

TEXPERT:  
AN APPLICATION OF MACHINE LEARNING TO TEXTURE RECOGNITION

BY

THOMAS CHANNIC

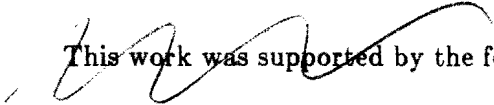
B.S., Maharishi International University, 1982

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

## ACKNOWLEDGEMENTS



This work was supported by the following agencies and grants:

— This work could not have been started, completed, or anything else without the help of my advisor, R.S. Michalski. Professors Robert E. Stepp and Arthur Baskin were also supportive, if not helpful, understanding, patient and downright nice guys. My fellow graduate students have always been an inspiration, particularly Carl Uhrik, and Stephen Borodkin, who not only gave useful advice but also were real regular nice guys.

Also worth mentioning are Wayne Woodmansee and Bill Motzer of Boeing for supplying ultrasound images, who, although I didn't get to know too well, did seem like reasonably nice guys.

And then there are all those whose contributions simply transcend the nice guy label: Richie, Maharishi, Gabriel, Michael and his eternal partner, D-E-F the second, Brother (in every sense) Paul, Mom, Dad, Plunk, Spunk, Carmen, and Professor Liz Dolske the Great, the most superconscious person ever to walk the halls of the Department of Computer Science at the University of Illinois.

## TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION .....	1
1.1. The Knowledge Transfer Continuum .....	2
1.2. Texture Recognition as Image Segmentation .....	4
1.3. Statistical Methods .....	5
1.4. Overview of the Thesis .....	6
2. TEXTPERT ALGORITHMS AND METHODS .....	8
2.1. Learning Algorithm .....	11
2.1.1. Features for Learning Texture .....	13
2.1.2. Iterative Learning .....	16
2.1.3. Incremental Learning .....	18
2.2. Testing Method .....	19
2.3. Image Pre-Processing .....	20
2.3.1. Averaging .....	20
2.3.2. Gray-Level Reduction .....	20
2.3.3. Event Generation .....	21
3. TEXTPERT ARCHITECTURE AND IMPLEMENTATION .....	22
3.1. Image Classification and Processing .....	22
3.1.1. Classifying an Image .....	22
3.1.2. Other Processing .....	24
3.2. Event Generation .....	24
3.3. Rule Acquisition and Induction .....	26
3.4. Rule Application .....	28
3.5. Image Regeneration .....	29
3.6. Transformed Image Testing .....	30
3.7. Implementation Summary .....	30
4. EXPERIMENTATION AND RESULTS .....	32
4.1. Learning Texture Using Iterative Learning .....	32
4.2. Learning Texture Using Incremental Learning .....	39
5. CONCLUSIONS .....	49

5.1. Effectiveness of Learning in Vision .....	49
5.2. Directions for Future Research .....	50
5.2.1. Descriptive Language Capability .....	51
5.2.2. Extended Automation Features .....	51
5.2.3. Statistical Image Processing .....	53
5.3. Final Remarks .....	53
APPENDIX A. Experiment Rule Output .....	55
A.1. Iterative Learning .....	55
A.2. Incremental Learning .....	57
REFERENCES .....	60

## CHAPTER 1.

### INTRODUCTION

Computers outperform all humans at certain tasks, long division and reliable storage of large quantities of data, to name a few. They outperform most humans at other tasks, like playing chess and diagnosing soybean diseases for example. Unfortunately there are many tasks which computers perform far worse than most humans, like understanding a human language and recognizing objects or other intelligible collections of information from a visual field. Because computers are utterly inferior to humans in these areas, the areas are appropriately included under the term artificial intelligence (AI).

Getting a computer to make sense out of electromagnetic waves (also known as computer vision) has often been considered as not really AI, but more of an engineering or an industrial problem. The most likely reason for this bias is that the problem involves a non-trivial transformation from an apparently continuous spectrum of light into discrete digital information suitable for processing by a computer. Nevertheless, the processing of digital information is by no means less trivial than the analog-to-digital (A/D) conversion of the image. If computer vision researchers agree on anything, they agree that the key to designing a computer that sees depends more on the intelligent manipulation of digital visual information than on how that information is derived from its source.

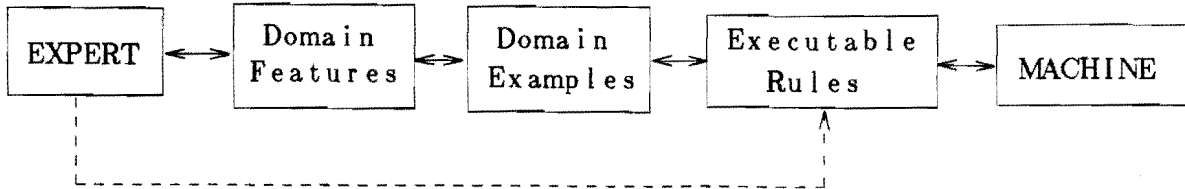
Until recently, all approaches to vision have been largely statistical in nature. Researchers sought the magic formula which would mathematically transform an image into easily discernible shapes and objects. Unlike these approaches, the research described in this thesis is based on the view that learned concepts play an essential role in texture recognition

and human vision in general. What we know determines to a large extent what we see. This idea could be described as a cognitive science approach. If this view is accurate, a computer vision system could enhance its capabilities by learning general and flexible concepts to guide its image processing strategy.

TEXPERT is a software system designed to acquire and apply knowledge for recognizing textures in a two-dimensional digital image. The system was designed primarily to test the usefulness of machine learning techniques in solving computer vision problems, particularly the problem of texture recognition. The system utilizes similarity-based learning (SBL) methods developed by Michalski [23]. Although there exist SBL methods which learn "by discovery" — i.e. without a human tutor, supervised methods were chosen here to allow an expert to incorporate as much knowledge into the learning process as possible. Thus the expert is allowed access to all points in what could be called a *knowledge transfer continuum* between man and machine.

### 1.1. The Knowledge Transfer Continuum

Figure 1.1 shows a series of steps by which knowledge can be transferred from a human expert to a machine. These steps can be referred to as a continuum because if any step is left out, a noticeable gap in performance typically occurs. For example, a bottleneck in expert systems [24] often occurs when a human expert attempts to define rules describing his knowledge without any consideration of features of the domain or examples of concepts in terms of the features. This unjustified "break" in the continuum is indicated by the dotted line in Figure 1.1.



**Figure 1.1:** Knowledge Transfer Continuum

---

The use of bidirectional arrows in Figure 1.1 is significant. They indicate that both the machine and the expert have input into the design and effectiveness of that step in the continuum. The degree to which either expert or machine has control of a particular step depends on the system and application. Ideally, tools for increased interaction at each step in the continuum should be available to the expert. Such tools have been developed by Reinke [2] in the context of the ADVISE Meta-Expert System [16]. These tools were used extensively in TEXPERT, and will be described in more detail in the next two chapters.

The TEXPERT environment provides a high-level of interaction between a knowledge engineer and the program. The system is menu and mouse driven, but still provides the high degree of flexibility necessary for learning across a wide range of visual problems. The next two sections discuss this range of problems, provide a brief description of notable approaches to these problems, and discuss how the TEXPERT philosophy compares and enhances the current methods.

## 1.2. Texture Recognition as Image Segmentation

In general, the problem of texture recognition can be thought of as equivalent to the problem of *segmentation* [18] — that is, segmenting an image in segments or regions on the basis of color, shape, or texture. Of all these segmentation criteria, texture seems to be the most general. In a real world environment, objects are readily distinguished among themselves by a distinct change in texture across the visual field. A book has a different texture than a desk-top which has a different texture than the wall which has a different texture than a picture frame and so on. In black-and-white photographs, where color is non-existent, texture is the arguably the only criteria for image segmentation.

In some ways, texture recognition is a more complicated problem than image segmentation. Inherent in the notion of texture is structure. A texture recognition program must provide some facility for recognizing structure or patterns that correspond to a common sense idea of texture. The search for *textons* [3,4], or elements of texture, has yielded few practical benefits for the texture recognition problem. Elements of texture, if they exist at all, vary greatly across regions of an image, and good descriptions for them are often difficult for humans to make.

Research has indicated that machine learning algorithms possess the capability of discovering simpler and more efficient rules for classification than experts in a domain [21]. Perhaps, a fairer statement is to say that machines perform better when allowed to generate their own rules from key examples described in terms of relevant features as provided by an expert, rather than by requiring the expert to condense all his knowledge directly into a rule formalism. Machine learning methodologies thus form an important step in the *knowledge transfer continuum* described above.



TEXPERT allows an expert to interact with the system at each step in the knowledge transfer continuum. The expert designs his own statistical methods for applying to the image, selects the examples he wants the system to use for learning, and can view, test, and incrementally modify the learned rules to improve system output. Thus TEXPERT is easily adapted to image segmentation problems which are not specifically texture related.

### **1.3. Statistical Methods**

The earliest work on texture recognition viewed the problem as one of pattern recognition [19,20]. Pixels were mapped into a feature space and clustered according to statistical pattern recognition methods. Over the years, the methods have become increasingly complex. Results of complicated statistical methods continue to show promise [22], but often at the expense of comprehensibility and adaptability across domains — that is, complex statistical methods make interaction with experts in the visual domain increasingly difficult if not impossible.

For example, consider a vision system to recognize parts on a manufacturing assembly line. Ideally, a system for an automotive assembly line could be easily modified to a tool and machine assembly line by interaction with an expert from the tool and machine domain. Use of complex statistical methods require detailed knowledge of these methods to locate parameters for modifying performance in a given domain, and furthermore, the effects of changing parameters is not always well-known or well-behaved.

Man-machine interaction in image processing is not new [14,15], but such systems also suffer from the drawback of being domain specific. The interaction of the expert is solely for the purpose of resolving problems within a given domain. The TEXPERT system provides a

general mechanism for any visual domain expert to tailor a texture recognition system to his/her domain.

Given a series of images classified by a human tutor into various regions of texture, TEXPERT generates rules that classify pixels of unclassified images into a known texture. The expert is involved to some degree in every stage of the learning process, to insure that the final rules of the system represent the fullest extent of her/his expertise.

An image that is input into the system is assumed to consist of a fixed number of distinct regions. For example, an image could consist of the following regions or textures: 1) a uniform background, 2) areas of random texture, and 3) areas of man-made texture. The regions of random texture might be characterized by high image intensity values relative to the background, irregularly shaped contours, and random size. The regions of man-made texture may be characterized by low image intensity values relative to the background and smooth contours, which consist of straight parallel lines in any direction. These descriptions represent logical combinations of image features. Textures defined in this way can easily be learned by almost any SBL algorithm.

#### **1.4. Overview of the Thesis**

The remainder of this thesis is organized as follows. Chapter 2 discusses the algorithms and methods used by TEXPERT. This discussion includes details about the learning algorithms used, the methodologies by which an expert can use these algorithms, and the overall framework in which these algorithms and methodologies are integrated.

Chapter 3 discusses the actual implementation of the system. Here the reader will find descriptions of the various components of the TEXPERT system. Also discussed in this

chapter are the facilities for enhanced interaction with an expert.

Experimental results are presented in Chapter 4. Experiments were performed on both actual and sample images. Conclusions and directions for future research are presented in Chapter 5. In this chapter the results from the previous chapter are discussed in depth, along with their ramifications for both computer vision and machine learning. References are provided at the end of the thesis.

## CHAPTER 2.

## TEXPERT ALGORITHMS AND METHODS

This chapter describes the algorithms and methods used in the TEXPert system. The system is built out of similarity-based learning methods developed by Michalski [23]. Learning, however, is only a part of the system, which also incorporates rule application strategies from expert systems [2,16] research in addition to standard image processing methods. These methods are combined to form the top-level algorithm for the system as shown in figure 2.1.

---

```
1. Input and classify the image;
   REPEAT
     define active sub-region for learning or testing;
     perform image pre-processing;
     generate events from pixels in active sub-region;
     if learning desired then
       rules = learn(events);
     if testing desired then
       test(events, rules)
     if further learning or testing desired then
       regenerate image;
     else
       generate results;
       done = true;
   UNTIL DONE
```

**Figure 2.1:** Top Level Algorithm

---

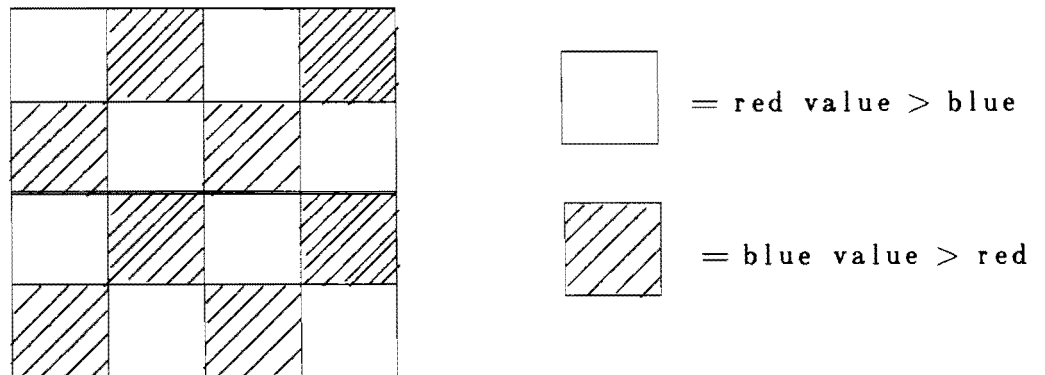
The first four steps in the algorithm involve image classification and pre-processing, and generation of events for learning or testing rules. Classification requires display of the image to a tutor or an expert who segments the image into regions or *classes*. An entire image need not always be classified. With large images, classification of only a part of the image is desirable. Once the sub-region of the image to be used for learning has been classified and defined, the input events to the learning algorithm can be generated. Often, generation of learning events first requires *image pre-processing* — use of convolution operators or other statistical methods to generate features for the image. For example, images can be processed using a "Mexican hat operator" [5], for the purpose of constructing an "on-edge" attribute. Each pixel in the image will have both a gray-level value and an "on-edge" value, which can be either the actual value or simply true or false depending on the result of the application of the "Mexican hat". These two values, and any other values produced by the application of other filters or operators to the image, together form an *event* which is essentially a point in an n-dimensional attribute space where n is the number of values or *attributes* associated with each pixel. Specific methods of attribute selection will be discussed in a subsequent section in the course of discussing the learning algorithm. Additional methods of pre-processing are also discussed in a later section.

The next stage in the algorithm is the actual learning phase. The learning phase takes as input classes or sets of *events* from the image classification and pre-processing phase, and outputs rules which correctly assign a pixel to a class based on the attribute values of the pixel. As a simple example, consider an image where every pixel has a red-intensity level and a blue-intensity level and that each level has three values — high, medium and low. Furthermore, suppose this image has two different "textures" or types of regions — that is, each pixel in the image is part of some region which is of one of the two types of regions in

the image. Occurrences of these two types of regions could exist anywhere in the image to form, for example, a checkerboard pattern as in figure 2.2. Now if one of the checkerboard regions has pixels where the red-intensity value is always higher than the blue-intensity value, and the other region's pixels always have a higher blue-intensity than red, then the rules for classifying pixels would look like these:

```
[blue_value(pixel) = high][red_value(pixel) = medium or low] or
[blue_value(pixel) = medium][red_value(pixel) = low]
then
[class(pixel) = 1]

[red_value(pixel) = high][blue_value(pixel) = medium or low] or
[blue_value(pixel) = medium][red_value(pixel) = low]
then
[class(pixel) = 2]
```



**Figure 2.2:** Two Textures Forming a Checkerboard

---

Details of the learning algorithm will be discussed in section 2.1.

The next phase of the algorithm is rule application or testing. In this phase, the rules output from the learning phase are applied to other images or other sections of the original image to evaluate the accuracy of the rule. These other images or other sections of images must also be classified by the expert or tutor in order to determine how well the rules were able to perform in relation to the expert. If the testing shows acceptable performance over a sufficient number of test cases, then the learning has been successful and these rules can be used reliably on unclassified images in the domain. If testing does not indicate acceptable rule performance, there are two options available for further learning: *iterative learning*, where further learning takes place on a regenerated image; or *incremental learning*, where the events/pixels are input to the learning algorithm along with a set of old rules, to produce modified rules which are accurate over all events/pixels. The iterative method is most useful in general, however the incremental method is frequently the method of choice when applying rules learned from one or more images in a domain to a new image. Both these methods will be discussed in more detail in the next section.

### **2.1. Learning Algorithm**

Although not the first step in the top-level algorithm, the learning phase is the heart of the TEXPert algorithm. The algorithm used is a modified version of the AQ algorithm, developed by Michalski, Reinke, Hong, Mozetic, and others [2,25,26]. The algorithm is a supervised covering algorithm. The fundamental input to the system are *events*, which are vectors in an n-dimensional attribute space where n is the number of attributes. Attributes can have values of several types. *Linear values* are ordered values such as integers. *Nominal*

*values* are non-ordered, for example, a shape attribute might have nominal values square, circle, triangle. Values may also be *structured* — that is, the values of the attribute may form a hierarchy. The shape attribute for example, may also have the value polygon, which could have as sub-values square and triangle, but not circle, which could be a sub-value of the value ellipse.

Given attributes with values of these types, the algorithm searches through a space of logical expressions relating attributes to values. The goal of the search is to find an expression, which is satisfied by the values for every event in one class, and which is not satisfied by the values of any event in any other class of events. The search is limited by a heuristic called the *lexicographical evaluation function* (LEF) which can be user-specified. The LEF evaluates candidate expressions and sub-expressions according to user-specified criteria. Only a small number of expressions are expanded at each stage in the search. The number of expressions to expand at each stage is also a user-specified parameter. The algorithm can be viewed as a very flexible beam search through the hypothesis space associated with the vector space defined by the attributes.

Several additions to the AQ algorithm have been made over the years. The important ones are worth mentioning here. Background knowledge can be provided to the algorithm. Such knowledge can be used to perform constructive induction, i.e. automatic generation of new attributes for the events inputted to the learning algorithm. Background knowledge to AQ can take one of two forms. *A-rules* describe mathematical combinations of values to form a single value for a new attribute. For example, the value for the attribute *length* may be multiplied by the value for the *width* attribute to produce a value for a new attribute called *area*. An example of this A-rule is shown in Figure 2.3(a). Figure 2.3(b) shows a



- 
- (a)  $\text{area} := \text{length} * \text{width}$
  - (b)  $[\text{discomfort\_level} = \text{high}] \text{ if}$   
 $[\text{temperature} > 90][\text{humidity} > 50]$

**Figure 2.3:** A-Rules and L-Rules

---

simple *L-rule* for AQ. L-rules create attributes on the basis of logical combinations of attributes and values. The L-rule in the figure says that if the temperature attribute has a value greater than 90 and the humidity attribute has a value greater than 50, then the new attribute discomfort-level should be constructed with the value high.

Of these two types of background knowledge, A-rules are the most applicable to the problem of image processing. A-rules provide for arithmetic combinations of intensity values to create new features for image pixels. An example of how A-rules can be used as image filters is given in the next chapter under "Rule Acquisition and Induction." Other ways of generating features for each pixel are discussed in the next section.

### 2.1.1. Features for Learning Texture

Many possibilities exist for computing features to recognize texture. The best features will be those which define an attribute space which most easily lends itself to the partitioning performed by the learning algorithm. The features generated by TEXPRT fall into one of three categories: neighboring gray-level values, simple statistics, and convolution filter

output.

### 2.1.1.1. Neighboring Gray-Level Values

These are the simplest attributes in the TEXPRT system. They consist only of the gray-level values of pixels neighboring the pixel with which the event is associated. For example, one could take the value for each immediate neighbor of a pixel to construct an event with eight attributes for each pixel. The first attribute would correspond to the value of the pixel to the central pixel's upper left, the second attribute would correspond to the pixel directly above the central pixel, and so on, as in Figure 2.4. Even the value of the pixel itself can often be used as an attribute, and is in fact the only attribute needed in the trivial

---

2	3	4	4	3	2
2	3	4	5	4	3
2	4	6	5	4	2
3	4	5	4	3	3
3	3	4	3	2	2

```
Event for central pixel (value = 6):
[upper_left = 3][above = 4][upper_right = 5]
[left = 4][right = 5]
[lower_left = 4][below = 5][lower_right = 4]
```

**Figure 2.4:** Possible Event Using Neighboring Gray Values

---

case when texture is determined by gray level alone.

A tradeoff is clearly apparent in the use of neighboring values as attributes. More information is available to the learning algorithm when more neighboring values are used, however with more information processing time steadily increases.

#### **2.1.1.2. Simple Statistics**

Simple statistics are linear combinations of the neighboring pixel values. Some examples of such statistics are sum of values, mean pixel value, maximum pixel value, and minimum pixel value. The main advantage of the use of these statistics is that they incorporate regional information about the pixel in one feature with only minimal additional computation. Although these statistics can be useful, the information is frequently limited in comparison with more involved statistics such as those discussed below.

#### **2.1.1.3. Convolution**

The third type of feature of benefit to a texture learning system are those that are calculated using filters or *convolution operators*. A convolution operator can be thought of as a coefficient template — a grid which is centered over a pixel in the image where each coefficient in the template multiplies the corresponding pixel value in the grid and the sum of these products is the value of the operator at the central pixel. An early edge detection template (the Kirsch operator) is shown in Figure 2.5. The equation in Figure 2.5 is the application of the Kirsch operator to the central pixel of Figure 2.4. The low value of the result indicates a small gradient in the vertical direction — that is, no horizontal edge through the central pixel is indicated.

---

1	1	1
0	0	0
-1	-1	-1

Value for central pixel of Fig. 2.4 (value = 6):  
 $(1)3 + (1)4 + (1)5 + (-1)4 + (-1)5 + -1(4) = -1$

**Figure 2.5:** Kirsch Operator

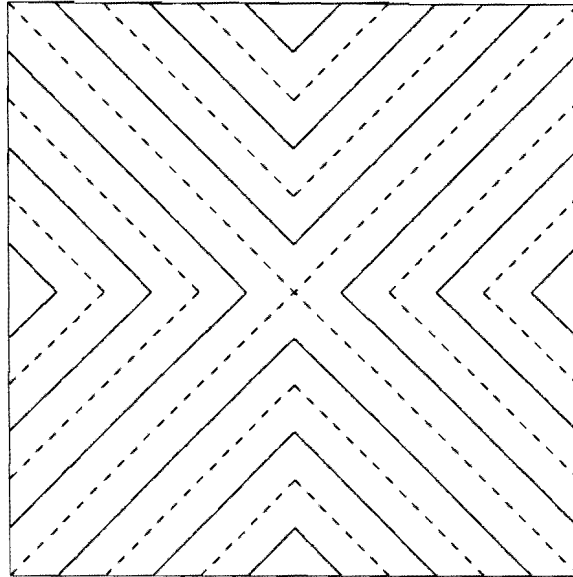
---

Researchers have suggested a wide range of convolution filters for various purposes. Even a summary of such filters extends beyond the scope of this thesis. The interested reader is referred to [6,11,12] for a comprehensive overview of the subject.

### 2.1.2. Iterative Learning

In the TEXPERT system, the learning algorithm described above is performed iteratively by using the classification of pixels from one iteration as the value of pixels for the next iteration. This method was first proposed by Michalski [1] and similar methods [13] have also been proposed. This method has also been called a pyramid or hierarchical architecture for vision.

Iterative learning is hierarchical because a new texture can be learned or recognized from a combination of old textures from a previous iteration. To illustrate how this works, consider the image in Figure 2.6. Such an image consists of three fundamental textures:



**Figure 2.8:** An Illustrative Image For Pyramid Learning

---

solid lines, dashed lines, and white space. At the first iteration of the learning algorithm rules can be derived for classifying pixels into one of these three classes as appropriate. For iterative learning a new image would be generated as follows: each pixel that was part of a solid line in the first image would be given a value of 1, each pixel that was part of a dashed line in the first image would be given a value 2, and all other pixels would be given the value 3. Now the new image serves as input to a new iteration of the learning algorithm. In this iteration, the new values of the pixels can be used to recognize the two textures (areas where lines run from top to bottom and left to right, and areas where lines run from top to bottom

and right to left) in the figure. The fact that there are two textures at the top-level and three textures at the first level suggests the pyramid as a model for the architecture.

In the above example, a new classification was required at each level of the pyramid, but this need not always be so. Iterations can occur using the original classification. This form of iteration is a form of relaxation and is a way of dealing with noise in the image. The idea is that only pixels whose attributes satisfy the learned rules to a high degree of consonance (see [2]) or certainty are replaced by a class designator in the next iteration. All doubtful pixels retain their original value for the next iteration. Once a pixel is classified, its value does not change for the next iteration. Thus each iteration becomes a means of classifying more and more pixels in the image until eventually the entire image is correctly classified.

### 2.1.3. Incremental Learning

At no time during either method of iterative learning are rules used in addition to events as input to the learning algorithm. Learning which uses rules as input is called *incremental learning*, and is entirely different than iterative learning. Incremental learning can be thought of as modifying the input rules based on new input events. Incremental learning is discussed in detail in [2,25,26].

As mentioned earlier, incremental learning is used in TEXPert when learning occurs over several images in the domain. The main advantage to incremental learning is efficiency. Cross-image learning could be effected by storing all the events for all the images, but this is not only cumbersome but often expensive. In fact, it is often advisable that incremental learning be used on a single image if that image is large. It is not always necessary, however, to invoke the learning algorithm several times to perform incremental learning on a single

image. The version of AQ described in the next chapter provides a mechanism for partitioning large event sets, and generating rules one set at a time until all the sets have been examined.

## 2.2. Testing Method

The testing phase consists of applying the set of rules generated during the learning phase to a separate region of the current image (or to another image in the domain) and evaluating the performance of the set of rules on this new region. In applying the rules, an event for each pixel in the testing region is generated in the same way events were generated for pixels in the learning area to learn the given rules. Each rule is applied to an event and an estimate of probability is calculated based on the degree to which the event satisfied the rule. The specific methods of calculating the estimate of probability are not discussed here. The reader is referred to [21] and [25] for details on these measures.

Once a measure of rule satisfaction is obtained for each rule, the event can be assigned to the class whose rule has the highest measure of satisfaction. In the case of the iterative-relaxation method described above, the rule satisfaction must exceed a given threshold parameter set by an expert. It can also be required that the winning measure exceed its nearest competitor by a certain threshold. If these conditions are not met, no classification occurs and the pixel maintains its original value for the next iteration. In the next iteration, the conditions may be relaxed — i.e. either or both the thresholds are lowered.

Once all the pixels in the testing region have been classified by the rules, the classified image can be used to evaluate the rule performance. Learning is considered successful when a reasonable level of accuracy has been maintained over a sufficient number of images in the

domain. Exact measures for “reasonableness” and sufficiency are provided by the expert.

### 2.3. Image Pre-Processing

As in any image system, additional processing of the image prior to the operations of one’s own system is often useful. For example many systems work with a Fourier transform of an image or “smoothed” images. In the design of the TEXPRT system, three methods of pre-processing are considered: averaging, gray-level reduction and event generation.

#### 2.3.1. Averaging

Averaging is a standard method of image reduction. Averaging an image replaces each  $m \times n$  region in the image by the average values of the pixels in that region. This method thus reduces the number of pixels in the image by the product  $mn$ . Such data reduction is often useful with large images, but should be used with care since information is often drastically reduced.

#### 2.3.2. Gray-Level Reduction

Gray-level reduction reduces the number of values that an individual pixel can have. This reduces the search space for the learning algorithm. The new value of the pixel is obtained by dividing it by:

$$\frac{MAX}{N}$$

where  $MAX$  is the maximum value of any pixel and  $N$  is the new number of values allowed for the pixel. If the minimum value for a pixel in the original image is not zero, the



minimum value can be subtracted from the pixel before the division described above.

### 2.3.3. Event Generation

The last pre-processing done on an image before learning is event generation. During this phase, an event is generated for each pixel in the image. For learning, there is an exact correspondence between pixels and events. Events are generated using an *event shape operator* which is specified by the user of the system. This operator extracts features (as described in section 2.1) from the image to form an event which is saved for subsequent input into the learning algorithm. Since the learning algorithm requires that pixels classified into similar classes be presented as such, event generation accomplishes this as well.

Additional details on event generation are covered in the next chapter as part of the discussion on the implementation of all the methods described above.

## CHAPTER 3.

### TEXPERT ARCHITECTURE AND IMPLEMENTATION

This chapter describes the implementation of the methods and algorithms described in the previous chapter. The system was implemented on a SUN-2 Workstation and uses the SunWindows window and graphics package. Development of the system was primarily in Pascal, although some portions of the system were written in C in order to interface with the SunWindows routines. This interface is described in detail in [7].

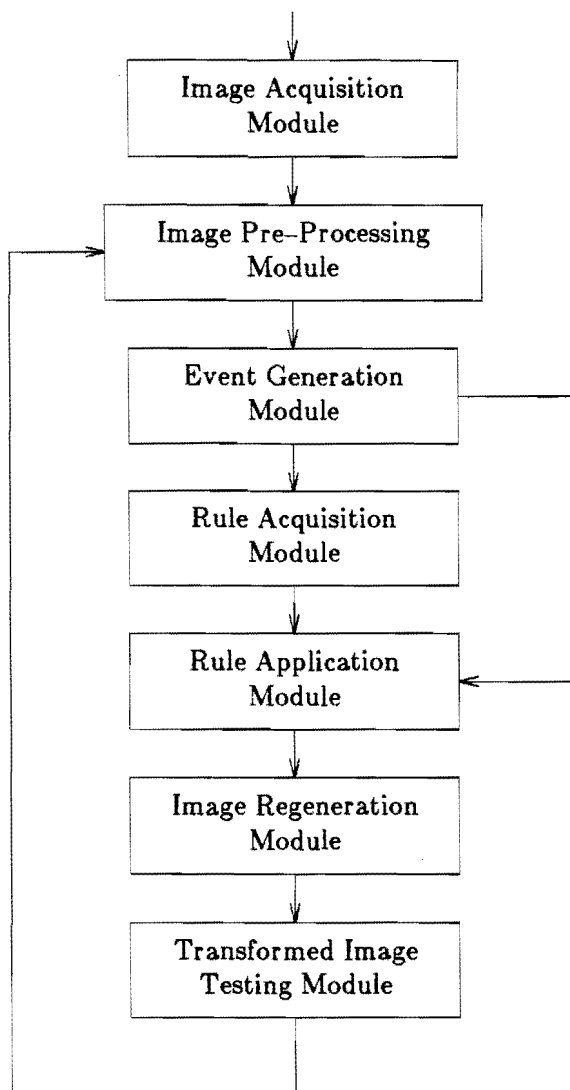
The general architecture of the TEXPERT system is shown in Figure 3.1. Each module is described in detail in a corresponding section below.

#### **3.1. Image Classification and Processing**

In the implementation of TEXPERT, image classification is considered as one possible, although usually essential, mode of processing. Classification occurs as the last step before event generation. Any pre-processing is required to take place before classification.

##### **3.1.1. Classifying an Image**

The TEXPERT system displays the fully pre-processed image to the user in the window in which the user started the program. The system can also be run outside the Suntools and SunWindows environment as long as this is done on the console and a mouse is available. Given the displayed image, the user can use the mouse to paint pixels which the user determines as belonging to a specific texture or region. Erasing is available in case the expert



**Figure 3.1:** TEXPERT General Design

---

changes his/her mind as (s)he is prone to do from time to time. The user is free to classify as much or as little of the image as (s)he desires.

As currently implemented, TEXPERT allows a user to classify images pixels into as many as 10 different classes. Once a pixel is classified into a class, and only then, is that class (or texture) assumed to exist in the image. However, only classes which exist in the learning area have rules produced in the learning algorithm. The user should, therefore, have pixels from all specified classes within the learning area specified during event generation (see next section).

### **3.1.2. Other Processing**

In the current implementation of TEXPERT, there are two filters for pre-processing data. The first is a gray-level reduction filter as described in the previous chapter. This filter reduces the number of gray-levels from 255 to 12. The maximum number of levels allowable as input to the learning algorithm is set to 58 by default. In practice, the fewer number of levels, the more efficient the algorithm.

The second filter available within the context of TEXPERT is the second-order Gaussian edge detector described in [5]. This filter produces a separate image file which can be read into TEXPERT simultaneously with the original image. These two images together can then be used to generate edge detection and "zero crossing" attributes for learning.

### **3.2. Event Generation**

Event generation is a crucial stage of the TEXPERT system. When a user defines the way events are to be generated, (s)he is in fact defining the attribute and hypothesis space in which the learning algorithm is to search. TEXPERT is designed with a flexible interactive means of specifying attributes for events.

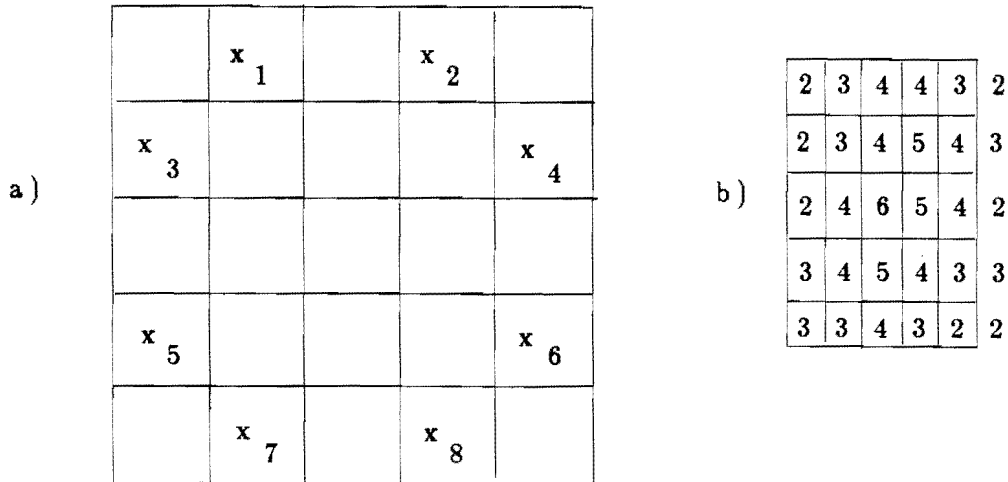
In the current implementation, a user is allowed to select one of four sizes for an *event template*. An event template is best thought of as a mask which is centered over the pixel for which the event is to be generated. A three-by-three template is shown in figure 2.4. Event templates are presently required to be square and to have an odd dimension so it can be centered over a pixel. The four templates range in size from three by three to nine by nine.

Once the user has selected a size for the template, he can then select which locations in the template (s)he would like to use as attributes for the event. Figure 3.2a) shows a five-by-five template with locations marked with attributes  $x_1$ ,  $x_2$ ,  $x_3$ , and so on. All other locations other than those with attributes do not generate an attribute for the event. Figure 3.2c) shows an event corresponding to the template as positioned in the sample section of image shown in part b). In this example, all the attributes are neighboring gray-level values.

After defining a template, a user can save the template in a file for future use. Thus a user need not always define a template for every event generation process.

In addition to specifying neighboring gray-levels as attributes, the user is also allowed to select other attributes for learning from a menu. These attributes are: minimum, maximum, sum, average, and the difference between the minimum and maximum values. These attributes are calculated over the neighboring gray-values specified in the event template.

Events are generated automatically as preparation to learning rules or testing them on an image. When invoking learning or testing on an image, the user is first asked to define a region in which to learn or test rules. The user can define a region either by entering coordinates directly from the keyboard or by selecting the desired boundaries of the area



c) Event for template positioned as in b):

$$\begin{array}{l}
 [x_1 = 3] [x_2 = 4] [x_3 = 2] \\
 [x_4 = 4] [x_5 = 3] \\
 [x_6 = 3] [x_7 = 3] [x_8 = 2]
 \end{array}$$

**Figure 3.2:** Possible Event Using Neighboring Gray Values

using the mouse. A separate file is written for each class of events — i.e. if there are  $n$  classes,  $n$  files are constructed with each file containing only the events from one class. In the case of testing only one file is created.

### 3.3. Rule Acquisition and Induction

After events have been generated for a specified learning area, TEXPRT builds an input file from the events and other appropriate files and feeds the input file into the learning

algorithm. The user can change the parameters of the algorithm by editing these files.

TEXPERT has enjoyed the use of several learning algorithms in its brief but promising history. Originally, GEM (Generalization of Examples by Machine) [2] was used. Some experiments were then performed using a LISP version of AQ11 developed by Jeff Becker at the University of Illinois. Currently TEXPERT uses the most recent version of AQ, AQ15 [26].

AQ15 provides all the enhancements to the AQ algorithm described in the previous chapter. Rules produced by AQ15 are saved in a file specified by the user. These rules can then be used for testing on the current image or any other image which is input into TEXPERT. These rules are also available for incremental learning as described earlier.

The A-rules facility of AQ15 provides a convenient way of implementing values of filters over the image as an additional attribute for events/pixels. Consider, for example, the Kirsch operator in figure 2.5. Remember each element in an operator is a coefficient for multiplying the corresponding pixel value in the image, and that the value of the filter is simply the sum of all these products. Then given an event template which generates attributes  $x_1$  to  $x_9$  as the pixels of a three-by-three region centered around the pixel for which the event is being generated, an A-rule which calculates the value of the Kirsch operator as attribute  $x_{10}$  is given in figure 3.3.

In general, any image filter can be declared using a single A-rule as in figure 2.5. The specification of A-rules is currently not an interactive feature of the TEXPERT system. A-rules must be added to the learning input file by means of an editor, and the learning algorithm must be invoked outside a TEXPERT session.

---

$$x_{10} := x_1 + x_2 + x_3 - x_7 - x_8 - x_9$$

**Figure 3.3:** A-Rule for Applying the Kirsch Operator of Figure 2.5

---

This, however, is not as an unreasonable requirement as it may seem. Given large images with many textures, the learning algorithm may require several hours of processing time to produce rules for the texture. When such processing times occur, it is not unreasonable to require that the user do some manual massaging of an input file.

If such long processing times do occur, TEXPRT provides a mechanism for running the learning algorithm as a background process, leaving the user free to perform other tasks (e.g. classification of an area for testing the rules after they are generated) while the learning algorithm is running. This feature is based on multiprocessing within the UNIX environment.

The expert of an image domain may find several other useful features of AQ15 as applicable to her/his domain. For a summary of these features the interested reader is again referred to [26].

### 3.4. Rule Application

Rule application (also known as rule exercising [9]) consists of generating events from a testing area and applying a set of rules to these events. Testing is also available through the AQ15 program which incorporates the tools and facilities of the ATEST program [2]. As



with the learning algorithm, the user is allowed to change the testing parameters by editing the default file used in generating the testing input file.

Rule application produces a file which is automatically used to regenerate an image as described below.

### **3.5. Image Regeneration**

Given the Rule Application Module above, image regeneration is a straightforward operation. The class to which a pixel is assigned by the induction rules replaces the pixel in the regenerated image. This image is maintained internally in the program and can be written to a file at the user's request.

Oftentimes only a section of an image will undergo testing. In this case, all pixels in the regenerated image which were not a part of any testing area in the original image will be retain their value in the original image.

Once testing is completed, the user is free to view the testing results directly on the screen. The "test results" mode is one of three modes of image display available to the user. The other modes are "raw image" and "classified image" mode. Thus the user can quickly see the differences between the original image, the classified image and the regenerated image that results from all testing done on the image in the current session. In "test results" mode, discrepancies between the classified image and the test results — that is, pixels for which the expert has assigned a different class than that assigned by the rules — are highlighted in the image. This gives the expert immediate feedback as to how well the rules agree with his initial classification.

### 3.6. Transformed Image Testing

Often the expert requires a more analytical method of how well the rules perform in relation to his classification. In the Transformed Image Testing Module, precise statistics are calculated and are saved in a file for future reference. These statistics include the number of pixels for each class (as determined by the expert) and then the percentage of these pixels which were classified into each possible class. A sample statistics file is shown in figure 3.4. These statistics were generated for a small region of an image which contained only two classes, class 0 and class 1. In this section of the image the expert classified 170 pixels into class 1. Of those pixels, the learned rules classified 89% of those pixels correctly (into class 1) and 11% incorrectly (into class 0). The statistics file can thus be thought of as a matrix, in which values on the diagonal represent the percentage of correct classification by the rules.

### 3.7. Implementation Summary

The TEXPRT system implementation provides a reasonably flexible environment for processing images through an machine learning algorithm. Given the large number of possible attributes, some restrictions are inevitable given that graduate student man-hours

---

CLASS	No. of pixels	% in class 0	% in class 1
0	1030	98	2
1	170	11	89

**Figure 3.4:** Sample Statistics File.

---

are not (contrary to popular belief) an unlimited resource. Nevertheless, sufficient features are available to perform sophisticated learning and texture recognition experiments. Such experiments are described in the next chapter.

## CHAPTER 4.

### EXPERIMENTATION AND RESULTS

To illustrate the capabilities of the TEXPERT system, experiments were performed on two-dimensional digital images obtained from ultrasound analysis of laminated material. Textures in these images represent areas of varying thickness or structural flaws in the material. The domain is well suited to testing the TEXPERT system for the following reasons:

- 1) The textures are complicated enough to make difficult the task of an expert to write rules to describe the image, but are still well within the scope of the language used by AQ15.
- 2) The textures lend themselves to both the iterative and incremental learning features of the TEXPERT system.
- 3) The problem is non-trivial and has practical ramifications for industry.

In the following sections, experiments are described for generating and testing rules in both the iterative and incremental learning modes. Results from applying these rules to the image are provided. The results are discussed in detail in the next chapter.

#### 4.1. Learning Texture Using Iterative Learning

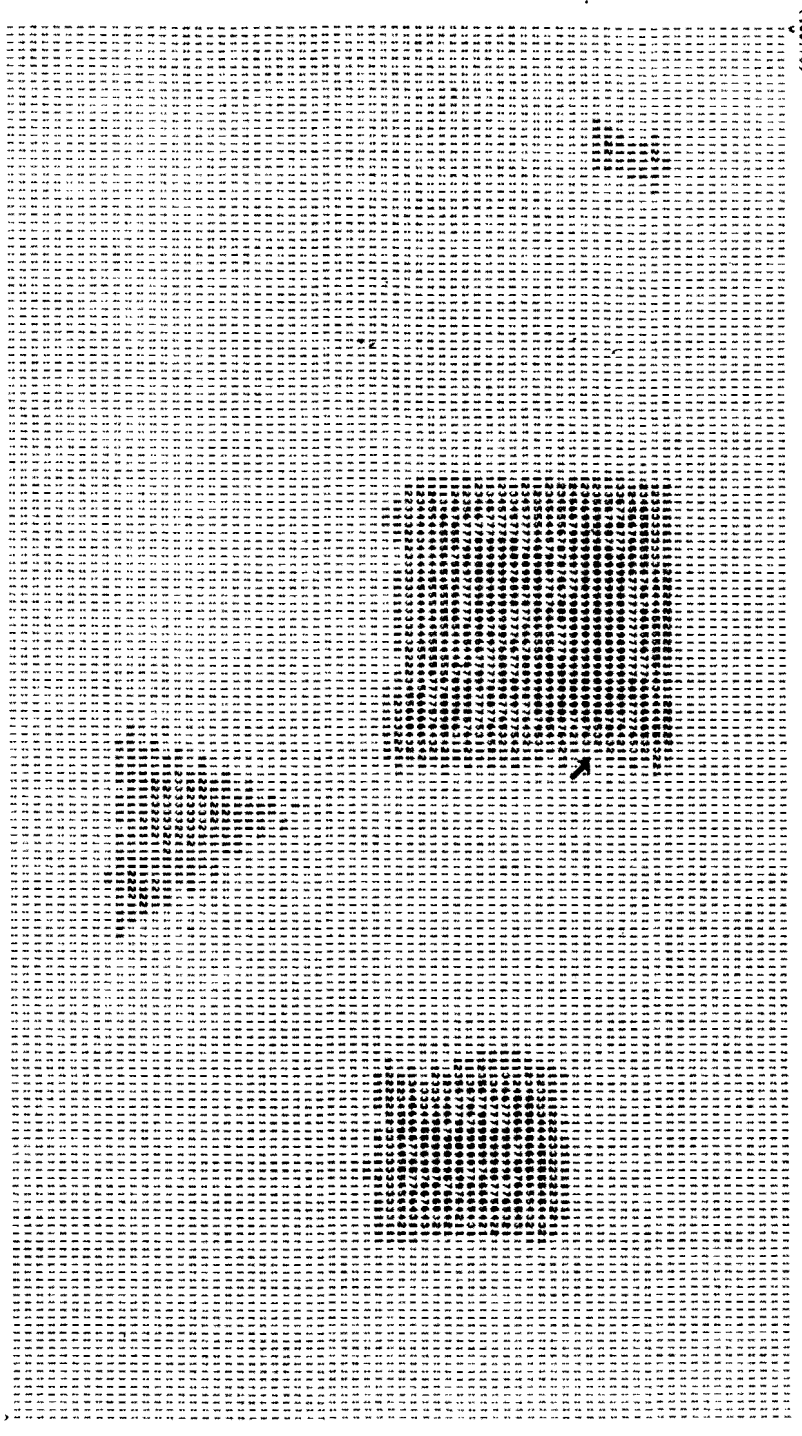
Figure 4.1 shows a section of the digital ultra-sound image used for the iterative learning experiments. The image window in the upper left corner of the main TEXPERT window indicates the area of the image which is visible in the display below. The user is also



IMAGE WINDOW (you are here)

(168, 75)

IMAGE: Binary  
413 Rows, 358 Cols  
Press Middle or Right Mouse Buttons for Menu



(168, 255)

Figure 4.1: Raw Image for Iterative Learning

guided by the coordinates at the upper left and lower right corners of the displayed section of the image. The actual size of the entire image is displayed at the top of the window with the name of the file which contains the raw image.

In this image (and for all the images in this chapter), there are two classes or textures. These are normal or class 0 texture and abnormal or class 1 texture. The reader can easily recognize four regions of abnormal texture on a background of normal texture in figure 4.1. Figure 4.2 shows these regions after they have been classified into class one by "painting" them using a mouse. Note that none of the background texture has been classified. This is unnecessary because the system assumes that all unclassified pixels automatically belong to class 0.

In order to learn rules for classifying pixels into one of these two textures, the user selects a training area from which TEXPRT generates events to give as input to the learning algorithm. Figure 4.2 also shows the learning area used for the iterative learning experiment.

For this experiment, events were generated using a simple three-by-three pixel event template to generate twelve attributes for each pixel. These attributes consist of the intensity value of the pixel, the intensity value of the eight neighboring pixel, the maximum and minimum values among these nine, and the difference between the maximum and minimum value.

Figure 4.3 shows the results of applying the rules learned to the learning area. The rules classified ninety-four per cent of the class 0 (normal) pixels correctly and ninety-seven per cent of the class 1 (abnormal) pixels correctly. During iterative learning, all correctly

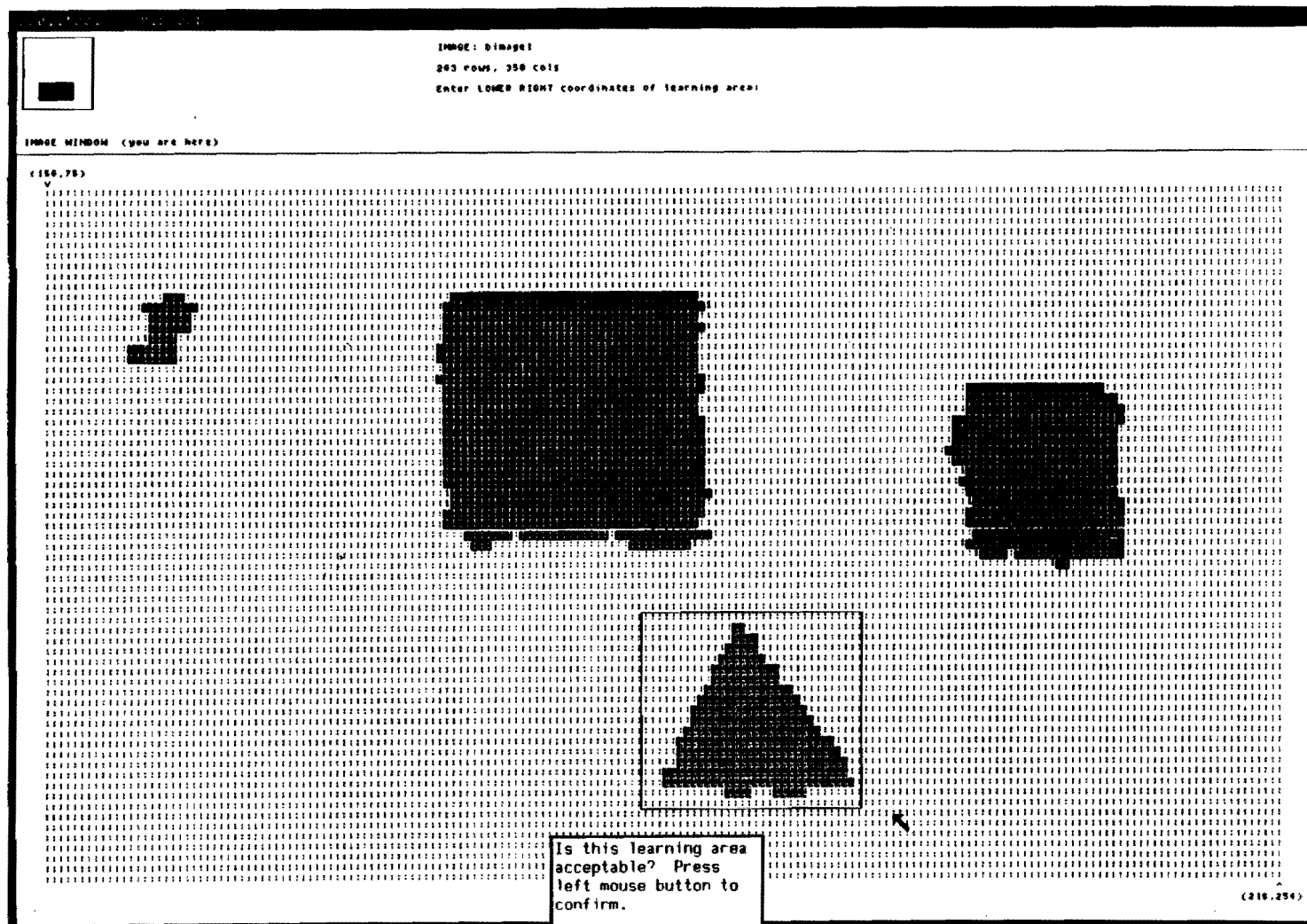


Figure 4.2: Classified Image for Iterative Learning

---

Testing statistics from image file "bimager1"			
Rows 191 to 209 and Columns 162 to 193 inclusive.			
CLASS	No. of pixels	% in class 0	% in class 1
0	351	94	0
1	257	0	97

**Figure 4.3:** Results of Applying Rules to the Learning Area

---

classified pixels are given a new value to indicate their class, whereas incorrectly classified pixels are not classified but retain their same value for the next iteration. In other words, in iterative learning either an event is classified correctly or it is not classified at all. For this reason, statistics on incorrectly classified pixels are not generated (or more correctly are not distinguished among the incorrect classes). Figure 4.4 shows the pixels which were not correctly classified as highlighted within the learning region.

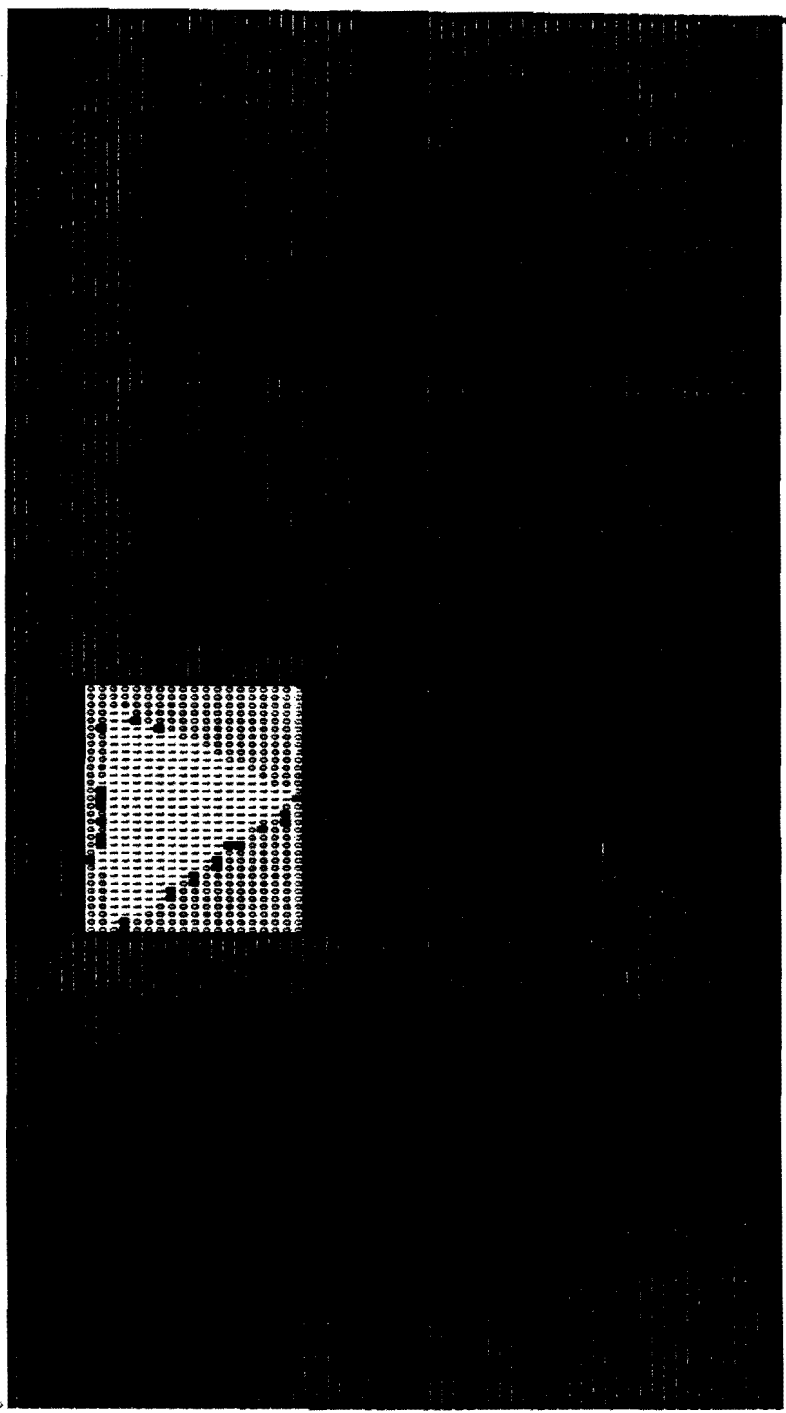
One may be surprised that a learning algorithm would not produce a one hundred per cent accurate classification on the pixels from which it generated its rules. Certainly this would be alarming in a simple blocks world domain. However, in digital images pixels with identical attributes are often found in different regions. In learning terminology, this means the same event belongs to two classes. These rules must somehow compete with each other for correct classification of these two identical events. The only possible solution to this problem is to change the events themselves. Replacing correctly classified pixels in the image with unique values provides the necessary change among the neighbor attributes of the problem pixels. Using the regenerated events from this learning area produces perfect results as shown in figure 4.5.





IMBET WINDOW (you are here)

IMBET: 01mep1  
243 row, 350 col  
Enter name of file for rule output:  
(enter RETURN for default - warning: old default will be overwritten)



(150, 285)

(150, 285)

Figure 4.4: Discrepancies in Iterative Learning

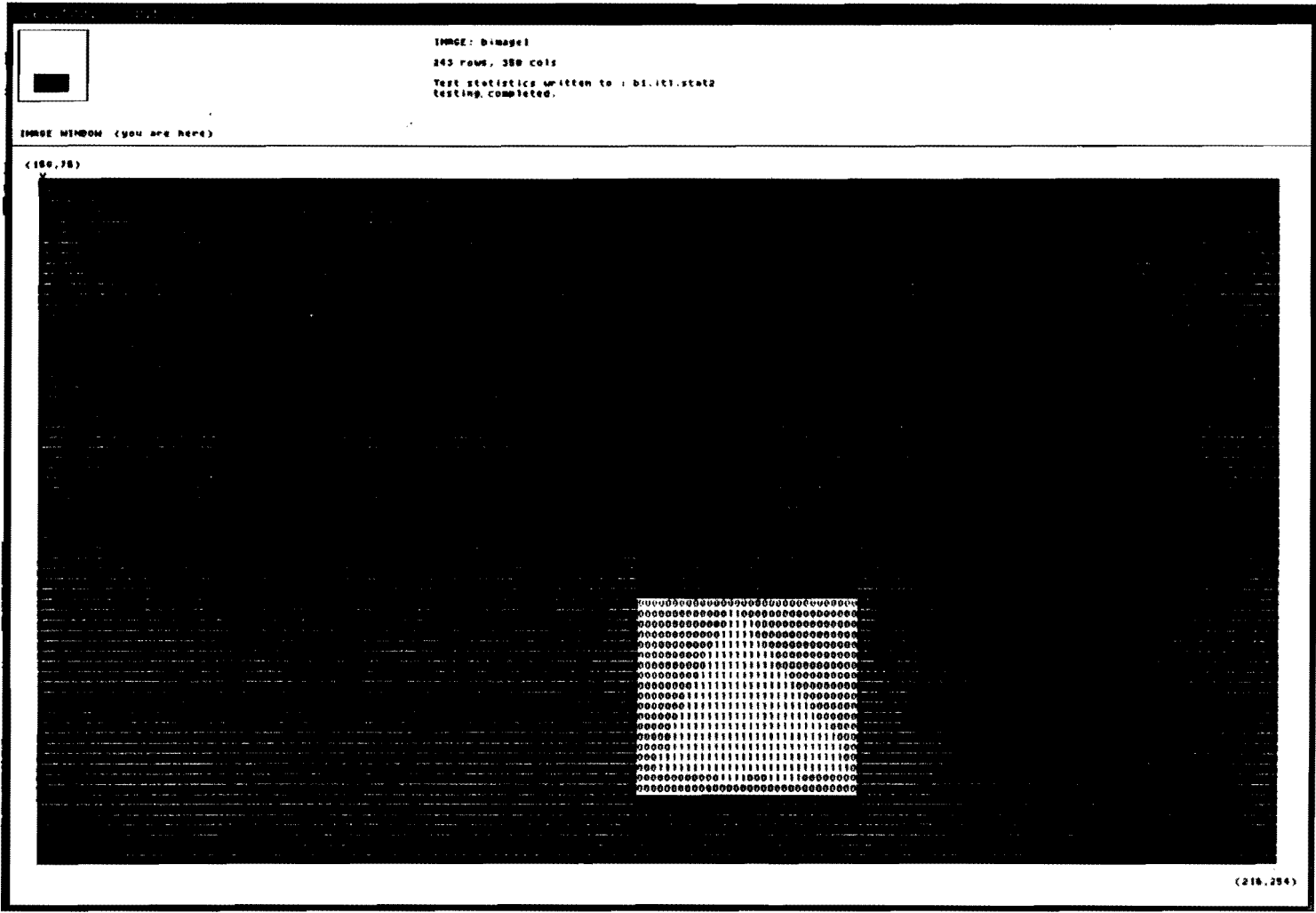


Figure 4.5: Iterative Learning Area without Discrepancies

Once satisfactory rule performance is achieved, the rules can be used to classify additional sections of the image or other images in exactly the same way as testing was done on the learning area. (Actual rules generated from iterative learning are included in the Appendix.) The next section on incremental learning provides results of more extensive testing of the system.

#### **4.2. Learning Texture Using Incremental Learning**

Although iterative learning provides a suitable method for dealing with noise and other inconsistencies in the images, a more powerful mechanism of learning will be required to modify rules across images or across widely varying textures within a single image. Incremental learning can provide this capability by allowing rules to be modified to reflect new information about a texture.

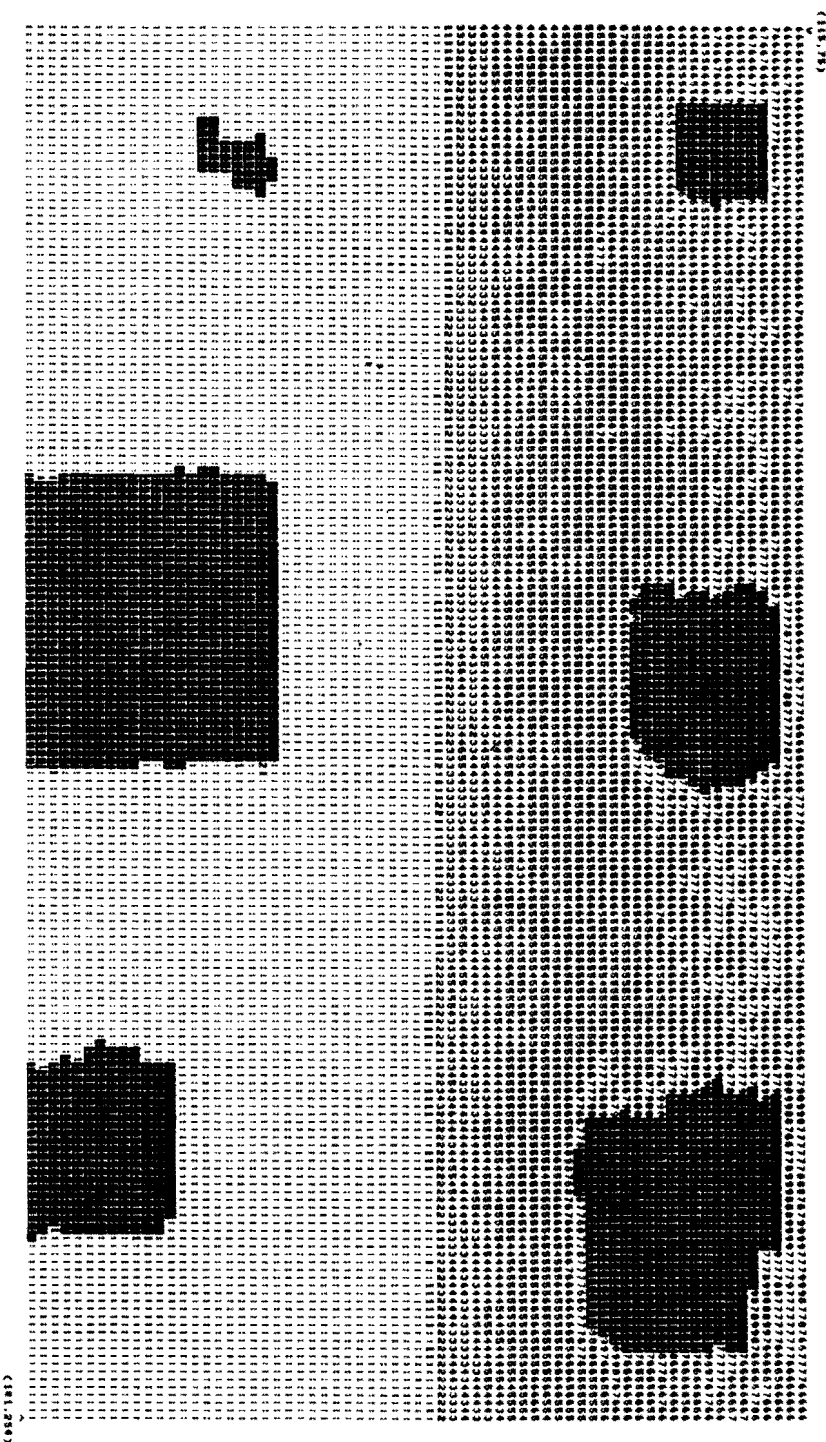
Consider the already classified image in figure 4.6. Again the user can readily distinguish the background and foreground textures. However, the background texture makes a sudden change in the middle of the image. For some applications, this could very well be considered a third texture and rules could be learned using three classes instead of two. However, in the domain of locating abnormal and normal textures, two classes should suffice.

To illustrate the power of incremental learning, choose for the base case of learning the region shown in figure 4.7. This is a region similar to that used in the iterative learning example in the previous section. Although this region appears to be small, it is reasonably sufficient for generating rules for other regions with a similar background texture. Figure 4.8 shows two separate testing regions for the rules generated from the area in figure 4.7. Since



IMAGE: BINARY1  
243 ROWS, 358 COLS  
Current Export File: export.dat

IMAGE WIDTH (USE FILE NAME)



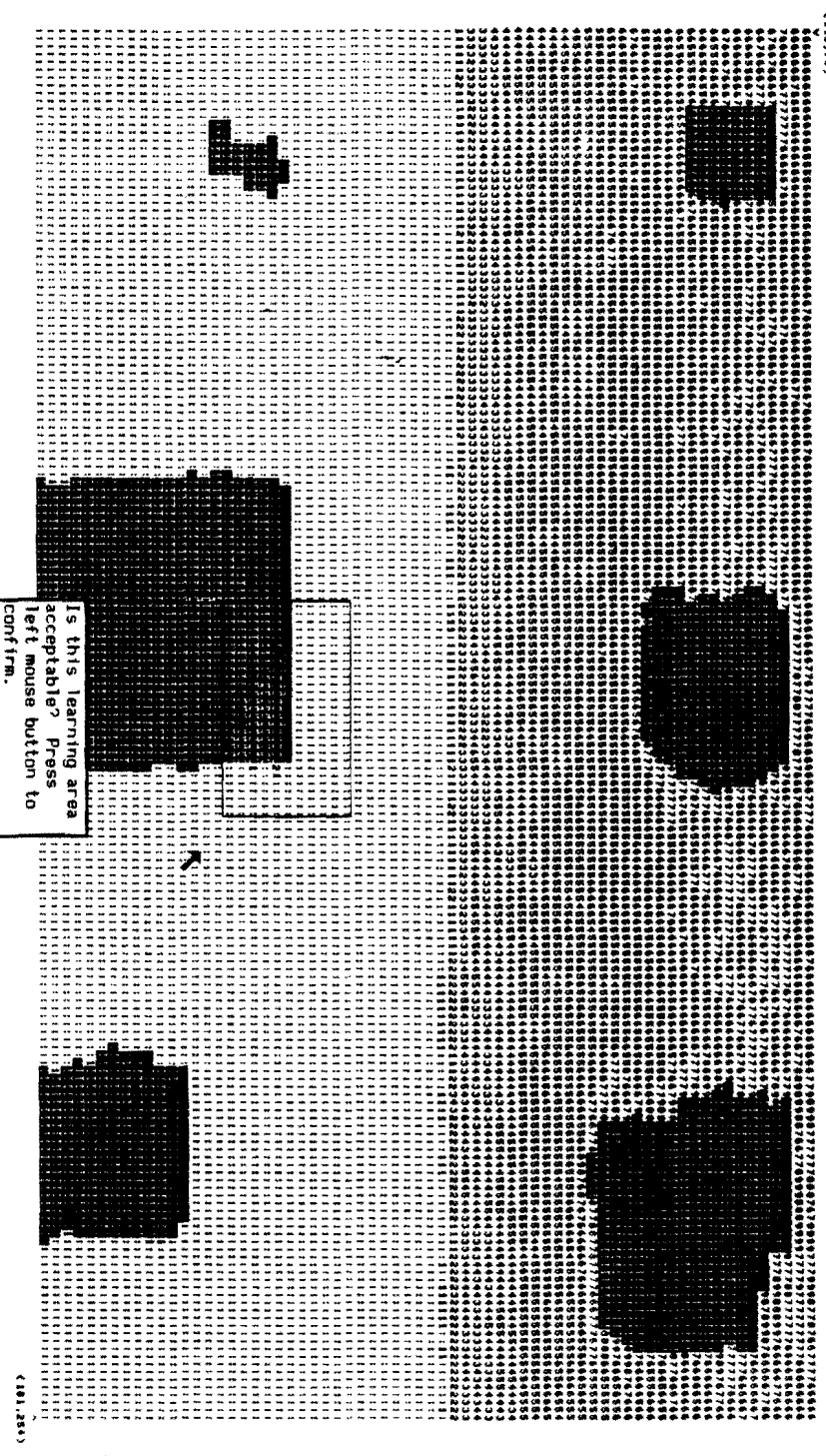
(384,250)

Figure 4.6: Classified Image for Incremental Learning

IMAGE WINDOW (you are here)

IMAGE : diaaset  
243 rows, 316 cols  
Enter LCMN RIGHT coordinates of learning area:

(18, 78)



Is this learning area acceptable? Press left mouse button to confirm.

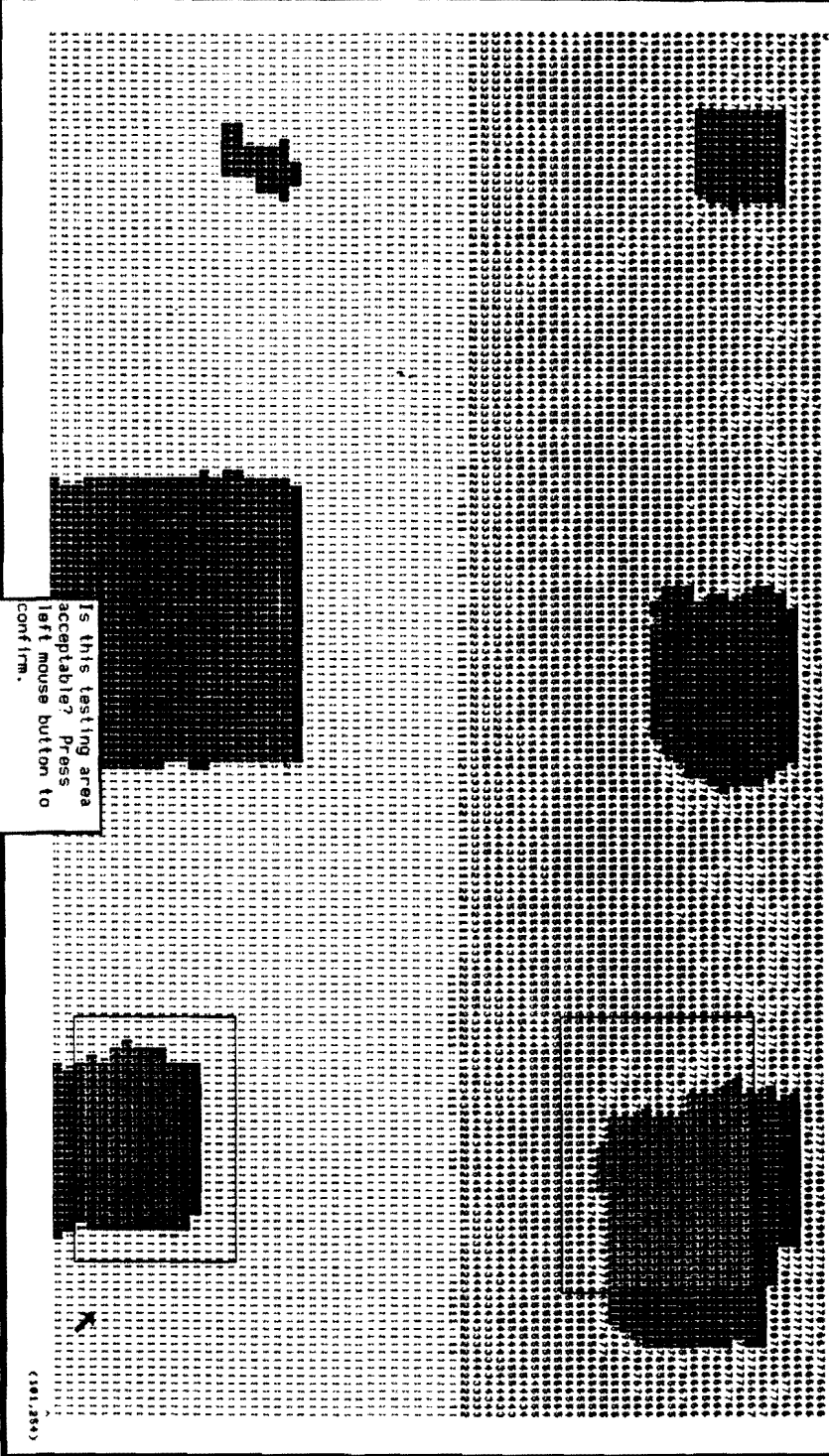
(181, 255)

Figure 4.7: Learning Area for Incremental Learning (Base Case)



IMAGE: Bismarck  
243 rows, 350 cols  
Enter LOWER RIGHT coordinates of testing area:

(181, 75)



Is this testing area  
acceptable? Press  
left mouse button to  
confirm.

(181, 254)

Figure 4.8: Two Testing Areas for Incremental Learning

the lower testing region most resembles the learning region, testing is highly successful, but not nearly as successful in the the upper testing region especially with regard to the background texture where only two per cent of the pixels were classified correctly (see figures 4.9 and 4.10 for the testing results).

In such cases of poor performance, iterative learning is not practical since many iterations would be required before the influence of the correctly classified pixels could spread among the incorrectly classified to produce an accurate set of rules. Several iterations means several applications of rule sets *whenever pixels in the image are to be classified*. For large images and domains where large numbers of images are required to be processed, as few as three or four iterations could prove prohibitive.

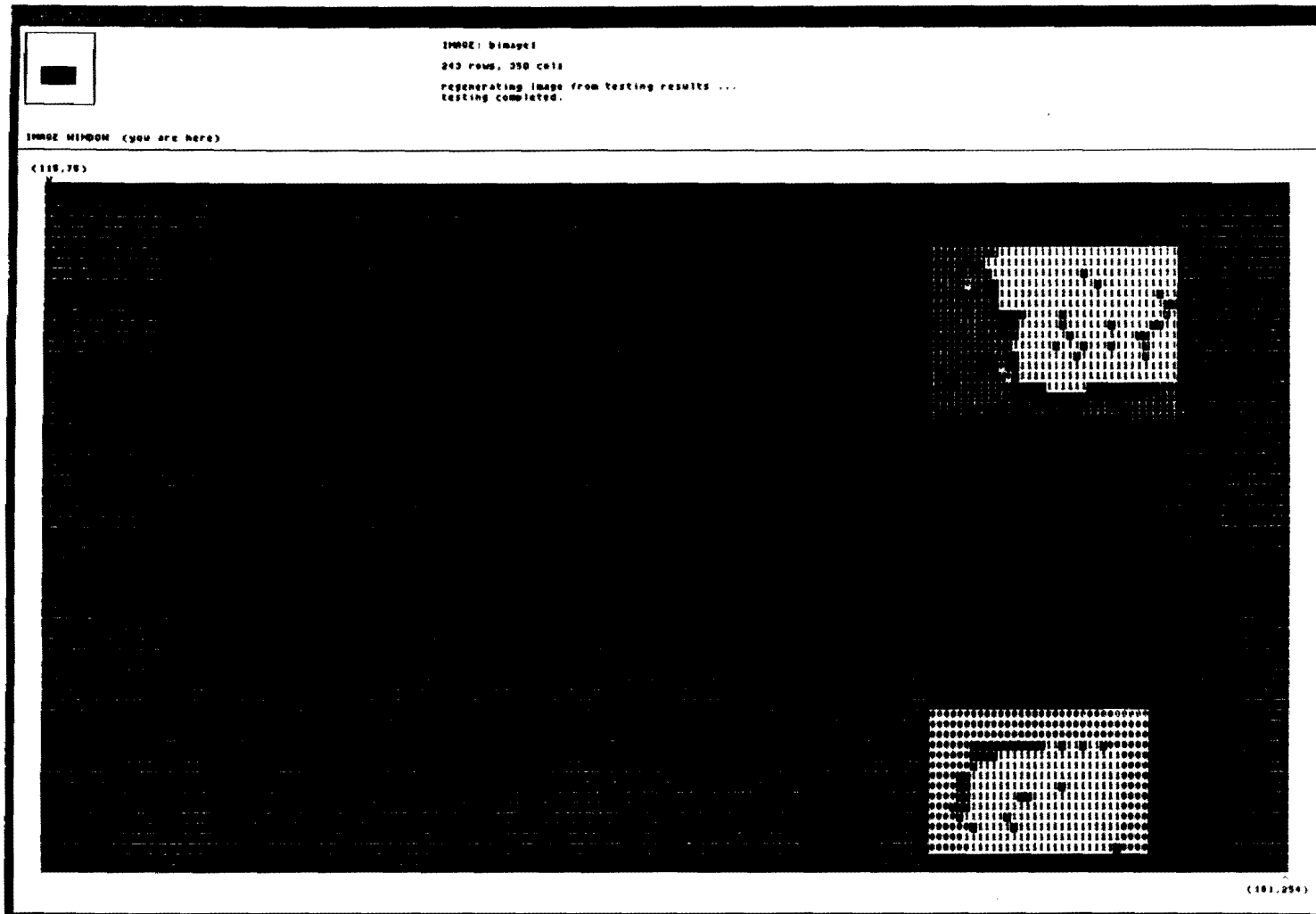
---

Testing statistics from image file "bimage1"			
Rows 166 to 179 and Columns 203 to 234 inclusive.			
CLASS	No. of pixels	% in class 0	% in class 1
0	197	99	1
1	251	13	87

Testing statistics from image file "bimage1"			
Rows 121 to 137 and Columns 203 to 238 inclusive.			
CLASS	No. of pixels	% in class 0	% in class 1
0	351	2	98
1	257	7	93

**Figure 4.9:** Results of Applying Rules to Two Testing Areas

---



**Figure 4.10: Test Results of Incremental Learning (Base Case)**



In such domains, incremental learning can be used to modify rules at a single level of iteration. Incremental learning takes additional events along with the original rules and/or events and generates new rules which can be successfully applied to all learning areas and similar sections of the image.

In this experiment, another learning area was chosen which was more similar to the problem testing area. This additional learning area is show in figure 4.11. (For a comparison of the rules for this experiment see the Appendix.)

The results of the incrementally learned rules are shown in figures 4.12 and 4.13. Although performance decreases slightly in the lower testing area, overall performance across both areas improved significantly. These rules perform well enough to form the basis for successful iterative learning, or in some domains may perform acceptably as is.

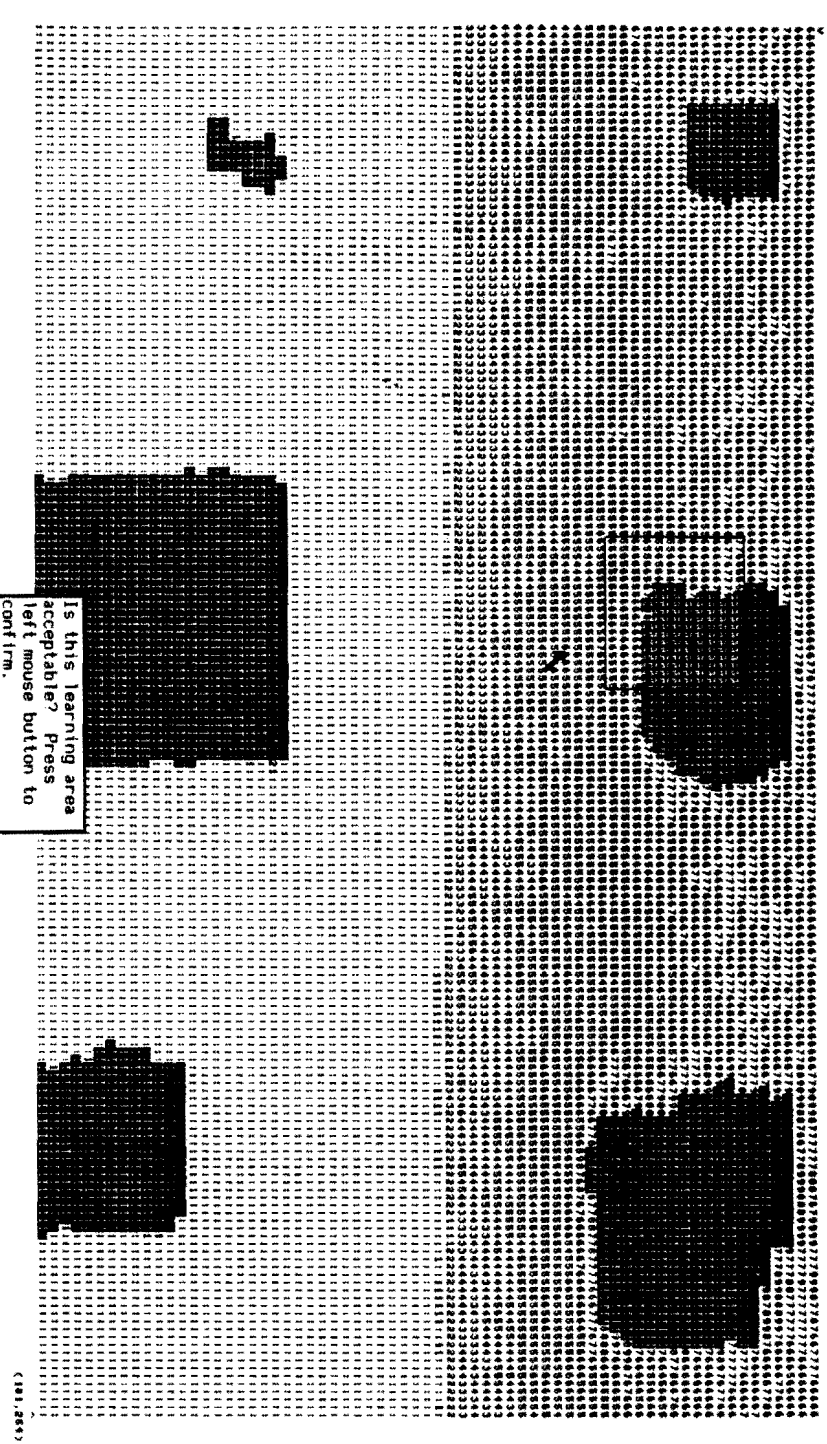
These experiments show only the fundamental capabilities of the TEXPERT system. Because of its great flexibility in choosing parameters, developing filters, and generating events, additional experiments will benefit largely from expertise in a given domain. A thorough discussion of TEXPERT strengths and weaknesses in relation to certain domains is included in the next chapter.



IMAGE: Bitmap1  
243 rows, 350 cols  
Enter LOWER RIGHT coordinates of learning area:

IMAGE WINDOW (Show area Name)

(188,78)



Is this learning area  
acceptable? Press  
left mouse button to  
confirm.

(181,285)

Figure 4.11: Second Learning Area for Incremental Learning

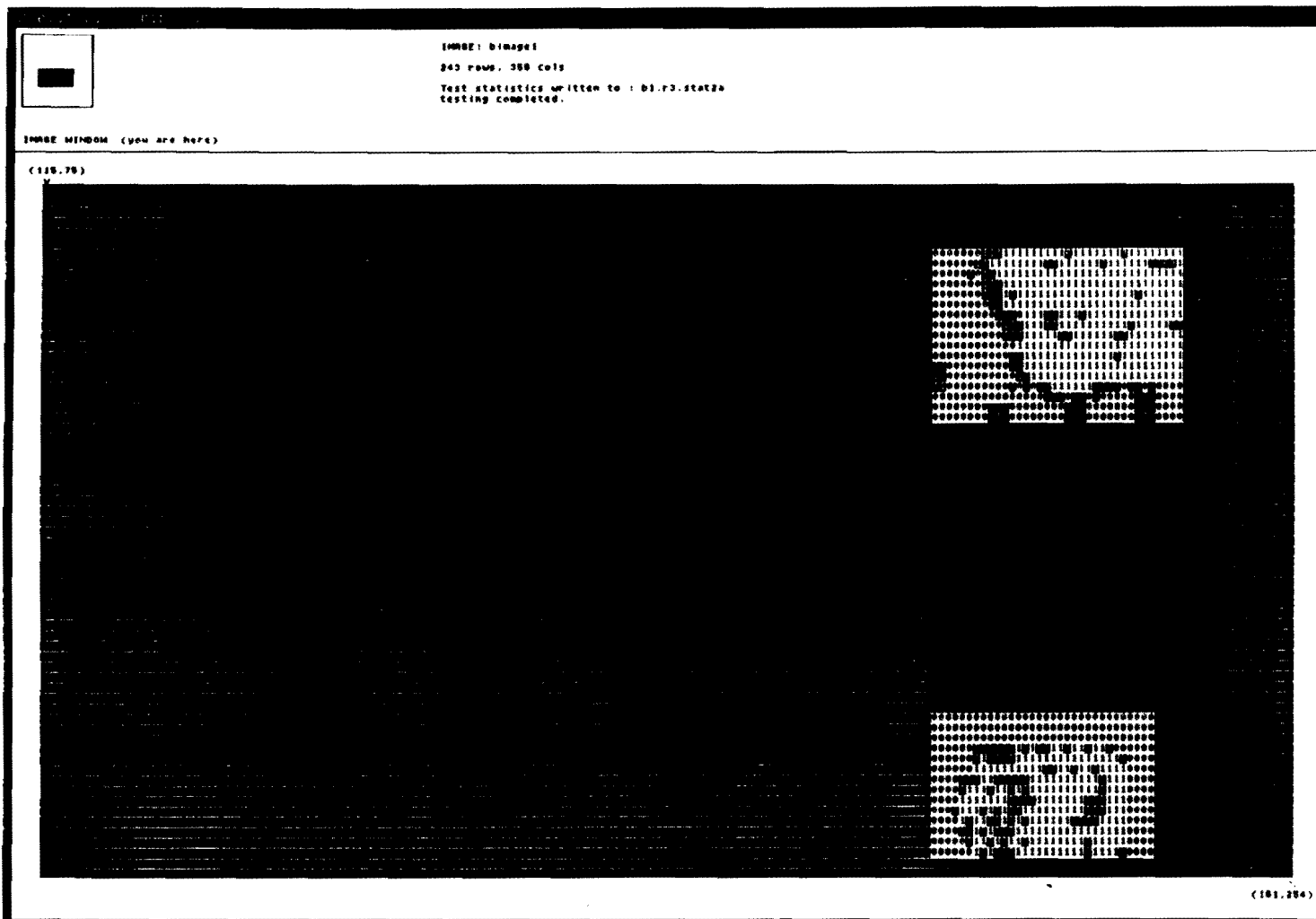


Figure 4.12: Results of Rules after Incremental Learning

---

Testing statistics from image file "bimage1"			
Rows 166 to 179 and Columns 203 to 234 inclusive.			
CLASS	No. of pixels	% in class 0	% in class 1
0	197	99	1
1	251	29	71

Testing statistics from image file "bimage1"			
Rows 121 to 137 and Columns 203 to 238 inclusive.			
CLASS	No. of pixels	% in class 0	% in class 1
0	351	75	25
1	257	7	93

**Figure 4.13:** Results of Applying Incremental Rules to Two Testing Areas

---

## CHAPTER 5.

### CONCLUSIONS

The goals of the TEXPERT system were not only to build a vision system which incorporates and learns expertise from an expert in a given domain but also to test the effectiveness of machine learning in a textural vision domain. This chapter discusses the degree to which the above goals were met, and on this basis, suggests several directions for future research in the endeavor to more fully synthesize machine learning and computer vision.

#### 5.1. Effectiveness of Learning in Vision

A system was created to allow a user to interactively tune the image and learning algorithm to produce acceptable rules. The interface is friendly and learning can be tailored according to the domain and the expertise of the expert.

The experiments in the previous chapter show the basic capabilities and effectiveness of similarity-based learning in a low-level vision domain. Reasonably accurate rules were able to be learned through both techniques of iterative and incremental learning. Effective rules were generated using only a small percentage of the actual pixels in the image.

Although test results are promising, they are by no means conclusive. Also, the question of efficiency for real-time applications was not directly addressed. Currently, testing is done with acceptable efficiency, but learning rules to use in testing can take approximately twenty or more minutes. This is acceptable only in cases where a large

percentage of testing is required. However, once satisfactory rules have been generated for processing a certain image domain, these rules need not be modified until significant changes have occurred in the domain. It would be worth a week or more of the expert's time to correctly train the system, if the automated classification using rules could save him a week or more of examining image after image without any automated help whatsoever.

Nevertheless, with a few of the possible enhancements described in the next section, the TEXPERT system could be a powerful tool in image processing. Besides detecting faults in ultra-sound images, several other areas of image processing could be aided by the incorporation of machine learning techniques into texture recognition processing. Satellite image processing and robotics vision systems could benefit from being taught by an expert in that domain. Robots could conceivably recognize objects solely on the basis of texture.

Whatever the future application of this system may be, the system can be improved in several areas at present. These enhancements are discussed in the next section.

## **5.2. Directions for Future Research**

When a child is born, one never says "Sure it's cute but it will be months before it can feed itself." Directions for growth are patiently attended and welcomed when they arrive. Likewise when a new system is "born," deficiencies should be thought of as potential areas of growth, and should be attended to with parental enthusiasm. Being a man-made computer-based offspring and not a naturally and miraculously evolving entity, however, even more patience may be required.

The areas for potential growth of the TEXPERT system comprise three major areas — namely, descriptive language capability, extended automation features, and statistical image

processing.

### 5.2.1. Descriptive Language Capability

The  $VL_1$  language used by AQ15 provides an adequate descriptive capability for utilizing the information in an image for the purposes of learning. Nevertheless, the rules generated can sometimes be difficult to understand (see Appendix). This problem can be alleviated somewhat by the wise use of the constructive induction facilities of AQ15, but a more promising solution to the problem would be the incorporation into the system of an algorithm to discover structures and substructures in the image on which basis more meaningful rules could be learned.

An interesting structure and substructure discovery algorithm which uses the  $VL_1$  language has been proposed by Holder [8]. This substructure information can easily be obtained by applying the substructure algorithm as a pre-processing stage prior to event generation, and either adding an attribute to each pixel which states that the pixel is part of a given substructure or actually changing the value of the pixel to a special value that corresponds to the specific substructure to which the pixel belongs. This structural information would make life easier for the expert who may not always be able to define an event generator to recognize low-level structures in images from the expert's domain. In fact, the expert may not even be aware of low-level structure at all.

### 5.2.2. Extended Automation Features

Although the interface to the system is friendly in the sense that it is largely menu-driven and mouse-oriented, several additional features of the system are strong candidates

for further automation. The primary area of consideration for further automation of the system is the area of event generation. As the system stands now, event generation must be completely defined by the expert. This is a great demand on her/his expertise. The system would be much improved if, once an image was classified, several learning experiments could be performed automatically, each with different methods of event generation. The event generation which produced the most accurate rules would be the final output of this extended automated learning process.

Since this multiple learning and event generation procedure could become computationally expensive, intelligent reduction of work would be useful if not required. Two modes of pre-processing would provide valuable information in reducing the amount of work done by the system.

The first way of reducing work is to perform experiments which generate events from attributes which are most relevant to the image. The most relevant attributes can be determined either before or after event generation. Methods for attribute reduction among a set of events has been researched and is discussed in [17]. To select attributes before event generation, some information must be extracted from the image — information which will determine which attributes would be more relevant to the specific images. For example, a Fourier transform may be done on an image which extracts information about the spatial frequencies in the image. High frequencies would suggest the use of a small template which would extract much of the information about the variations of intensity occurring in the near vicinity of the pixel, whereas low frequencies would suggest the use a larger template in which near neighboring values of the pixel are not used as attributes.



It may be possible that given enough image input into the system, rules could be learned which determine the most likely means of effective event generation based on features found during image preprocessing. Such a system would be learning how to learn to recognize texture. This "meta-learning" system could significantly reduce the burden of event template definition currently placed on the expert.

The second way of reducing work is to reduce the number of events which are inputted to the learning algorithm at each learning phase. In large images, many events are often redundant and need not be input into the algorithm. Selecting representative events for learning as discussed in [10] would effectively reduce the amount of work done by the learning algorithm.

### **5.2.3. Statistical Image Processing**

The last area for enhancing the capabilities of the TEXPERT system is in adding the power of raw statistical image and signal processing methods. As the system stands now, only "local" processing techniques are available. "Global" processing such as Fourier transforms cannot be easily used in generating events. A facility for producing related images and allowing multiple related images (such as stereo images) to be input into the system would provide much more information to the learning algorithm, which would most likely result in more powerful and meaningful rules.

### **5.3. Final Remarks**

This chapter has described the power and problems of using similarity-based learning methods to solve texture recognition and image segmentation problems in digital images.

Directions of research to alleviate the problems have been discussed and potential new applications have been suggested. Although promise has been demonstrated, much work remains to be done. Nevertheless, a synthesis of machine learning and computer vision does appear to offer practical insights into the very difficult problem of enabling computers to "see."

## APPENDIX A.

## Experiment Rule Output

## A.1. Iterative Learning

This section contains the AQ15 output for the iterative learning experiment. Two sets of rules are shown: the first set consists of the rules derived from the raw image, the second set consists of the rules derived from the image which was regenerated from the application

---

```
class0-outhypo
# cpx
1  [x2&x3 < 2|[x4 = 0 v 1 v 4]|x5&x7 = 0 v 1 v 3|
2  [x2 = 0 v 1|[x5&x9 < 2|[x7 = 0 v 1 v 4|
3  [x1 = 0 v 2 v 4|[x3 < 1|[x4 = 0 v 2 v 3|[x5 = 0 v 1 v 4|[x6 = 0 v 1 v 3|
4  [x1 < 1|[x6 = 0 v 1|[x7 = 0 v 2 v 4|[x9 < 2|
```

```
class1-outhypo
# cpx
1  [x6 = 2 v 4|[x7 = 2|
2  [x5 = 2|[x7 = 1 v 3|
3  [x1 = 1 v 3|[x4 = 2 v 3|[x9 = 2|
4  [x3 = 1|[x7 = 2 v 4|[x9 = 2|
5  [x4&x5 = 2 v 3|[x9 = 2|
6  [x2&x6 >= 2|[x4&x5 = 2 v 3|
7  [x2 >= 2|[x5&x6 = 2 v 4|
8  [x1 = 1|[x7 = 2|
9  [x2&x6 >= 2|[x3 = 2|[x4 = 1 v 4|
10 [x2&x6 >= 2|[x4 = 1 v 4|[x7 = 2 v 4|
11 [x2 = 2|[x6 = 2 v 4|
12 [x3 = 1|[x4 = 2 v 3|[x7 = 3|
13 [x4 = 1 v 4|[x5 = 2 v 4|[x7 = 3|
```

```
This learning used (milliseconds of CPU time):
System time:      31683
User time :      1901850
```

## Rules From Raw Image

of the first set of rules to the image.

The first set of rules may appear difficult to understand because of the nature of the attributes. If one bears in mind that the x5 attribute represents the value of the pixel itself, the rules can be interpreted as specializations of the condition that a pixel belongs in class0 if its value is 0 or 1 or 3 (or sometimes 4) and belongs in class1 if its value is 2.

The second set of rules use special values 14 and 15 for pixels which were correctly classified into class0 and class1, respectively. These rules are much simpler to understand and essentially give a single condition for those pixels already classified correctly, and then extend these conditions for misclassified pixels in the vicinity of correctly classified pixels.

---

```

class0-outhypo
#   cpx
1   [x5 < 1 v 15]
2   [x5 <= 14] [x7 < 14]
3   [x4&x6 < 1 v 15]

class1-outhypo
#   cpx
1   [x5 = 15] [x6 = 1 v 15]
2   [x4 = 1 v 15] [x5 = 15]
3   [x5&x6 = 1 v 15] [x7 = 14]
4   [x4&x5 = 1 v 15] [x7 = 14]

```

```

This learning used (milliseconds of CPU time):
System time:      10784
User time   :      288833

```

### Rules From Regenerated Image

---

## A.2. Incremental Learning

This section consists of the two sets of rules derived during the incremental learning experiment. The first set of rules is best understood by focusing on the x11 attribute in complex (cpx) #1 in both rules. x11 is the maximum value of all pixels immediately surrounding the pixel to be classified. With the specializations provided by the other conditions in the complex, pixels with surrounding maximum value equal to zero through three inclusive or equal to ten belong to class0, otherwise they belong to class1. This

```
class0-outhypo
# cpx
1 [x1 < 5 v 7 v 10] [x4 < 3 v 7 v 8 v 10] [x6 = 0..2 v 5 v 8 v 11..13]
  [x11 = 0..3 v 10]
2 [x1 = 0..4 v 6 v 7 v 13] [x2 < 2 v 4 v 10] [x3 < 9 v 10 v 12]
  [x6 < 4 v 6..8] [x7 = 0..3 v 8 v 9 v 11 v 13]
  [x9 = 0 v 1 v 4 v 6..8 v 10 v 12]
```

```
class1-outhypo
# cpx
1 [x7 = 4..7 v 10 v 12] [x11 = 4..9 v 11..13]
2 [x4 = 3 v 7 v 8 v 10] [x6 = 4 v 6..8]
3 [x6 = 9 v 10] [x7 = 10 v 12] [x10 = 6 v 8 v 9]
4 [x2 = 2 v 4 v 10] [x11 = 4..9 v 11..13]
5 [x4 = 8 v 10..13] [x7 = 10 v 12] [x11 >= 11]
6 [x1 = 8 v 9 v 11 v 12] [x4 = 8 v 10..13] [x11 >= 11]
7 [x1 = 5 v 8..12] [x6 = 3 v 4 v 6 v 7 v 9 v 10]
8 [x1&x2 = 10] [x4 = 8 v 10..13]
9 [x1 = 8 v 9 v 11 v 12] [x6 = 9]
10 [x9 = 9 v 11 v 13] [x10 = 6 v 8 v 9] [x11 >= 11]
11 [x6 = 8 v 9 v 11] [x9 = 9 v 11 v 13] [x11 >= 11]
12 [x2 = 2 v 4 v 10] [x4 = 3 v 7 v 8 v 10]
13 [x6 = 8] [x11 >= 11]
14 [x6 = 4 v 6 v 7]
15 [x9 = 2 v 3 v 5 v 9 v 11 v 13] [x11 = 4..9 v 11..13]
16 [x1 = 5 v 8..12] [x4 = 3 v 7 v 8 v 10]
```

This learning used (milliseconds of CPU time):

```
System time: 77534
User time : 4833467
```

### Rules From First Learning Area

generalization performs well in the first testing area but poorly in the second because the maximum value of neighboring pixels varies more among class0 pixels in the second testing area.

The second set of rules shows how the maximum value attribute was appropriately modified to more correctly distinguish class0 and class1 pixels in the second testing area.

---

```

class0-outhypo
# cpx
1 [x4 < 3 v 9 v 12 v 13] [x6 < 2 v 9 v 10] [x7 < 8..10]
  [x10 < 2 v 3 v 5 v 11] [x11 < 4 v 5 v 11 v 12]
2 [x1 < 9 v 11 v 12] [x2 = 0 v 1 v 4 v 7 v 8 v 11..13]
  [x7 = 0..4 v 7 v 8 v 12] [x9 = 0..2 v 4 v 6..8 v 10 v 13] [x10 < 2 v 5]
3 [x1 = 0..3 v 7] [x2 = 0 v 4 v 9..13]

class1-outhypo
# cpx
1 [x1 = 4..6 v 8..13] [x9 = 3 v 5 v 9 v 11 v 12] [x11 = 4 v 5 v 11 v 12]
2 [x1 = 4..6 v 8..13] [x7 = 9 v 10]
3 [x2 = 2 v 3 v 5 v 6] [x10 = 2 v 3 v 5 v 11]
4 [x1 = 4..6 v 8..13] [x2 = 2 v 3 v 5 v 6 v 9 v 10] [x6 = 2 v 9 v 10]
5 [x1 = 4..6 v 8..13] [x4 = 3 v 9 v 12 v 13] [x9 = 3 v 5 v 9 v 11 v 12]
6 [x1 = 4..6 v 8..13] [x9 = 3 v 5 v 9 v 11 v 12] [x10 = 2 v 3 v 5 v 11]
7 [x2 = 1..3 v 5..8] [x4 = 3 v 9 v 12 v 13] [x7 = 5 v 6 v 9..11 v 13]
8 [x1 = 4..6 v 8..13] [x2 = 2 v 3 v 5 v 6 v 9 v 10] [x11 = 4 v 5 v 11 v 12]
9 [x2 = 2 v 3 v 5 v 6] [x7 = 8..10]
10 [x2 = 1..3 v 5..8] [x6 = 2 v 9 v 10] [x9 = 3 v 5 v 9 v 11 v 12]
11 [x2 = 1..3 v 5..8] [x9 = 3 v 5 v 9 v 11 v 12] [x11 = 4 v 5 v 11 v 12]
12 [x1 = 9 v 11 v 12] [x4 = 3 v 9 v 12 v 13]
13 [x2 = 1..3 v 5..8] [x7 = 9 v 10]
14 [x2 = 1..3 v 5..8] [x6 = 2 v 9 v 10] [x7 = 5 v 6 v 9..11 v 13]
15 [x1 = 4..6 v 8..13] [x10 = 2 v 5]
16 [x2 = 2 v 3 v 5 v 6] [x11 = 4 v 5 v 11 v 12]
17 [x1 = 4..6 v 8..13] [x2 = 2 v 3 v 5 v 6 v 9 v 10] [x4 = 3 v 9 v 12 v 13]
18 [x1 = 9 v 11 v 12] [x11 = 4 v 5 v 11 v 12]
19 [x2 = 2 v 3 v 5 v 6] [x4 = 3 v 9 v 12 v 13]
20 [x2 = 1..3 v 5..8] [x10 = 2 v 5]
21 [x2 = 1..3 v 5..8] [x7 = 5 v 6 v 9..11 v 13] [x11 = 4 v 5 v 11 v 12]
22 [x1 = 4..6 v 8..13] [x4 = 3 v 9 v 12 v 13] [x7 = 5 v 6 v 9..11 v 13]

```

This learning used (milliseconds of CPU time):  
 System time: 12500  
 User time : 4035117

### Rules From Second Learning Area

---

When the maximum value attribute is considered (and this is true for a large number of events since the complexes are ordered according to the number of learning events which satisfy the complex), class0 pixels can take on a significantly wider range of values, whereas the class1 pixels can take on fewer values.

## REFERENCES

- [1] Michalski, R. S., "AQVAL/1—Computer Implementation of A Variable-Valued Logic System VL1 and Examples of Its Application to Pattern Recognition", *Proceedings of the First International Joint Conference on Pattern Recognition*, pp. 3-17, Washington, DC, 1973.
- [2] Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System", M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1984.
- [3] Julesz, Bela, "Experiments in the Visual Perception of Texture", *Scientific American*, 232, pp. 34-43, April 1975.
- [4] Julesz, Bela, "Textons, the Elements of Texture Perception, and their Interactions", *Nature*, 290, pp. 91-97, March 1981.
- [5] Marr, David, *Vision*, W. H. Freeman and Company, New York, 1982.
- [6] Ballard, D.H., Brown, C.M., *Computer Vision*, Prentice-Hall, Englewood Cliffs, N.J., 1982.
- [7] Channic, T.D., "Editing Network-Structured Knowledge Bases in the ADVISE System", Report No. UIUCDCS-F-85-934, Department of Computer Science, University of Illinois, Urbana, IL, 1985.
- [8] Holder, L., "Discovering Substructure in Examples," M.S. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.
- [9] Uhrik, C.T., "A Rule Exerciser for Knowledge Base Enhancement in Expert Systems," M.S. Thesis, Department of Computer Science, University of Illinois, 1985.
- [10] Michalski, R.S., Larson, J.B., "Selection of Most Representative Training Examples and Incremental Generation of VL<sub>1</sub> Hypotheses," Report No. UIUCDCS-R-78-867, Department of Computer Science, University of Illinois at Urbana-Champaign, May, 1978.
- [11] Gonzales, R.C., Wintz, P., *Digital Image Processing*, Addison-Wesley, Reading, MA, 1977.
- [12] Rosenfeld, A., Kak, A.C., *Digital Picture Processing*, Academic Press, New York, NY, 1976.



- [13] Nagin, P.A., Hanson, A.R., Riseman, M., "Region Relaxation in a Parallel Hierarchical Architecture", *Real-Time/Parallel Computing*, Onoe, Preston, Rosenfeld, eds., pp 37-61, Plenum Press, NY 1981.
- [14] Kodaira, N., Kato, K., Hamada, T., "Man-Machine Interactive Processing for Extracting Meteorological Information from GMS Images," *Real-Time/Parallel Computing*, Onoe, Preston, Rosenfeld, eds., pp 297-324, Plenum Press, NY 1981.
- [15] Hanaki, S., "An Interactive Image Processing and Analysis System," *Real-Time/Parallel Computing*, Onoe, Preston, Rosenfeld, eds., pp 219-226, Plenum Press, NY 1981.
- [16] Michalski, R.S., Baskin, A.B., Uhrik, C., and Channic, T., "The ADVISE.1 Meta-Expert System: The General Design and a Technical Description," Intelligent Systems Group Reports, Department of Computer Science, University of Illinois, January 1987.
- [17] Baim, P.W., "The Promise Method for Selecting Most Relevant Attributes for Inductive Learning Systems," Report No. UIUCDCS-F-82-898, Department of Computer Science, University of Illinois at Urbana-Champaign, September, 1982.
- [18] Laws, K.I., "Textured Image Segmentation", PhD dissertation, Department of Engineering, University of Southern California, 1980.
- [19] Tou, J.T., Gonzales, R.C., *Pattern Recognition Principles*, Addison-Wesley, Reading, MA, 1974.
- [20] Fukunga, K., *Introduction to Statistical Pattern Recognition*, Academic Press, NY, 1972.
- [21] Michalski, R.S., Chilausky, R.L., "Knowledge Acquisition by Encoding Expert Rules Versus Computer Induction From Examples: A Case Study Involving Soybean Pathology," *International Journal of Man-Machine Studies*, **12**, pp. 63-87, 1980.
- [22] Silverman, J., Cooper, D., "Bayesian Clustering for Unsupervised Estimation of Surface and Texture Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **10**, 4, pp. 482-495, July 1988.
- [23] Michalski, R.S., "A Theory and Methodology of Inductive Learning," *Machine Learning*, Michalski, R.S., Carbonell, J. and Mitchell, T. (eds.) pp. 83-134, Tioga Press, Palo Alto, CA, 1983.
- [24] Michie, D., "Experiments on the Mechanization of Game Learning," *Computer Journal*, **25**, 1, pp. 105-112, 1982.

- [25] Michalski, R.S., Mozetic, I., Hong, J.R., Lavrac, N., "The AQ15 Inductive Learning System," Report No. UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois at Urbana-Champaign, July, 1986.
- [26] Hong, J.R., Mozetic, I., Michalski, R.S., "AQ15: Incremental Learning of Attribute-Based Descriptions from Examples," Report No. UIUCDCS-F-86-949, Department of Computer Science, University of Illinois at Urbana-Champaign, May, 1986.