability, speed, and portability.

Automation is one of the key ideas behind Sherlock. To reduce the cost of manual inspection it would not be good enough to develop a semi-automatic system that would require an operator. The whole inspection process has to be automated, starting after image acquisition and ending with labeling of flaw regions.

Generality was the other major principle driving this work. I wanted to make a system that was applicable to any type of image based flaw detection i.e. a system that could find any type of flaw in any type of image. This objective was motivated by the large number of sponsors of the NDE center. Each of them have their own type of flaw detection problem, and if I could develop a general purpose system, it would be useful for all of them. Of course developing such a system is an enormous challenge, and I was not really expected to achieve this, but this area is of such importance that even some encouraging results would be interesting.

Reliability is of course one of the most crucial factors in designing an inspection system, and the obvious comparison Sherlock would face is the performance of human inspectors. There are several industrial inspection systems on the commercial market, but many of them are outperformed by manual inspectors. It turns out that it is very difficult to emulate the performance of the human eye and to incorporate a human's experience into a computer algorithm.

Speed is an other important factor of industrial inspection. Often, an inspection is preferred in real time in an assembly line fashion. Many computer algorithm that perform well are extremely time consuming. A trade-off is therefore often necessary between speed and reliability.

My last design objective was portability. I wanted to involve the sponsors of the NDE centers as much as possible in my work since they are the experts and also the people who would eventually use a system like Sherlock in their daily work. It was therefore important that my work was easily ported to their machines. It was equally important that my program was easy to use. This is also a sort of portability that is important in a busy schedule.

3.3. Short About Important Design Decisions

To provide generality I chose to base Sherlock on a pattern recognition scheme. By equipping Sherlock with feature extractors that can pick out a lot of different information and classifiers capable of forming many types of decision boundaries, a wide variety of flawdetection problems can be solved.

A pattern recognition scheme is also very modular and that can help improve Sherlock's reliability. By choosing appropriate feature extractor and classifier the inspection process can be tailored to each particular detection problem. If for example gradient information is important for one detection problem, gradient information can be extracted without including any other type of information which would only confuse the classifier. What classifier to choose can likewise be tailored to the particular problem at hand. If none of the implemented feature extractors or classifiers fit an encountered detection problem, better feature extractors and classifiers can be implemented without having to change or redo the detection scheme. This way Sherlock can be expanded to reliably solve new detection problems without redoing the whole work.

Another important design decision was to implement Sherlock in MS Windows. The other alternative would be a MOTIF/UNIX implementation, but porting applications between different hardware platforms are a lot more problematic in practice than what hardware vendors want us to believe. An MS Windows application is therefore a lot more appealing since no porting is necessary between platforms running MS Windows. An other advantages of an MS Windows implementation is that most people seem to have access to a PC and are actually relying a lot on MS Windows based text editors and presentation programs. A computer based inspection system in MS Windows is therefore convenient since inspection results can be pasted directly into reports or presentations generated in another MS Windows application.

Another aspect of an MS Windows implementation is that it is user-friendly and easy to use. By learning the basic principles of how the program works, a person with no schooling in pattern recognition can try the different algorithms and see how well they perform. This way my work would be easily available for evaluation by the NDE sponsors.

A major counter argument for implementing Sherlock on a PC is of course speed considerations. UNIX workstations are in general much faster than PC's. This is starting to change though. PC's are getting increasingly faster and are closing the gap. In addition, PC's can be equipped with extremely powerful plug-in cards to tailor the system to the specific needs. If speed and computation power is the bottleneck, one can use a DSP board which can boost the performance up to workstation level and above. If one wants to hook up the PC with a camera or any other image acquisition system, there are a variety of frame grabbers and interface modules to choose among. DSP boards and interface modules are also availble for workstations, but they are fewer and extremely expensive. Another argument in favor of PC's is that UNIX workstations are often slowed by network jams. Considering all these factors, I think PC's are performance-wise the best platform for computer-based inspection. If we include price in the picture PC's become even more attractive.

3.4. System Overview

The basic working principles for a completed Sherlock in an industrial setting is illustrated in Fig 3.1. Inspection can either be done in real-time (in an assembly fashion) or on previously acquired images.

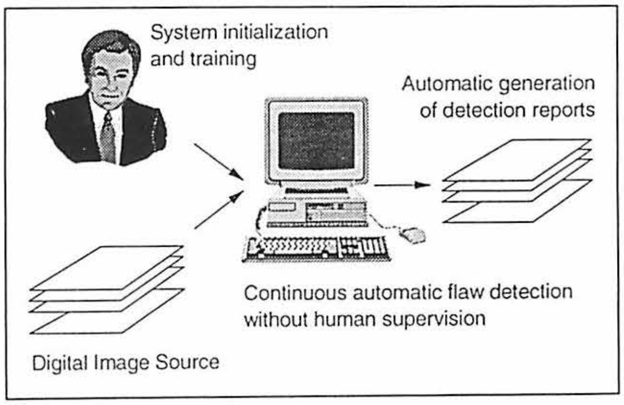

System initialization and training
Automatic generation of detection reports
Continuous automatic flaw detection without human supervision
Digital Image Source

Fig 3.1
System overview of Sherlock

Before the inspection can take place, an operator is needed for the training process. The operator chooses methods of preprocessing, feature extraction, and classification based upon known examples of flaws and non-flaws. After training the classifier, the system is ready to run, and can then continuously inspect digital images for flaws without human supervision. To check the detection results, an operator can print out the automatic generated detection reports.

For this thesis, all the fundamental stages in the above described system were studied and implemented. Two tasks remain before Sherlock can be installed in an industrial facility. First of all it needs a macro unit for automatic loading of images and execution of chosen algorithms. In addition, it is also missing the capability of automatic generation of detection reports. None of these tasks is interesting from a research point of view and were therefore not implemented because of lack of time.

3.5. Detection Scheme

Sherlock is based on a pattern recognition scheme as described in Chapter 2 and illustrated in Fig 3.2. The input image is mapped to feature space using different feature extractors and boundaries between flaws and non-flaws are then found by any of the classifiers. This scheme could also be used for 1-D data such as ultrasonic A-scans. To accomplish this, a new battery of feature sets would be needed.

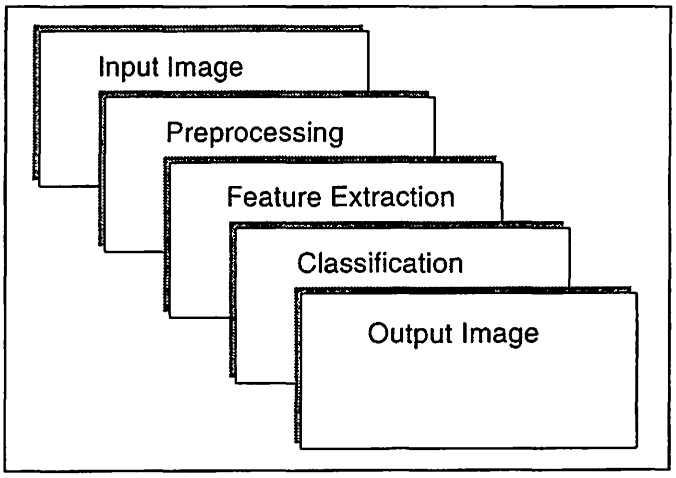


Fig 3.2
The basic processing stages of Sherlock.

The processing stages of Sherlock are illustrated in Fig. 3.2. Input images are preprocessed if necessary before feature extraction. The feature vectors are then fed into the classifier which dumps the classification result on the screen as an image in which flaws and non-flaws are colored differently.

3.6. Preprocessing

Preprocessing can be any type of image processing that would ease the detection task. A simple preprocessing step would be noise cleaning. Sherlock supports three types of noise cleaning: spatial averaging, median filtering, and Butterworth low-pass filtering.

Other preprocessing techniques could be removal of trend or geometry. Trend removal is removing a general background trend. This is often done by fitting and subtracting a polynomial. Geometry removal is the process of removing intensity variations in the image due to the structure and geometry of the specimen being inspected. Either of Ulmer's [1992] or Siwek's [1994] methods could be used for this purpose. Neither trend or geometry removal is implemented in Sherlock. Implementations of such algorithms can be quite time consuming and is outside the scope of this work.

3.7. Feature Extraction

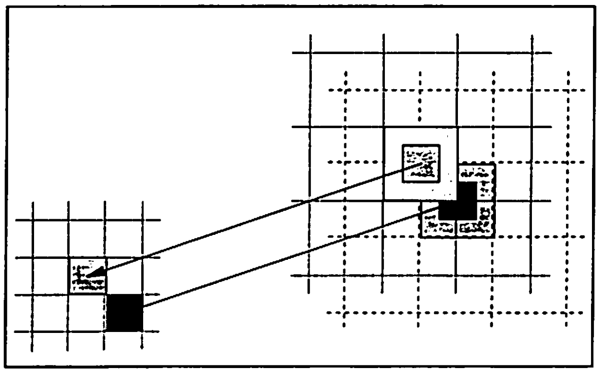


Fig 3.3
Illustration of feature extraction.

Feature extraction in Sherlock is done by moving a rectangular window on top of the input image shown in Fig 3.3. Within this window a set of numerical characteristics is calculated, for example, the mean and the variance. Each of these numerical quantities represent a feature as defined in chapter 2 while they collectively define a feature vector that is associated with the center pixel. By moving the window pixel by pixel, each pixel is represented with a feature vector. When this vector is classified as being a flaw or a non-flaw, so is the pixel.

In most instances, flaw detection does not require 1-pixel resolution. To save time and computation it can be useful to move the window more than one pixel at the time. By moving the window *k* pixels at the time, the computation is reduced with a factor of k^2 which is paid for with a resolution of *kxk* pixels in the output image. Consider the case in which the window is moved four pixels in each direction. This means that the calculated vectors are effectively associated with the 4x4 pixel center areas of the window. Each vector therefore represents 16 pixels which of course only requires 1/16 of the original computation and cannot give a better resolution than 4x4 pixels in the output image. In

Sherlock's terminology the window size is called *feature support* and *k*, the number of pixels the window is moved each time is called *grid size*.

An interesting interpretation of the feature extraction process is that it can be looked at as a mapping from one image (the input image) to *n* images where *n* is the number of features in the feature vectors. Fig. 3.4 is an illustration of this. If the mean was the first element in the feature vector, an image could be created by combining all the means. I call images created from the feature vectors *feature maps*. I use the term *map* because these images gives associations to maps where different elevations are colored differently.

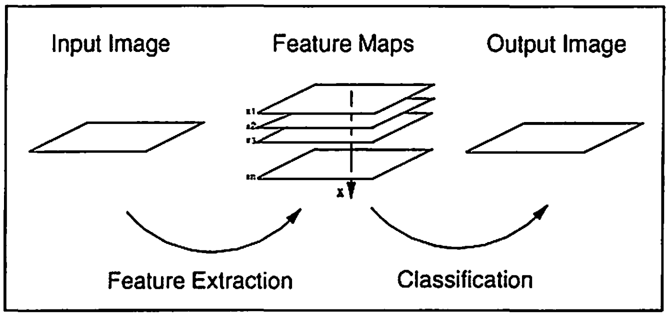


Fig 3.4
Illustration of how feature extraction can be looked at as a mapping from one image to n images (feature maps) where n is the order of the feature vector. Feature vectors can be assembled by collecting intensities of pixels with the same coordinates.

Feature maps can be utilized when evaluating a feature's capability to discriminate between flaws and non-flaws in a feature. The easier a flaw can be distinguished from the rest of a feature map, the higher is this feature's discriminatory capability. If it is absolutely impossible to see the flaw in a feature map, it's feature contains very little information, if any, that is useful for classification purposes.

By looking at Fig. 3.4 it is apparent that the classification can be considered as a mapping from *n* images to one image. This interpretation is of less importance and is not emphasized in Sherlock.

3.8. Implemented Feature Sets

3.8.1. Introduction

The feature sets implemented in Sherlock can be divided into two groups: texture descriptors and geometry features. These feature sets give complementary type of information and cover a broad information range, making Sherlock applicable for a wide variety of flaw types.

Geometry features are the most intuitive and these descriptors give information about edges, contours, and geometric shapes. Many geometry features are transform based. Image transforms can often be interpreted as a 2-D expansion analog to the familiar 1-D Taylor series. The first terms in such series contain the most energy because they describe the general signal trend whereas the later terms describe the smaller signal fluctuations. By analyzing such image expansions certain terms can prove useful in discriminating between flaws and non-flaws.

The other feature category is textures, and texture can be defined as a mixture of edges and irregularities that combined have the appearance of a more or less homogeneous pattern. The examples in Fig 3.5. are natural textures taken from the Brodatz album. Brodatz textures are commonly used in texture analyses.

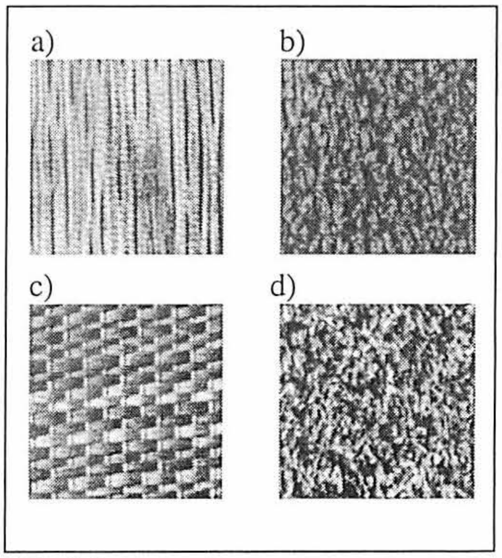


Fig 3.5
Examples of four natural textures

In flaw detection, textures can be a useful way to describe flaws. Some flaws can be described in terms of edginess or geometry. Examples of an edge flaw would be cracks, and a geometry flaw would for example be a spherical-type void. Many flaws, however, does not have a particular shape or geometry associated with them. In those cases texture features can be valuable descriptors.

There are three types of texture features that are implemented in Sherlock. All three of these feature sets are statistically based. Statistics alone do not necessarily completely describe a texture, but they can often give enough information to discriminate between different texture types.

The rest of this chapter will discuss the implemented features in greater detail starting with the statistical texture features.

3.8.2. First Order Histogram Features

First order histogram features are revealing properties of the ordinary pixel intensity distribution in the neighbor-hood, feature support, of the center pixel(s) (see Fig. 3.6). The term *first order* is used because such histograms corresponds to the continuous first order pdf.

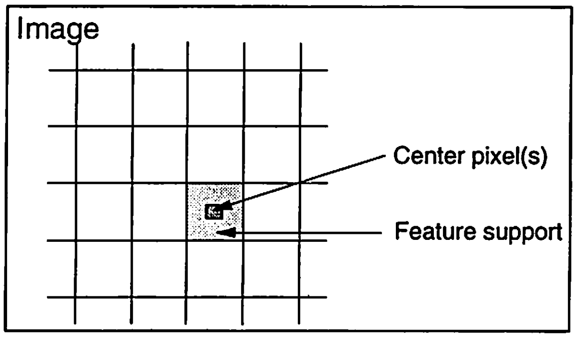


Fig 3.6
Feature extraction

Since no location information (within the feature support) is included, there might be several subimages that have equal or similar first order histograms. However, information about the intensity distribution alone can often be sufficient to find the flaws.

The first order histogram is as earlier mentioned a discrete approximation to a continuous density function. Denoting the density function P(b) the approximation can be formulated as

$$P(b) \approx \frac{N(b)}{M} \qquad (3.1)$$

where b is the intensity value, $N(b)$ is the number of pixels with intensity value b and M is the number of pixels in the feature support.

Altogether, there are six first order histogram features implemented in Sherlock: *mean, standard deviation, skewness, kurtosis, energy,* and *entropy.* These features are defined as follows:

3.8.2.1. Mean

The mean (see eq. 3.2) is the simple arithmetic mean and gives the average pixel intensity over the feature support. The usefulness of this feature alone is often rather limited, but combined with other features, the mean is often quite significant.

$$\begin{aligned}
&\textit{Mean,} && S_m = \sum bP(b) \\
&\textit{Standard deviation,} && S_D = \left[\sum (b-\bar{b})^2 P(b)\right]^{1/2} \\
&\textit{Skewness,} && S_S = \frac{1}{\sigma_b^3}\sum (b-\bar{b})^3 P(b) \\
&\textit{Kurtosis,} && S_K = \frac{1}{\sigma_b^4}\sum (b-\bar{b})^4 P(b) \\
&\textit{Energy,} && S_N = \sum [P(b)]^2 \\
&\textit{Entropy,} && S_E = -\sum P(b)\log_2 P(b)
\end{aligned} \tag{3.2}$$

3.8.2.2. Standard Deviation

Standard deviation (see eq. 3.2) is also a well-known statistical property. This is essentially the average deviation from the mean over the feature support. A standard deviation of 2.4 means that the pixels on average deviate from the mean by 2.4 intensity increments.

The standard deviation does an excellent job in detecting edges and spikes on relative horizontal surfaces. If there are significant background trends, some caution is needed in interpreting the standard deviation. Such trends would, of course, also spark large standard deviations without warranting labeling the area of interest as an area with edges or spikes. In such situations, it can still be possible to discriminate between areas of flaws

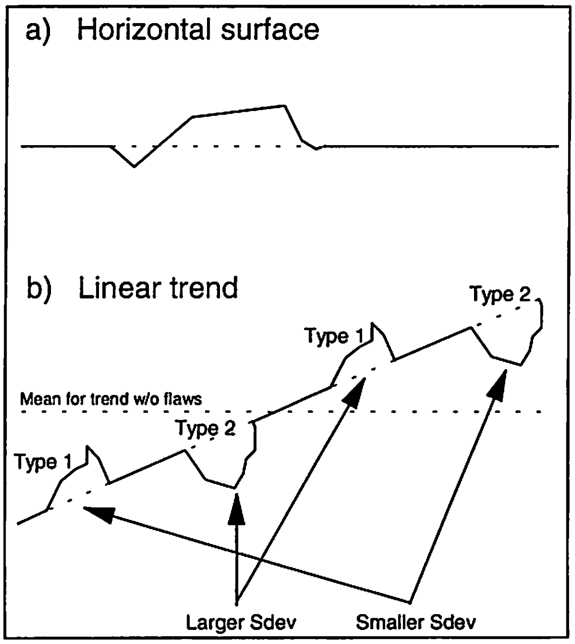


Fig 3.7
Comparison of different interpretations of the standard deviation. For horizontal surfaces (a) flaws will trigger large values. If there is a trend as in (b) the same type of flaws can have different effects depending on their locations.

and non-flaws if the trend is stable and the flaw significant enough. In areas with flaws, the standard deviation would be either larger or smaller depending on the nature and location of the flaw (Fig 3.7.).

3.8.2.3. Skewness

Skewness (eq.3.2) is the third central moment and is a measure of how symmetric the histogram is about the mean intensity value. Mathematically it is the mean cube deviation from the mean where the deviation is measured in standard deviations.

For a perfectly symmetric distribution, the skewness is zero. The skewness can obviously also take both negative and positive values where negative values in general means that the histogram is heavier on the lower intensity range, and positive values means that the histogram is heavier in the higher intensity range. The qualifier "in general" is used because the skewness is very sensitive to outliers. The skewness can be negative and still have the centroid above the mean if there are some sufficiently extreme outliers in the lower intensity values (see Fig 3.8.). Interpreting the actual skewness of a distribution based on the statistical skewness defined in eq. 3.2. can therefore be dangerous. However, in flaw detection we are not concerned about finding a perfect measure of skewness, but rather a descriptor that can distinguish between flaws and non-flaws. In certain instances, the outliers are exactly the characteristics that enable us to do that, and then the statistical skewness is useful.

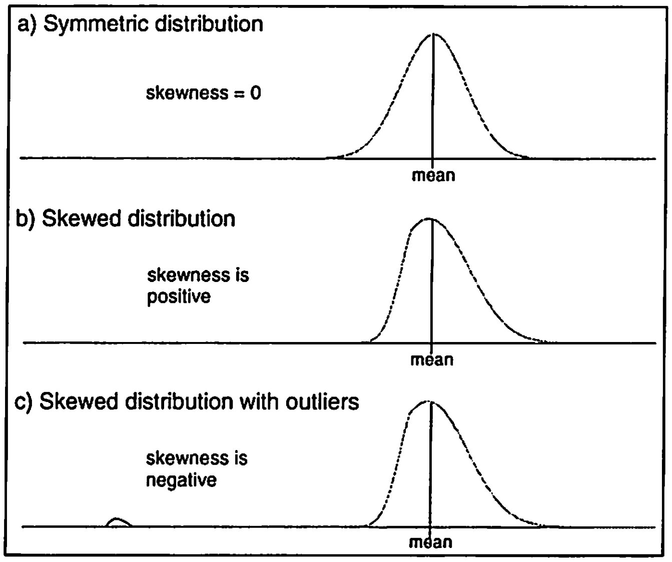


Fig 3.8
Properties of statistical skewness

3.8.2.4. Kurtosis

Kurtosis (see eq. 3.2) is the fourth central moment and can be used as a measure of relative flatness, or alternatively peakiness, of the intensity distribution. Kurtosis is like skewness normalized with the standard deviation and is of this reason also sensitive to outliers -- in fact even more so since the deviation from the mean is raised to the power of four.

If considering only unimodal distributions, the kurtosis increases with peakiness. As seen from Fig. 3.9. uniform, gaussian, and laplacian distributions have kurtosis of 1.8, 3, and 6 respectively [Alpesh,1994]. In general distributions are neither unimodal nor symmetric. Then the kurtosis can be interpreted as a measure of busyness in the image that suppresses small deviations and weights larger.

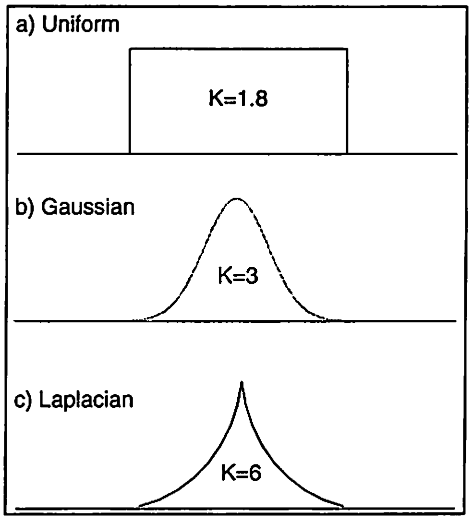


Fig 3.9
Values of kurtosis for some distributions showing that kurtosis can be used as a measure of flatness or peakiness.

3.8.2.5. Energy

Energy as defined in eq. 3.2 is a type of uniformity measure. This quantity is the smallest when all the probabilities are equal i.e. when the distribution is uniform [Gonzales, 1991]. Combined with mean, and variance, this can be a useful feature for detecting crack-like flaws.

3.8.2.6. Entropy

Entropy as defined in eq. 3.2 is also a sort of busyness feature. This feature is smallest for uniform distributions [Gonzales, 1991] and increases as the image becomes busier.

Entropy weight frequencies differently as shown in Fig 3.10 a). From the graph we can see that maximum weight of about 0.5 is given to probabilities slightly less than 0.4. This does of course not suggest that uniform distributions with probabilities slightly less than 0.4 will give the largest entropy values, but rather that distributions where many intensity values have small probabilities will be favored. This is because the number of entropy terms in such distributions increases faster than the contribution of each term is reduced. This can be seen in Fig 3.10 b) which graphs entropies of different uniform distributions. The entropy in uniform distributions is equal to the log_2 of the constant probability since the number of terms equals one over $P(b)$.

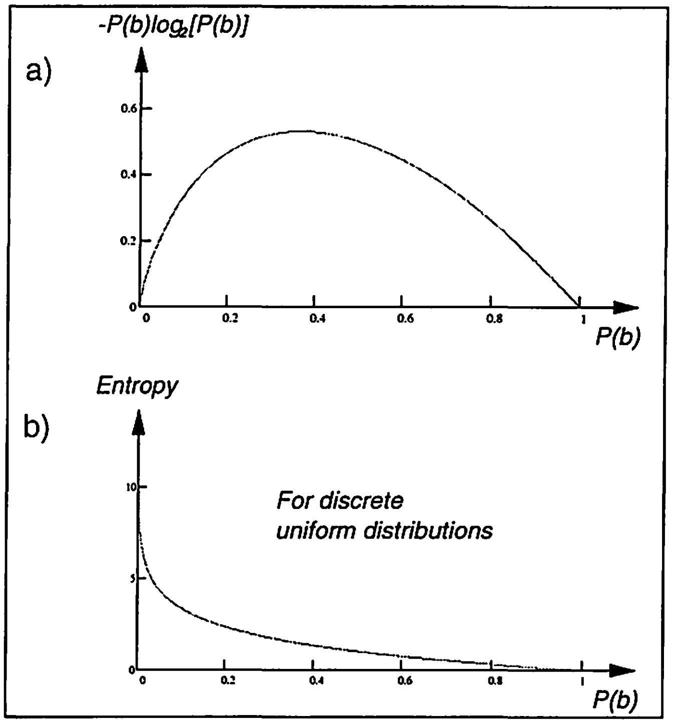


Fig 3.10
Illustration shows in a) entropy weighting of different probabilities and b) entropies of different uniform distributions.

Another interesting characteristic of entropy is that the minimum average bits required to represent a signal is bounded downward according to Shannon by its entropy as defined in eq. 3.2 [Gonzales,1991]. This result would also enable us to deduce that images having many small intensities with small probabilities give large entropies. Such images are of course a lot more complicated than rather uniform images with few intensities and must be represented with a larger number of bits.

3.8.3. Second Order Histogram Features

Second order histogram features are as the name says second order statistics. "Second order" means that two pixels are considered at a time and that the distribution approximated is the second order probability density. These two pixels are separated with a distance *r* and an angle θ as illustrated in Fig 3.11.

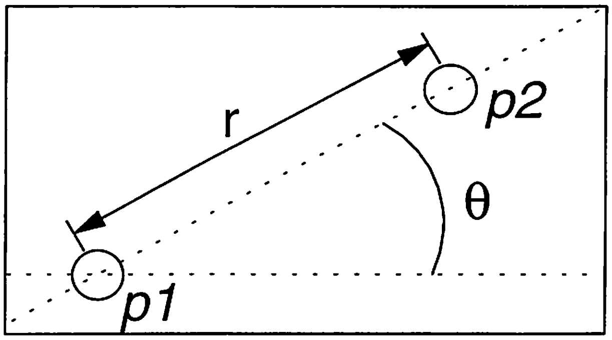


Fig 3.11
Relation between two pixels a and b used in second order histogram analyses

Since second order histogram features are calculated by considering two spatially separated pixels, they will contain spatial information. This is a great advantages over first order histogram features. By considering how the correlation between the two pixels *p1* and *p2* (as defined in Fig 3.11) varies, important discriminatory texture information can be extracted. Jules [1973] found found first order and second order statistics are sufficient in texture discrimination. Textures that have pairwise similar first order and second order distributions cannot in general be discriminated by the naked eye. Some counterexamples of this claim have been found, but for most cases Jules' conclusion is valid [Pratt,1991].

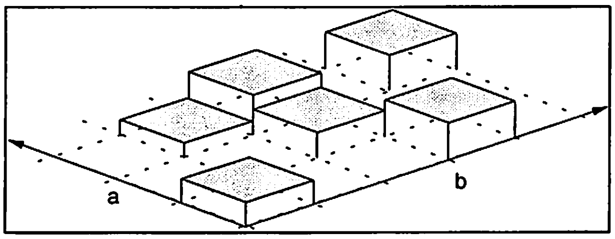


Fig 3.12
Illustration of a two dimensional histogram, often called a co-occurrence matrix.

The histogram created by accumulating frequencies of intensity pairs is two dimensional and is often referred to as the co-occurrence matrix. Fig 3.12 shows such a matrix where the heights of the columns indicate the probability of different gray scale pairs. The extracted second order features are measures of the energy spread of the diagonal of this matrix.

The co-occurrence matrix is a discrete approximation to the second order pdf:

$$P(a,b) \approx \frac{N(a,b)}{M} \qquad (3.3)$$

where *a* and *b* are gray scale values of *p1* and *p2* respectively, *N(a,b)* is the number of occurrences of the intensity pair *a* and *b*, and *M* is the number of pixels in the feature support. Often no discrimination is made between the cases [*p1=a*, *p2=b*] and [*p1=b*, *p2=a*], and then the co-occurrence matrix becomes symmetric about the diagonal.

I have implemented six different second order histogram features in Sherlock. These are *autocorrelation, covariance, inertia, absolute value, inverse difference,* and *second order energy.* Mathematically they are defined as follows:

$$\begin{aligned}
&\text{Autocorrelation,} && S_A = \sum_a \sum_b abP(a,b) \\
&\text{Covariance,} && S_C = \sum_a \sum_b (a-\bar{a})(b-\bar{b})P(a,b) \\
&\text{Inertia,} && S_I = \sum_a \sum_b (a-b)^2 P(a,b) \\
&\text{Absolute Diff.} && S_V = \sum_a \sum_b |a-b| P(a,b) \\
&\text{Inverse Diff.} && S_B = \sum_a \sum_b \frac{P(a,b)}{1+(a-b)^2} \\
&\text{Energy,} && S_G = \sum_a \sum_b P^2(a,b)
\end{aligned} \qquad (3.4)$$

3.8.3.1. Autocorrelation

Autocorrelation is the mean of the product between the intensity pair (a,b). This defines a two dimensional surface spanned by *a* and *b* with maximum along the diagonal (Fig. 3.13). Since the autocorrelation also increases with larger values of *a* and *b*, it can be interpreted as a blend of a correlation and energy information.

3.8.3.2. Covariance

Covariance is a correlation measure between the two pixels. It is a scaled version of the well-known correlation coefficient and is therefore large positive for strong positive correlation and large negative for strong negative correlation. If the two pixels are uncorrelated, the covariance equals zero.

3.8.3.3. Inertia

Inertia (Fig 3.14) is a physical interpretation of the mean square difference between *a* and *b*. In this interpretation *(a-b)* and *P(a,b)* are analogous to distance and mass respectively.

Since inertia is a measure of how much *a* and *b* differ raised to the power of two, small intensity fluctations will be suppressed while large intensity differences will be favored. This makes inertia an excellent feature for detection of edges. Edges are often very important flaw descriptors which explains why inertia is a very useful feature in flaw detection.

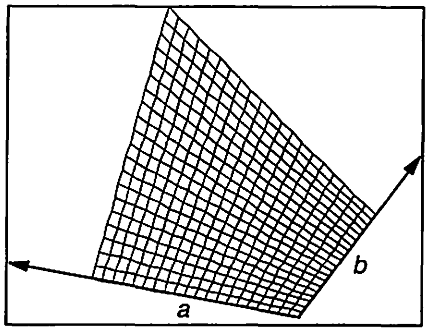


Fig 3.13
Illustration shows how autocorrelation are largest on the diagonal while increasing with a and b.

3.8.3.4. Absolute Difference

Absolute difference is the mean intensity difference between the two pixels *p1* and *p2*. For this reason it is also a useful edge detector. However, the difference is not raised to the power of two and does therefore not numerically separate relative uniform regions from edgy reasons as well as inertia. Absolute difference is of same reason less sensitive to outliers.

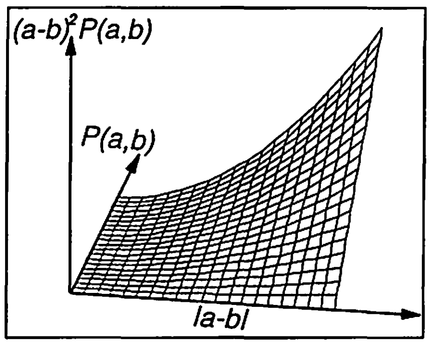


Fig. 3.14
Inertia increases linearly with P(b) and quadratic with (a-b).

Since both absolute difference and inertia provides edge information, they are related. Inertia is related to absolute difference as image energy is related to image mean. Note here that the image energy is not energy as defined in eq. 3.2 which is energy of the probability, but energy of the image intensity as defined as:

$$\textit{Energy of image:} \qquad E = \sum_b b^2 P(b) \qquad (3.5)$$

3.8.3.5. Inverse Difference

Inverse difference (eq. 3.4) is also a measure of how *p1* and *p2* differ. The inverse difference has large values for uniform surfaces, but as soon as *p1* and *p2* start to differ the value diminish quickly.

Inverse difference is the third and last features that measures the difference in intensity values between two and two pixels. The two other are inertia and absolute difference. A valid question is then what information the inverse difference is adding that the other two features don't. In fact, one could ask why we would need three features that measures the intensity difference between the two pixels at all; wouldn't one be sufficient?

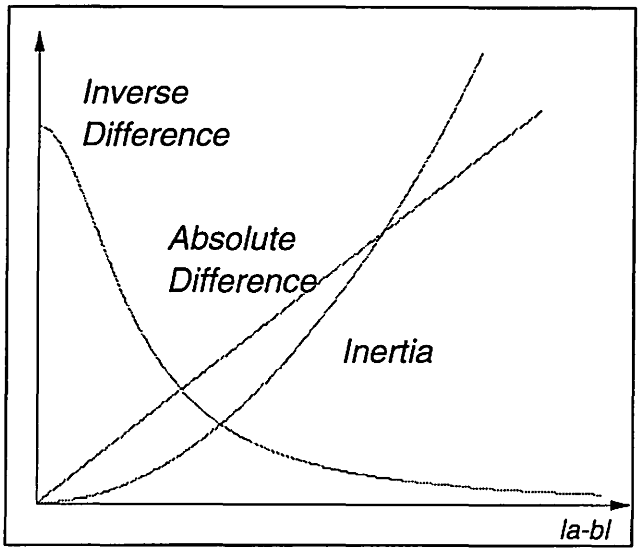


Fig 3.15
A graph illustrating weighting of intensity differences between pixel 1 and pixel 2 for the three second order difference features.

By looking at the graph in Fig 3.16 one can see how these three features differs and complement each other. These features discriminatory capacity over a particular interval range is determined by their dynamic range in this interval relative to its total dynamic range. Comparing a feature's discriminatory capacity for different regions can therefore effectively be done by comparing its first derivatives. By doing so, we can see that inverse difference obviously has a large discriminatory capability for small intensity fluctuations while having a poor resolution for medium and large intensity differences. This feature is therefore very suitable for distinguishing between surfaces that are quite uniform. Inertia on the other hand has a great discriminatory capability for different kinds of edginess because of its exponential growth. However, for the uniform-like surfaces inverse difference could separate, inertia would do a poor job. The most general of the three is the absolute difference which is a linear measure of how the two pixels differ. The absolute differences are not as good at separating uniform surfaces or distinguishing edges as inverse difference and inertia respectively, but with a suitable classifier it could do a decent all-round job.

3.8.3.6. (Second order) Energy

Second order energy as defined in eq. 3.4 is as its first order counter part a probability energy and is a measure of uniformity. The more uniform the surface is, the higher the feature value. This can be understood by using the same argument as in discussion of inertia, this feature has best discriminatory capability in cases where the probability is large. In the lower probability region this feature has a poor resolution. One can therefore conclude that this feature can discriminate between uniform-like surfaces. Outliers (small probabilities) will have a small influence on this feature which means that it will do a poor job in detecting sharp edges and noise-spikes.

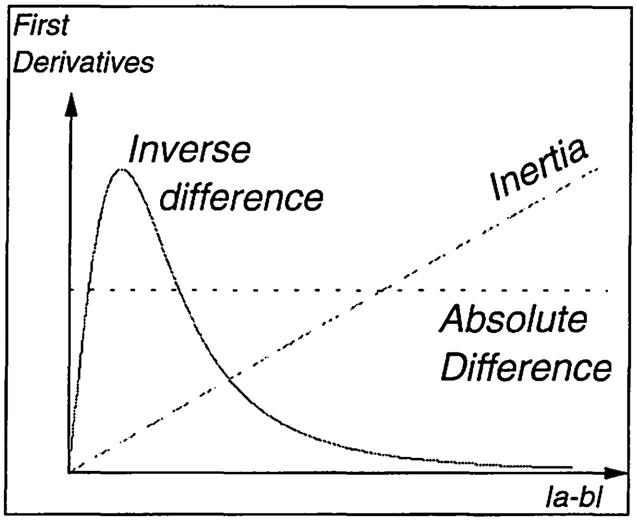


Fig 3.16
Graph of first order derivatives scaled independently. These graphs shows the features discriminatory capacity distributed over the intensity difference range.

3.8.4. Focused Second Order Difference Features

This is a set of features suggested by the author that can be useful in discriminating between similar edginess over the whole intensity difference range from uniform to extremely peaky.

The idea is to generalize and improve on the strengths of the traditional second order histogram features *inertia*, *absolute difference*, and *inverse difference.* As discussed earlier, all three of these are useful intensity difference measures -- inertia for high differences, inverse difference for low differences, and absolute differences for an all round useful descriptor (se Fig 3.16.).

An apparent weakness these features have is that none of them particularly addresses information in the midrange intensity difference interval. For detection problems where high resolution is required in this interval these features are pushing more of the discriminatory responsibility over to the classifier.

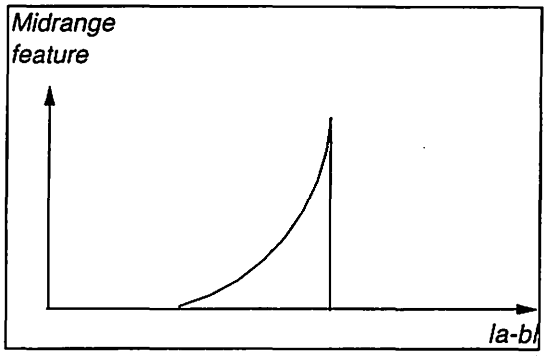


Fig 3.17
A feature useful for discriminating between edges in the midrange intensity difference interval.

One improvement would therefore be to tailor a feature for this region. Such a feature could for example look something like the feature illustrated in Fig 3.17.

Two important characteristic of this midrange feature are worth noting. First of all this feature is zero outside the midrange interval. This means that it only focuses on what it was designed for, the midrange intensity differences, without getting disturbed by other irrelevant information. Secondly, the feature is monotonically increasing in the interval of interest. This means that a particular feature response can be tracked back to a particular intensity difference, and this one-to-one mapping is often useful in the classification process. The shape of the mapping function should be chosen such that the classes are separated as far a part as possible.

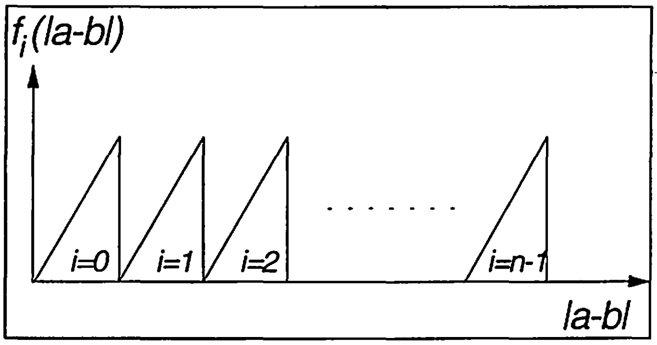


Fig 3.18
Linear focused difference features.

The focused difference features are a generalization of the idea behind the midrange feature. By using a number of such features scattered over the whole or most of the intensity difference range, edge discrimination can be done with high precision for a wide variety of edges. An example of linear focues difference features is shown in Fig. 3.18

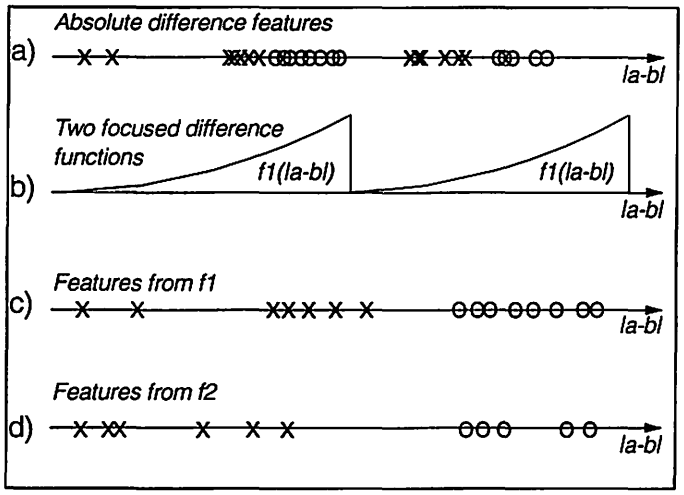


Fig 3.19
Simplifying a multi-modal problem by using two focused difference features. **X** *and* **O** *denotes features from class1 and class2 respectively.*

The advantages focused difference features have over traditional difference features is that they can be tailored to the particular problem at hand. First of all they can be focused on the particular interval of interest, and thereby greatly increase edge resolution. In addition, they can reduce multi-modal problems to unimodal problems. This can be extremely useful in the classification process. Consider the original absolute difference features in Fig 3.19a. Difficulties are here encountered because the two classes, labeled X and O, are each multimodal and are clustered in two or more clusters. By choosing appropriate difference features, for example those shown in Fig 3.19 b), the problem is reduced to a unimodal problem as seen in Fig 3.19 c) and d).

Focused difference features can of course be used in combination with other features as well. The local mean would for example be a good complementing feature since focused difference features contain very little information about the general intensity trends in the image.

3.8.5. Cosine Transform Features

The discrete 2-D cosine transform extracts spectral information and is widely used in image compression because of its good energy compaction property and computational efficiency. The author has found cosine transform features most suitable for detection of edges and determining orientation in textures. The cosine features can also do a good job in suppressing simple trends.

Mathematically, the discrete 2-D cosine transform pair is defined as:

$$\begin{aligned} C(u,v) &= \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \\ f(x,y) &= \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} C(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]. \end{aligned} \tag{3.6}$$

where $\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{N}} & \text{for u } = 0 \\ \sqrt{\dfrac{2}{N}} & \text{otherwise} \end{cases}$

where N x N is the dimension of the image $f(x,y)$.

To better understand the information extracted by the discrete 2-D cosine transform, it is useful to consider the cosine transform as an image expansion. Well-known examples of 1-D expansions are Taylor expansion and Fourier series. What these 1-D expansions are doing is emulating the original signal using a set of basis functions. The 2-D cosine transform is doing exactly the same in two dimensions. Its basis images are defined by:

$$B^C_{u,v}(x,y) = \cos\left[\frac{(2x+1)u\pi}{2N}\right]\cdot\cos\left[\frac{(2y+1)v\pi}{2N}\right] \tag{3.7}$$

The total number of basis images equals to N^2 where N as before is the image dimension in each direction. Basis functions for $N=4$ are illustrated in Fig 3.20 using dark to signify low values and light shades for high pixel values. Combining these basis images, the original image can be completely represented.

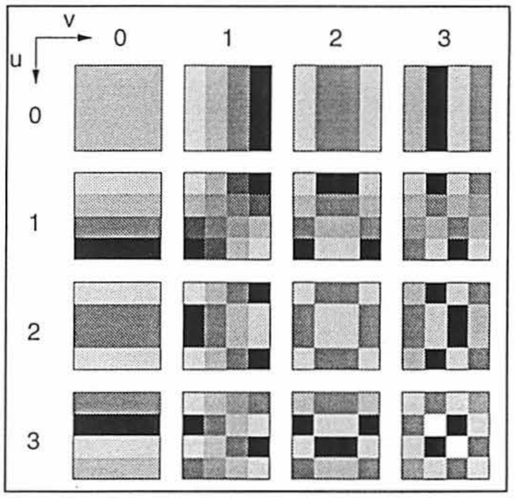


Fig 3.20
Discrete 2-D cosine transform basis functions for N=4

For classification purposes, only the 9 first coefficients are used. For a 4x4 image this corresponds to the individual weighting of the 9 upper-left basis images in Fig 3.20. These coefficients represent the lower frequencies which is where most of the information is. By choosing these coefficients as features the information loss is minimized. Due to the excellent energy compression properties of the cosine transform, the actual information loss using these features is usually small.

By studying the discrete cosine basis functions, it is easy to understand how the cosine transform can provide useful features in image analyses. Since the basis function are representing increasingly higher spatial frequencies, the cosine coefficients can be used to discriminate between different types of edginess. By observing that horizontal and vertical frequencies are increasing independently it is also apparent that the cosine transform can produce good descriptors for determining texture orientation. For example. the basis image corresponding to u=0 and v=2 will be weighted heavily for horizontal edges, but will completely ignore any vertical trends and edges. For determining orientations that are not strictly horizontal or vertical one would rely on coefficients that are associated with basis images that contain the appropriate combination of horizontal and vertical frequencies.

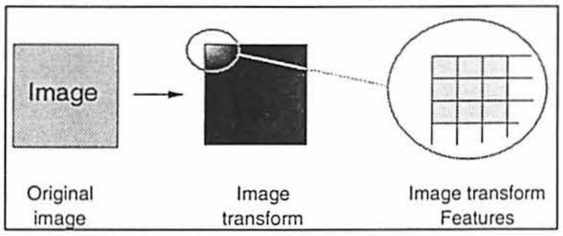


Fig 3.21
Illustration of extracting transform features (shaded gray).

3.8.6. Hadamard and Walsh Transform Features

Other image transforms can of course also be used to extract geometric and spectral information, and in Sherlock two other well-known transforms are implemented: the Hadamard transform and the Walsh transform. These transforms are computationally extremely efficient. They don't have as good energy compaction properties as the cosine transform, but the information they extract is similar to that of the cosine transform.

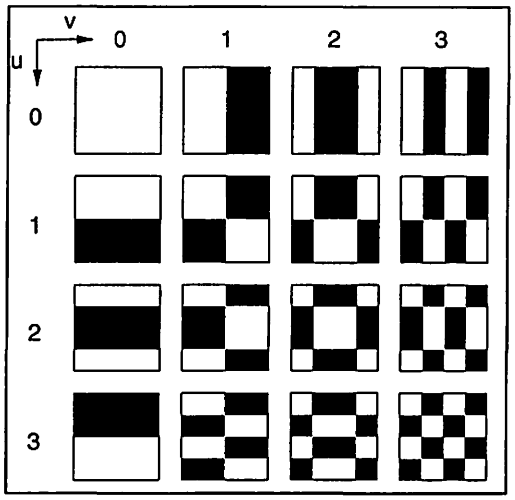


Fig 3.22
Ordered Hadamard basis images for N=4

The Hadamard and the Walsh transforms have binary basis images which structurally resembles the basis images of the cosine transform. This explains why these three transforms extract similar type of information. Fig 3.22 shows the ordered Hadamard basis images for N=4. For this case, the Walsh transform has exactly the same basis images ordered differently.

Mathematically the Hadamard and the Walsh transforms are defined as:

$$T(u,v) = \frac{1}{N}\sum_x \sum_y f(x,y) B_{u,v}(x,y)$$
$$f(x,y) = \frac{1}{N}\sum_u \sum_v T(u,v) B_{u,v}(x,y) \qquad (3.8)$$

where $T(u,v)$ is the transform, $f(x,y)$ is the original image and $B_{u,v}(x,y)$ is the set of basis images. The basis images for the ordered Hadamard and Walsh transform are respectively defined as:

$$B^{H}_{u,v}(x,y) = (-1)^{\sum_{i=0}^{n-1}\left[b_i(x)p_i(u)+b_i(y)p_i(v)\right]} \qquad (3.9)$$

$$B^{W}_{u,v}(x,y) = (-1)^{\prod_{i=0}^{n-1}\left[b_i(x)p_{n-1-i}(u)+b_i(y)p_{n-1-i}(v)\right]} \qquad (3.10)$$

where $b_k(a)$ is the kth bit of the binary representation of a, summation is a modulo 2 summation, and $p_k(a) = b_{n-k}(a) + b_{n-k-1}(a)$.

Because of the better energy compaction, the cosine transform would in general be the preferred choice over the Hadamard and Walsh transform. However, if computation time is a consideration, either of the two latter transforms would be a serious alternative. Lastly, it should be mentioned that all of these three transforms can be implemented in fast algorithms analogous to the FFT.

3.9. Implemented Classifiers

3.9.1. Introduction

This chapter is discussing the implemented classifiers in details. A number of classifiers was implemented because it is important for a general purpose package as Sherlock to have a good selection of different classifiers from which to chose. A particular classifier will have its strengths, but will not be superior for every classification problem. Classifiers differ in how they determine decision boundaries, in training requirements, and classification speed. The optimal classifier doesn't exist, but an ideal classifier for a given problem will fit the chosen feature set while not being excessively complex and time consuming. Before discussing the classifiers, feature normalization will be addressed. This is an important area, particular for classifiers using distances in feature space as a measure of similarity of two feature vectors.

3.9.2. Feature Normalization

Feature normalization is often a crucial link between the extracted feature vectors and the chosen classifier. Feature normalization can, if ignored or done poorly, completely ruin the classification result. The type of feature normalization discussed here is individual feature normalization for calculation of distances between two feature vectors in feature space using the Euclidean distance function.

The Euclidean distance *d* between two *n* dimensional vector *A* and *B* is defined as

$$d(A,B)=\sqrt{\sum_{i=0}^{n-1}(a_i-b_i)^2} \qquad (3.11)$$

Such a distance measure would weight features differently depending on their numerical magnitude. Consider a feature vector containing the features *weight* and *height* in pounds and feet, respectively. The distance between two such features in feature space would be largely dominated of the difference in weight. This occurs because weight can vary by tens of pounds from person to person while the height only fluctuates with a foot or so. This large numerical unbalance would bias the classification by effectively suppressing the information content in the *height* and only relying on the *weight* feature. Such a biasing is of course completely unjustifiable because the information content of a feature is not related to its numerical deviation. Finding distances between features in the feature space must therefore be normalized in order to weight individual features equally. If a particular weighting is desired, the weighting should be proportional to the feature's discriminatory capacity and nothing else.

Four different weighting strategies are available in Sherlock. Two of these normalizes the features itself and two are including weights in the distance function.

A common normalization method is to scale the features individually. In Sherlock one has the opportunity to scale the features from 0 to 1, or to scale the features by subtracting the feature mean and dividing by the feature standard deviation. These are common techniques that both ensure that distances between feature vectors depend quite equally on the individual features.

The method of introducing weights in the distance function has the advantages that the original features are not changed. This can be useful in feature analysis. By normalizing the features independently, some of the relational information between different features is lost. Two types of weights are available in Sherlock. The individual feature distance can either be normalized with respect to standard deviation or the maximum deviation from the mean (suggested by the author). Of the two, the maximum deviation is found to be the most sensitive to outliers. This can be both positive or negative depending on whether outliers are important or not for the classification. Which weighting that does the best job depends from application to application.

3.9.3. K-mean

The K-mean (see Table 3.1) is a commonly used clustering algorithm. It is an unsupervised classifier that doesn't need any training, but organizes the features in K spherical clusters using a distance function of choice. The algorithm independently minimizes the squared within cluster distance which is defined as:

$$J_j = \sum_{X \in S_j(k)} \left\| X - Z_j(k+1) \right\|^2 \qquad (3.12)$$

The K-Mean Algorithm
1) Input number of clusters
2) Input initial cluster centers
3) Assign all feature vectors to closest cluster center.
4) Assign new cluster means as new cluster center.
5) IF cluster centers have changed THEN go to 3)
6) K-mean has converged.

Table 3.1
The K-mean algorithm

where X denotes feature vectors, Z_j(k+1) is the new cluster center for cluster j, $j=1,2,...,K$, and $S_j(k)$ are the members of cluster j after iteration k.

Initially the K-mean algorithm requires the number of desired clusters (classes) in the output image, and initial guesses of where these clusters are located in feature space. Since the latter often is pretty hard to know ahead of time, the first K feature vectors are generally chosen as initial cluster centers. All feature vectors are then assigned to the closest cluster center. When all feature vectors are assigned, new cluster centers are calculated to minimize the objective function in eq. 3.12. It is easy to show that this cluster center simply is the mean of the cluster vectors:

The first derivative of the objective function in eq. 3.12 is

$$\frac{\partial J_j}{\partial Z_j(k+1)} = (-2) \cdot \sum_{X \in S_j(k)} \left\| X - Z_j(k+1) \right\|$$

which should be set to zero to find the optimal cluster center:

$$\sum_{X \in S_j(k)} \left\| X - Z_j(k+1) \right\| = 0$$

which gives:

$$\sum_{X \in S_j(k)} X = N_j \cdot Z_j(k+1)$$

where N_j is the number of feature vectors in $S_j(k)$.
The optimal cluster center is then:

$$Z_j(k+1) = \frac{1}{N_j} \sum_{X \in S_j(k)} X$$

which is the cluster mean.

The disadvantages of the K-mean is that it is relative slow, particularly for large data sets with high dimensional feature vectors. Another weakness is that the K-mean doesn't necessarily converge to a global minimum since it is a gradient descent algorithm. Therefore it should be run several times with different initial cluster centers to increase the chances of finding the global optimal classification. The requirement that the desired number of classes has to be known a head of time is also in general considered a weakness.

The advantages of the K-mean clustering algorithm is that it doesn't need any training data, but groups data that are clustered together and separates those which are far apart. K-mean can for this reason also be a useful data analysis tool. A lot can be learned about the data by running the K-mean algorithm with various numbers of desired clusters.

3.9.4. Fuzzy-C

The Fuzzy-C clustering algorithm (see Table 3.2) is generally considered a more powerful clustering algorithm than the K-mean. Like the K-mean, the Fuzzy-C requires the user to input the number of clusters. Assigning the feature vectors to the clusters is then based on a fuzzy principle. Initially each feature vector is given a random membership value to each of the clusters restricted by the constraint that a feature vector's memberships should add up to 1. A feature vector's membership value is based on the feature space distance between the vector to the relevant cluster center with respect to the distances to all the other cluster centers. Normalization of the features is therefore necessary. Once all the membership values are found, new cluster centers are calculated by weighting each feature vector according to its membership value.

The Fuzzy-C minimizes the objective function:

$$J_m(\mathbf{U},\mathbf{V}) = \sum_{k=1}^{N}\sum_{i=1}^{C} u_{ik}^{m} d_{ik}^{2} \qquad (3.13)$$

where m is a fuzziness parameter, **U** denotes all the memberships values, **V** denotes all the cluster centers, k indexes feature vectors, and i indexes clusters. The distance from vector k to cluster center i is denoted by d_{ik} and the membership value of vector k belonging to cluster i is denoted u_{ik}. The objective function in eq. 3.13 can be interpreted as the total moment about all the cluster centers using all the feature vectors in calculating the moment about each cluster center. In this calculation the feature vectors are weighted by the membership values raised to the power of m.

The Fuzzy-C Algorithm
1) Input number of clusters
2) Randomly initialize membership values. A feature vectors membership should sum to 1.
3) Assign membership values according to eq. 3.15
4) Calculate new cluster centers according to eq. 3.14
5) If cluster centers have changed or difference < tolerance THEN go to 3)
6) Fuzzy-C has converged.

Table 3.2
The Fuzzy-C algorithm

The cluster centers that minimize the objective function in eq 3.13. are found by the same technique as used when finding the optimal cluster center for the K-mean. The result is:

$$\textit{Cluster center:} \qquad \mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^{m}\mathbf{x}_k}{\sum_{k=1}^{n} u_{ik}^{m}} \qquad (3.14)$$

where i and k as before indexes clusters and feature vectors respectively. The optimal cluster center is recognized as being the centroid. This makes sense when comparing with the result from the analyses of K-mean. The K-mean weights all vectors equally. If the memberships in eq. 3.14 were all equal to 1, the centroid would be reduced to the simple mean.

Optimal membership functions are found by applying Lagrange multipliers to the variables u_{ik} [Bezdek, 1982]. The result is:

$$\textit{Fuzzy membership:} \qquad u_{ik} = \left[\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{\frac{2}{m-1}} \right]^{-1} \qquad (3.15)$$

We see that the membership value of a feature vector belonging to a cluster i is a function of the relative distance between the feature vector and the cluster center i with respect to the distances to all the other cluster centers.

The fuzzy index m>1 is obviously an important parameter. This parameter can be interpreted as a fuzziness of membership assignment and should be set proportionally to the uncertainty and noisiness in the acquired data. When $m \rightarrow 1^+$, the Fuzzy-C becomes a hard clustering algorithm emulating the K-mean. This is equivalent to having a high confidence in the accuracy of the data. If m $\rightarrow \infty$ eq. 3.15 becomes 1/c. This means that a feature vector is assigned equally to all the clusters. Since this will be true for any vector and any clusters, all cluster centers will coexist in the simple mean of all the feature vectors. What this means is simply that the data is so noisy that no discrimination can be made. No good method of determining the fuzziness parameter m is developed. For good to reasonable good data, values between 1 and 3 have been found to work well by the author.

The disadvantages of the Fuzzy-C is similar to those of K-mean. Fuzzy-C does not guarantee convergence to an optimal minima, and it needs to be given the desired number of clusters. The Fuzzy-C is in addition slower than K-mean because of its increased complexity, and it also has the problem of determining an appropriate value of the parameter m. Another problem occurs if any of the feature vectors becomes identical to any of the cluster centers. This would cause a division by zero in eq. 3.15. However, such an event is rare due to the high precision in a computer.

An advantages of the Fuzzy-C is that it can be useful for classification of noisy data. By choosing the fuzziness parameter m the classifier is told how reliable the acquired data is. By assigning degrees of membership to all the clusters, noise has a smaller influence for patterns close to cluster boundaries. A hard assigning of such noisy boundary patterns could lead to arbitrarily classification due to the noise.

3.9.5. Nearest Neighbor

The nearest neighbor is a simple intuitive supervised classifier. Given a set of prototypes, unknown feature vectors are assigned to the same class as the closest prototypes. *Closeness* here means any type of similarity measure. The similarity measure in Sherlock is the Euclidean distance.

Two serious weaknesses of the nearest neighbor is illustrated in Fig 3.23. Because the nearest neighbor classifier relies on a hard classification, border vectors can be misclassified due to noise. This is a problem for all hard limiting classifiers, but is more serious for the nearest neighbor classifier because it only consider one prototype in its classification, disregarding the information in the other prototypes.

Another serious problem with the nearest neighbor is its sensitivity to a poorly representative set of prototypes. Fig 3.23 b) shows how one prototype missing from a) completely changes the decision boundary. Feature vector 3 that in a) was comfortably inside the class 1 region is suddenly moved into the class 2 domain. All supervised

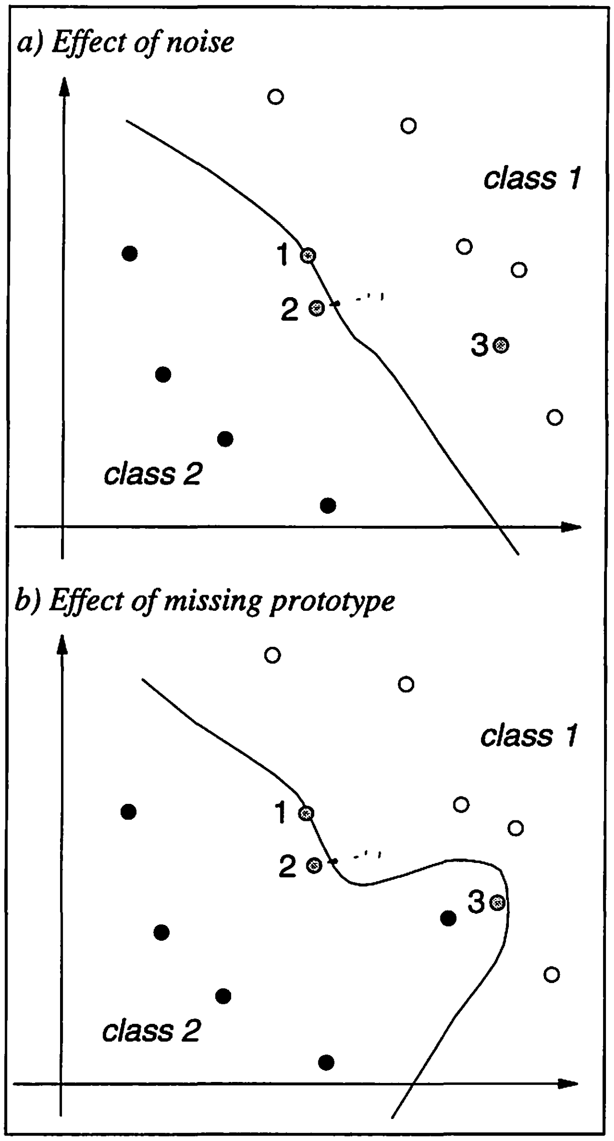


Fig 3.23
Illustration shows hard classification of three border vectors. Vector 2 belongs to class 1, but is classified as class 2 because of noise. Vector 2 is wrongly classified as belonging to class 1 because class 2 missed an important prototype. The importance of the missing prototype is shown in b).

classifiers are sensitive to poor prototypes, but again because the nearest neighbor only relies on one prototype in the classification process, it suffers more serious consequences.

Another serious drawback with the nearest classifier is that it can be quite slow. This is particular true for cases with many prototypes and high dimensionality of the feature vectors.

The advantages of the nearest neighbor classifier is that it is very easy to understand and it is also simple to implement. For well-behaved data and a representative set of prototypes, the nearest classifier can do a good job.

3.9.6. The Single Layer Perceptron

The Single Layer Perceptron network, hereafter called the Perceptron algorithm [Rosenblatt, 1957], is a reward punishment based supervised classifier. It is one of the few neural networks that guarantees convergence if the classes are linearly separable.

The perceptron algorithm optimizes the objective function:

$$J(\mathbf{w},\mathbf{x}) = \frac{1}{2}\left(\left|\mathbf{w}^T\mathbf{x}\right| - \mathbf{w}^T\mathbf{x}\right) \qquad (3.16)$$

where **w** and **x** denote weights and feature vectors, respectively. This function is of course minimum i.e. zero for $\mathbf{w}^T\mathbf{x} \geq 0$.

The derivation of the perceptron algorithm is based on a gradient descent technique. If $\mathbf{w}^T\mathbf{x} < 0$, the weight vector is incremented in the negative direction of the gradient of the objective function in eq. 3.16.

$$\mathbf{w}(k+1) = \mathbf{w}(k) - c\left\{\frac{\partial J(\mathbf{w},\mathbf{x})}{\partial \mathbf{w}}\right\}_{\mathbf{w}=\mathbf{w}(k)} \qquad (3.17)$$

where c is a positive constant which determines the amount of correction. The first derivative of the perceptron objective function is:

$$\frac{\partial J}{d\mathbf{w}} = \frac{1}{2}\left(x \cdot \operatorname{sgn}(\mathbf{w}^T\mathbf{x}) - \mathbf{x}\right) \qquad (3.18)$$

where *sgn*(*x*) is the sign function:

$$\mathrm{sgn}(a) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{if } a < 0 \end{cases} \qquad (3.19)$$

Substituting eq. 3.19 into 3.18. and eq. 3.18. into 3.17 we get the perceptron algorithm for updating of weights:

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) + c\begin{cases} 0 & \text{if } \mathbf{w}_j^T(k)\mathbf{x}(k) > 0 \\ \mathbf{x}(k) & \text{if } \mathbf{w}_j^T(k)\mathbf{x}(k) \le 0 \end{cases} \qquad (3.20)$$

where *j* denotes the class. There are many ways to implement the perceptron algorithm, and in the above implementation one particular weight vector is associated with each class.

An alternative formulation of the perceptron algorithm is called the *reward-punishment principle*. A formulation for a two class problem could be:

$$\begin{aligned} &\textit{if}\ \ \mathbf{x}(k) \in \textit{class}\ 1 \ \ \textit{and}\ \ \mathbf{w}^T(k)\mathbf{x}(k) \le 0 \\ &\quad \mathbf{w}(k+1) = \mathbf{w}(k) + c\mathbf{x}(k) \\ &\textit{ifelse}\ \ \mathbf{x}(k) \in \textit{class}\ 2 \ \ \textit{and}\ \ \mathbf{w}^T(k)\mathbf{x}(k) \ge 0 \qquad (3.21) \\ &\quad \mathbf{w}(k+1) = \mathbf{w}(k) - c\mathbf{x}(k) \\ &\textit{else}\ \ \mathbf{w}(k+1) = \mathbf{w}(k) \end{aligned}$$

In this formulation, two classes are separated with one feature vector. The *punishment-reward principle* is nothing else than modifying the weight vector up if it is too small and down if it is to big.

The decision boundaries of the perceptron algorithm are linear which is easily understood by considering the test criteria in eq. 3.20 or 3.21. If we consider eq. 3.20 which is the most general, one can see that the weight vector is defining a linear decision boundary in feature space dividing the patterns into an in-class-*j* region and an outside-class-*j* region. These decision boundaries are general because the feature vectors are augmented with a constant which means that for example a vector of 6 elements becomes a vector with 7 elements, the 7th element being a constant usually set to 1. The decision boundary is therefore given by

$$\mathbf{w}^T\mathbf{x} = w_0x_0 + w_1x_1 + \cdots + w_{n-1}\cdot 1 = 0 \qquad (3.22)$$

where n is the number of elements in the augmented feature vector. This is clearly a linear decision boundary.

As mentioned before the perceptron algorithm converges if the classes are linearly separable, and for such problems the algorithm is an excellent classifier. Once the perceptron network is trained i.e. the weights are found, the classifier is also extremely fast. For linearly separable classes, the perceptron algorithm is probably one of the best around.

The disadvantages of the perceptron algorithm is of course that if the classes can't be separated by linear decision boundaries, it won't converge and does a real poor job. No good stopping criteria has been found for the nonlinear case [Duda, 1973]. Another disadvantages is that the training i.e. finding the weight vector(s) can be pretty time consuming.

3.9.7. The Fuzzy-Perceptron

The Fuzzy-Perceptron [Keller, 1985] is an extension of the traditional perceptron algorithm to classification problems where the classes are not linearly separable. A successful extension of the powerful perceptron algorithm would be an excellent general purpose classifier.

The motivation of the fuzzy perceptron is that traditional crisp perceptron is not converging, even to a reasonable good solution, when the classes are linearly inseparable because boundary vectors are weighted just as much as vectors in the center of the class region. Boundary vectors are often not very characteristic for the classes they represent and can therefore be considered as outliers. By weighting such outliers less and concentrating on classifying the more important general class vectors, reasonable decision boundaries should be obtained. These decision boundaries would not necessarily classify all the boundary vectors correctly, but the majority of the more typical patterns would be correctly labeled.

The weight updating procedure for a two-class problem is

$$\mathbf{w}(k+1) = \mathbf{w}(k) + |\mu_1(k) - \mu_2(k)|^m c\mathbf{x}(k) \qquad (3.23)$$

where k is iteration, c and m are constants, and $\mu_1(k)$ and $\mu_2(k)$ are memberships of feature vector $\mathbf{x}(k)$ for class 1 and class 2 respectively. These memberships add up to 1. The fuzzy perceptron modifies the weight vector with a weighted version of the current vector. If this vector belongs completely to any of the classes, i.e. that any of the membership functions are 1, the updating becomes crisp. If this pattern has a membership of 0.5, no adjustment is done of the weight because this pattern cannot be placed on either side of the decision boundary.

The two-class membership functions for feature $\mathbf{x}(k)$ belonging to class j are given as follows:

$$\mu_j(k) = 0.5 + \frac{\exp\left(f(d_{\bar{j}} - d_j)/d\right) - \exp(-f)}{2\left(\exp(f) - \exp(-f)\right)} \tag{3.24}$$

$$\mu_{\bar{j}}(k) = 1 - \mu_j(k) \tag{3.25}$$

where $\bar{j}$ is not class j, d_q is the distance from the mean of class q, d is the distance between the two class mean, and f is a parameter controlling how fast the function decreases. A plot of this membership function for different values of f is shown in Fig 3.24. The function is 1 if the pattern is equal to the mean of its class and 0.5 if it is equal to the mean of the other class.

The fuzzy perceptron is like the traditional perceptron algorithm also guaranteed to converge if the classes are linearly separable. It won't converge if the classes are not linearly separable, but in such cases it is easier to define a good stopping criteria. An intuitive criteria would be that the fuzzy perceptron should be stopped when all the patterns having a larger membership than a certain tolerance are correctly classified. Keller suggests the following threshold:

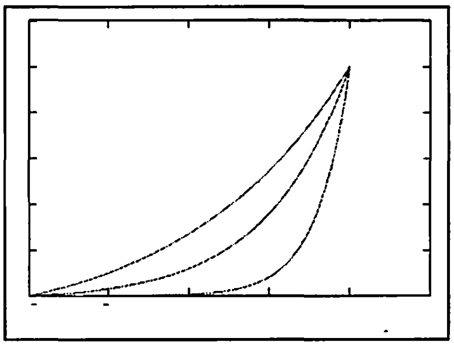


Fig 3.24
Membership functions for different values of f.

$$Th = 0.5 + \beta \tag{3.26}$$

$$\beta = \frac{1 - e^{-f}}{2(e^{f} - e^{-f})} + \varepsilon \tag{3.27}$$

where ε is a constant the user chooses. Equation 3.27 is found by setting membership functions for class 1 and class 2 equal to each other.

The disadvantages of the fuzzy perceptron algorithm suggested by Keller is that the membership functions require that each class is clustered relatively densely in one cluster only. In addition, there are three parameters that needs to be determined: m, f, and ε. Because of the computation of the fuzzy membership functions and the weight correction size, the fuzzy perceptron is also a little bit slower to train.

Once the fuzzy perceptron is trained and three good parameter values are found, the fuzzy perceptron is a powerful classifier for problems with few and unimportant boundary vectors.

3.9.8. Sorting Fuzzy

The sorting fuzzy classifier is a classifier developed by the author and is based on a feature-wise fuzzification of the feature vectors. A vector is given a fuzzy score depending on how similar it is to one of the prototypes i.e. fuzzy rules. Scores are given feature for feature and then combined.

Fuzzy memberships for a fuzzy rule based on a prototype is illustrated in Fig 3.25. In this example there is a total of three features, and the rule is based on the prototype [a,b,c]. The membership functions in this illustration are piece-wise linear, but could be any monotonic function. Each side of the prototype

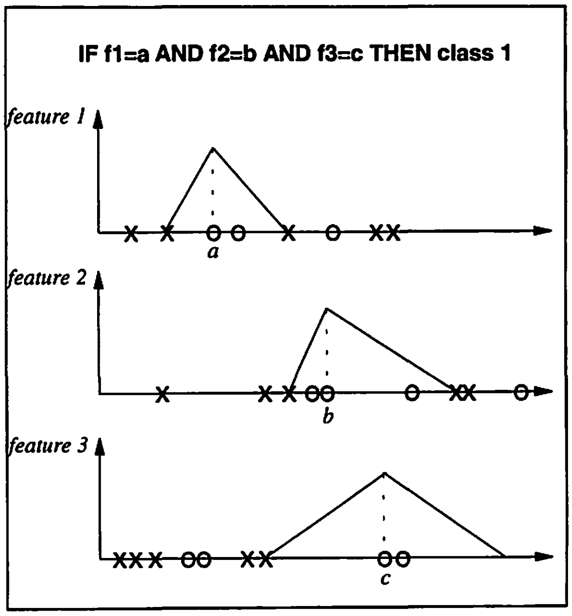


Fig 3.25
Illustration of feature-wise fuzzification in Sorting Fuzzy classifier

feature is fuzzified independently, monotonically decreasing to zero when hitting a prototype feature belonging to another class. In order to enable this, the prototypes have to be sorted feature-wise. If the prototypes are rather limited and important prototypes might be missing, the fuzzification could alternatively decrease to zero when hitting any other prototype feature regardless of class. If one would want to reduce a large number of fuzzy rules that contained a lot of redundancy, one could use trapezoidal instead of triangular membership functions. These membership functions would be constant at 1 between neighboring prototype features.

In the case of the prototype feature being the extreme left or right feature (see feature 3 in Fig 3.25) it is difficult to define a good membership function. In the Fig 3.25. the extreme-side fuzzy function is simply a mirror image of the other side of the prototype. A constant membership function at 1 would be another useful alternative.

When an unknown pattern is tested against one of the fuzzy rules it gets a fuzzy score for each feature. These scores then need to be combined in order to create a vector score. The author has suggested four ways of doing this:

$$S_a = \sum_i s_i \qquad (3.28)$$

$$S_s = \sum_i s_i^2 \qquad (3.29)$$

$$S_f = \sum_i f(s_i) \qquad (3.30)$$

$$S_m = \prod_i s_i \qquad (3.31)$$

where s_j fuzzy score of feature *i*. Of these four methods, the latter is the most strict. Any feature with a small score would have a large impact on the total result. The addition method in eq. 3.28 is not as sensitive to such cases. This method has more of an average information and can in fact be interpreted as a scaled average value. The summed square method in eq. 3.29. is somewhere in between the two mentioned before. The method in eq. 3.30 is a generalization of the two eq. 3.28. and 3.29.

Once the vector score for one fuzzy rule is found, scores for the other fuzzy rules have to be calculated. An easy way to classify the unknown feature vector is to give it the same class as the prototype the fuzzy rule with the highest score was based on.

The major disadvantages of the sorting fuzzy classifier is that it is relatively slow. Unknown vectors have to be tested against all fuzzy rules, and that can take time if the number of rules are large. In most cases, however, it is possible to reduce the number of rules without significantly reducing the performance.

The good properties of the sorting fuzzy classifier are that it solves detection problems with separated clusters within each class, that it trains extremely fast, and is easy to understand. The Sorting fuzzy classifier is a good all-round classifier when speed is not too much of a concern.

4. INSPECTION STRATEGIES

Once the system is trained it runs by itself without human supervision. How well the system performs is of course highly dependent on how well the training was done. The four decisions that has to be made in the training processes are listed below:

1) Choose feature support and grid size
2) Choose feature set
3) Choose training data
4) Choose classifier

Of these 2) and 3) are very important and 1) and 4) are less crucial.

4.1. Choosing Feature Support and Grid Size

The choice of these parameters will influence the classification result, but within reasonable limits, the values of these parameters are of less importance.

The *feature support* must be small enough to give local information, but should also be large enough to provide enough data to justify using for example statistics if the chosen features are statistical. If one attempts to discriminate between textures, the feature support must be large enough that a representative texture patch will fit inside. In general, the author has found a window size of 10x10 to be a good all-round size.

The *grid size* is a parameter whose sole purpose is to adjust the resolution in the output image. If this parameter is chosen as 4 the output image will have a resolution of 4x4 pixels. The lower this number is, the more computation is needed. For most problems, a resolution of 1x1 pixel is not required so the inspection speed can be increased by allowing a lower resolution. By adjusting the *grid size* parameter the user can optimize the resolution/computation ratio for the particular problem at hand.

An example of the effects of changing the *feature support* and *grid size* is shown in figure 4.1. As seen from this figure, increasing the *feature support* has an averaging effect while reducing the *grid size* increases the resolution, revealing more detailed information.

4.2. Choosing Features

This decision is one of the most important in the training process. Although a good classifier theoretically could discriminate between classes that are poorly separated in feature space, the general rule is that garbage in gives garbage out. For a successful classification, it is important that the input information is as good as possible. Fortunately, there are good ways of determining the discriminatory capacity of a feature. For this purpose Sherlock is equipped with the command *View Feature Map.* This command was used to generate the images used in Fig 4.1.

A feature map does provide a lot of information about a feature's ability to distinguish between flaws and non-flaws. However, feature maps give only information about individual features and don't tell how two or more features cooperatively would perform.

In the following sub-sections, examples of images and their respective feature maps are discussed. The idea of evaluating the usefulness of a feature by looking at it's feature map could easily be understood from one example alone, but several examples are provided to give the reader a better feel for what information the different features are extracting.

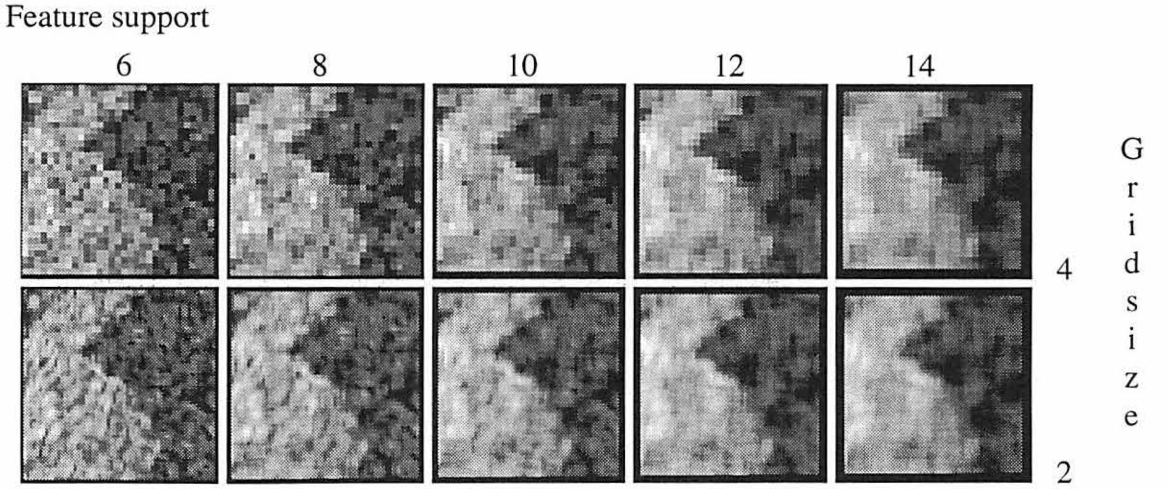


Fig 4.1
Illustration of the effects of feature support and grid size. The images shown are feature maps of the standard deviation of an image of two Brodatz textures.

4.2.1. A Simulated Texture Image

The first example is an image with four simulated texture regions (Fig. 4.2). The human eye can easily see two circles in the middle, but partly hidden by the largest circle is also a rectangle on the left. The fourth texture is of course the background.

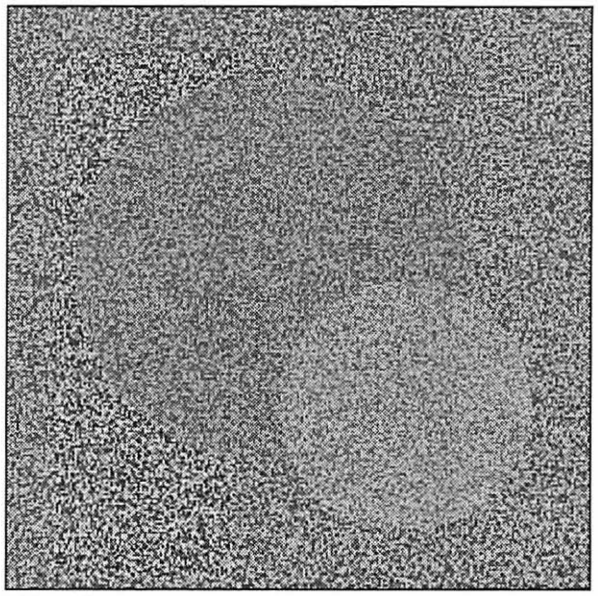


Fig 4.2
Simulated texture image.

Feature maps of three different feature sets are shown in Fig 4.3-5. By considering these feature maps one can see that the hardest region to identify is the rectangle. Several features can distinguish between the background and the combined circle area, and other features can separate the small circle from the rest. But, only one feature, *the standard deviation*, is capable of correctly identifying the rectangle. In fact, this feature is also able to identify the background. Clearly, the *standard deviation* should be chosen as one of the features to classify textures of these type. In addition, it would need a feature capable of separating the two circles. For this task there are many features to choose among.

Of the three feature sets, the first order histogram features performed best while the cosine transform features did a really poor job. The poor performance of the cosine transform features is of course easy to understand since they contain a lot more edge and geometry information than statistical.

The conclusion of the feature map analyses is that two features are necessary in the classification of the textures in Fig 4.2. One of these features must be *the standard deviation* and the other feature could be any feature that correctly separates the two circles (*mean, first order energy, autocorrelation, second order energy,* and *the (0,0) Cosine Transform feature*).

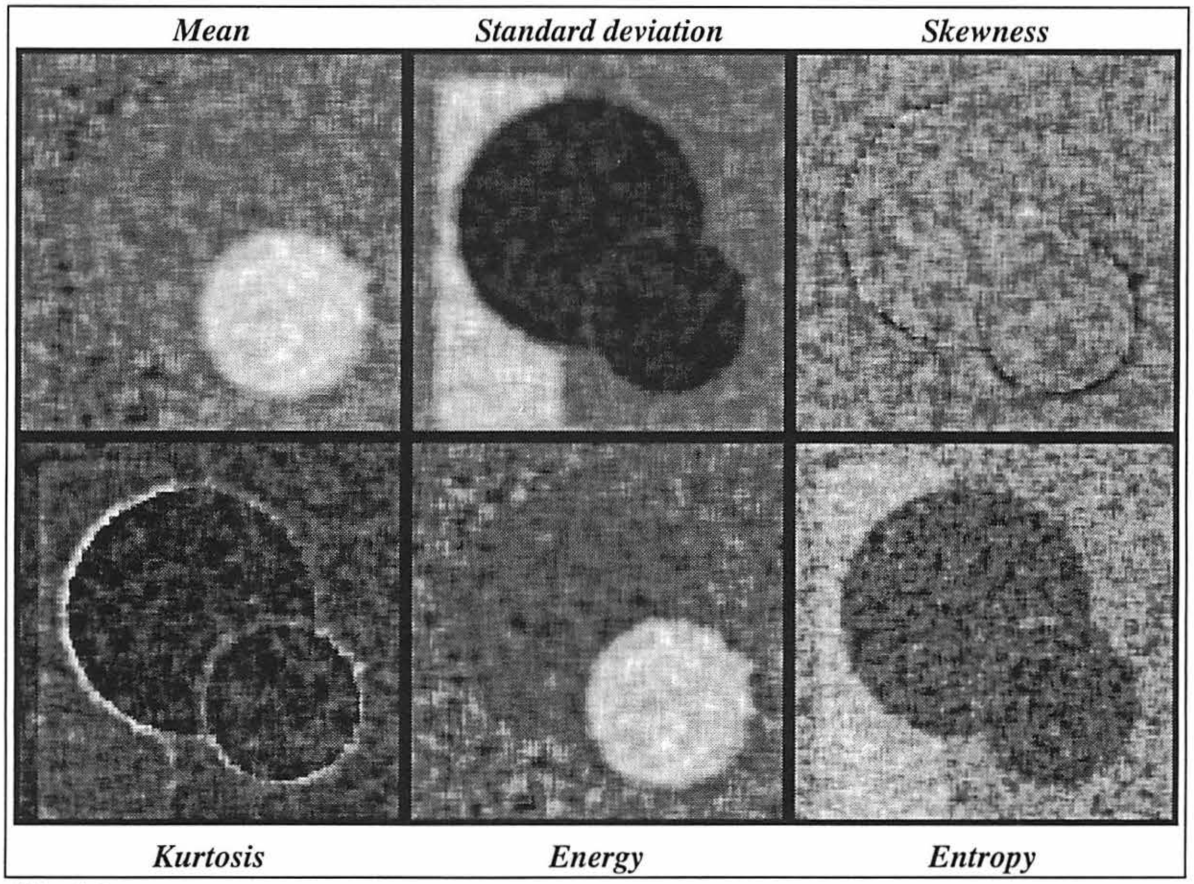


Fig 4.3
First order histogram features.

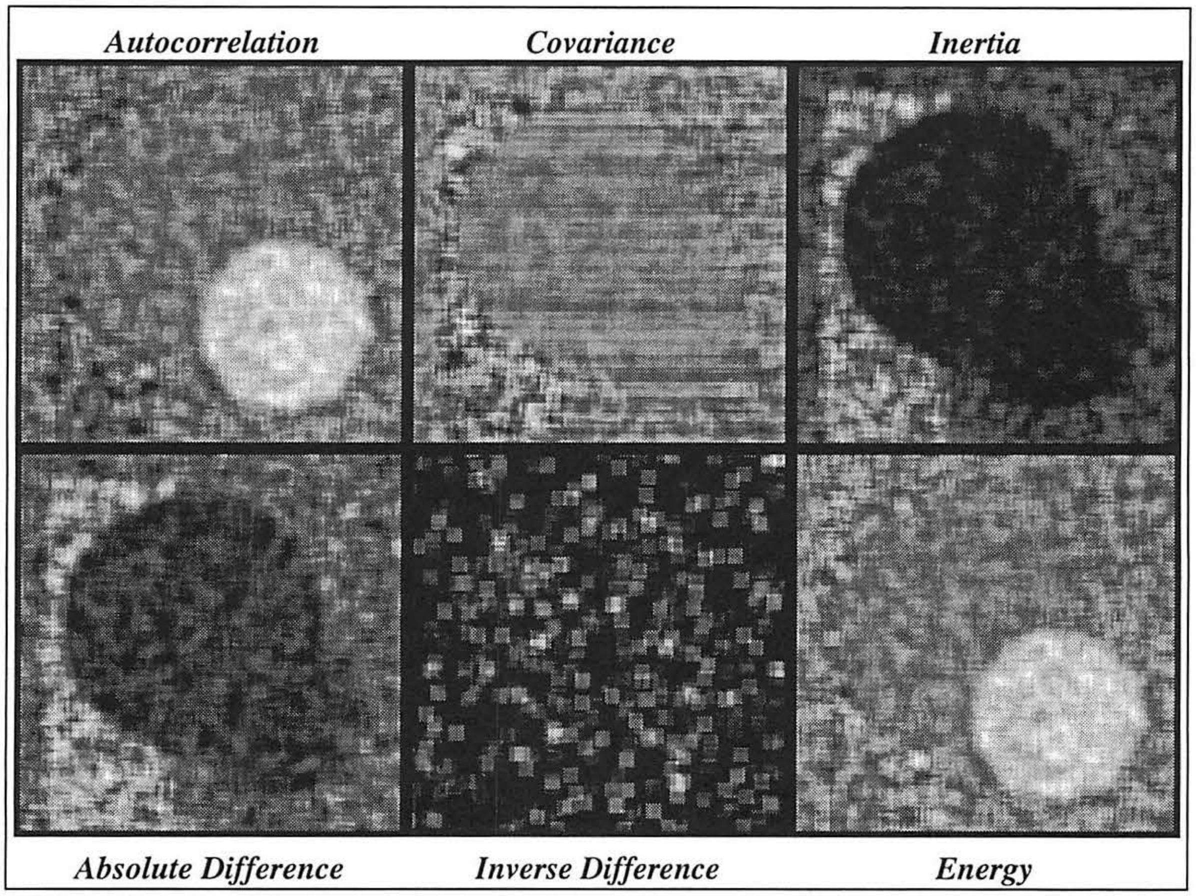


Fig 4.4
Second order Histogram features.

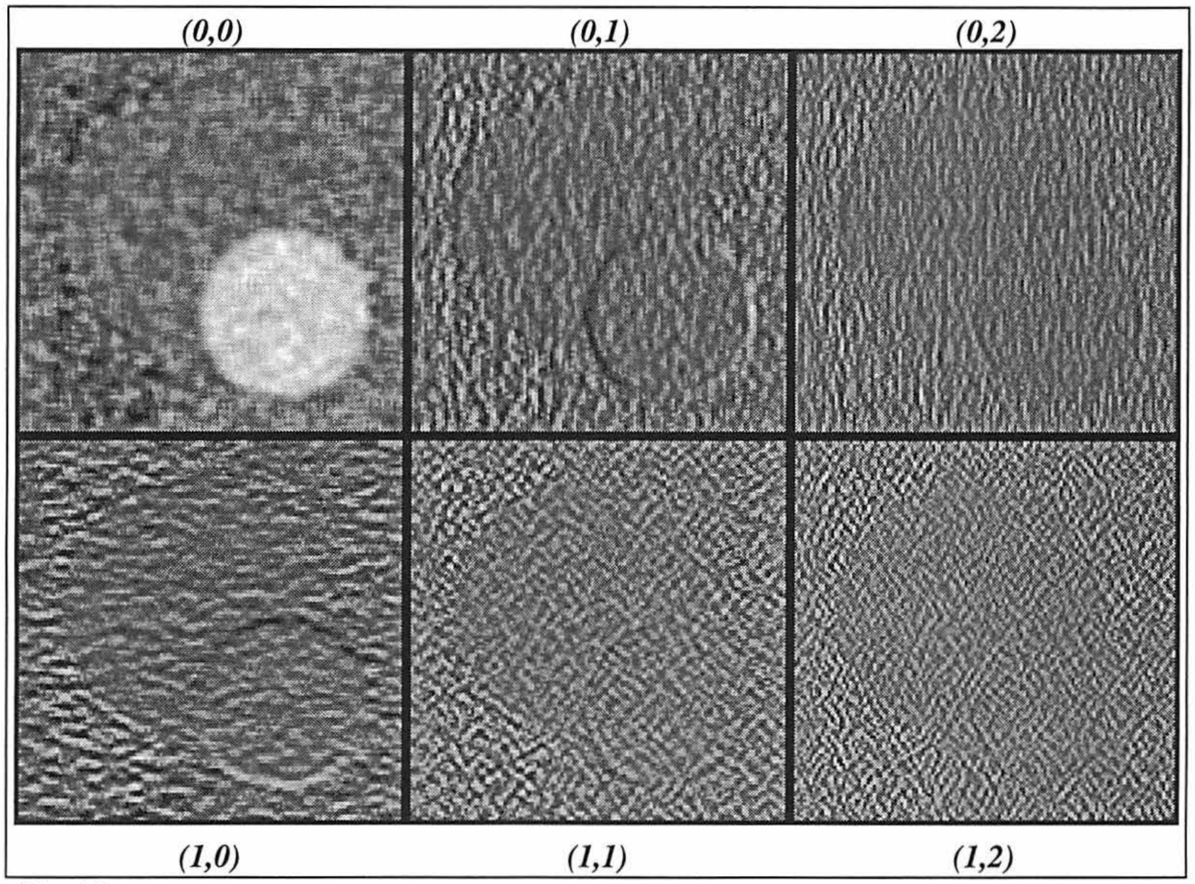


Fig 4.5
Cosine Transform features. (coordinates corresponds to the transform matrix)

4.2.2. Two Natural Brodatz Textures

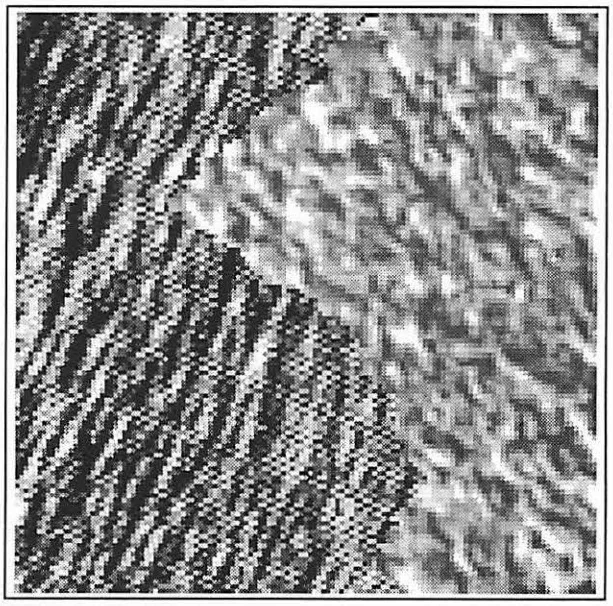


Fig 4.6
Two natural textures

The problem discussed in this section is texture segmentation. Fig 4.6 shows an image that consists of two Brodatz textures. The objective in here is of course to separate the two textures.

Feature maps of this image are found in Fig 4.7-9.

Since there are only two classes in this image we are ideally looking for a feature that alone can separate the two. It turns out that several features can do this including *standard deviation, entropy, inertia, and absolute difference.* Of these the *entropy* does the best job.

As in the last example, the cosine transform does a rather poor job. The explanation for this is of course the same as earlier. There is little or no geometrical information that the cosine transform features can extract that separates the two textures. By the naked eye one can see that the two textures differ in orientation. However, this is not significant enough that the cosine transform can pick it up.

It is interesting to note that the first order histogram features (*standard deviation and entropy*) do a better job than the second order histogram features (*inertia and absolute difference).* For texture classification, spatial location is considered very important which would lead us to expect the second order histogram features to be the best. However, using the feature map, one can clearly see what information is extracted by each feature.

The features of choice, if two features were chosen, would be *standard deviation* and *entropy.*

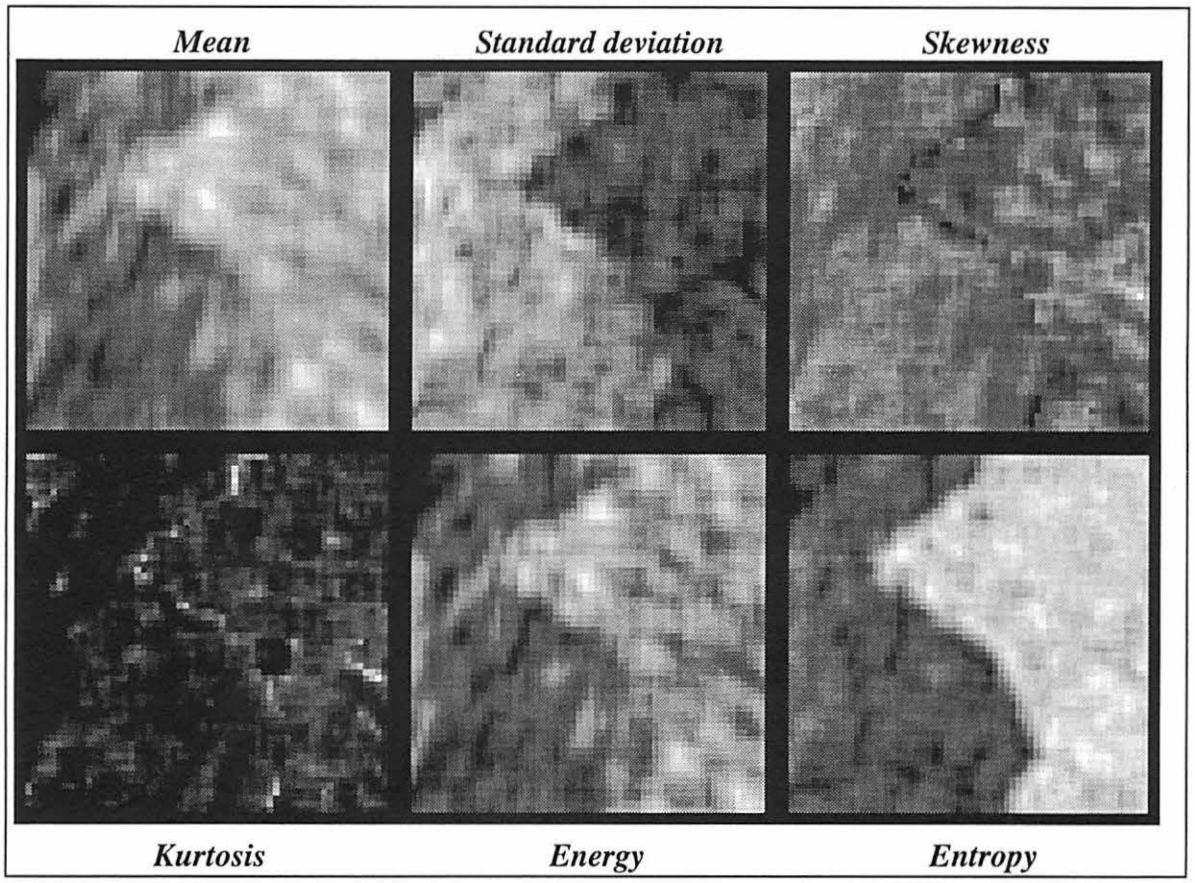


Fig 4.7
First order histogram features.

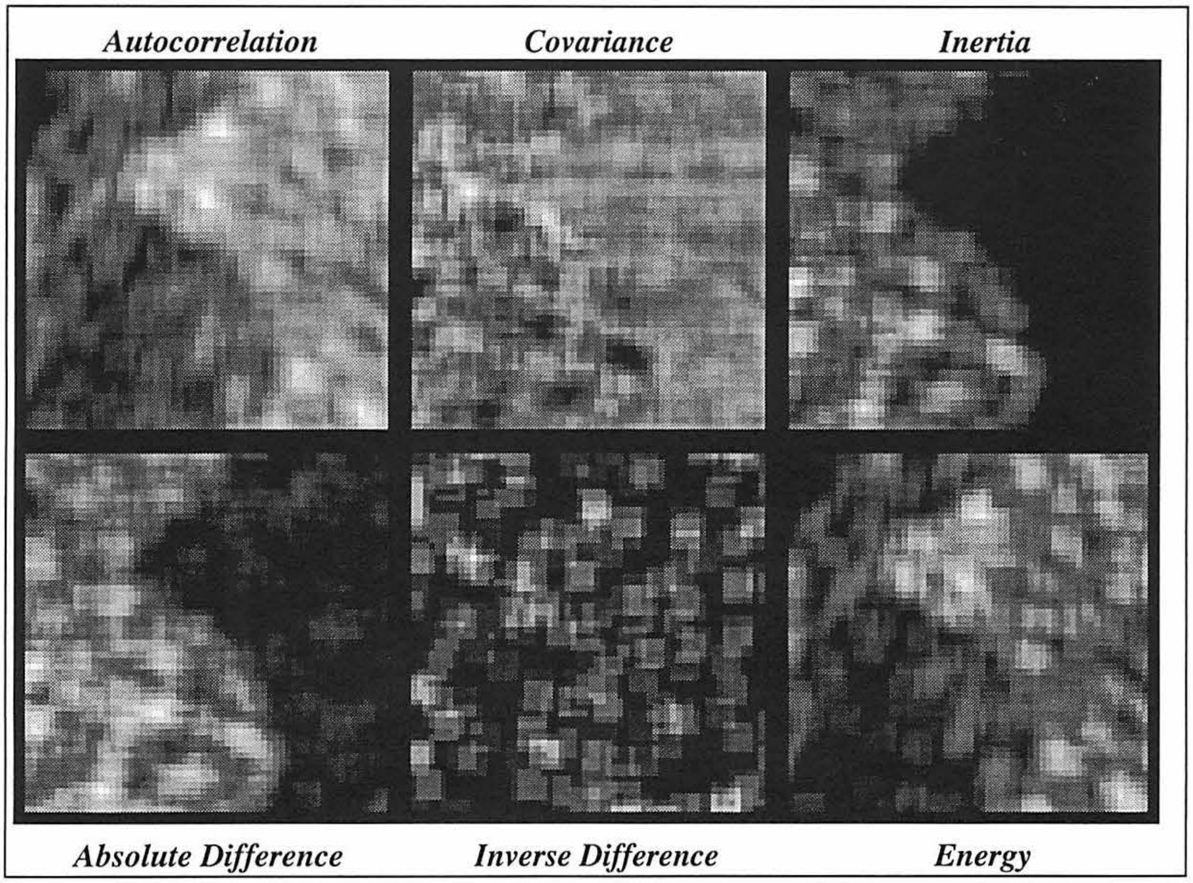


Fig 4.8
Second order histogram features.

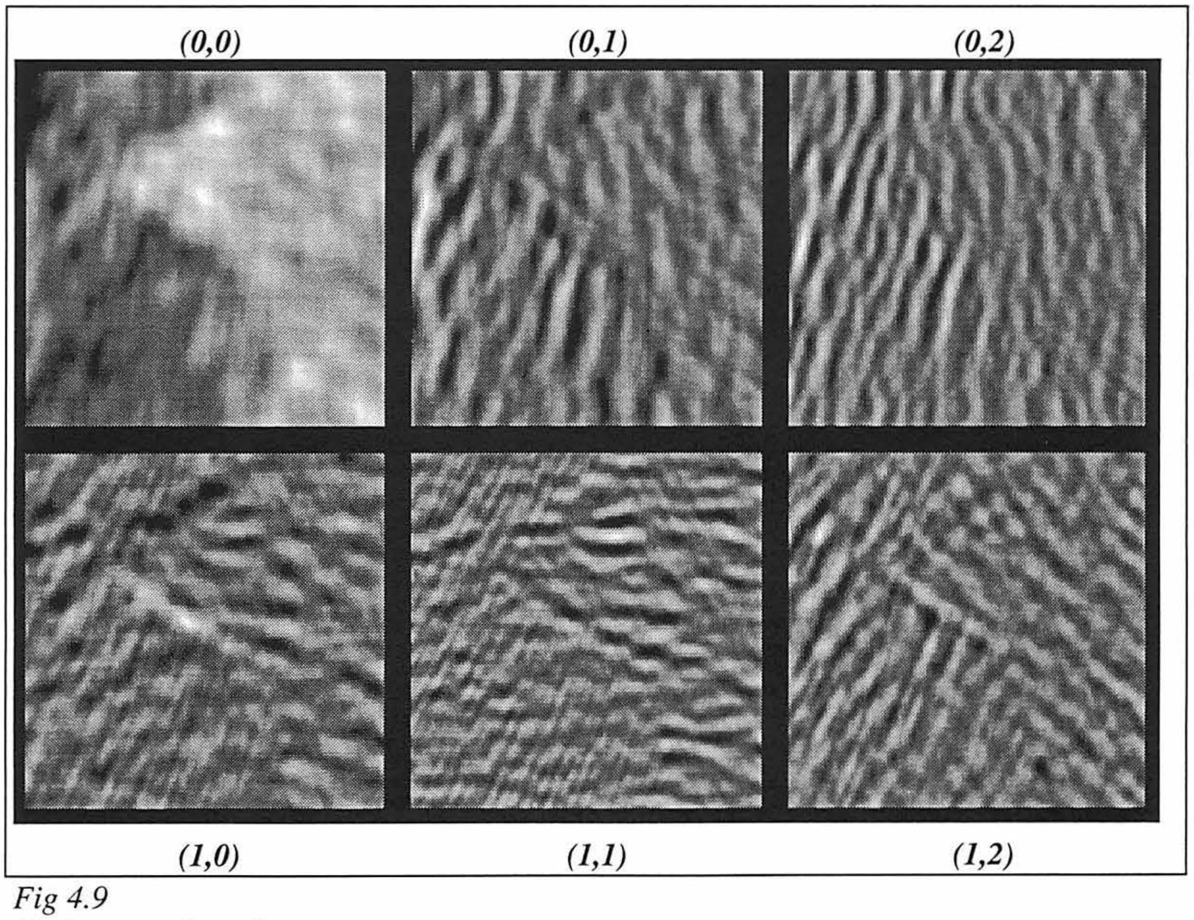


Fig 4.9
Cosine transform features.

4.2.3. Two Geometry Brodatz Textures

The two textures in Fig 4.10 are both of textiles. The human eye can easily discriminate between these two textures, but it turns out that his task is harder for a computer.

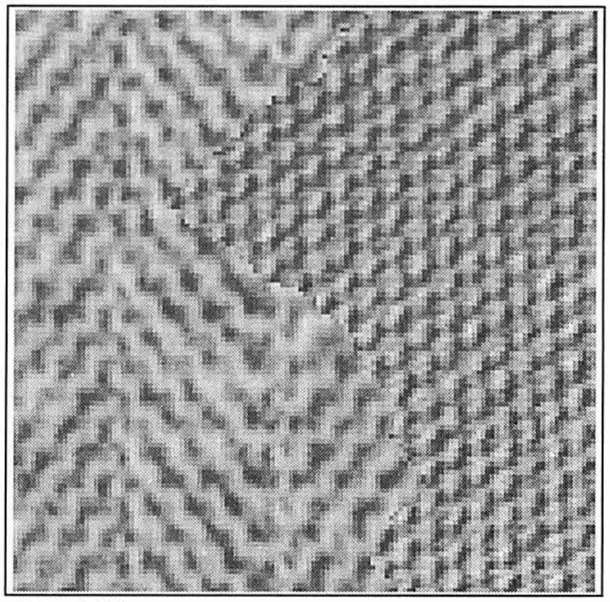


Fig 4.10
Two geometry Brodatz textures.

Considering the first order histogram features (Fig. 4.11), none of the features is particularly useful. The *mean*, *standard deviation,* and *energy* all contributes a little bit, but the two textures are far from separated.

Second order histogram features (Fig. 4.12.) do a better job. In particular, *inertia* and *absolute difference* separate the two textures reasonably well. The other second order histogram features does a poor job, *inverse energy* being the poorest.

The information extracted by the cosine transform features is very interesting. None of these features actually separates the textures completely, but by further processing of (*1,1*) and (*1,2*) the two texture regions should be discriminated.

The conclusion of this feature map analysis is that none of the features could alone adequately classify the two textures. The best attempt was made by the cosine transform features. By further processing of some of these features, a very good classification should be possible.

Interesting to note, is that they eye and the computer are extracting very different information when evaluating an image. The simulated texture image (Fig 4.2) was hard to classify using the naked eye, but was easy to classify for the computer. The geometry textures are easy to distinguish between using the human eye, but a lot harder to classify by a computer.

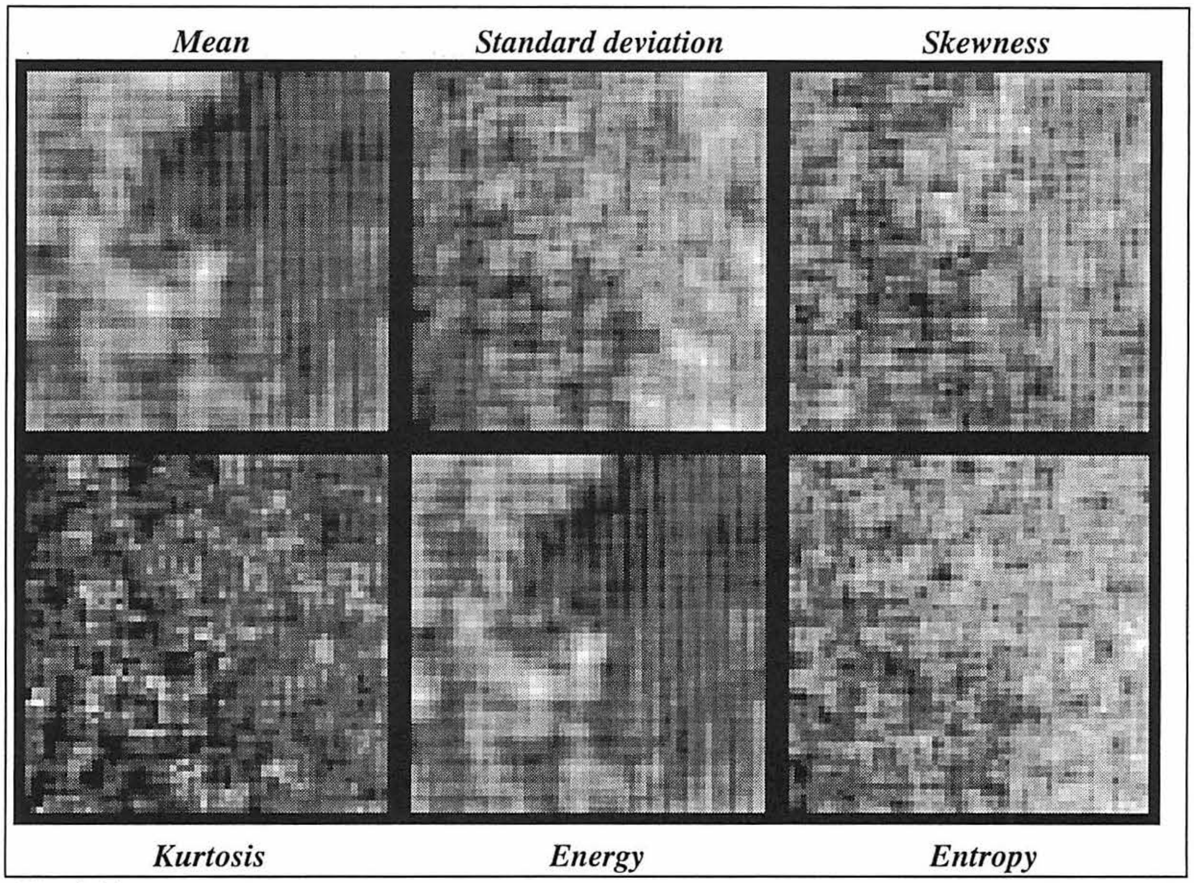


Fig 4.11
First order histogram features.

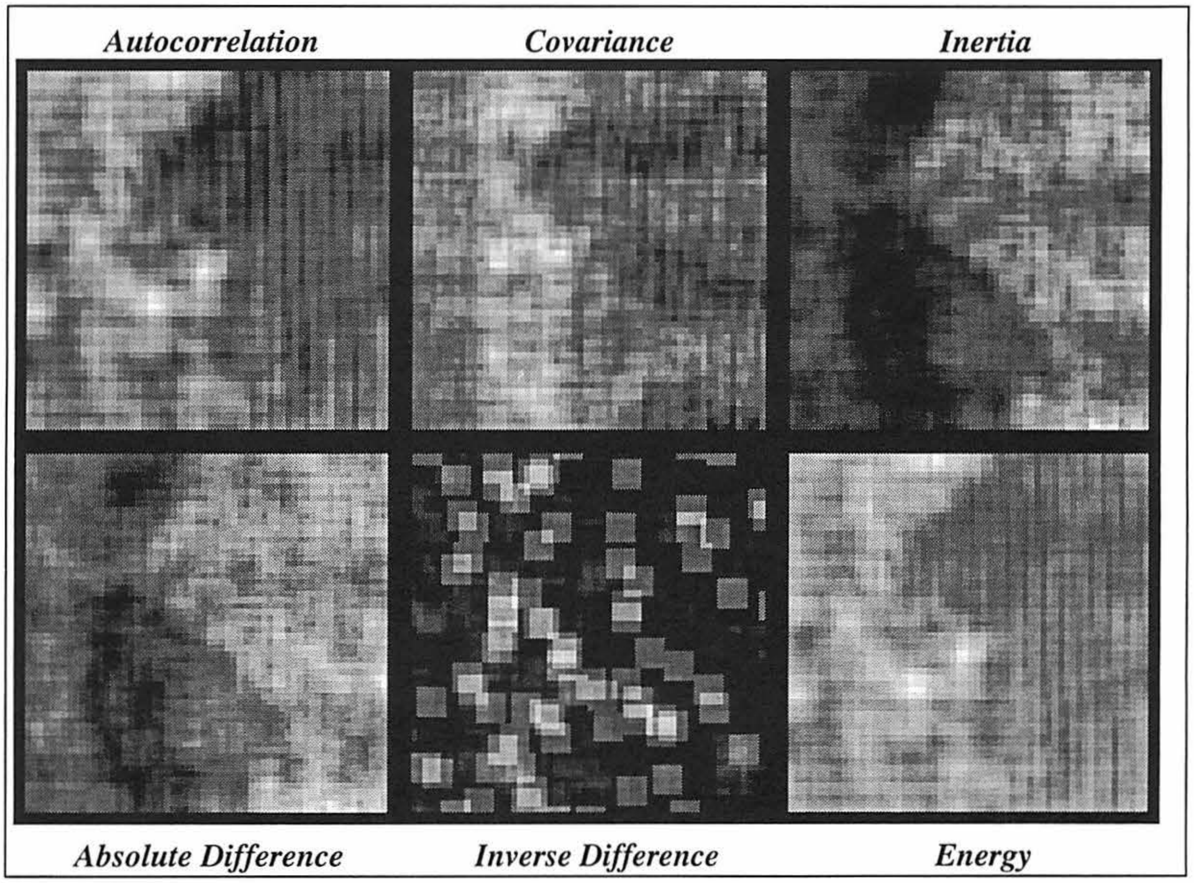


Fig 4.12
Second order histogram features.

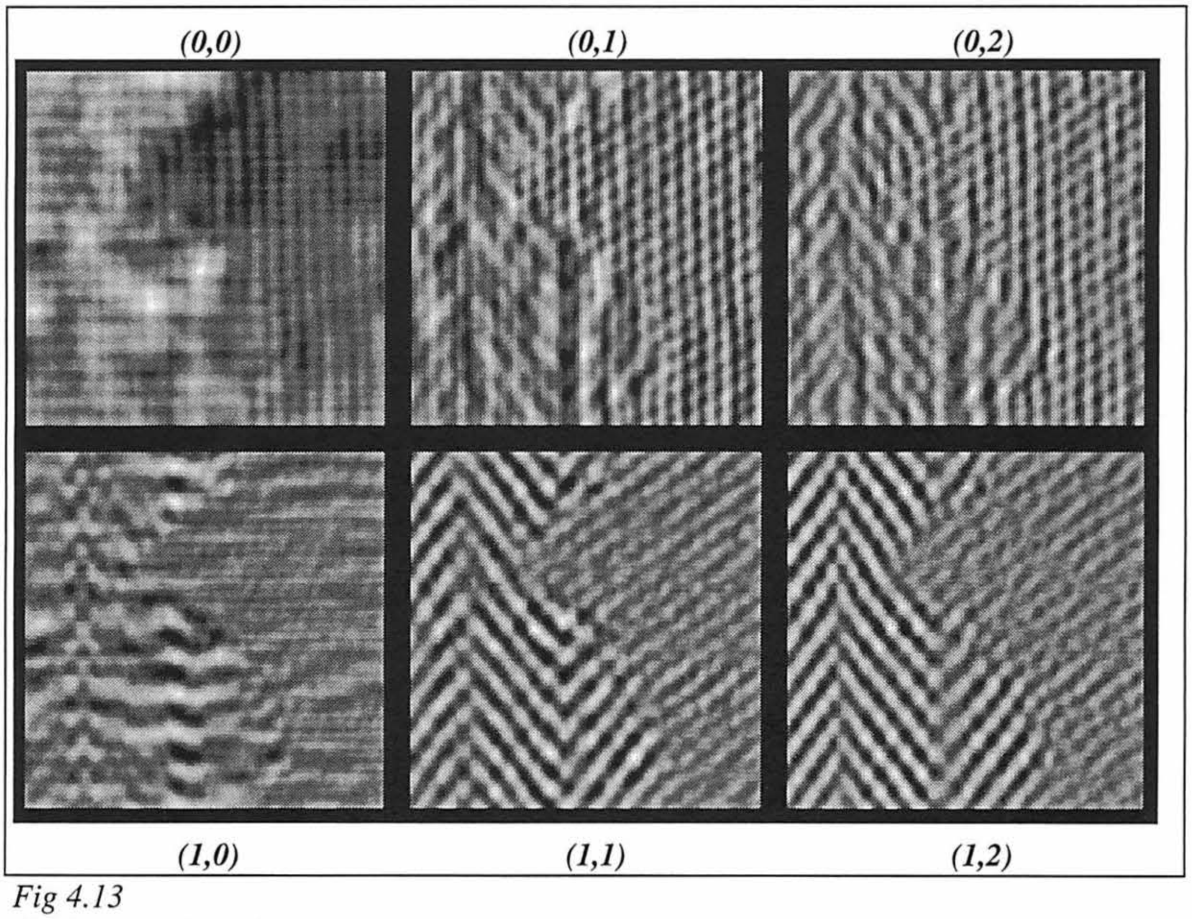


Fig 4.13
Cosine transform features.

4.2.4. Void Flaw in Busy Image

This is an x-ray image that contains an inclusion-like flaw in the lower part of the image while containing a horizontal trend, vertical bars, and handwriting.

The objective in this image could either be to remove the handwriting, extract the flaw, remove the trend, or identify the vertical bars.

Feature maps of this image are displayed in Fig 4.15-17.

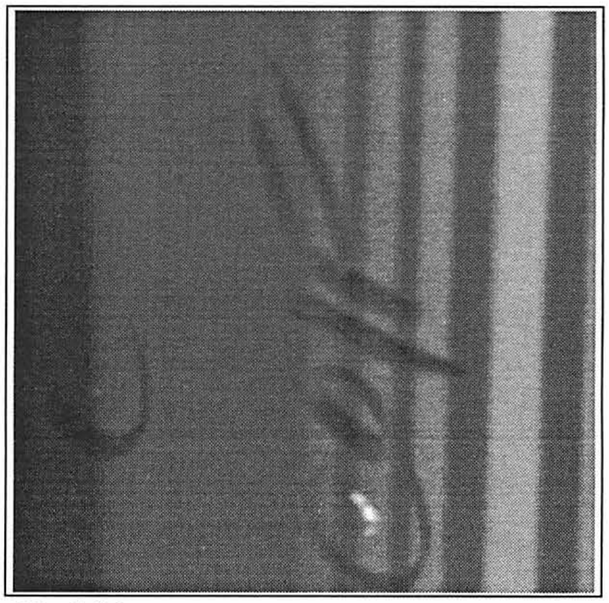


Fig 4.14
Void flaw in busy image

The first order histogram features are capable of extracting a wide variety of information in this image. The flaw is reasonably well identified by the *standard deviation*, and all the edges in the image are found by the *skewness, kurtosis,* and *entropy.* This information could be useful in identifying the handwriting or the vertical bars.

The second order histogram features did a beautiful job in identifying the flaw. The *covariance* and *inertia* completely separated the flaw from the rest of the image. In these feature maps, the trend and the handwriting are completely gone. The vertical bars are almost gone, too; a couple of weak edges are barely visible in the background.

The three lower cosine transform features did an excellent job in removing the trend and the vertical bars. The reason for this is that these feature maps contain no horizontal frequencies. By using the three lowest features, the flaw and the handwriting should both be possible to identify.

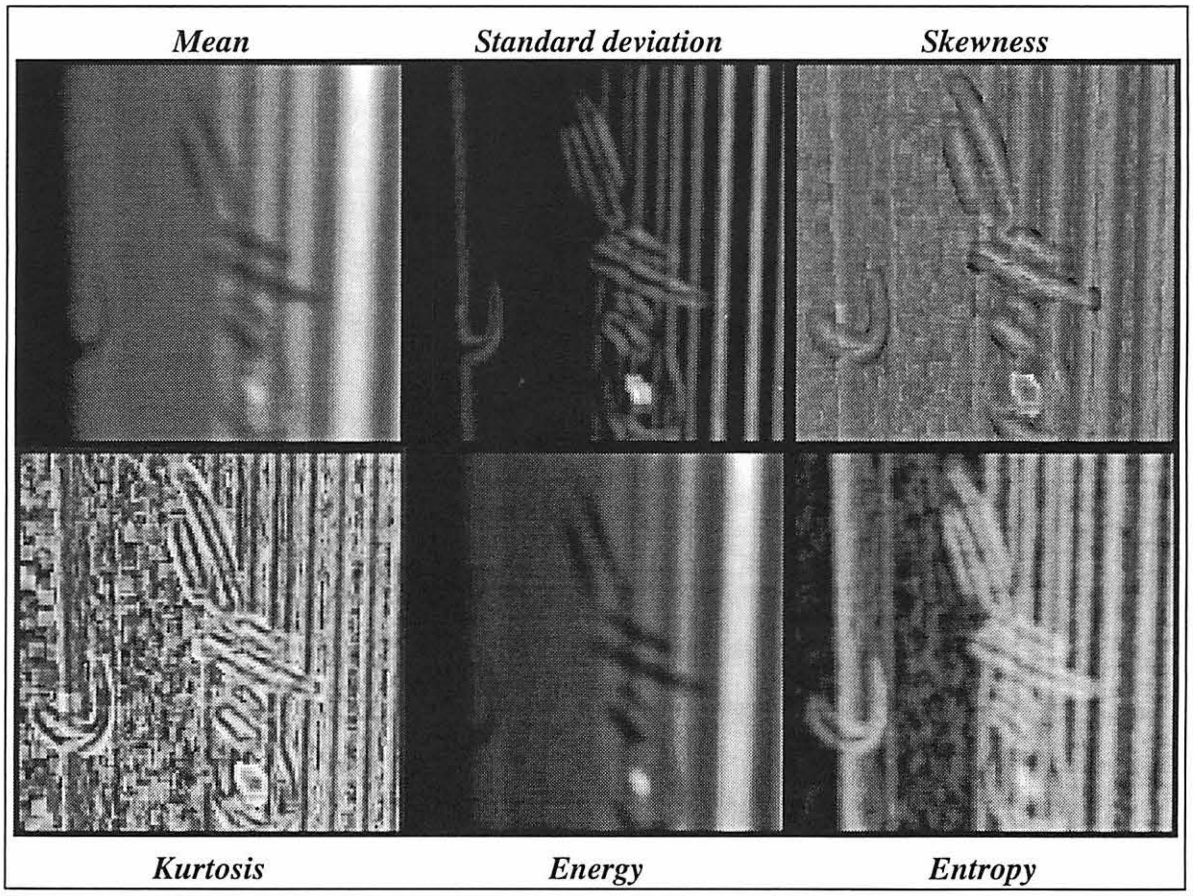


Fig 4.15
First order histogram features.

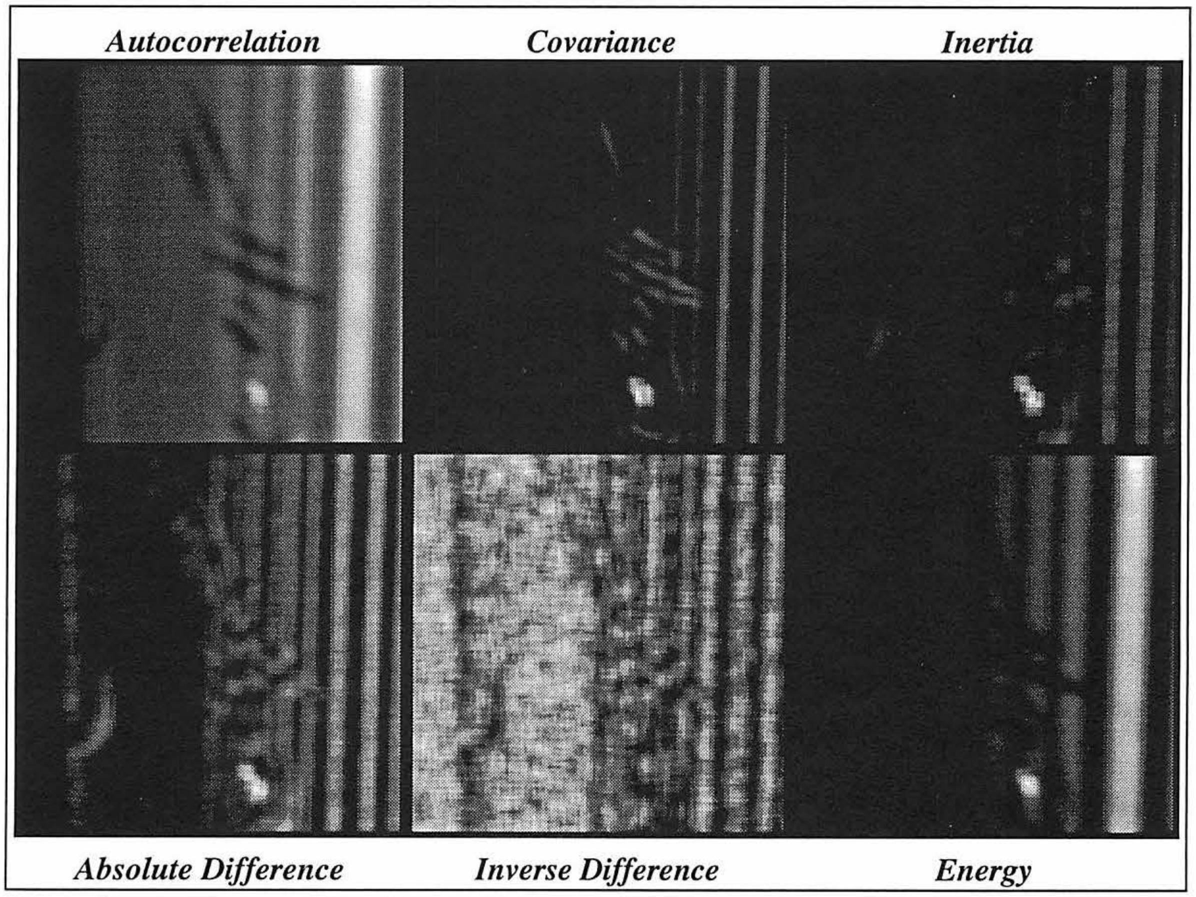


Fig 4.16
Second order histogram features.

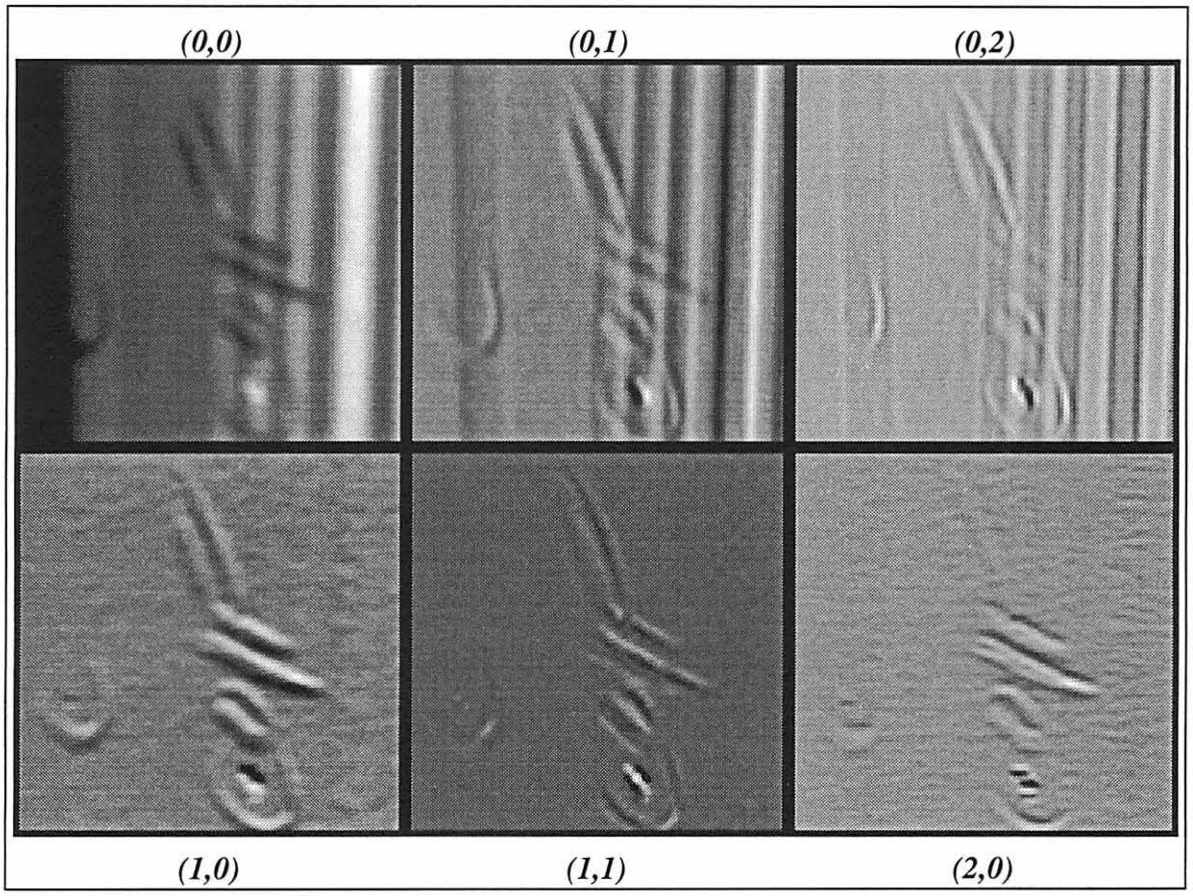


Fig 4.17
Cosine transform features.

4.2.5. Infra Red Galaxy Image

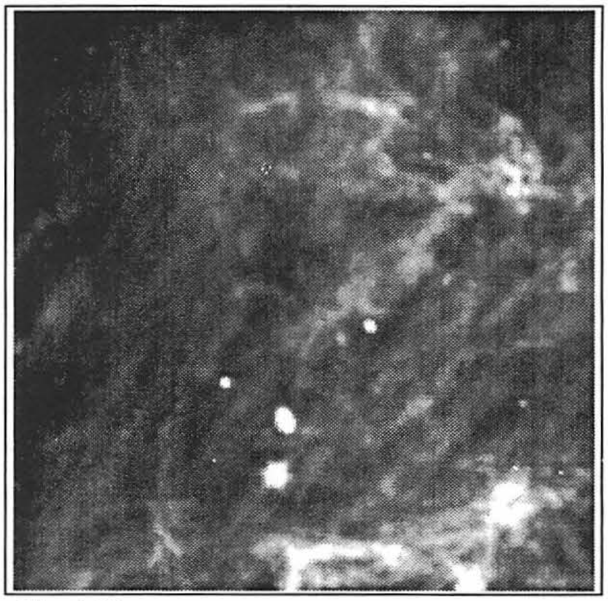


Fig 4.18
Infra red galaxy image.

This is an astronomical image of an extragalactic field. It is an infrared image at 100 micron wavelength, and the four bright spots in the middle are known galaxies. The objective in this image is to find unknown galaxies that are hidden by the cloud-like structures called infrared *cirrus clouds*. Most of the infrared cirrus emmision is from our own galaxy.

By analyzing the feature maps (Fig 4.19-21) we can see that the first order histogram features that are most useful for this purpose is *standard deviation* and *entropy*. The *standard deviation* clearly separated the four known galaxies, and the *entropy* identifies four areas in the bottom part of the image as structures having similar characteristics as two of the known galaxies. This is a very interesting result! Could it be that the entropy has identified four unknown galaxies?

The interesting suggestion by the *entropy* is backed by the result of the *inverse difference*. This feature identifies the same six structures as having the same characteristics. The other second order histogram features cannot substantiate this claim although most of them identify most of the known galaxies.

The cosine transform features, although identifying the known galaxies, are not identifying other structures in the image that has characteristics that are similar to the known galaxies.

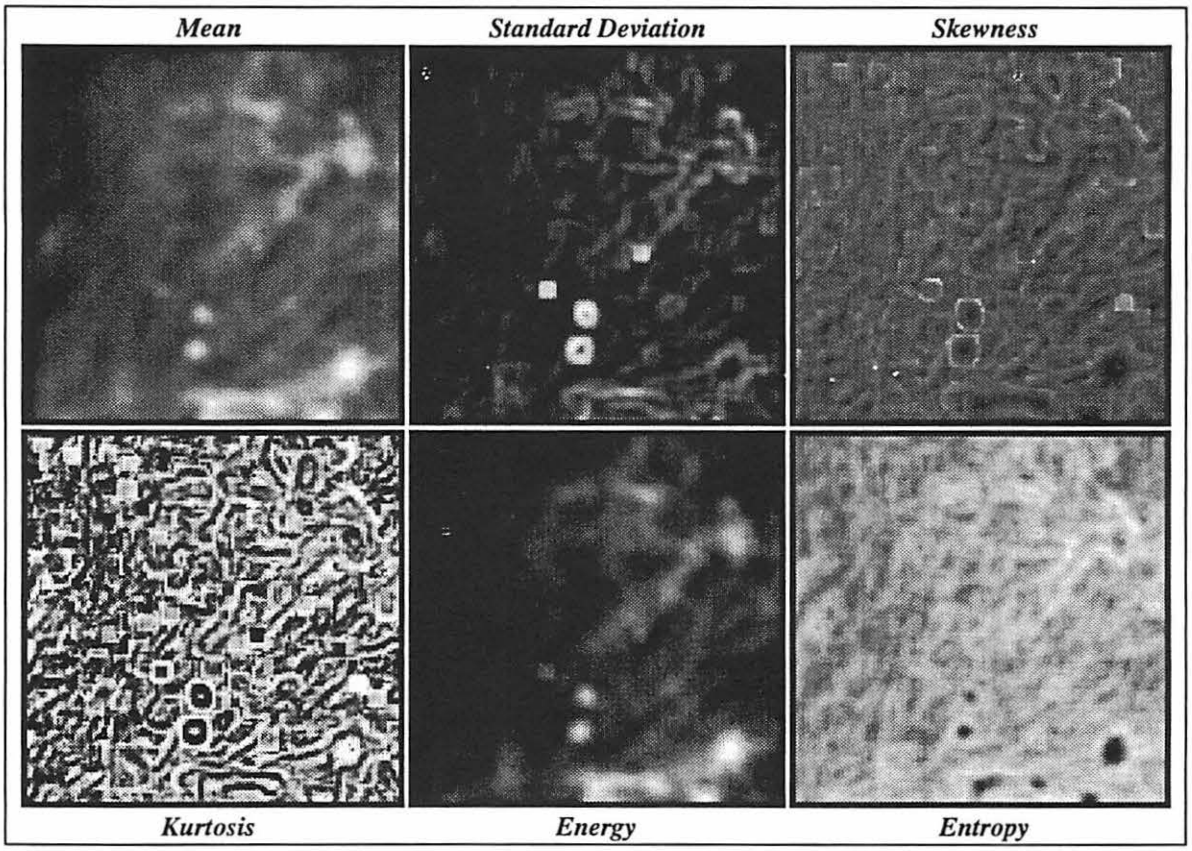


Fig 4.19
Feature maps of infra red galaxy image, first order histogram features.

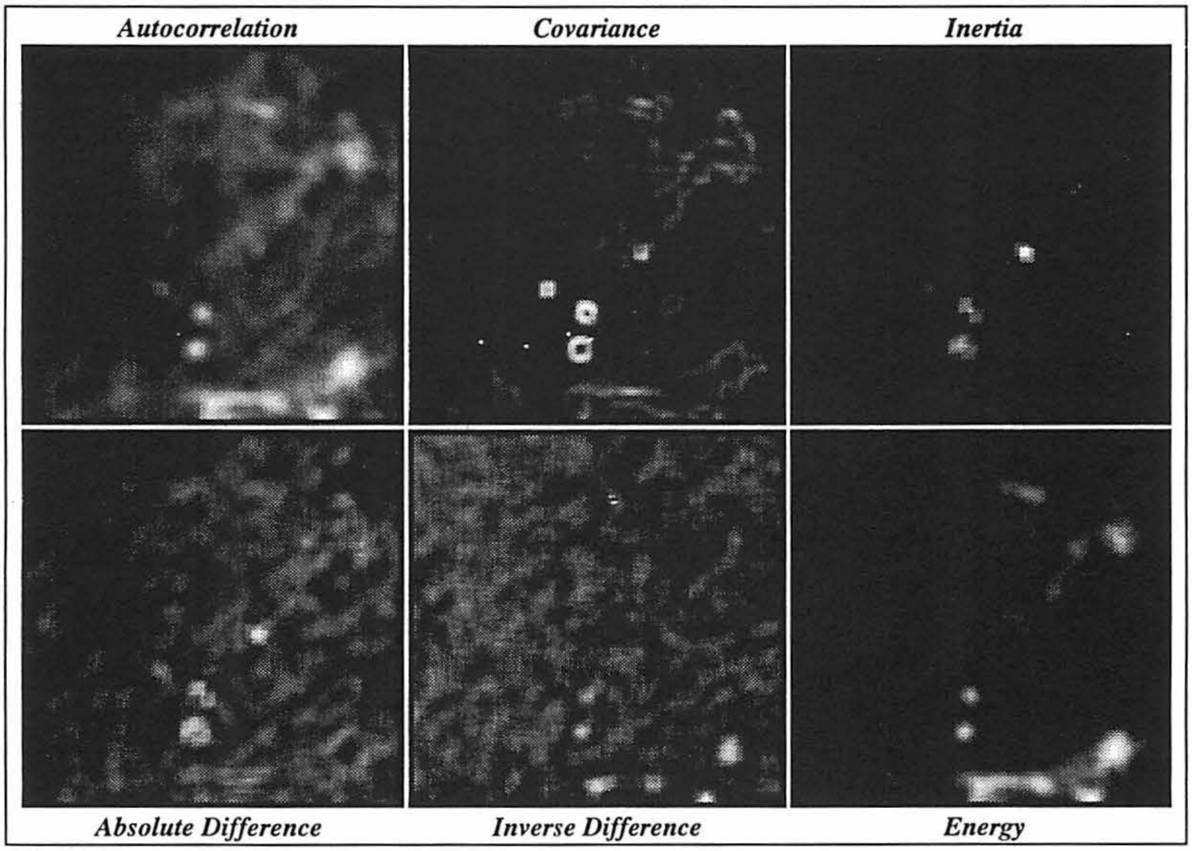


Fig 4.20
Feature maps of infra red galaxy image, second order histogram features.

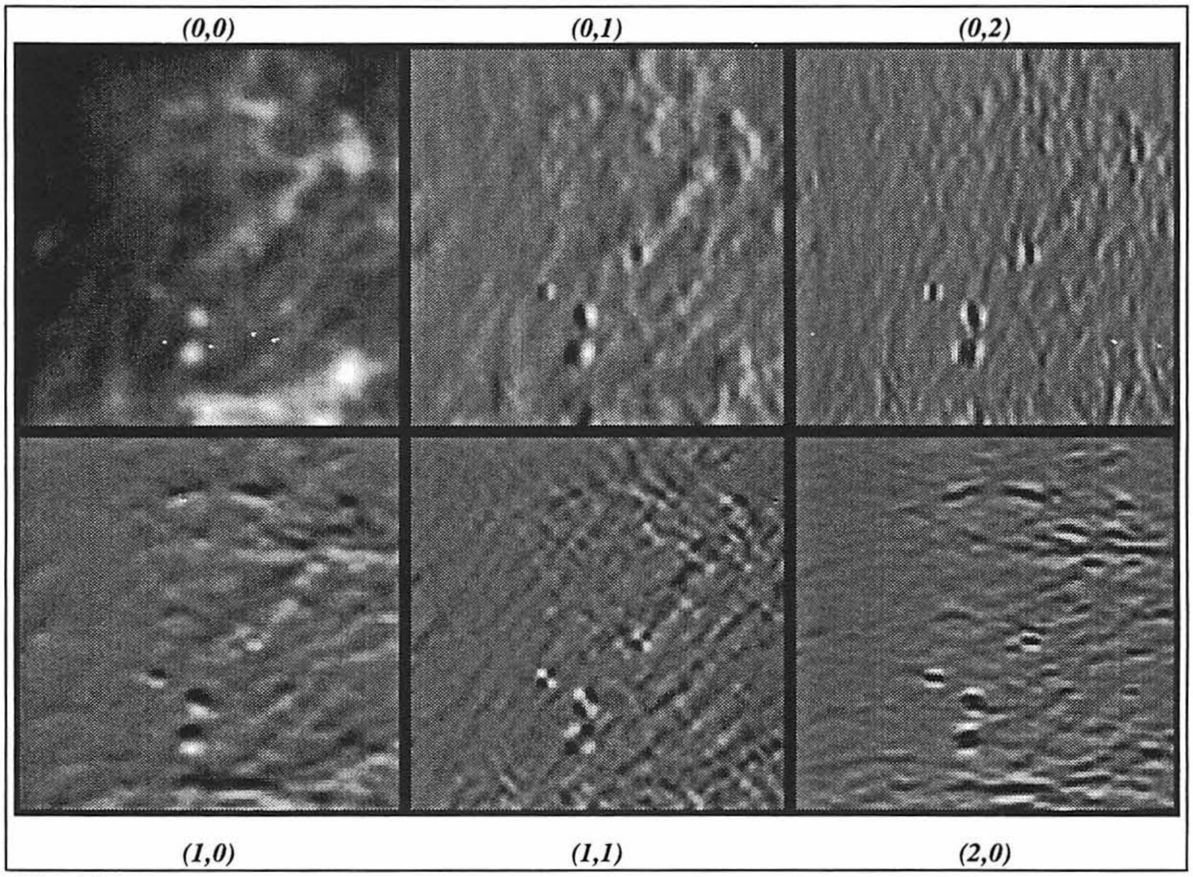


Fig 4.21
Feature maps of infra red galaxy image, cosine transform features.

4.2.6. Fossil Skeletal Details in X-ray Image

The image in Fig 4.22 is an x-ray image of a fish fossil and the objective with this image is to extract skeletal details.

The feature maps of this image is shown in Fig 4.23-25.

A couple of the first order histogram features did an excellent job in finding vertebras (*standard deviation* and *entropy*). However, none of the rib bones could be found using these features.

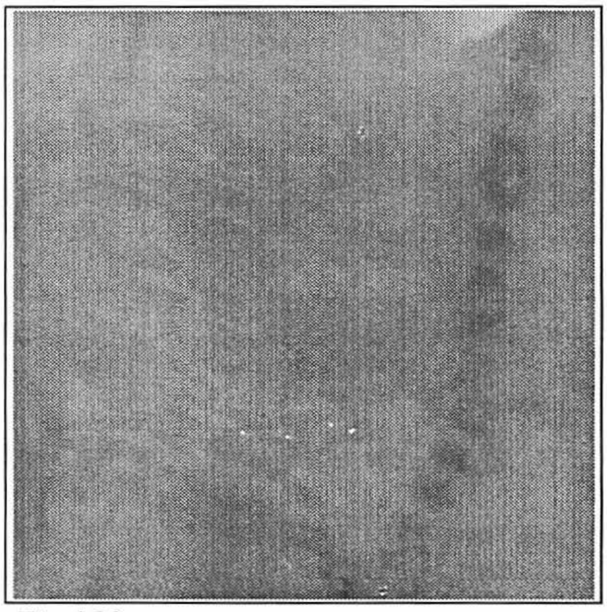


Fig 4.22
Fossilized fish skeleton in x-ray image.

The second order histogram features turned out to be the least useful features in this image. However, it is interesting to note that these features identified noise probably introduced in the image acquisition. In the three difference features (*inertia, absolute difference,* and *inverse difference*) one can find strong vertical structures. These are caused by inference between vertical structures in the original image and the distance parameter in the second order histogram extraction.

The cosine transform features were extracting a lot of interesting details. In most of these feature maps, the vertebras very clearly identified. Several did also show weak contours of rib bones.

Considering how difficult a task it is to extract information in the image in Fig 4.22, the three feature sets did a fairly good job. None of them were able to convincingly identify rib bones, but the spine and vertebras were easily found.

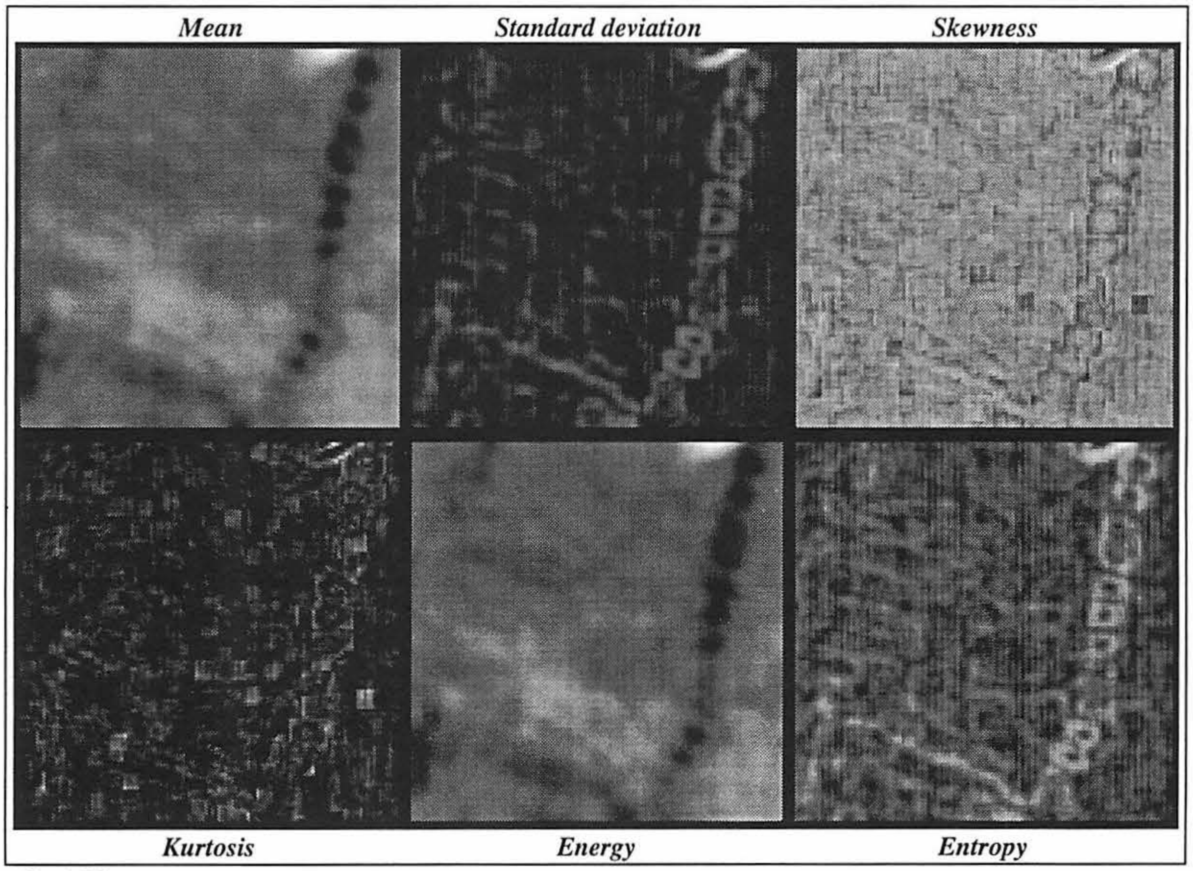


Fig 4.23
First order histogram features.

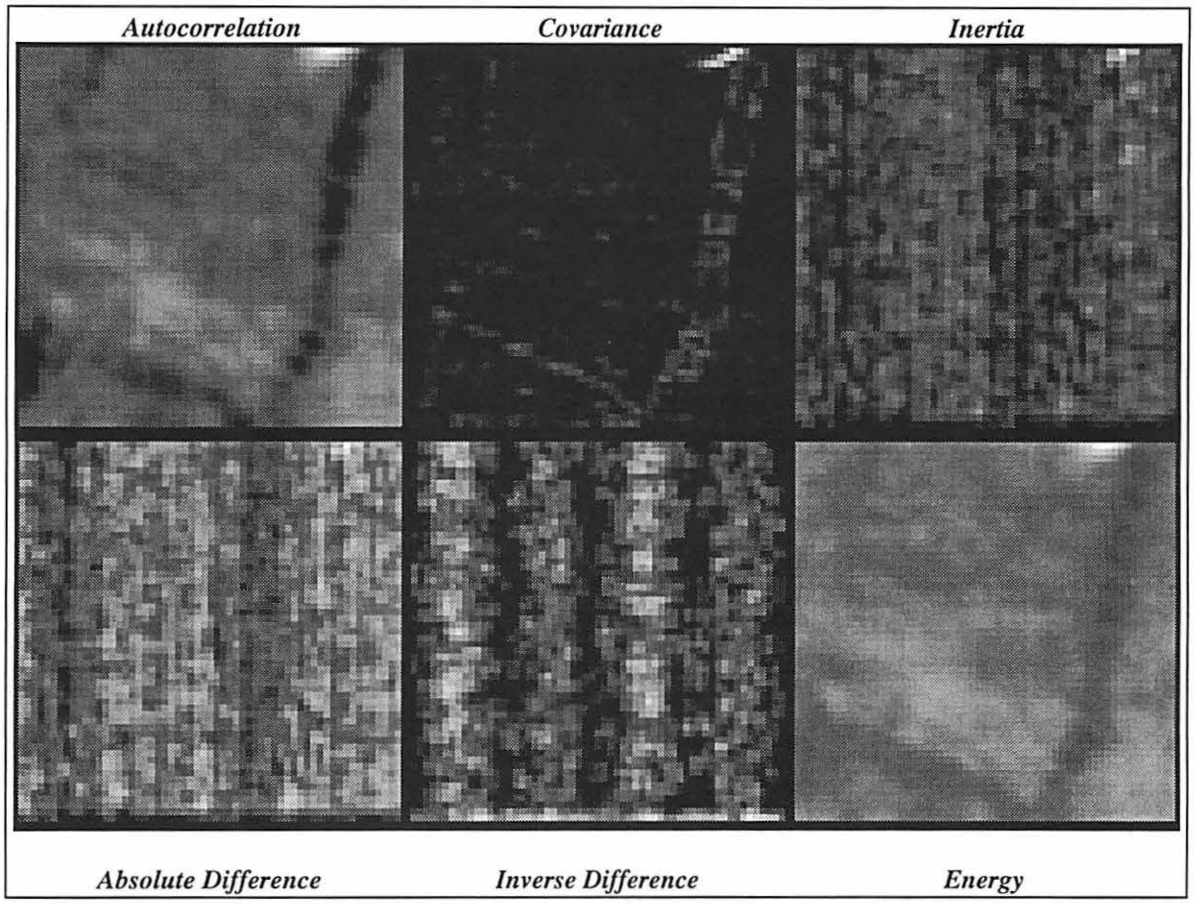


Fig 4.24
Second order histogram features.

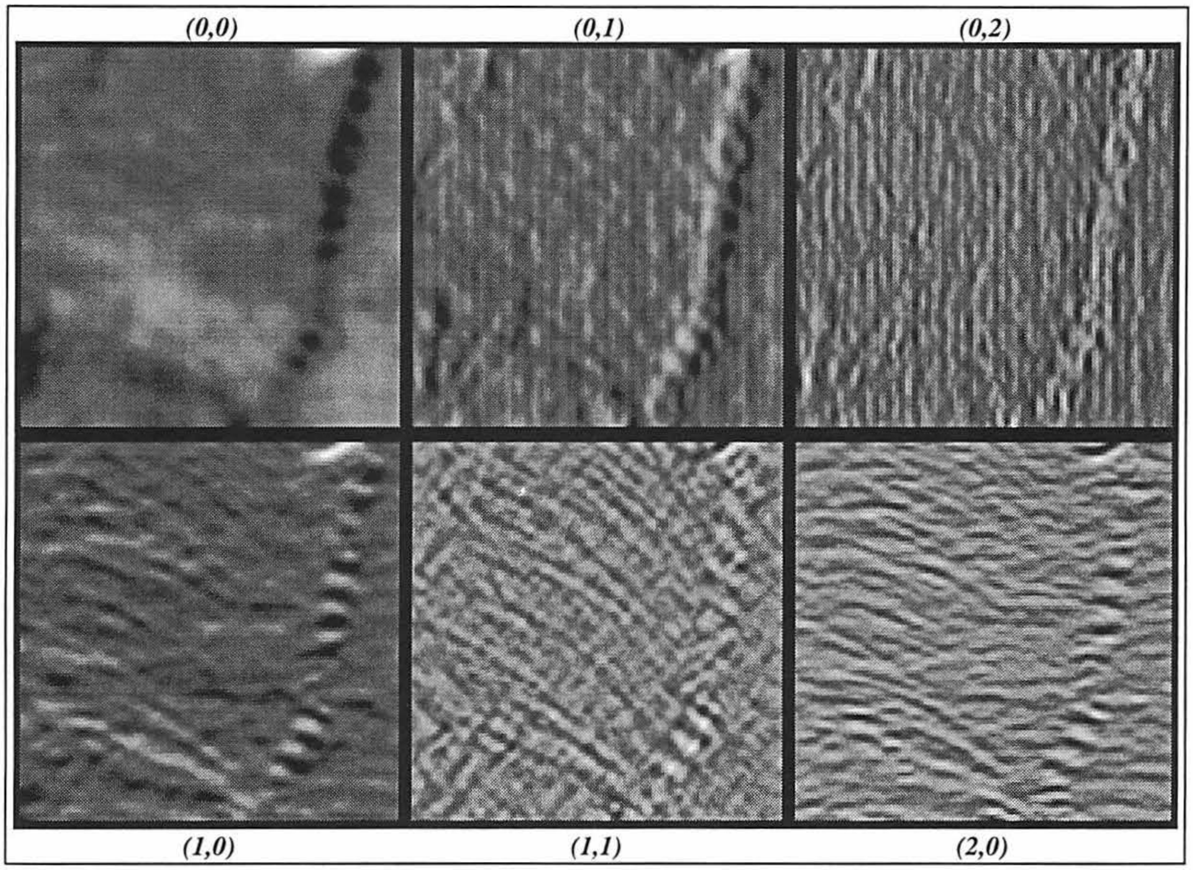


Fig 4.25
Cosine transform features.

4.3. Choosing Training Data

This topic has been addressed earlier as well, but it is so important that it is worth repeating. Since Sherlock is a set of general techniques that are trained to solve particular problems, it is extremely important that the training data is representative for the actual data to be classified. The knowledge incorporated in the trained system comes exclusively from the training data. If the training data is incomplete, the trained system will become incomplete also.

Ideally, all types of flaws and non-flaws should be represented, but this is difficult to guarantee. This is partly due to the fact that computers are extracting different information than a human. Even though, prototypes of all visual flaw variations have been included, the actual numerical flaw variations calculated by the computer can be incomplete. To design a good set of training data, it is therefore important to consult feature maps.

Another good rule is the more data the better. However, this approach can slow the training process unnecessarily if there is a lot of redundancy present. Nevertheless, such an approach can prove useful to cover as many flaw variations as possible.

4.4. Choosing Classifier

The general rule for choosing a classifier is that is must match the feature set. First of all, the classifier has to be able to extract the discriminatory information in the features. Secondly the features cannot conflict with any of the assumptions the classifier is based on. These two requirements are closely related.

An example of a classifier that has a pretty strict requirement is the Single Layer Perceptron. This classifier requires that the classes are linearly separable in feature space. For data that don't comply with this, the classifier performs poorly. Other classifiers with strict requirements are the K-mean and Fuzzy-C classifier. These both require the data to be unimodal i.e. each class has no more than one cluster.

Classification speed is of course also an important issue. The classifier should be as fast as possible without sacrificing reliability. This is of course a tough requirement

5. CLASSIFICATION RESULTS

5.1. Introduction

The first part of this chapter reports on unsupervised classification. This is not directly related to automatic flaw detection, but it tells a lot about the extracted features. Many examples of feature extraction were given in last chapter, and the unsupervised classification can be looked at as a measure of how well the features are able to extract valid information. The reason why these results are included here is that feature extraction is such an important part in finding flaws.

The last part of this chapter reports on supervised classification. This is what an automatic flaw detection system would rely upon. Here the extracted features are carried through the whole flaw detection process. Training data (prototypes) are extracted from known data, and the knowledge acquired by the system is then used to classify unknown data. Real data are used in this section to demonstrate Sherlock's ability to solve real world problems.

5.2. Simulated Texture Image

The first example of unsupervised classification is the simulated texture image (Fig 4.2) studied in the chapter of inspection strategies. The objective is to make the classifier label the four texture regions correctly. Classification results using different methods of feature normalization are used to demonstrate the importance of this subject. As seen from Fig. 5.1, all four texture regions are correctly identified using an appropriate normalization.

5.3. Shrinkage Cracks

The second example of unsupervised classification is using the same strategy as in last example to find shrinkage cracks in railroad frogs. Classification results using various normalization methods are also discussed here. The classifier of choice this time is the *Fuzzy C* clustering algorithm. This performs similarly to the K-mean, but has more flexibility due to the fuzzy parameter m which was constant at 2 for the results shown in Fig 5.2-3. Because of the low contrast in the images, the histogram equalized version of the original images are displayed to better evaluate the classification result.

5.4. Classification of Flawed Rugs

The results reported here are examples of automatic supervised classification that could have been implemented in a real-time system for monitoring production of rugs.

The objective of the inspection is to identify regions with flaws. The rugs consists of .25" fibers that all are pointing in the same angle out of the rug basement. Sometimes these fibers get skewed making the wrong angle, and sometimes the fibers themselves have problems. Both of these cases are defined as flawed rug regions which the manufacturer would like to automatically identify.

The images for the inspection to be reported were acquired using a light source, a camera, and a frame grabber. Flaw regions in this setting appeared as darker regions in the image.

A total of four images were inspected (Fig 5.4-5.7) of which one (Fig 5.4) was used as the training image. In this image, three prototypes were extracted from flaw regions and three prototypes were extracted from non-flaw regions. The white rectangle that appears in the original (top) picture in Fig 5.4-5.7 was a piece of tape that was used as aid in the image acquisition.

The extracted features were normalized using a mean-sdev scheme to compensate for differences in mean and variance between the images. The features extracted was the first order histogram features, and the classifier was the single layer perceptron network.

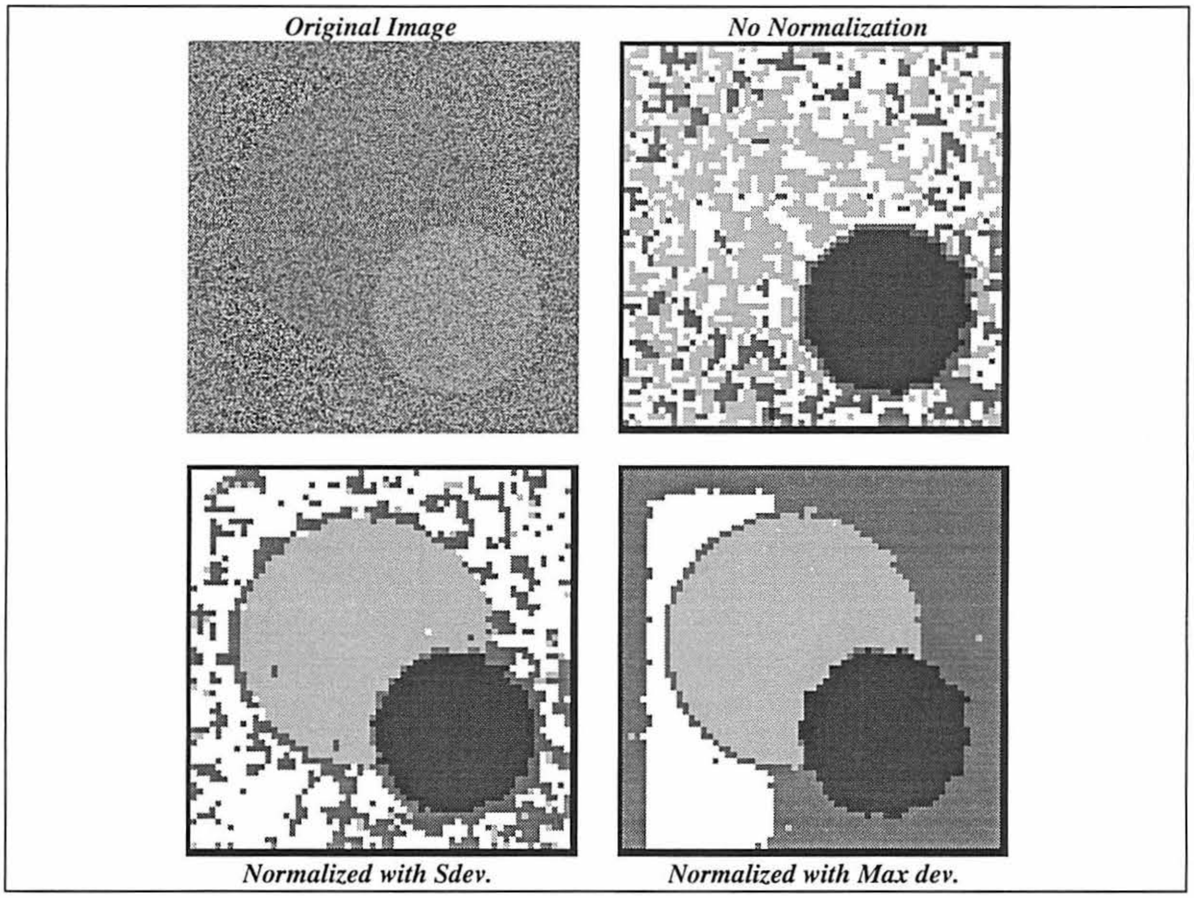


Fig 5.1
Classification of simulated textures using all six first order histogram features, feature support=10, grid size=4, and K-mean,

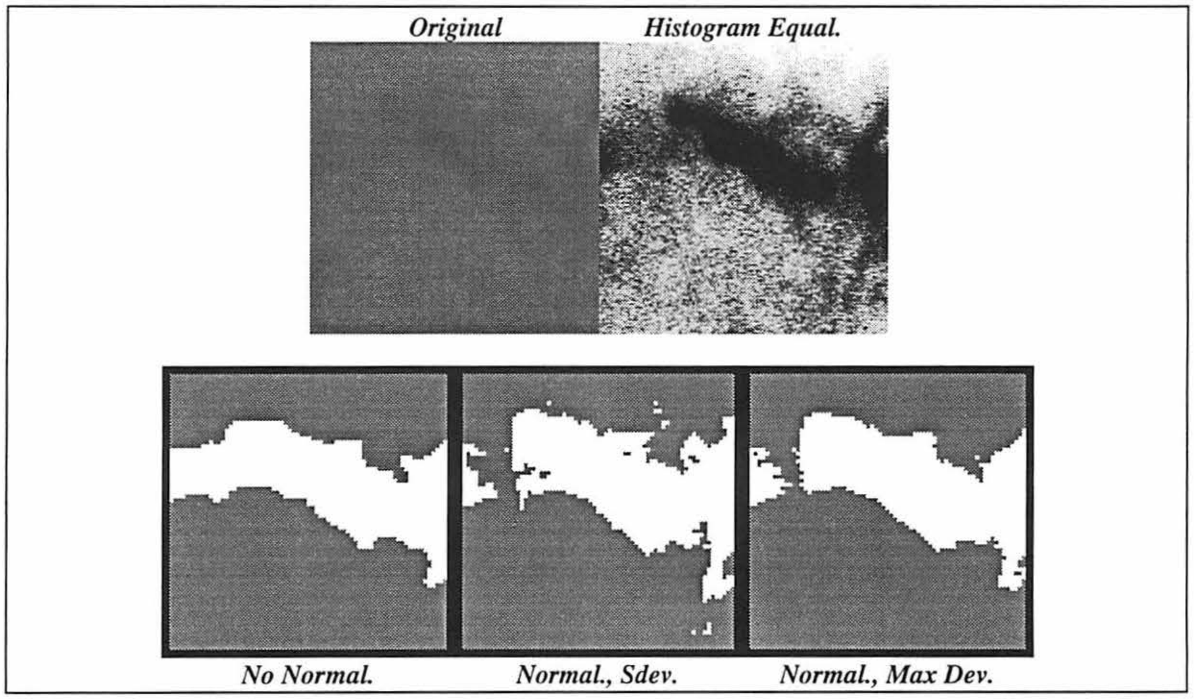


Fig 5.2.
Identification of shrinkage cracks using first order histogram features, feature support=10, grid size=2, and Fuzzy-C (m=2). All processing was done on the original image. The histogram equalized result is shown to clarify the dark flaw structure.

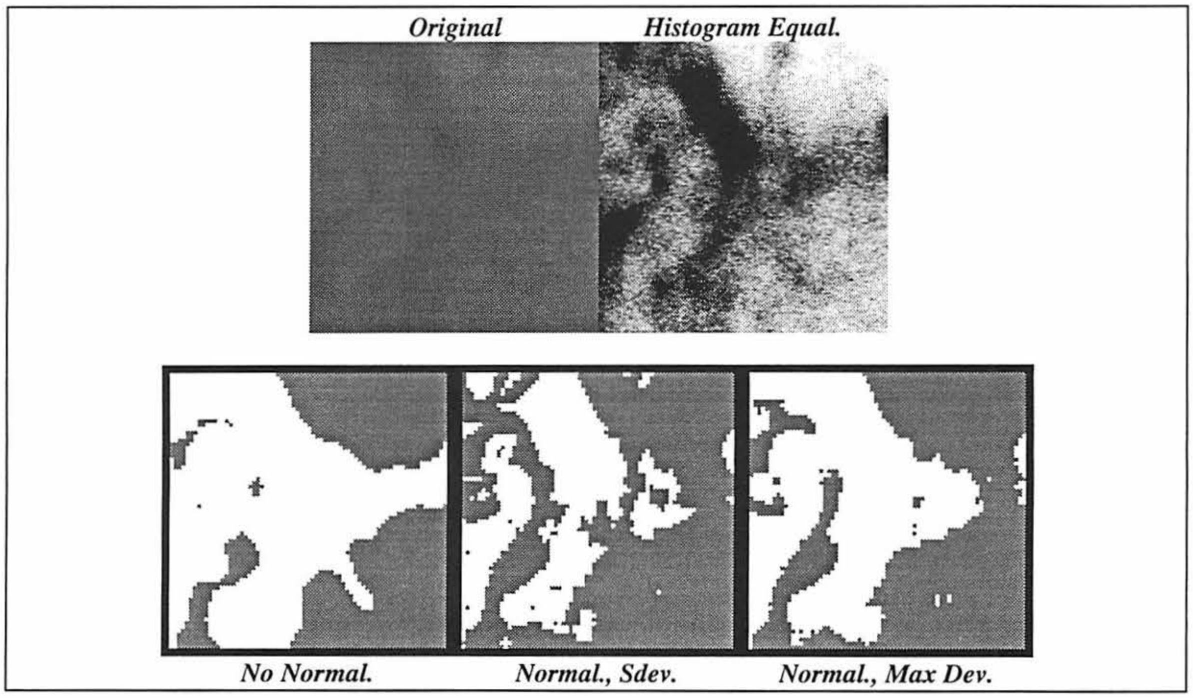


Fig 5.3
Identification of shrinkage cracks using first order histogram features, feature support=10, grid size=2, and Fuzzy-C (m=2),

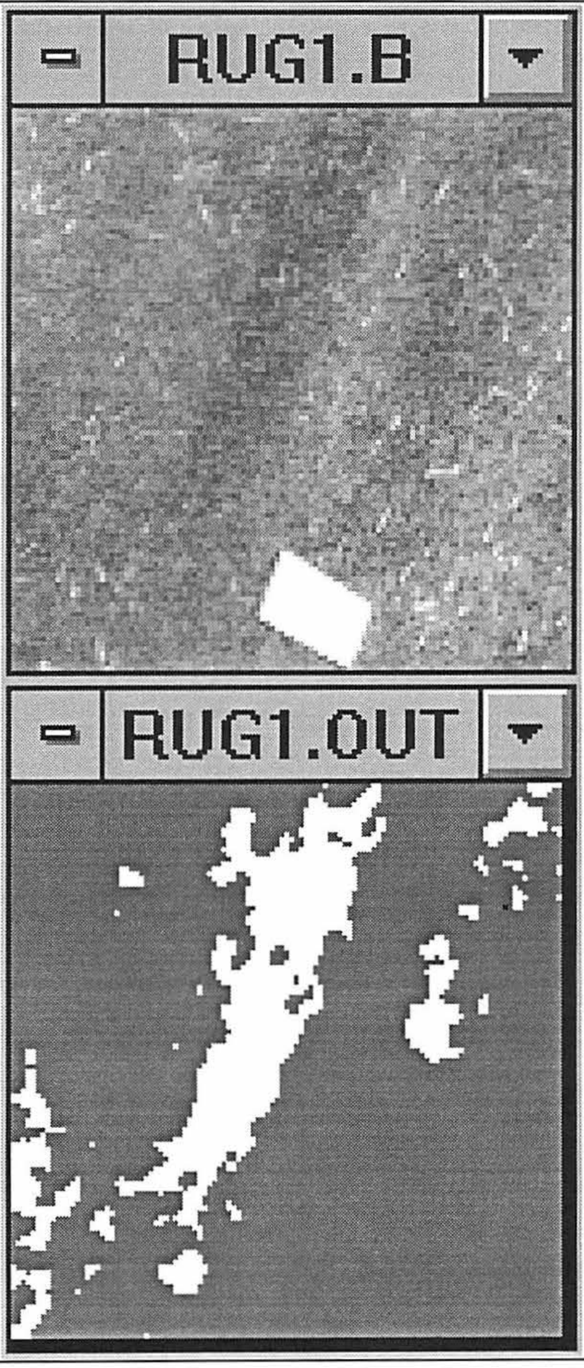


Fig 5.4
Identification of flawed rug regions using first order histogram features, feature support=10, grid size=4, and the single layer perceptron network. A total of six prototypes from this image were used in the classification process.

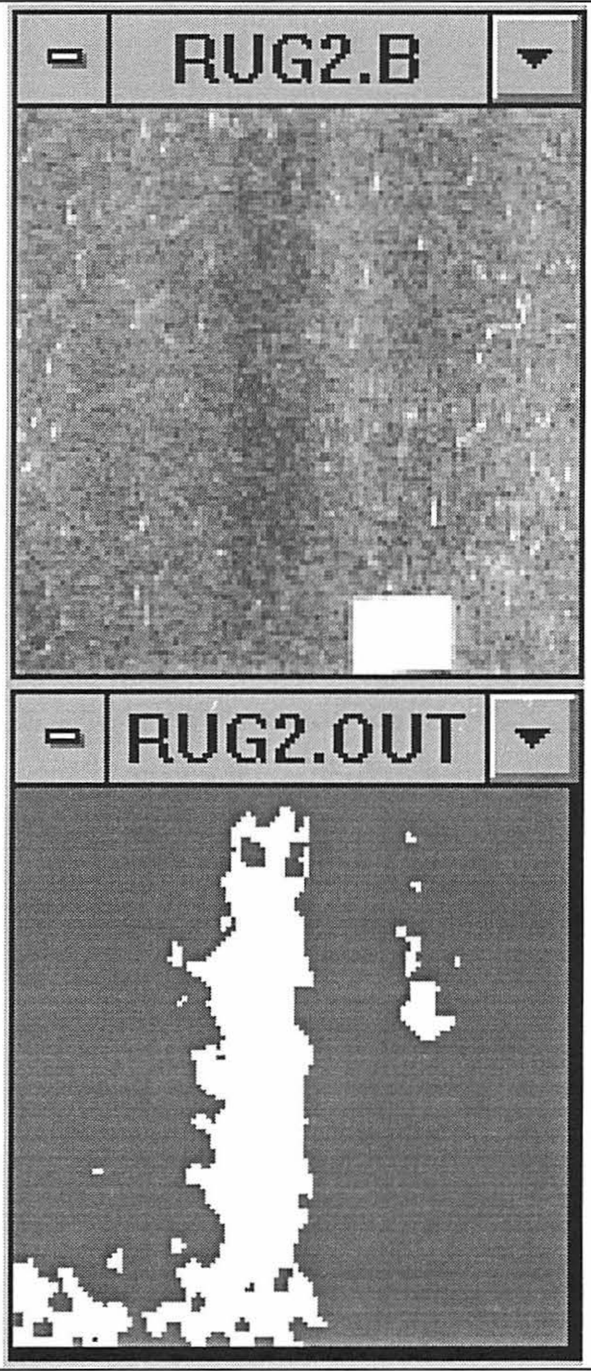


Fig 5.5
Identification of flawed rug regions using first order histogram features, feature support=10, grid size=4, and the single layer perceptron network. A total of six prototypes from the image in Fig 5.4 were used in the classification.

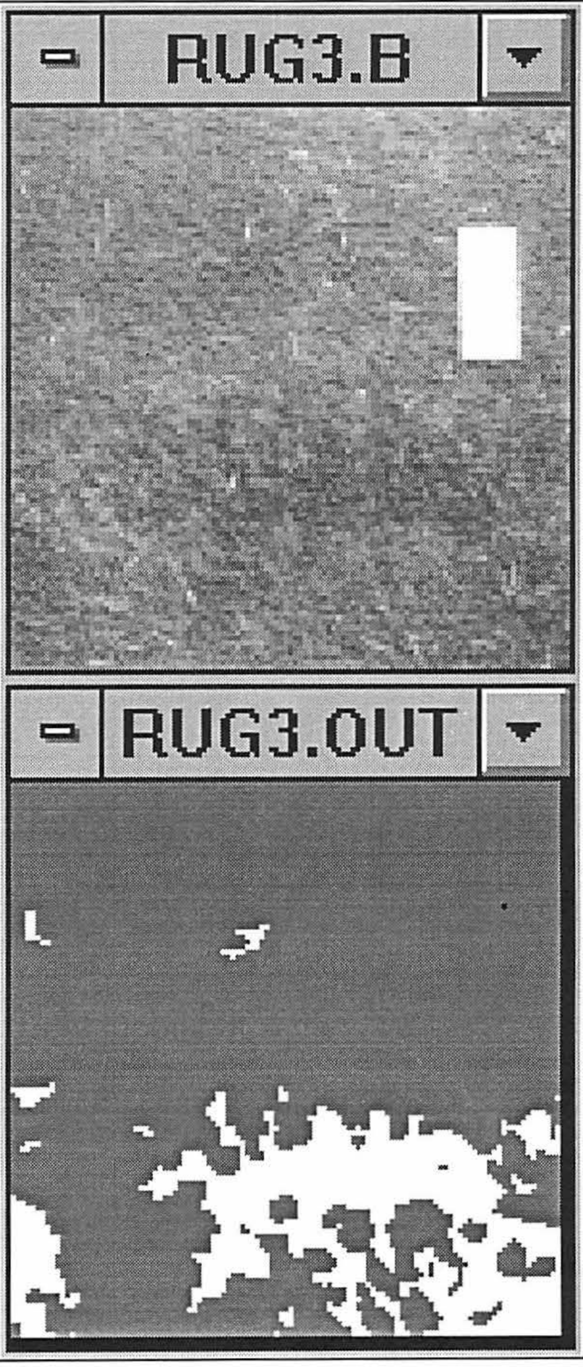


Fig 5.6
Identification of flawed rug regions using first order histogram features, feature support=10, grid size=4, and the single layer perceptron network. A total of six prototypes from the image in Fig 5.4 were used in the classification.

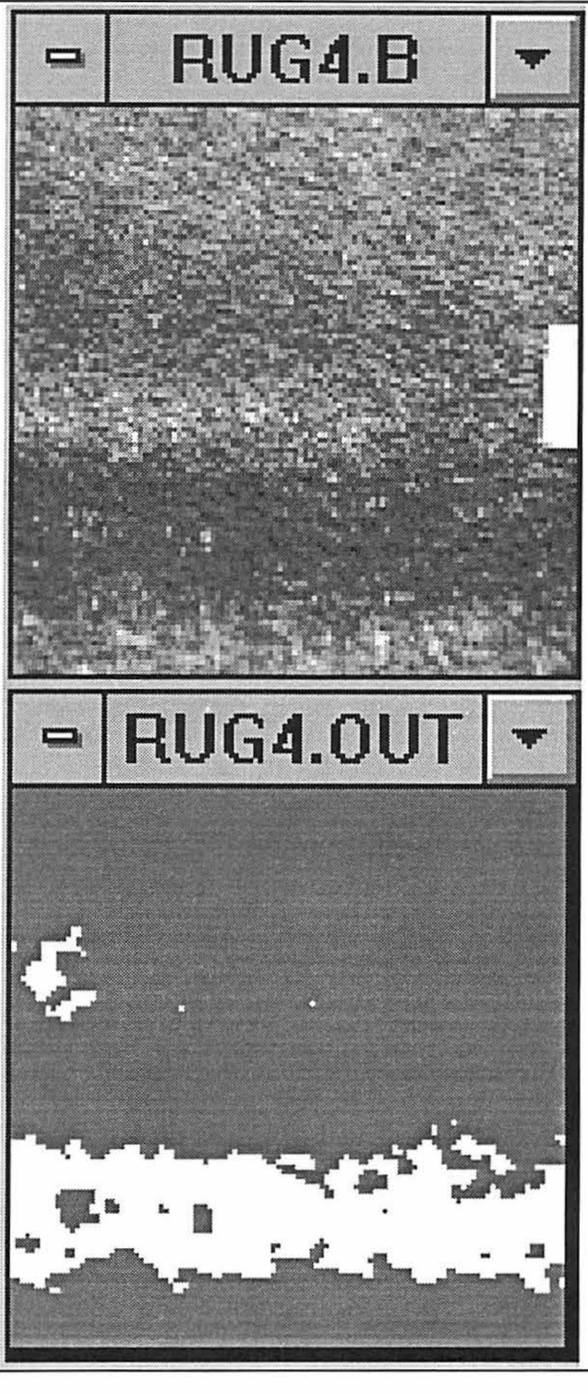


Fig 5.7
Identification of flawed rug regions using first order histogram features, feature support=10, grid size=4, and the single layer perceptron network. A total of six prototypes from the image in Fig 5.4 were used in the classification.

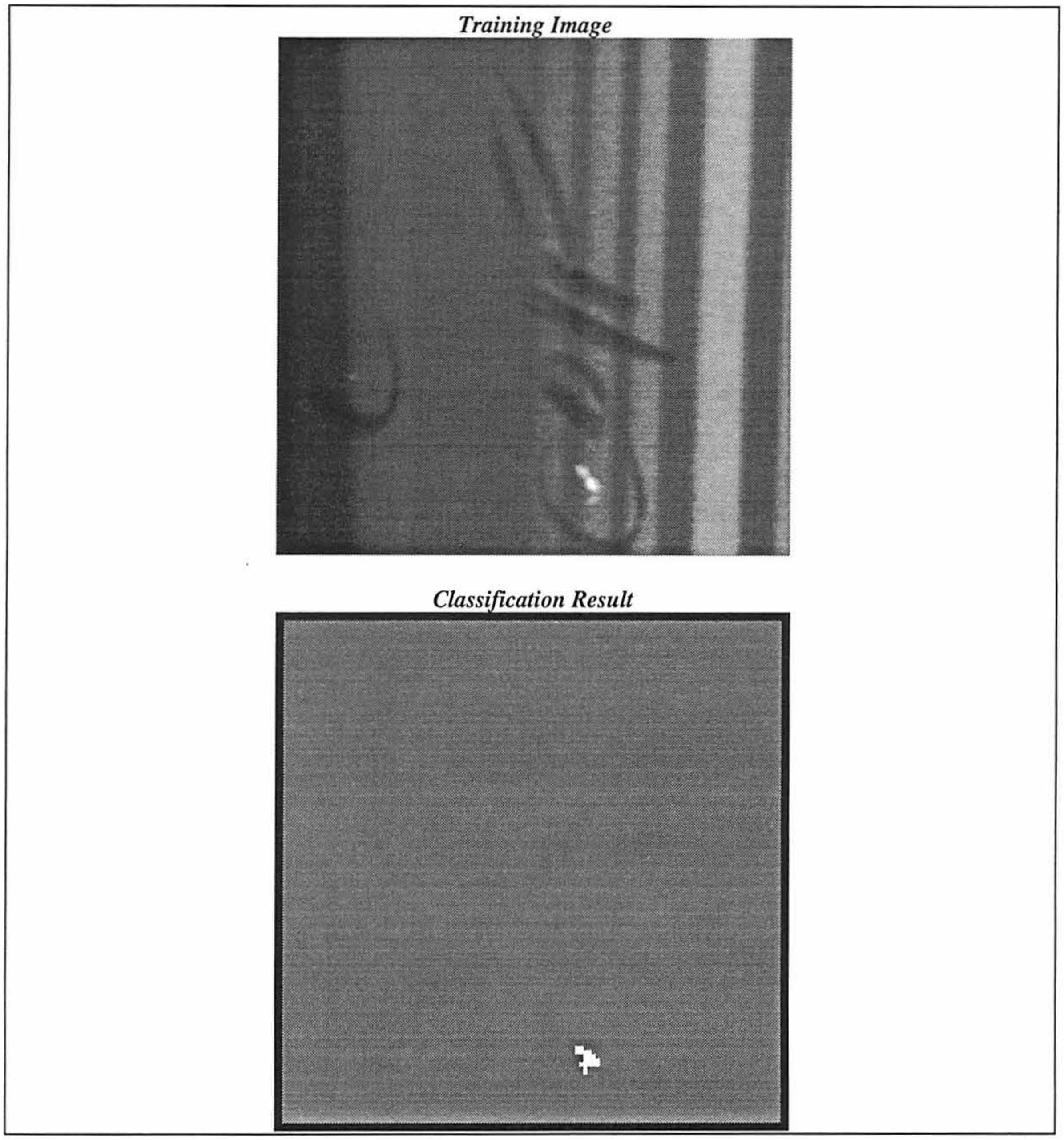


Fig 5.8
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image were used in the classification.

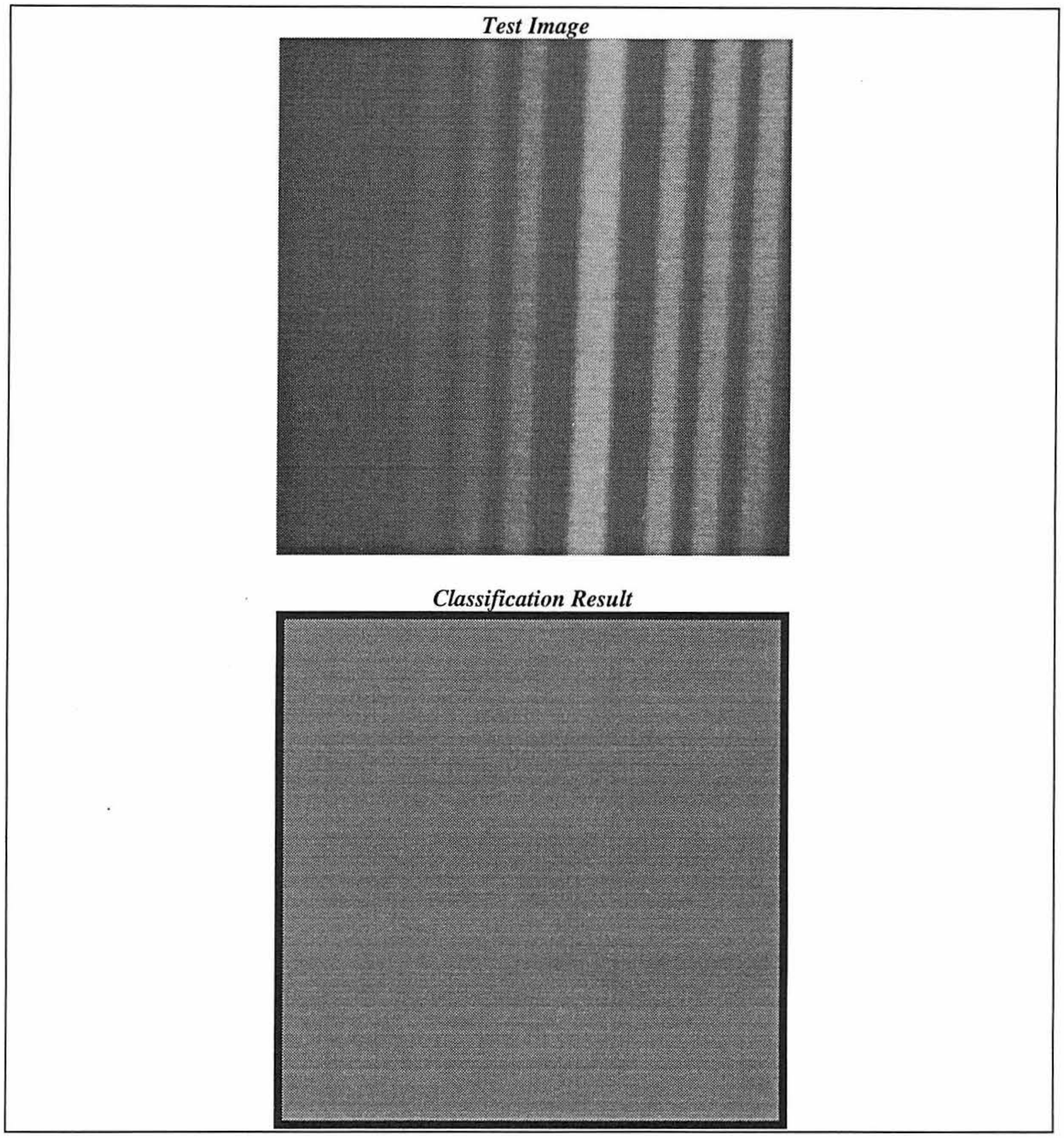
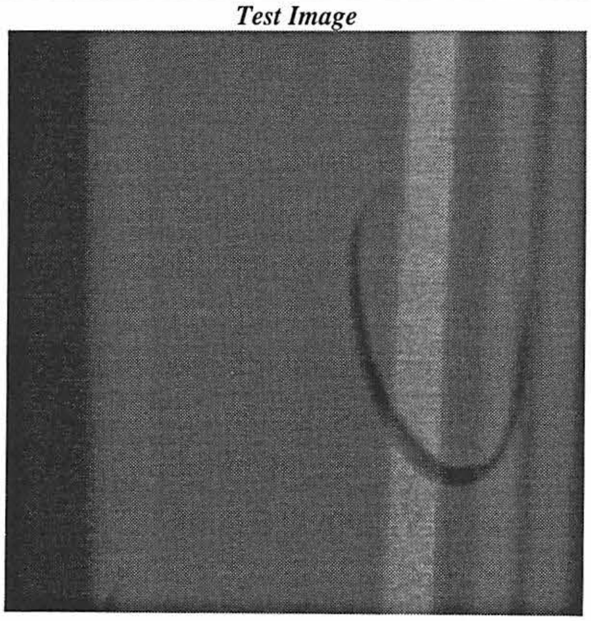


Fig 5.9
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a low-contrast test image with vertical bars, but no flaws.



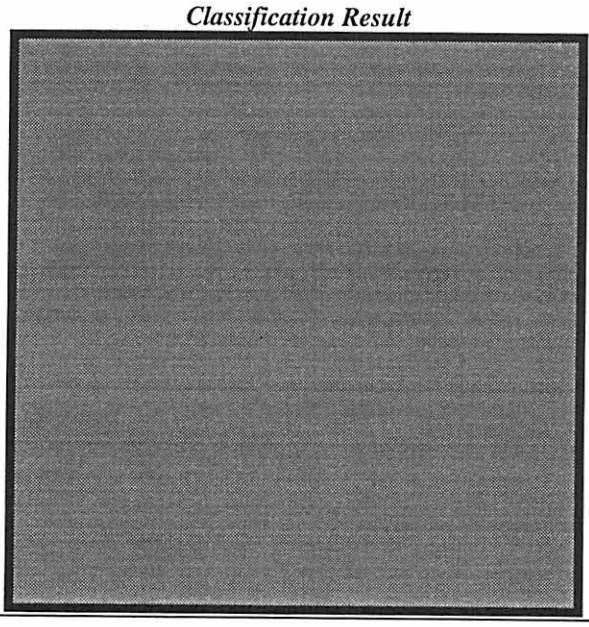
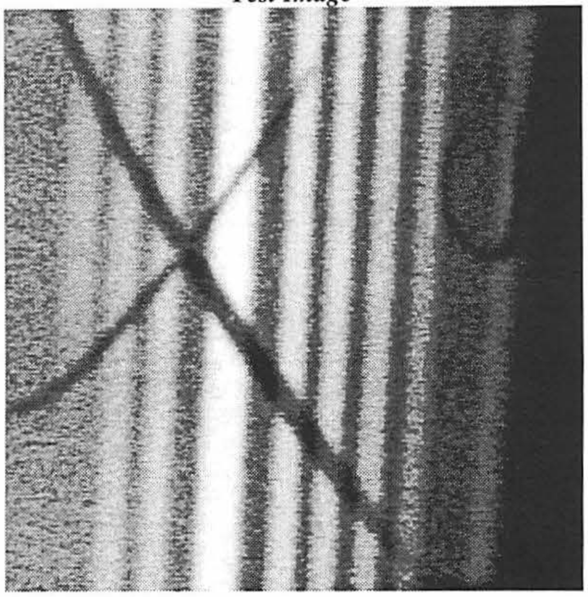


Fig 5.10
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a low-contrast test image with hand writing, but no flaws.



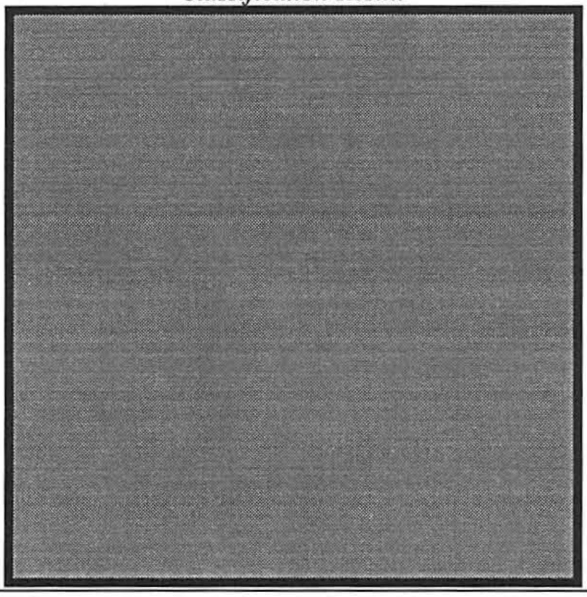


Fig 5.11

Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a high-contrast test image with hand writing, but no flaws. The results show that intensity alone can not be used to identify flaws.

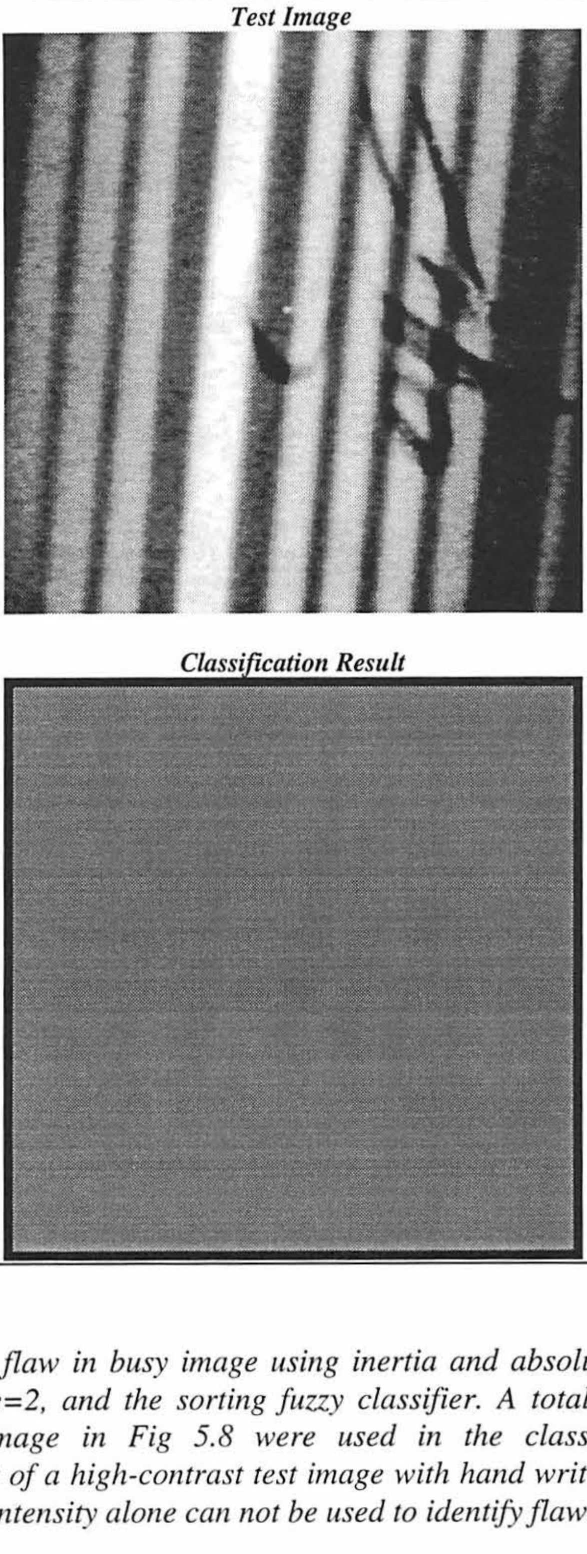


Fig 5.12
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a high-contrast test image with hand writing, but no flaws. The results show that the intensity alone can not be used to identify flaws.

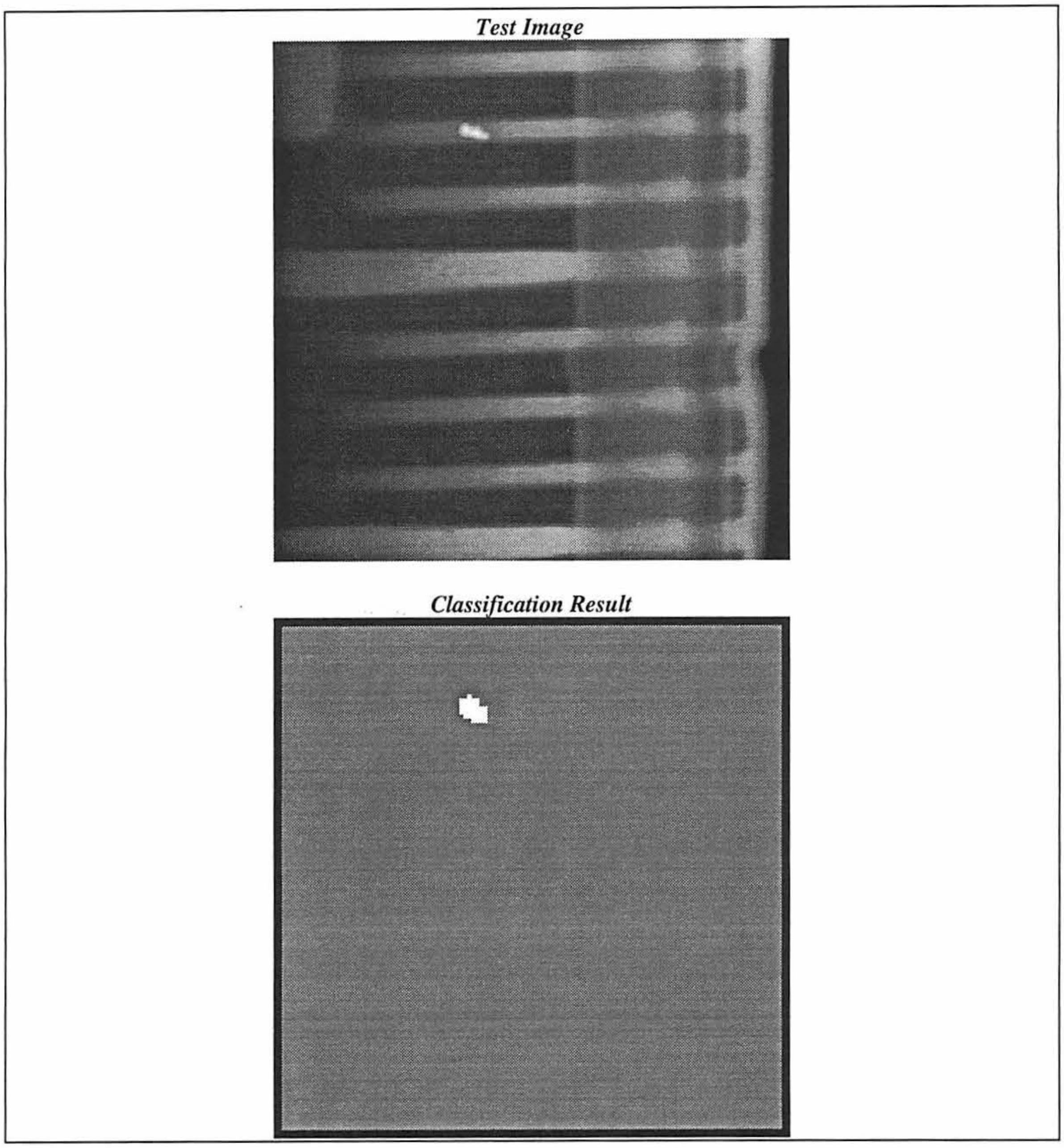


Fig 5.13
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a test image containing a flaw in upper half of the image. This flaw was found.

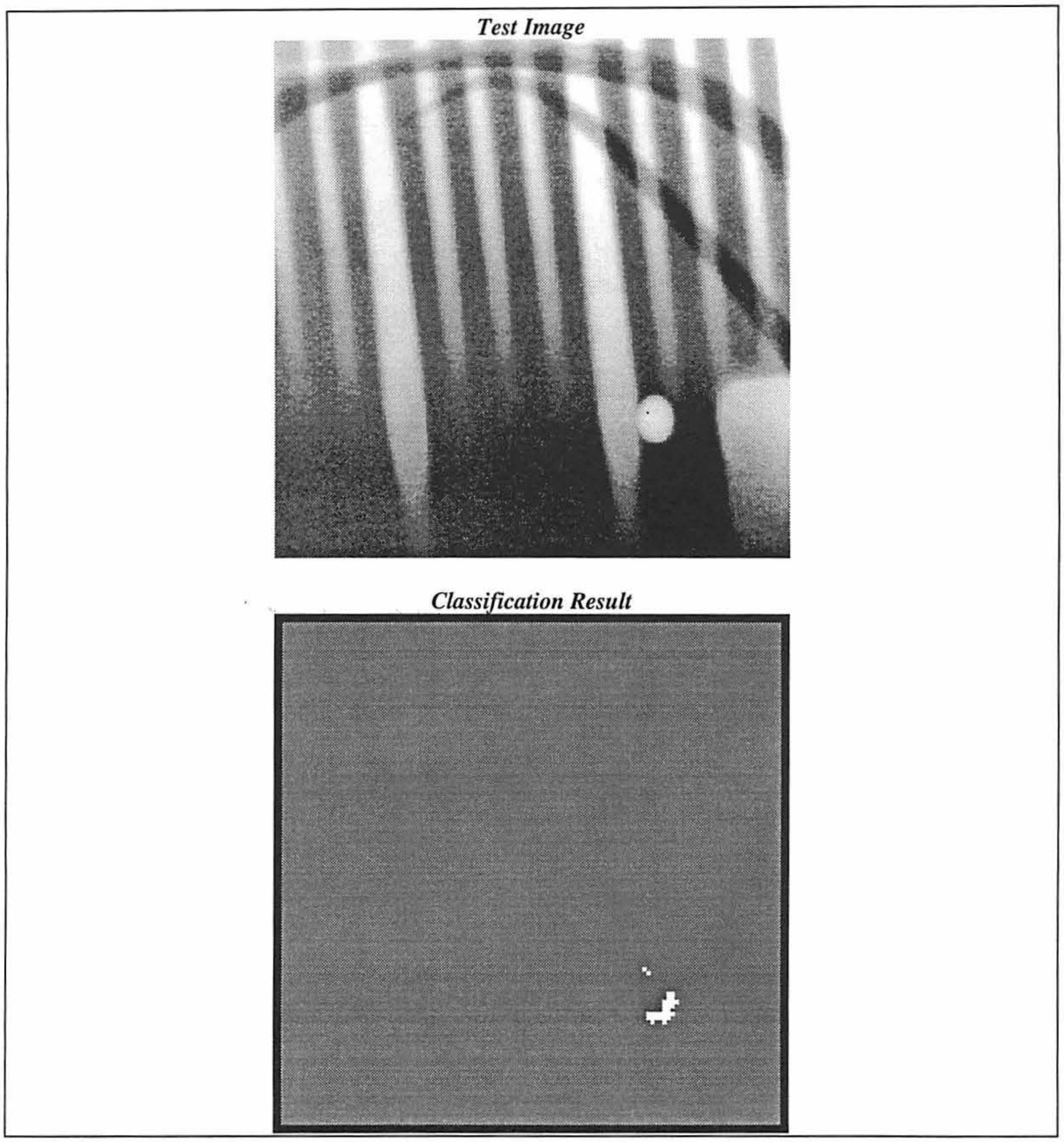


Fig 5.14
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a test image containing a flaw in bottom half of the image. This flaw was partially found.

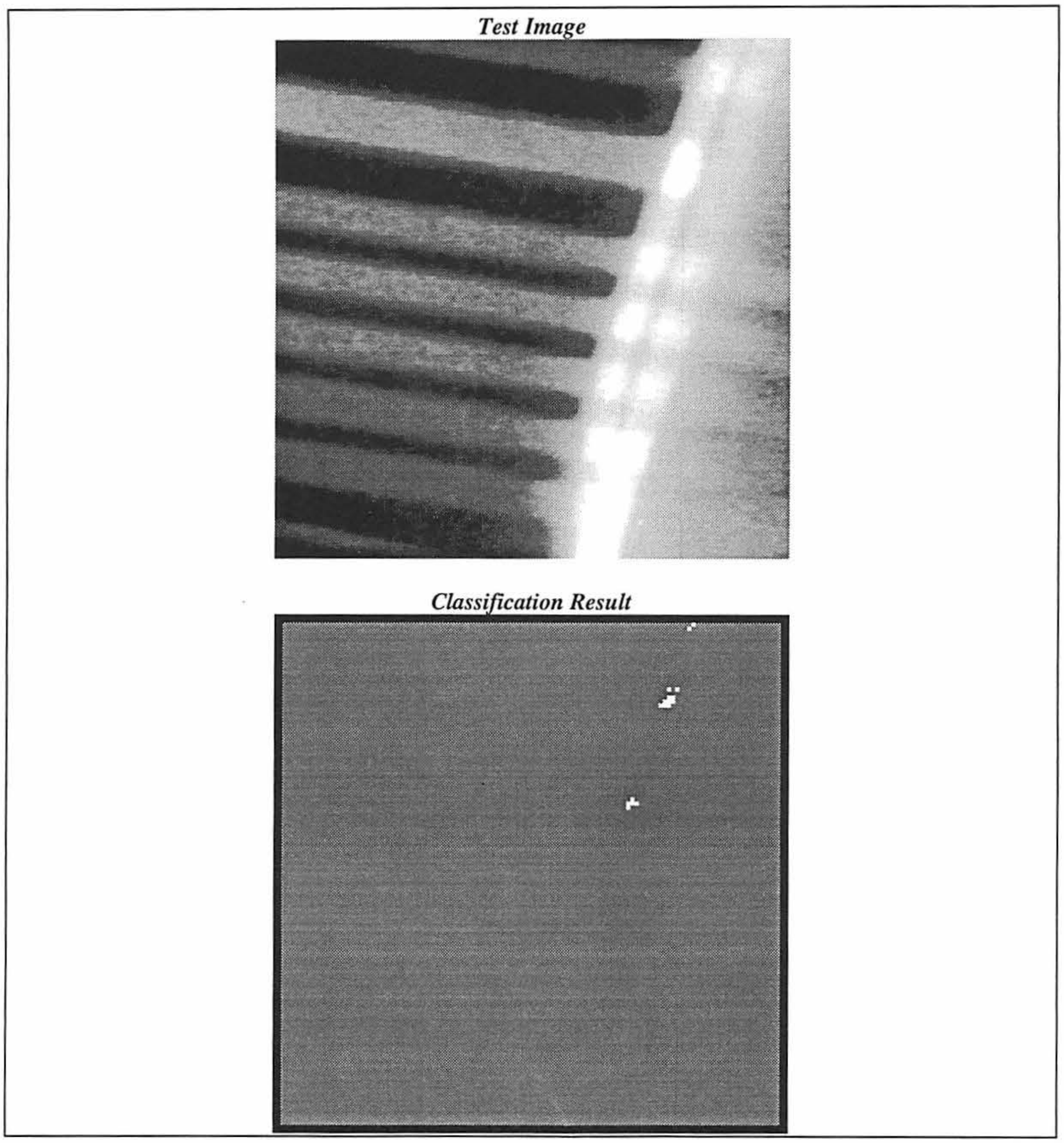


Fig 5.15
Identification of void flaw in busy image using inertia and absolute difference, feature support=10, grid size=2, and the sorting fuzzy classifier. A total of fifteen prototypes from the training image in Fig 5.8 were used in the classification. The above classification is result of a high-contrast test image containing no flaws. Flaws were incorrectly identified.

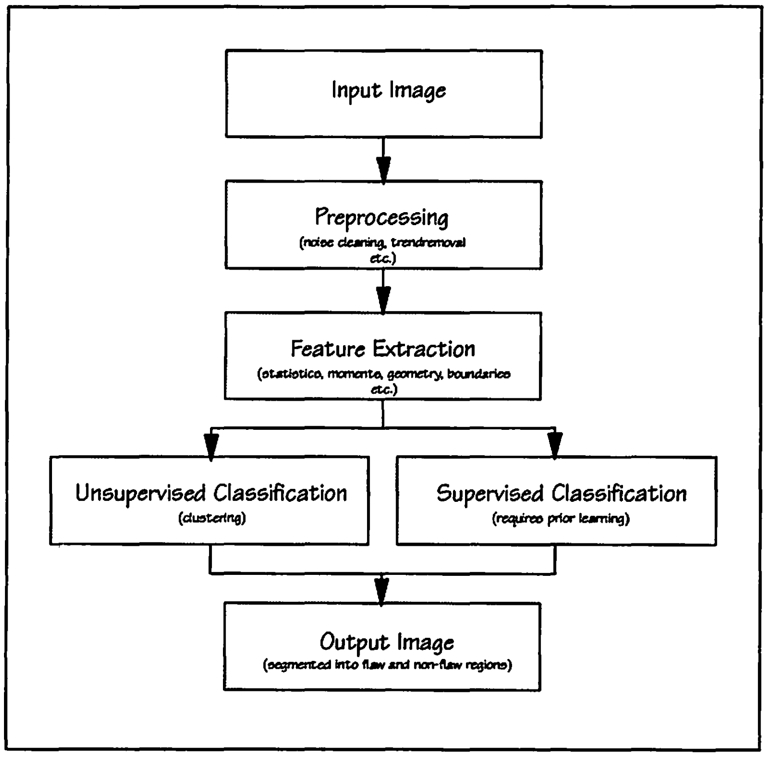
5.6. Discussion

The good results reported earlier in this chapter are very encouraging. These results suggest that it is feasible to make a general purpose flaw detection package that can be trained to solve a wide variety of detection problems.

The results of particular interest are the results from 5.4 and 5.5. In these sections, one image was used as training data, and based on a few prototypes from this image, the computer was set to inspect completely unknown images. Two things have to be considered in evaluating the classification of these two image series. Firstly, the training data in both cases was very limited. In an industrial setting one would perhaps use all vectors in twenty images to provide prototypes that cover more of the expected feature space. Secondly, the problems were very different. The rug problem was finding a low-contrast flaw, and the void-flaw was a high-contrast flaw that had to be found in a busy image. When the results were still very good it suggests that a modular pattern recognition scheme is a feasible way to make a general purpose flaw detection package.

6. USER MANUAL FOR SHERLOCK V1.1

6.1. Processing Principles


Input Image
Preprocessing
(noise cleaning, trendremoval etc.)
Feature Extraction
(statistics, moments, geometry, boundaries etc.)
Unsupervised Classification
(clustering)
Supervised Classification
(requires prior learning)
Output Image
(segmented into flaw and non-flaw regions)

Sequence of processing for identifying flaws using Sherlock:

1) Preprocessing (optional)
   Any type of processing of the image that will ease the task of identifying flaw regions.

2) Feature Extraction
   a) Choose feature support (Parameters→Feature Support)
   b) Choose grid size (Parameters→Feature Support)
   c) Choose type of feature set (Features→...)

3) Classification
   i) Unsupervised Classification
      a) Choose appropriate ***unsupervised*** classifier (K-mean or Fuzzy-C)
   ii) Supervised Classification
      a) Train
      b) Extract features to classify
      c) Classify

6.2. Menus

File	Description
Open...	Open PGM file from disk
Save...	Save active image as PGM file
Close	Close active image
Write	Write data to disk
Read	Read data to disk
Print	Print active image
Print Preview	Show print preview of active image
Print Setup...	Change printer setup
Exit	Terminate program

File→Write/Read	Description
Feature vector	Write/read extracted image descriptors
Training data	Write/read classified feature vectors
Neural net weights	Not implemented
Output nodes	Not implemented
Cluster centers	Not implemented
Output Image	Write/read segmented image in own ASCII format
Feature Map	Write/read image based upon a specified feature in own ASCII

Edit	Description
Copy Image	Copy active image to clipboard
Copy Features	Not implemented
Copy Classification result	Not implemented

View	Description
Toolbar	Toggle Toolbar
Status Bar	Toggle Status Bar
Feature Chart	Not implemented
Classification Chart	Not implemented
Feature Map	Creates a new image based upon a specified feature
Histogram	Creates a histogram based upon the active image
Toggle Grid	Toggle grid in active image
Zoom	Magnify or shrink active image

View→Zoom	Description
In	Magnifies active image using current zoom factor
Out	Shrinks active image using current zoom factor
Normal	Restore active image to original size
Change Zoom factor	Change zoom factor (default is 2)

ImgProc	Description
Noise Cleaning	Not implemented
Trend Removal	Not implemented
Geometry Removal	Not implemented
Histogram Equalization	Generate a histogram equalized version of active image
Edge detection	Not implemented
Unsharp masking	Not implemented
Smoothing	Not implemented
Thresholding...	Create a binary image at specified threshold value
To Img struct	Convert output image or feature map to an ordinary image

Features	Description
First order Histogram	Generate first order histogram based on active image
Second order Histogram	Generate second order histogram based on active image
Autocorrelation	Generate autocorrelation features from active image
Cosin Transform	Generate features based on the Cosine Transform
Walsh Transform	Generate features based on the Walsh Transform
Hadamard Transform	Generate features based on the Hadamard Transform
Zernike Moments	Not implemented
Karhunen Loeve	Not implemented
Wavelet	Not implemented
Geometry	Not implemented
Extract Characteristics	Extract feature-wise characteristics
Normalize	Normalize features
Feature Selection	Not implemented

Features→Extr. Char.	Description
Feature Mean	Calculate mean for each feature
Feature Sdev	Calculate standard deviation for each feature
Feature Max	Find max for each feature
Feature Min	Find min for each feature

Features→Selection	Description
Scale 0 to 1	Scale all features linearly to values between 0 and 1
Scale mean/sdev	Scale features by subtracting mean and divide by sdev

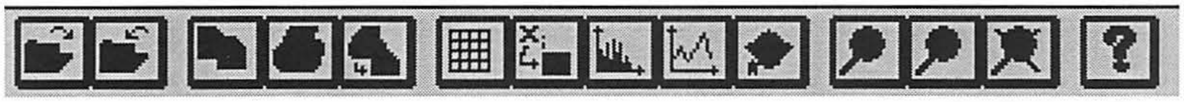
Classifiers	Description
K-mean...	Traditional clustering algorithm
Fuzzy-C...	Fuzzified version of K-mean
Min-Max Clustering	Not implemented
Nearest Neighbor	Simple supervised classifier
Perceptron	Single layer neural network
Sorting Fuzzy	Supervised fuzzy classifier
LMSE	Not implemented
Stochastic	Not implemented
Biased classifier	Not implemented
Histogram classifier	Not implemented
Min-Max Neural Net	Not implemented
Fuzzy Backprop	Not implemented
Mendel Classifier	Not implemented
Back Prop Neural Net	Not implemented

Parameters	Description
Feature Support...	Define square over which the features are extracted
Grid Size...	Define square which each feature vector represents
Grid Color...	Define grid color
TrainingFlag	Toggle TrainingFlag

Window	Description
Duplicate	Duplicate active image
Arrange Icons	Arrange icons at the bottom of the screen
Cascade	Cascade images
Tile Horizontal	Tile images horizontal
Tile Vertical	Tile images vertical

Special	Description
Merge...	Not implemented
Show Merge Line	Show the border in the included combine images

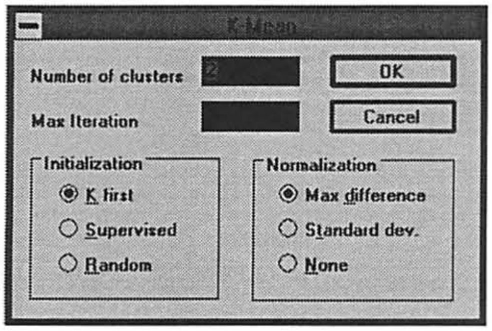
6.3. The Toolbar



a b c d e f g h i j k l m n

Button	Command	Description
a	File Open	Open PGM file on disk
b	File Save	Save active image as PGM file
c	Copy Image	Copy active image to clipboard
d	Print	Print active image
e	Print Preview	Show print preview of active image
f	Toggle Grid	Toggles grid
g	Create Feature map	Creates a new image based upon a specified feature
h	Create Histogram	Creates a histogram based upon the active image
i	Display Statistics	Not implemented. Has currently same function as d
j	Toggle Training Flag	Toggle training of active image
k	Zoom In	Magnifies active image using current zoom factor
l	Zoom Out	Shrinks active image using current zoom factor
m	No Zoom	Restore active image to original size
n	About	Displays information about copyright and tech. sup.

6.4. Dialog Boxes


K-Mean

Number of clusters 2

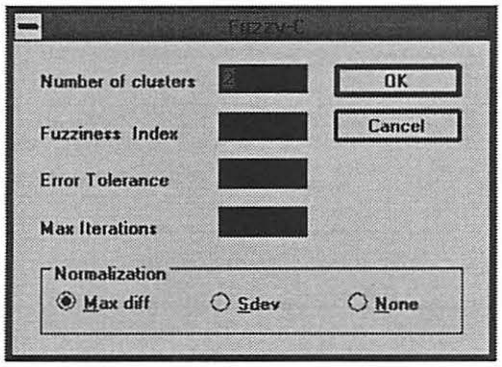
Max Iteration

Initialization: [✓ K first] Supervised Random

Normalization: [✓ Max difference] Standard dev. None

K-mean

Number of clusters:	Specifies the number of desired segment types in the output image. Two is commonly used to distinguish between flaws and non-flaws.
Max Iteration:	This specifies the maximum number of iteration before the iteration is terminated. A dialogbox will inform the user if the iteration is terminated without convergence.
Initialization:	Determine how the algorithm initially chooses initial cluster centers.
K first:	Choose the K first feature vectors
Supervised:	Use specified feature vectors identified using the training data routine
Random:	Choose k random feature vectors.
Normalization:	Attempts to give all features equal weight in the classification process.
Max difference:	Divide the difference between each feature in the cluster center and a feature vector with the maximum deviation from the mean for that particular feature.
Standard. dev:	Divide the difference between each feature in the cluster center and a feature vector with the standard deviation of that particular feature.
None:	No normalization.



Fuzzy-C

Number of clusters: 2

Fuzziness Index:

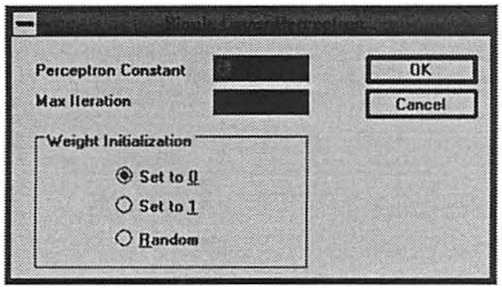
Error Tolerance:

Max Iterations:

Normalization: [✓ Max diff] Sdev None

Fuzzy-C

Number of clusters:	Specifies the number of desired segment types in the output image. Two is commonly used to distinguish between flaws and non-flaws.
Fuzziness Index:	Must be larger than 1. The larger the value the more fuzziness is assigned to the feature vectors.
Error Tolerance:	Terminate iteration if error is smaller than this value
Max Iteration:	Terminate iteration if number of iterations exceeds this value
Normalization:	See K-mean



Single Layer Perceptron

Perceptron Constant:

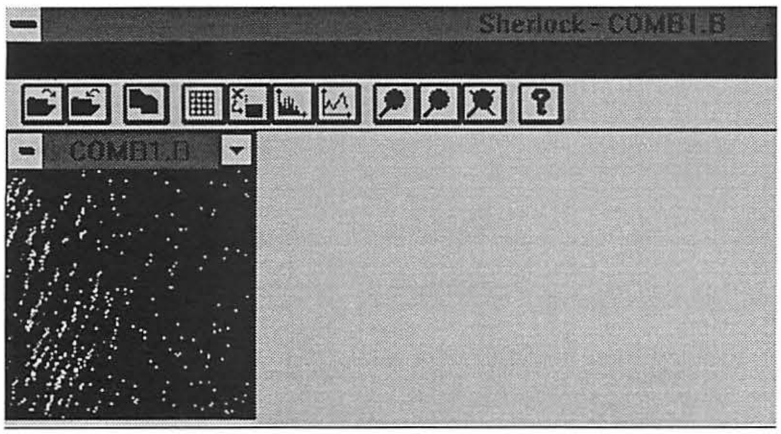
Max Iteration:

Weight Initialization: [✓ Set to 0] Set to 1 Random

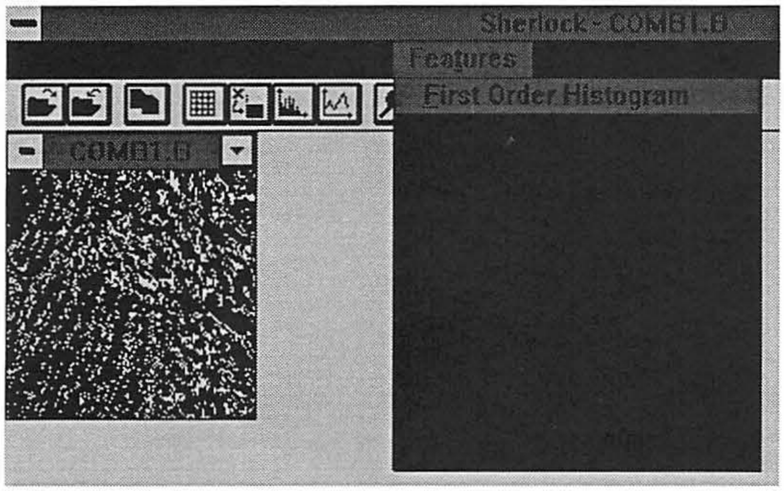
Perceptron

Perceptron Constant:	Positive constant. The larger this constant is, the longer search jumps are made in each iteration.
Max Iteration:	Terminate iteration if number of iterations exceeds this number. Should be at least a seven digit number.
Weight Initialization:	Determines how to initialize the weights.

6.5. Example of Basic Processing



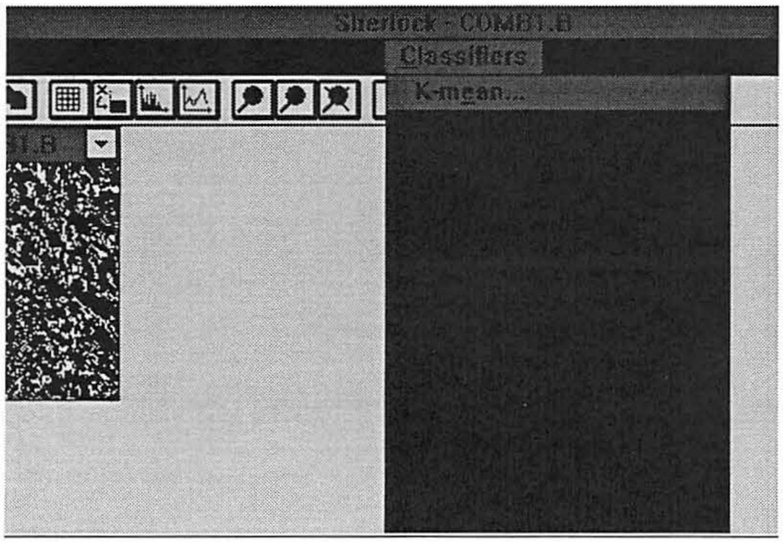
LOADING AN IMAGE
comb1.b is loaded by choosing File➔Open or button #1



EXTRACTING FEATURES
Features are extracted by choosing one of the three first options in the the features menu.

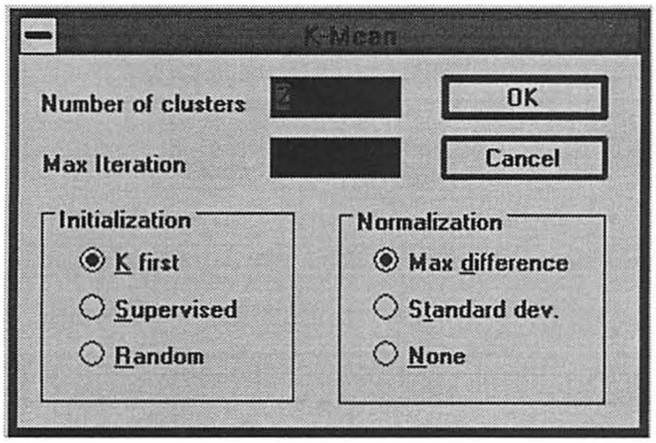
For our example we choose "First Order Histogram"

NOTE: Extracting features can take some time, especially for large images.



UNSUPERVISED CLASSIFICATION
The feature vectors are classified by choosing one of the classifiers in the menu seen to left.
The classification result is then used to generate a segmented image.

For our example we choose K-mean



K-Mean

Number of clusters: 2

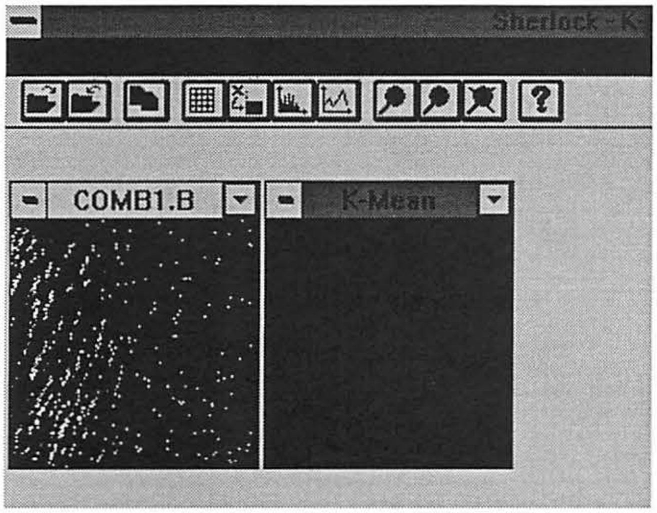
Max Iteration:

Initialization: [✓ K first] Supervised Random

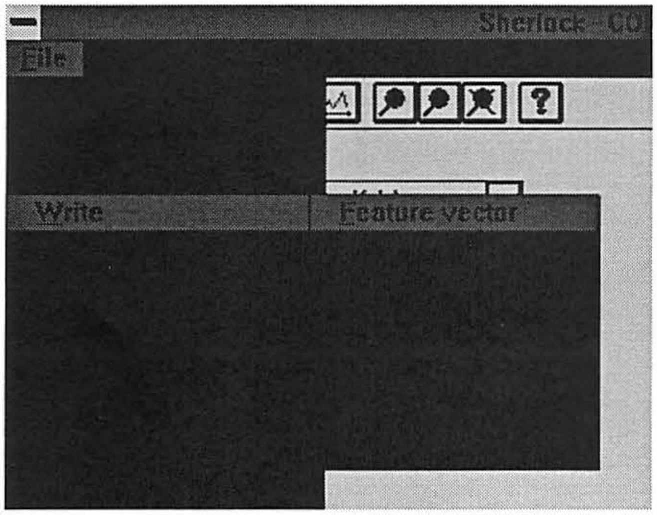
Normalization: [✓ Max difference] Standard dev. None

OK Cancel

Choosing K-mean pops up a dialog box. Since the image we have opened has two different textures only, we accept 2 clusters. The other parameters are also accepted as they are. Hit the OK button.

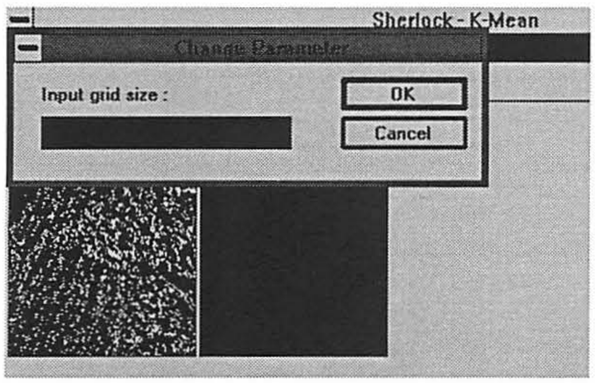


After a short time, an output image should be displayed. The edge between the two textures are quite jagged. This is due to the grid size parameter. To get a smoother edge the grid parameter has to be set smaller which gives a higher resolution. The cost of this is more computation.



SAVING FEATURE VECTORS
Next operation in our example is to store the current feature vectors which will be overridden next time a new set of features are extracted. Saved feature vectors can be retrieved from disk later.

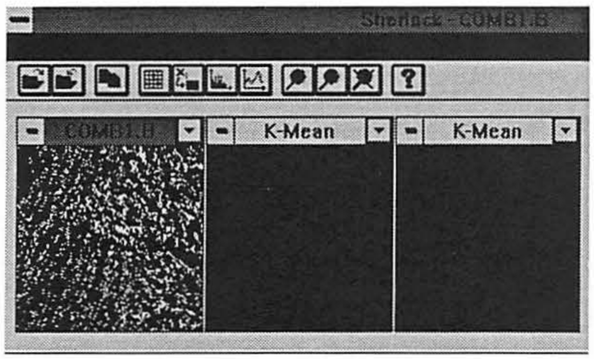
To save feature vectors choose "File➔Write➔Feature vector"



CHANGING GRID PARAMETER
To increase the resolution in the output image, one can increase the grid size parameter in the parameter menu.

To get a smoother edge in our example we change the grid size from the default 4 to 2. We then extract features again, because the old features are not valid since the grid size is changed.

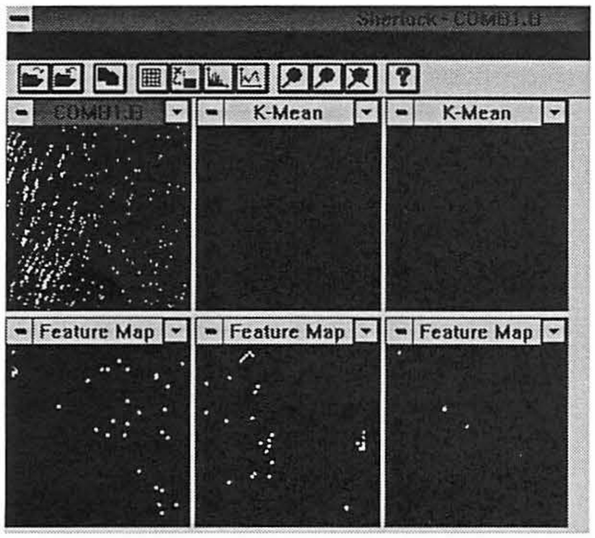
NOTE: Click comb1.b with the mouse before extracting new features. This is necessary to tell the program from which image it is supposed to extract features



After extracting new features (still first order histogram) and classified these by K-mean, we should get another output image as shown to left. This image clearly has a smoother edge. It is probably a good idea to store new feature vectors for later use.

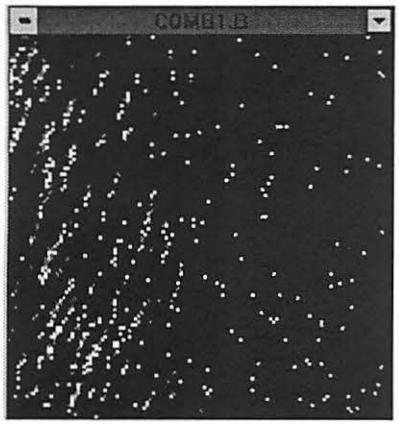
To compare with the actual edge, one can choose "Special→Show Merge Line". (The result of this operation is not shown.)

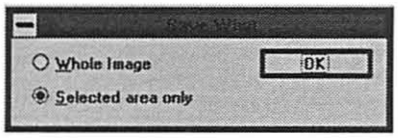
DISPLAYING FEATURE MAPS
Feature maps are useful to determine how well the features have separated two classes in feature space. Each feature map is created from the values of one individual feature. This image then tells how well this particular feature distinguishes between different regions in the image.

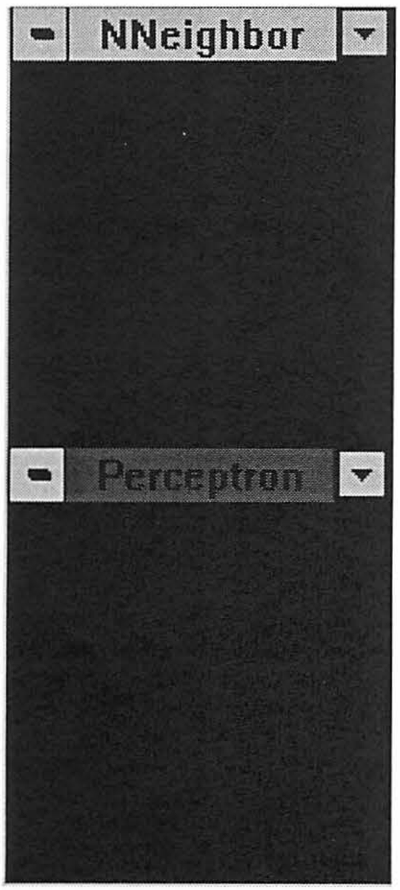


Displaying features can be done either by choosing "View→FeatureMap" or by clicking the 7th button in the toolbar. Desired feature can then be selected in a dialog box.

To the left is shown the three first features of a total of six for the image comb1.b using a 2x2 grid and a feature support of 10x10.







GENERATING TRAINING DATA
Next step is to generate training data so we can try supervised classifiers which in general are more powerful.
To extract training data we have to

1) Extract features
2) Set training flag
3) Identify classes
4) Save to file

For our example:
1) Load old features which we previously stored
2) Set training flag for active image by clicking comb1.b and choosing Parameters→TrainingFlag
3) Use the mouse to identify two separate classes as done in illustration to left. Classes are separated by color. Colors are chosen by # of left-button mouse clicks. Right mouse button clears the color assignment. Which color you assign to each class doesn't matter. All that matters is that the classes have different colors.
4) Save training data by
   a) File→Write→Training Data and specifying file name
   b) Choosing "Selected area only" in the next dialog box. The option "Whole Image" would have created a third class, namely the background, and stored all the feature vectors as training data.

THE NEAREST NEIGHBOR CLASSIFIER
Next we want to classify the image using the NN classifier.
1) Load prototypes
   Choose Classifiers→Nearest Neighbor→ReadPrototypes
   In the file dialog box, type in the name of the training data file.
2) Load/extract feature vectors that you want to classify. For our example we already have features loaded.
3) Classify feature vector and generate an output image.
   Choose Classify→Nearest Neighbor→Classify

To left is shown output using max dev as normalization.
Perfect border is displayed by Special→Show Merge Line

THE PERCEPTRON CLASSIFIER
1) Load training data
   Choose File→Read→Training Data
2) Train the Perceptron algorithm
   Choose Classifiers→Perceptron→Train
3) Load/Extract data to classify.
   Choose File→Read→FeatureVector or extract new features.
   Type of features must correspond with the type of features used in the training data
4) Classify
   Choose Classifiers→Perceptron→Classify

6.6. System Requirements

Computer:	IBM compatible
Processor:	486DX33 or better
RAM:	4 MByte or more
Viedeo Card:	256 colors or more
Software:	MS Windows 3.1 or higher

6.7. Technical Support

Contact:

Jorn Lyseggen
Iowa State University
310 Durham
Ames, IA 50011

Phone office:	(515) 294-4955
Phone home:	(515) 232-3530

E-Mail: lys@iastate.edu

7. CONCLUSION AND FUTURE WORK

In this thesis, a new prototype system for automatic image-based general purpose flaw detection has been presented. The system is based on a modular pattern recognition scheme, a strategy that to the authors knowlede has not been tried before in automatic flaw detection.

Another important aspect of this system is that once it has been trained it can run completely unsupervised.

Versatility in this system is achieved by incorporating a wide variety of feature sets and a battery of different classifiers. Using these general techniques, the system can be tailored to solve detection problems in any image modality.

The system has been demonstrated for very different types of images and detection problems. Encouraging results have been reported in:

- texture discrimination in optical images
- identification of galaxes in infra red images
- detection of shrinkage cracks in x-ray images
- identification of flawed rug areas in optical images
- detection of weld flaws in x-ray images.

Of these, the two latter was done in a series of images. One of the series were low-contrast flaws in noisy optical images, and the second serie was a high-contrast flaw in busy x-ray images. The results of this classification were very good which demonstrates the feasibility of developing a reliable automatic inspection system by the proposed pattern recognition scheme.

The reported prototype system has been licensed by the Center for Advanced Technology Development and shipped to a dozen of the sponsors at the NDE center. Very positive feedback has been received.

Improvements that can be made in future work are:

- development of more and better feature extractors
- development of a macro language that will automate execution of succesive commands
- development of a module that automatically generates inspection reports
- implementation of time consuming algorithms on a DSP board to enable real-time inspection
- development of a confidence measure that tells how confident any classification is

All of these will be undertaken by the author of this thesis after graduation.

BIBLIOGRAPHY

Aloimonos J. and Schulman D. (1989), *Integration of Visual Modules,* Academic Press Inc., Boston

Basart J., Zhang Z., and Lyseggen J., (1993) *Automatic Flaw Detection,* Review of Progress in Quantitative Nondestructive Evaluation, Vol. 13A, D.O. Thompson and D.E. Chimenti, Plenum Press, New York

Bezdek, J. C. (1982), *Pattern Recognition with Fuzzy Objective Function Algorithms,* Plenum Press, New York

Brodatz, P. (1956), *A photographic album for artisits and designers,* Dover, New York

Branco S.(1992), *Fast Learning and Invariant Object Recognition,* John Wiley & Sons, New York

Brown, R. G. and Hwang, Y. C. (1985), *Introduction to Random Signals and Applied Kalman Filtering,* John WIley & Sons, New York

Brown, R. A. (1992), *Image processing workstation software development and feature size measurement methods for NDE X-ray images,* M. S. Thesis, Iowa State University

Chen, C. H. (1973), *Statistical Pattern Recognition,* Hayden Book Company, Rochelle Park, New Jersey

Dayhoff, Judith E. (1990), *Neural Network Architectures,* Van Nostrand Reinhold, New York

Dasarathy B. V. (1990), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques,* IEEE Computer Society Press, Washington

Doering E. R. (1987), *Detection of anomalies in digital images using pixel classification*
M. S. Thesis, Iowa State University

Duda, R. and Hart, P. (1973), *Pattern Classification and Scene Analyses,* New York: Wiley

Fukunaga, K. (1972), *Introduction to statistical pattern recognition,* Academic Press, New York

Gatot, C. (1988), *Feature Extraction in medical radiographic images,* M.S. Thesis, Iowa State University

Gonzales, R. W. and Woods, R. E (1991), *Digital Image Processing,* Addison Wesley Publishing Company, New York

Grasselli, A. (1969), *Automatic Interpretation and classification of images,* Academic Press, New York

Gray, J. N. and Inanc, F. (1990), *A CAD Simulation Tool for X-ray NDE Studies,* Review of Progress in Quantitative Nondestructive Evaluation, Vol. 9A, D.O. Thompson and D.E. Chimenti, New York: Plenum Press, pp 391-398

Halmshaw, R. (1987), *Nondestructive Testing,* London, England: Edward Arnold

Haralick R. M. and Shapiro L. G. (1990), *Computer and Robot Vision,* Addison Wesley Publishing Company, New York

Hopfield, J. J. (1982), *Neural networks and physical systems with emergent collective computational abilities,* Proc. Natl. Acad. Sci. 79: pp 2554-58

Julesz, B. (1962), *Visual Pattern Discrimination,* IRE Trans. Information Theory, IT-8, 1, February 1962, pp 84-92

Julesz, B. (1973), *Inablitiy of Humans to Discriminate Between Visual Textures That Agree in Second Order Statistics - Revisited,* Perceptrion, 2, 1973, pp 391-405

Kandel, A. (1982), *Fuzzy Techniques in Pattern Recognition,* John Wiley & Sons, New York

Kasturi, R and Jain R. C. (1991), *Computer Vision: Principles,* IEEE Computer Society Press, Whasington

Keller, J. A. and Hunt, D. J. (1985), *Incorporating Fuzzy Membership Functions into the Perceptron Algorithm,* IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-7, no. 6, pp. 693-699

Kosko, B. (1992), *Neural Networks and Fuzzy Systems,* Prentice Hall, Englewood Cliffs, NJ

Li, Z. C. (1989), *Computer Transformation of Digital Images and Patterns,* World Scientific, London

Lippmann, Richard P. (1987), *An introduction to computing with neural nets,* IEEE ASSP Magazine, pp 4-22 (April)

Lyseggen, J. and Basart, J. (1994), *A MS Windows Package for Automatic Image Based Flaw Detection,* Proceeding of 3rd Annual Midwest Electro-Technical Conference, April 8-9, pp 39-42

Lyseggen, J. and Basart, J. P. (1994), *Automatic Flaw Detection Using 2nd order Statistics and Fuzzy Logic.,* Review of Progress in Quantitative Nondestructive Evaluation, Vol. 14, D.O. Thompson and D.E. Chimenti, Plenum Press, New York. To be published.

Moharir, P. S. (1992), *Pattern-Recognition Transforms,* John Wiley & Sons Inc., New York

Meisel, W. (1972), *Computer-Oriented Approaches to Pattern Recognition,* Academic Press, New York

Nadler M, and Smith E.P. *Pattern Recognition Engineering,* John Wiley & Sons Inc., New York, 1992

Van Otterloo, P. J. (1990), *A coutour-Oriented Approach to Shape Analyses,* Prentice Hall, New York

Oppenheim, A.V. and Schafer R. W. (1989), *Discrete Time Signal Processing,* Prentice Hall, New York

Patrick, E. A. and Fattu, J. M. (1986), *Artificial Intelligence with Statistical Pattern Recognition,* Prentice-Hall, Englewood Cliffs, New Jersey

Pratt, W. K. (1991), *Digital Image Processing,* John Wiley & Sons Inc, New York

Rosenblatt, F. (1957), *The perceptron: A perceiving and recognizing automation,* Cornell Univ., Ithaca, NY, Project PARA, Cornell Aeronaut. Lab. Report. 85-460

Rummelhart, D. E. and McClelland, J. L. (eds.) (1986), *Parallel Distributed Processing, vol 1,* M.I.T. Press, Cambridge, MA

Sankar K. P. (1986), *Fuzzy Mathematical Approach to Pattern Recognition*
John Wiley & Sons, New York

Siwek, E. M. (1994), *Application of the X-ray measurement model to image processing of X-ray radiographs.* M.S. Thesis, Iowa State University

Specht, Donald F. (1990), *Probabilistic Neural Networks,* Neural Networks 3:109-118

Tou, J.T. and Gonzalez R. C. (1974), *Pattern Recognition Principles,* Addison-Wesley Publishing Company, Massachusetts

Ulmer, K. W. (1992), *Automated Flaw Detection Scheme for X-ray Image in Nondestructive Evaluation,* M.S. Thesis, Iowa State University

White, H. (1989), *Learning in Artificial Neural Networks: A Statistical Perspective,* Neural Computation, vol. 1, no. 4, 425-469, Winter 1989

Wilson, R. and Spann M. (1988), *Image Segmentation and Uncertainty,* John Wiley & Sons Inc., New York

Wang, Z. and Klir G. J. (1992), *Fuzzy Measure Theory,* Plenum Press, New York

Wong, C.Y. (1987), *Regression filtering and edge detection,* M.S. Thesis, Iowa State University

Zadeh, L. A. (1965), *Fuzzy sets,* Inform. Contr. vol 8, pp. 338-353

Zheng, Y. (1987), *Image analyses, modeling, enhancement, restoration, feature extraction and their application to NDE and astronomy,* Ph.D. Thesis, Iowa State University