Interactive Machine Learning for Refinement and Analysis of Segmented CT/MRI Images

Erol Sarigul

A doctoral dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

A. Lynn Abbott, Chair
Daniel L. Schmoldt
Amy E. Bell
Earl Kline
Richard Conners
Anbo Wang

17th September, 2004
Blacksburg, Virginia

Keywords: Image segmentation, machine learning, decision trees,
postprocessing, user interface.

Interactive Machine Learning for Refinement and Analysis of Segmented CT Images

Erol Sarigul

(ABSTRACT)

This dissertation concerns the development of an interactive machine learning method for refinement and analysis of segmented computed tomography (CT) images. This method uses higher-level domain-dependent knowledge to improve initial image segmentation results.

A knowledge-based refinement and analysis system requires the formulation of domain knowledge. A serious problem faced by knowledge-based system designers is the knowledge acquisition bottleneck. Knowledge acquisition is very challenging and an active research topic in the field of machine learning and artificial intelligence. Commonly, a knowledge engineer needs to have a domain expert to formulate acquired knowledge for use in an expert system. That process is rather tedious and error-prone. The domain expert's verbal description can be inaccurate or incomplete, and the knowledge engineer may not correctly interpret the expert's intent. In many cases, the domain experts prefer to do actions instead of explaining their expertise.

These problems motivate us to find another solution to make the knowledge acquisition process less challenging. Instead of trying to acquire expertise from a domain expert verbally, we can ask him/her to show expertise through actions that can be observed by the system. If the system can learn from those actions, this approach is called *learning by demonstration.*

We have developed a system that can learn region refinement rules automatically. The system observes the steps taken as a human user interactively edits a processed image, and then infers rules from those actions. During the system's *learn mode*, the user views labeled images and makes refinements through the use of a keyboard and mouse. As the user manipulates the images, the system stores information related to those manual operations, and develops internal rules that can be used later for automatic postprocessing of other images. After one or more training sessions, the user places the system into its *run mode*. The system then accepts new images, and uses its rule set to apply postprocessing operations automatically in a manner that is modeled after those learned from the human user. At any time, the user can return to learn mode to introduce new training information, and this will be used by the system to updates its internal rule set.

The system does not simply memorize a particular sequence of postprocessing steps during a training session, but instead generalizes from the image data and from the actions of the human user so that new CT images can be refined appropriately.

Experimental results have shown that IntelliPost improves the segmentation accuracy of the overall system by applying postprocessing rules. In tests two different CT datasets of hardwood logs, the use of IntelliPost resulted in improvements of 1.92% and 9.45%, respectively. For two different medical datasets, the use of IntelliPost resulted in improvements of 4.22% and 0.33%, respectively.

To My Family

Acknowledgments

I am extremely thankful to my advisor Prof. A. Lynn Abbott, for his patience and valuable guidance in this study. Without his support, patience, and understanding, this dissertation would not be exist. I owe a lot to him. His support and assistance during years of this study will not be forgotten.

I would like to thank my committee members: Dr. Schmoldt, Dr. Conners, Dr. Bell, Dr. Kline, and Dr. Wang for their feedback and stimulating question during the course of this research. I would like to specially thank to Phil Araman and Matt Winn for their help for creating the reference images fo hardwood log dataset and training part of IntelliPost. My sincere appreciation goes to Phil Araman for giving me financial support to help me finish this research.

I would like to thank my family especially my wife, Emine for her emotional support, patience, and love during this research. I would like to thank her help by taking care of our sons so that I could more time to study to finish it. No words would be enough to express how her help is appreciated. My sons, Alperen and Yusuf, are my emotional inspiration to finish this study. When I needed emotional support, they were there to give it enough.

Special thanks and appreciation go to also my good friends: Anbumani Subramanian, and Sang-Mook Lee for taking time to have stimulating discussion for my research, their help, and friendship.

I would like to express my deepest appreciation to my parents, and their emotional support and patience during this study and never ending prayer for me. Plain words are not enough to express my feeling for them. Finally, I would not be able to finish this thesis without the help of God. My ultimate appreciation goes to him.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Problem definition and motivation

Increasing demand and limited forest resources continue to drive the hardwood industry to seek more productive means of converting logs to lumber. Conventional log sawing practices waste considerable amounts of valuable wood, largely because most defects that adversely affect board quality are at unknown locations inside the logs. Traditionally, the sawmill operator chooses a sawing strategy by visually examining the exterior of the log, modifying the strategy as sawing exposes the log interior. This method has several drawbacks. Among the most noticeable drawbacks are the following: first, the outside bark distortion provides only limited information about internal features to experts. Second, human experts can be heavily affected by boredom, fatigue, and working conditions.

Developing nondestructive sensing and analysis methods that can detect and identify interior defects with high accuracy is critical to future efficiency improvement for sawmills. Studies have shown, for example, that the commercial value of lumber can be improved by

11% to 21% through the careful selection of sawing strategies, particularly if internal defect locations were known [152, 180, 186]. Those studies assume that knowledge of internal defects is available, and that information is used to choose the best sawing position and method.

Due to the fact that most defects of interest are internal, a nondestructive sensing technique is needed. Different sensing methods have been tried, including nuclear magnetic resonance [34], ultrasound [76], and x-rays. Because of its efficiency, resolution, and extensive usage in medicine, x-ray computed tomography has received extensive testing in round wood applications [27, 134, 61, 74, 124, 172, 179, 205]. If complete internal information of the log is known, it is possible to optimize the sawing strategy of the log based on both dimension and defect locations. Such a sawing procedure is outlined by Guddanti and Chang [33], and is illustrated in Figure 1.1.

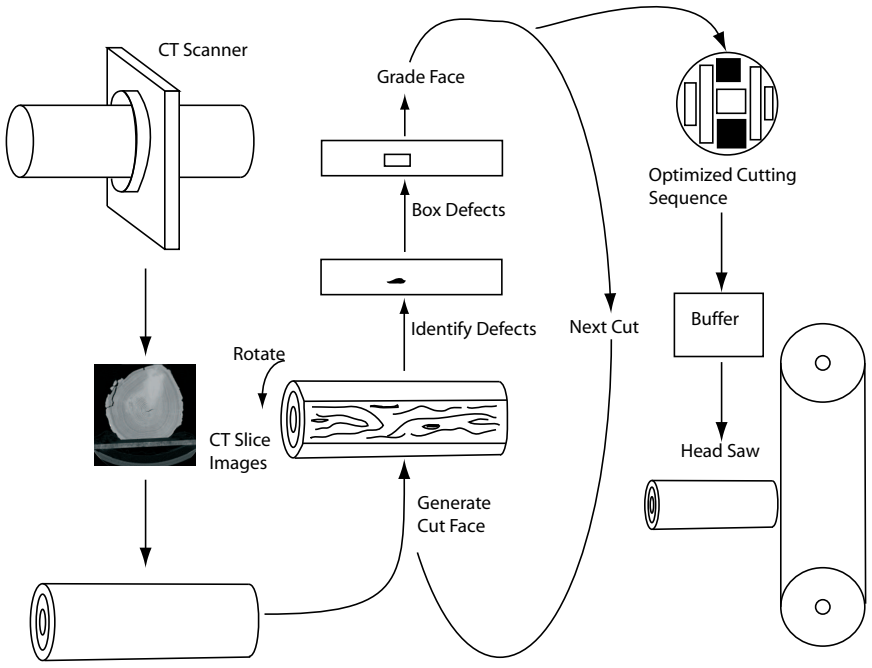


Figure 1.1: Overall CT based sawmill system (adapted from [33]).

Utilizing CT (Computed Tomography) imaging technology brings several hurdles to overcome. It is necessary to have some automated technique to interpret scan information so that

the saw operator can be presented with the information needed to make proper sawing decisions. The outcome of CT imaging cannot be readily synthesized into a three-dimensional (3D) mental model by human operators [207]. Due to these limitations, there is a need to have an automated technique to detect, identify, and label internal defects, and then construct 3D model of a log using this information. The early work on automatically labeling internal log defects proved the feasibility of utilizing CT images for this purpose. In those studies, a variety of methods have been employed to segment different regions of a CT image and then identify or label those segmented regions. Quite often, image segmentation methods are based on threshold values derived from a CT image's histogram [179, 203, 172]. In some cases, texture-based methods have been used for defect labeling [61, 204, 18], and not for segmentation. Knowledge-based classification [206, 202], analyzing shapes [61, 172], and morphological operations [172] have been used to label defects.

While previous studies have demonstrated feasibility, they present some serious limitations. First, classification accuracy was reported anecdotally based on training and single test sets. No extensive statistical methods have been found in the literature. Second, they were not designed to be used in real time operation. Third, texture and density information are very important for human differentiation of regions in a CT/MRI. In some cases, image segmentation algorithms can utilize texture information for detecting regions, and for partitioning regions into defects.

These limitations have been addressed by Li, Abbott, and Schmoldt [105, 164]. They have developed an automated detection and identification system for hardwood log defects that has demonstrated highly accurate labeling of log defects in CT slices. In contrast to the previous global approaches that separates the tasks of segmentation and region labeling, their approach operates on using local pixel neighborhoods primarily, and effectively combines segmentation and labeling into a single classification step.

Although the classification performance of the ANN based classifier was reported to be quantitatively high, they noted that there was a need for a postprocessing module to refine the classified image further. Because the initial ANN-based classifier depends primarily on local information, and identifies defects on a pixel-by-pixel basis, misclassification tends to occur at small, isolated locations. They have chosen a hard-wired, fixed postprocessing approach that is based on mathematical morphology to process such misclassified regions. While this approach has shown some success, it is limited in its ability to handle all cases of postprocessing needs.

This dissertation describes a new system that replaces their fixed postprocessing module with with an adaptive system that is capable of "learning" new sets of postprocessing rules. The system operates by observing the postprocessing operations as a human user interactively edits segmented CT/MRI images. As the user manipulates the images, the system stores information related to those manual operations, and develops internal rules that can be used later for automatic postprocessing of other images. After one or more training sessions, the user accepts new images, and uses its rule set to apply postprocessing operations automatically in a manner that is modelled after those learned from the human user. The focus of this dissertation is to introduce the learning method by which the system develops postprocessing rules, and to present results that demonstrate the efficiency of this approach.

1.2 Background

The work described in this dissertation builds on a previously developed methods that use artificial neural networks (ANN) for image segmentation [105, 164, 163, 157, 158, 159, 161, 160]. These authors described an approach that uses ANN to classify pixels individually, using small neighborhoods of CT density values as input and assigns a label (knot, decay,

split, bark, or clear wood) to each pixel in the image.

The overall classification system described in [105, 164, 163] consists of three modules as shown in Figure 1.2: (1) a preprocessing module, (2) an artificial neural network (ANN) module, and (3) a postprocessing module.

The preprocessing module distinguishes an object from background (air) and internal voids, and normalizes CT density values to accommodate density values. The extraction of foreground objects from background is performed by Otsu's thresholding method [135] using a histogram of a CT slice. This thresholding method assumes bimodal distribution of pixel values and selects the threshold level by minimizing within-group variance. A second objective of the preprocessing module is to normalize values, so that the ANN based classifier can work consistently for different CT/MRI slices. Because Hounsfield numbers are directly related to density, CT/MRI images vary dramatically for different species and by moisture content. If normalization is not applied, there will be no consistent relationships among similar regions across CT/MRI images, and the ANN classifier would be unable to learn any useful patterns. To ensure consistency of similar defect regions in CT/MRI images, histogram normalization is performed as follows:

$$x_{norm} = \frac{1}{x_a}\left[x_0 + \frac{x_a - x_{cw}}{1 + \exp\, a(\frac{x_{cw}}{2} - x_0)}\right] \tag{1.1}$$

This formula transforms original density values x_0 to normalized values x_{norm} giving approximately the same density values to important areas of CT/MRI images. The translation anchor value x_a is arbitrarily selected to be greater than the peak density value (x_{cw}) of foreground object in the histogram. The constant a is empirically determined.

The ANN module labels each non-background pixel of a slice using histogram-normalized values from small windows, typically of size 3×3×3 or 5×5, centered on the pixel location to

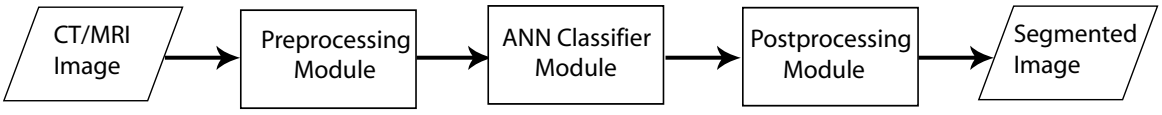


Figure 1.2: Overall image segmentation system.

be classified as illustrated in Figure 1.3. Each histogram normalized value in the neighborhood along with radial distance serve as an feature vector to the ANN. The radial distance of a pixel under consideration from the centroid of the foreground region of the slice is provided as an additional input to the ANN classifier. The radial distance information provides contextual information that helps in classification. The output nodes of the ANN represent each class to be detected. The class associated with the output node that has the largest value for a given input vector is assigned as corresponding label for that input.

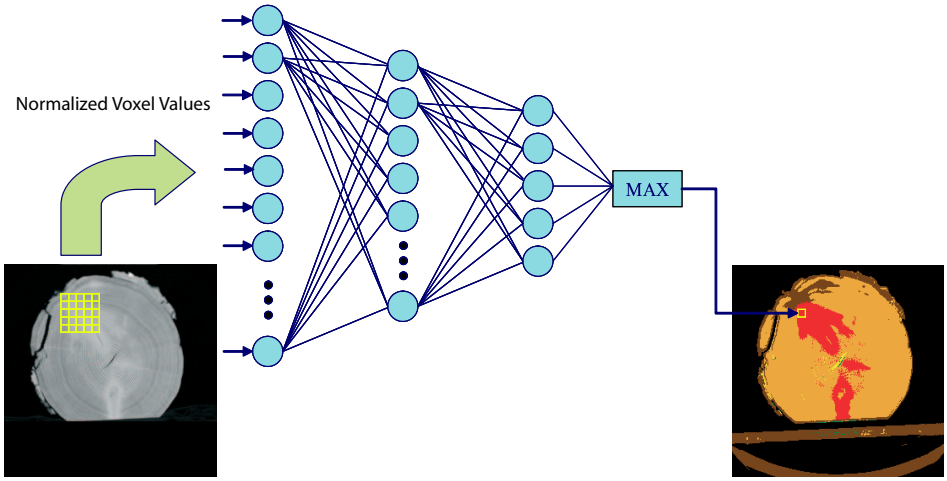
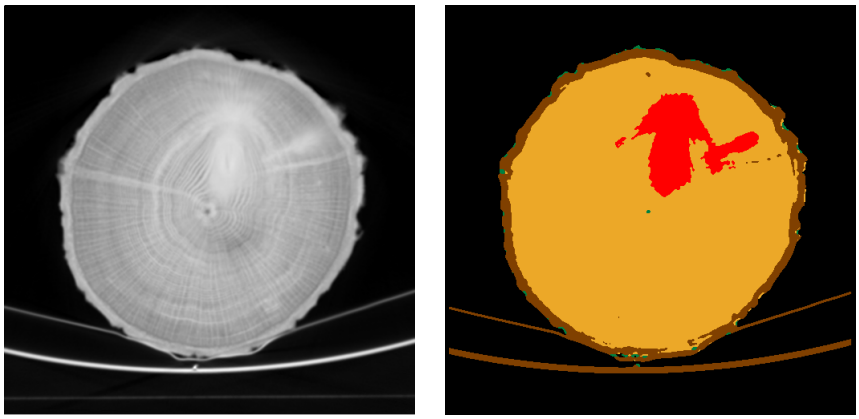


Figure 1.3: The architecture of the ANN classifier. The normalized values are collected within a neighborhood of each pixel and then fed to the trained ANN classifier to determine the class of that pixel. The Max function is used after the output layer to select the class assignment of the pixel. The 5×5 window at the left is drawn out of proportion for the sake of clarity.

The classification algorithm performs the task of segmentation and labeling in a single step. The network is trained using a conventional back-propagation method [123]. Training samples are collected from multiple CT or MRI slices and provided as a training dataset.

Figure 1.4 shows visual results of typical output of a trained ANN classifier. It is clear from visual inspection that ANN classifier works well for this case.

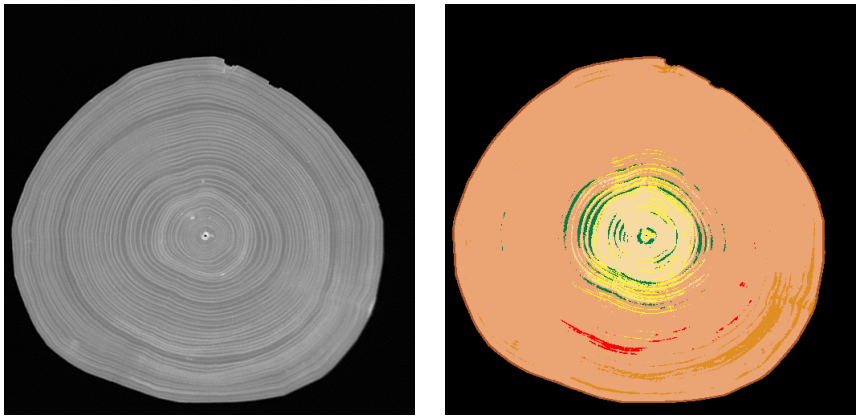


(a) (b)

Figure 1.4: Typical output of ANN classifier for red oak log. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.

Because the ANN classification algorithm primarily uses local information along with radial distance, spurious misclassifications can occur as shown in Figure 1.5. In this case, the ANN classifier is misled by density information near the center of the log. Small density changes in the center cause the ANN to label regions as heartwood instead of sapwood. Some annual rings and low density regions are incorrectly labeled as split and decay respectively. A postprocessing procedure can be used to correct such misclassified regions thereby improving overall segmentation performance. They have proposed a postprocessing method that was based on morphological image processing.

Many of these incorrect labels have negligible effect on statistical classification accuracy, which depends on pixel counts alone. Qualitatively, however, the removal of several small regions and the smoothing of region contours can be desirable. Most of the needed changes can be accomplished with relatively simple postprocessing steps. The difficulty lies in the development of postprocessing rules that determine when to apply these simple steps. Hard-wired postprocessing rules that indiscriminately remove all regions smaller than some



(a) (b)

Figure 1.5: Using just density information is not enough, as illustrated with this sugar maple log slice. The ANN classifier can be misled by density alone. This result demands postprocessing that could use high level information. Light tan color represents sapwood, beige color represents heartwood, yellow represents split defects, green represents decay, red represents knot, and dark tan color represents live bark.

threshold may also remove valid defect regions such as splits, which are relatively small. Because it is difficult to manually specify an exhaustive set of postprocessing rules that will work well for all possible situations, the emphasis of this research has been to allow the machine to develop its own rules, based on observations of a human user.

1.3 Brief survey of image segmentation

The main goal of this study is to improve image segmentation performance for the system described in the previous section. Since this study is related to the image segmentation problem, we will survey previously developed segmentation algorithms and analyze their strengths and weaknesses.

The primary goal of segmentation is to detect and identify which parts of an image should be grouped together according to some similarity criterion. Segmentation is a process

of partitioning the image into disjoint regions such that each region is uniform and the union of any two adjacent regions is nonuniform.

Hundreds of image segmentation algorithms have been proposed in the literature. In spite of this, it is still a subject of on-going investigation. Out of all previous studies, there is no single method which can perform well for all image processing applications. Issues related to segmentation involve selection of similarity criteria, the approach to selecting the image partition, and the effect relative to the overall image processing system.

Since we deal with CT/MRI image segmentation, we will restrict our review primarily to segmentation methods that have been developed for gray-level images. Fu and Mui [60] categorized segmentation methods into three classes: (1) threshold-based methods, (2) edge-based methods, and (3) region-based methods. Haralick and Saphiro [81] divide image segmentation techniques as follows: (1) measurement-space guided spatial clustering, (2) single-linkage region growing schemes, (3) hybrid-linkage region growing schemes, (4) centroid-linkage region growing schemes, (5) spatial clustering schemes, and (6) split-and-merge schemes. According to [81], the difference between clustering and image segmentation is that in clustering, the grouping is done in measurement space, while segmentation is done in the spatial domain of the image. The rest of this section follows Fu and Mui [60] along with additional categories proposed by Pal and Pal [136]. Those categories are: (1) threshold-based methods, (2) edge-based methods, (3) region-based methods, (4) statistical and Bayesian methods, and (5) neural networks and fuzzy logic based methods.

Threshold-based methods

The threshold-based methods are the most intuitive, old, simple and popular for image segmentation. Threshold-based image segmentation methods can be done using global infor-

mation such as image histograms, or local information such as a co-occurrence matrix. If a threshold value is used for the entire image, then such method is called global thresholding. If a separate threshold value is determined for each subregion, then it is called local thresholding [156, 23, 187, 97] or adaptive thresholding [38, 132, 197]. Sahoo et al. [156] surveyed thresholding methods and attempted to evaluate the performance of some of them.

Sometimes thresholding methods are categorized as bi-level and multi-thresholding. Bi-level thresholding separates an image into two types of regions: object and background. Because of this, bi-level thresholding methods are a form of two-class classification.

When a gray level image contains distinct regions, its histogram usually shows different peaks so that each indicates one region and adjacent peaks are likely separated by a deep valley. The bottom of each valley can be chosen as a threshold. If this is the case, a thresholding algorithm can perform detection of such valleys and therefore determine threshold values that separate foreground regions from background. However, clearly separated peaks in a histogram are not always present in real applications and selection of a threshold is not a trivial task. Therefore many methods [155, 75, 67, 141, 156, 23, 187, 97, 178, 137, 135, 4, 50, 195, 117, 148, 153, 194, 188, 189, 48, 32, 2, 109] have been proposed to determine optimal thresholds in such cases. For example, Otsu [135] uses class a separability measure that is maximized. In his method, the maximized ratio of the between-class variance to the local variance determines the threshold. Cheriet et al. [37] proposed a recursive extension of Otsu's thresholding algorithm. The algorithm was developed for segmentation of bank checks. The algorithm recursively separates the largest peak from the rest of the image until all peaks are processed. As with the Otsu's method, this performs well for images with bimodal histograms.

The two dimensional histogram of an image is considered by Li et al. [104] for finding a threshold for segmentation. In a two dimensional histogram, both individual pixel values and

local gray level averages are used. The authors showed that the Fisher linear discriminant can be used for such histogram data to obtain an optimal projection where the data clusters are better defined and therefore easier to separate by choosing proper thresholds. But this approach demands more computational cost than that of the one dimensional counterpart.

Edge-based methods

Edge-based methods assume that abrupt changes in gray level intensity values are present at region boundaries. A variety of methods have been proposed [155, 75, 67, 201, 95, 92, 111, 112, 84, 140, 29, 47, 66] to find edges in an image. Edge detectors provide local information which does necessarily form a closed path [69].

Ahuja et al.[4] developed a method that uses pixel neighborhoods for image segmentation. Their method uses a supervised classification technique (Fisher's linear classifier) to detect edges. They concluded that a pixel value and its neighborhood provide rich information to use in segmentation.

A set of algorithms was developed by Prager [146] to segment natural scenes by analyzing edges. The main focus in these algorithms is to locate edges of an object as accurately as possible. The output of these algorithms provide a set of line segments associated with such attributes as length and confidence.

Edge-based methods have not been very successful because small gaps in edge-based curves cause two dissimilar regions to be merged. Since finding edges requires differentiation which is sensitive to noise, edge-based segmentation methods have provided limited success for image segmentation problems.

Region-based methods

Region-based segmentation methods use region growing algorithms to perform image partitioning. Region growing algorithms use one or more "seed" pixels and then grow regions by incorporating surrounding pixels based upon certain similarity criteria. If the adjacent pixels are similar to a particular seed, they are merged with the region that belongs to the seed point. The process continues until every pixel in the image is assigned to one region. There are many algorithms [35, 3, 125, 10, 17, 62, 88, 110, 167] proposed in the literature.

One region growing segmentation framework is proposed by Chang and Li [35]. The seed points are obtained by regional feature analysis. The algorithm is known as Fast Adaptive Segmentation. The algorithm divides an image into small regions which are assumed to be homogeneous. Then primitive regions are tested against similarity criteria. If the similarity test is affirmative, then these regions are merged to form larger regions. The process continues until no more merges are possible.

One can see that selection of seed points is very important in region-based segmentation methods. The selection of seed points can be done either manually or automatically. In the case of automatic selection, important characteristics of an image are used to determine seed points. For example, peaks in an image histogram can be used [85] for seed point selection. Adams and Bischof [3] studied how effective seeded region growing algorithms are when selection is manual. They concluded that such algorithms are fairly robust, fast, and no parameter adjustment is required. However, the order of processing heavily affects seeded region growing algorithm performance. Mehnert and Jackway [125] addressed this shortcoming and tried to make seeded region growing algorithm more predictable and more parallel.

Several studies [139, 53, 22, 196, 62, 45, 71, 39, 162] were aimed at combining the strength

of edge-based and region-based segmentation methods to provide more robust segmentation method. One survey [59] investigated available methods that combine edge and region information and compared their performance.

Statistical and Bayesian methods

Statistical and Bayesian methods use "feature space" to segment images. These methods transform a image's pixel data into feature space, and then partition the space based on their statistical properties. Many studies [93, 78, 6, 72, 79, 154, 41, 42] are available in the literature. Statistical methods in image segmentation problem have attracted many researchers since it allows for formal mathematical analysis of the image segmentation problem as opposed to using heuristic or ad hoc methods in image segmentation.

Spatial interaction models like Markov Random Field or Gibbs Random Field have been used to segment images. Geman and Geman [63] have proposed a method for image restoration based on stochastic relaxation and relaxation labeling. Derin et al. [49] improved the one-dimensional Bayes smoothing algorithm [6] for application to two dimensional image problems.

Comer and Delp [41] proposed a method for the segmentation of textured images using a multiresolution Bayesian approach. Their method employs a Multiresolution Gaussian Auto-regressive (MGAR) model for the pyramid model representation of an image. First, Gaussian pyramid decomposition is applied to the image and decomposed images are stored in the nodes of a binary tree. The Markov Random Field model is then used to classify pyramid image data. Finally, an optimization criterion is used for image segmentation. This criterion minimizes the expected number of misclassified nodes in the multiresolution lattice. Expectation Maximization (EM) is used for parameter estimation.

A major drawback of the statistical and Bayesian based image segmentation is that they have relatively high computational cost. Another problem with these methods is the requirement for a stochastic image model which can be difficult to obtain in advance.

Neural networks and fuzzy logic methods

Any vision system should be reasonably fast, and reasonably robust to random noise and component failure. Segmentation methods based on neural networks and fuzzy set theory aims to achieve these goals.

Several authors [28, 138, 20, 8, 64, 43] have attempted to use neural networks for image segmentation. One study [28] used a 3 layer feed-forward neural network. The number of neurons in the input layer is determined by the number of features computed for each pixel, and the number of neurons in the output layer determines the number of region types in the segmentation problem. Their algorithm performs initial segmentation using self organizing maps (SOM) based on color and texture information of regions. The SOM network contains 64×64 nodes for segmentation. Average color, position, size, rotation, texture (Gabor filters) and shape are collected to construct feature vectors. Each feature vector is given to a multilayer perceptron with 28 input nodes and 11 output nodes. This algorithm was implemented for the segmentation of outdoor images.

Papamarkos et al. [138] developed another segmentation algorithm that is based on self organizing maps. Their approach is similar to multithresholding where the output of the network determines the number of homogeneous regions. Multilayer neural networks have been used to segment noisy images. The weight updating rules are determined so that they reduce fuzziness in the system. This study aimed to combine the advantages of fuzzy sets (reasoning from imprecise/incomplete knowledge) and the robustness of neural networks.

1.4 Our approach

As we described in the beginning of the chapter, the ultimate goal of segmentation is to partition an image into meaningful regions. Segmentation algorithms are used tasks such as measurements, visualization, registration, content based search, and reconstruction.

A typical segmentation problem could be divided into three different steps: (1) preprocessing which applies low level image filtering and enhancement operators to condition an image, (2) the primary segmentation algorithm, (3) and a postprocessing step which performs final refinement on the segmented image based on predefined requirements. The intervention of a human operator is often necessary to check accuracy of the result produced automatically, and to make manual correction if it is needed.

This dissertation describes the development of a postprocessing module that is built previously developed ANN based segmentation algorithm. The postprocessing module uses higher-level domain-dependent knowledge to improve initial image segmentation results. There are several motivations for such an approach. First, the initial ANN-based classifier depends primarily on information from very small image neighborhoods, and identifies defects on a pixel-by-pixel basis. It therefore ignores such domain specific information such as defect shape, size, and proximity to other defects within the log. Second, the domain expert driven postprocessing approach that we describe here can employ different form of domain expertise for different defects. One possible solution is shown in Figure 1.6. In this method, the user directly intervenes into the segmentation process to assist in selecting rules.

In general, the problem is difficult because of the inherently variability of wood. Although much of the work reported above has yielded results that are good in a quantitative sense, subjective evaluation suggests that the results could be improved in a qualitative sense by refining the resulting shapes and extent of detected defect regions in the images. For

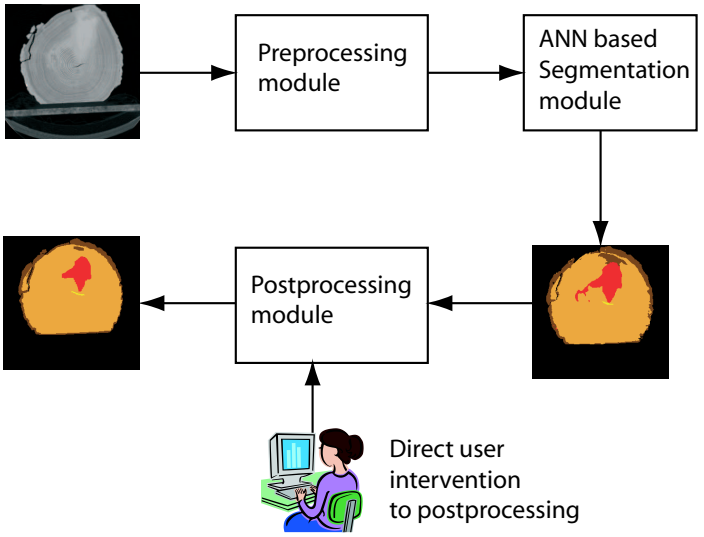


Figure 1.6: Putting the human user into the loop. Adding an interactive postprocessing module makes the overall image segmentation system adaptable.

example, small spurious defect regions may have only a small effect on pixel-wise statistical classification accuracy, but are objectionable to the human observer.

1.5 Needs

A *knowledge-based system* is one that encodes domain specific knowledge in a form so that the system can use for automated reasoning.

Designing a knowledge-based postprocessing system requires the formulation of domain knowledge. A serious problem faced by knowledge-based system designers is the knowledge acquisition bottleneck. Knowledge acquisition is very challenging and an active research area in the artificial intelligence community. Commonly, a knowledge engineer needs to have a domain expert to formulate acquired knowledge for use in an expert system. That process is rather tedious and error-prone. Despite its difficulty, this process should be done carefully, and requires lengthy interviews and discussion with a human expert. The domain

expert's verbal description can be inaccurate or incomplete, and the knowledge engineer may not correctly interpret the expert's intent. In many cases, the domain experts prefer to do actions instead of explaining their expertise.

These problems motivate us to find another solution to make the knowledge acquisition process less challenging. Instead of trying to acquire expertise from a domain expert verbally, we can ask him/her to show expertise through actions that can be observed by the system. If the system can learn from those actions, this approach is called *learning by demonstration* [46, 108].

There are several difficulties in designing a knowledge-based postprocessing module for image analysis, however. Although many postprocessing steps can be easily implemented, different situations (possibly depending on the species of wood, on particular defect types, on the intended use of a log, and on personal preferences of the user) may require different types and degrees of region refinement. For these reasons, we have developed the *IntelliPost* system that can learn postprocessing rules automatically. The system observes the steps taken as a human user interactively edits a processed image, and then infers rules from those actions. During the system's *learn mode*, the user views labeled images and makes refinements through the use of a keyboard and mouse. As the user manipulates the images, the system stores information related to those manual operations, and develops internal rules that can be used later for automatic postprocessing of other images. After one or more training sessions, the user places the system into its *run mode*. The system then accepts new images, and uses its rule set to apply postprocessing operations automatically in a manner that is modeled after those learned from the human user. At any time, the user can return to learn mode to introduce new training information, and this will be used by the system to update its internal rule set.

The system does not simply memorize a particular sequence of postprocessing steps

during a training session, but instead generalizes from the image data and from the actions of the human user so that new CT images can be refined appropriately. Because it learns from a human "teacher," this approach represents a form of supervised machine learning. However, the level of supervision is relatively mild by traditional machine-learning standards, because the teacher does not need to be knowledgeable concerning internal feature spaces or representations for rule selection. Because of its ability to accept new training inputs over time, the system is said to perform "incremental" (or "dynamic" or "on-line") learning. This contrasts with many machine-learning systems, which require all training data to be made available at the beginning. Such systems perform "batch" (or "static") learning.

1.6 Contributions of this study

1. *One of the first image segmentation systems based on learning by demonstration.*
   Many researchers have described applications which employ learning by demonstration. Some of these include robot programming, text editing, and graphical figure editing. The degree of interactivity, providing incremental learning capability through user feedback, makes the system one of the first image segmentation systems to learning by demonstration. To our knowledge this will be the first to use learning by demonstration in the field of image analysis.

2. *Application of mathematical morphology to CT image postprocessing.*
   Mathematical morphology provides useful tools to analyze regions in digital images based on their shapes. Morphological image processing transforms an image based on geometric characteristics of regions in an image, and can be used to eliminate irrelevant features from an image. In our application, the shape of a region conveys very crucial information. Mathematical morphology provides very effective methods for extracting such information, and then transform regions based on this. In other words,

mathematical morphology provides both probing capability to analyze the shape of a region, and operators to process a region in an image. These powerful features of mathematical morphology persuaded us to develop a postprocessing operation library that is tailored specifically for postprocessing operations that can be used in both run mode and learn mode by the system.

3. *Developed a procedure to measure overall image segmentation performance.*
   There is no universally accepted performance metric that measures how good a proposed image segmentation method performs. We have developed and implemented a procedure that provides "ground truth" from a human expert for evaluating our system's results. In this procedure, a human expert uses an existing graphical image editing program to create reference segmented images. The image editing program lets the user (a human expert) manually segment defect regions in a typical CT slice, for example. Manually segmented regions constitute ground truth for measuring the segmentation performance of the system. In biomedical research, ground truth is widely available to evaluate implemented image segmentation algorithms. Due to the absence of available "ground truth" database for hardwood logs scans, we have developed this procedure to generate such a database.

1.7 Possible application areas of this study

Although the motivation has been to implement a postprocessing system for detecting and identifying internal wood defects, this system can be used for other segmentation applications. Since the system provides interactive editing capability of segmented regions in a typical CT slice, this study should be useful for medical applications, for example. In biomedical applications, the performance and accuracy of a segmentation algorithm is critical. Another area that can benefit from this work is remote sensing. In remote sensing

applications, images are often segmented into different regions (crops, forest, etc.) by some automated segmentation method. The result can often benefit from refinement through a system such as the one described here.

Although the acquisition method of MRI (Magnetic Resonance Image) image is different from that of CT, both imaging methods show similar information which is cross section of an object. The implemented image segmentation algorithm would be suitable for segmentation problems of MRI.

1.8 Outline of the thesis

This thesis organized as follows. Chapter 2 describes previous work on learning by demonstration. Chapter 3 presents an overview of morphological image processing, and then relevant machine learning theory. Chapter 4 introduces the *IntelliPost* (intelligent postprocessing) system, which is based on learning by demonstration. Chapter 5 presents results that have been obtained using the system. Chapter **??** summarizes and concludes the study.

Chapter 2

Historical Review of Learning by Demonstration

2.1 Introduction

Many forms of machine learning can be viewed as teacher-student interaction in which a teacher provides training examples and a computer or "student" generalizes from the training examples. The teacher then tests the student's capability by providing test examples and observing the responses. During the learning stage, there is often no immediate response from the student on how the knowledge is perceived. Therefore, this strategy may waste valuable resources, including time and effort.

Learning by demonstration takes the interaction model one step further: the student asks the teacher to solve one example case and the teacher then solves the presented example. By observing the way of solution for the presented example, the student generalizes from the way the example is solved. In this way, the student can direct the teacher's attention

to areas that are unclear. Therefore, the student can take the initiative, rather than being passive in the learning stage.

This chapter begins by discussing the terminology used in a related field knows as *programming by demonstration*. Then we will briefly introduce relevant experimental programming by demonstration systems that have been developed. Cypher and Lieberman [46, 108] give a broad overview of the origin and history of programming (and learning) by demonstration. Those experimental systems can be grouped in three different categories: text editing systems, programming aid systems, and finally graphical editing systems.

2.2 Definitions and their usage

In this section we will present a range of terms that are widely used in the demonstrational learning or programming literature. In those studies researchers do not agree on a unique definition of the concept of demonstrational learning or programming. Among the range of terms, the most used are learning by demonstration, programming by demonstration, programming by example, demonstrational interfaces, and adaptive (or intelligent) interfaces.

Before we begin to present the definition of these terms, there is a need to understand the difference between "learning" and "programming". In the glossary of *Watch What I do*, one of the first books dedicated to demonstrational learning and programming [46], Myers and Maulsby describe "learning" in the following way:

> Simon defined learning as changes in a system that result in improved performance over time on tasks similar to those done previously. A dictionary definition is that it is acquiring knowledge or skill through study, experience or teaching. Whether a computer system "learns" or merely "induces generalizations" is of-

> ten a subject of debate, because typical generalization procedures and concept representations are simplistic and brittle. Simon's definition seems to suggest continuous, cumulative improvement as the acid test, and the dictionary hints at the breadth and depth of background knowledge applied and the ability to learn from various kinds of input.

According to their definition, "learning" is a progression toward a better performance in the system response. The "learning" system should have inference and generalization capabilities that give system ability to give reasonable answers to unseen input. The definition of "program" is in the same glossary is as follows:

> A program is usually defined as "a sequence of instructions that are executed by a computer," so any system that executes the user's actions can be considered programmable. However, for the purposes of this book, it is useful to characterize how programmable the systems are. Therefore, Myers defines the term programmable to be systems that include the ability to handle variables, conditionals and iteration. Note that it is not sufficient for the interface to be used for entering or defining programs, since this would include all text editors. The interface itself must be programmable. This distinction is not particularly easy to make, and there can be arguments about whether a particular system should be called programmable or not.

According to this definition, we can understand that programming is the process of specifying a sequence of operations. If we compare the process of "learning" and "programming", "learning" involves a much deeper understanding of a user's operations than "programming" in a way that it requires inferencing and generalization ability to capture a user's intention.

On the other hand, the outcome of programming is a sequence of commands that deals with manipulating variables, conditionals, and iteration.

Programming by demonstration provides a means for a user to "program" a computer by demonstrating to it examples of what a user wants to accomplish. The outcome is a program that is generated by the computer. During the demonstration stage, the user must perform operations in the correct order and at correct times, so the system must infer the boundaries and preconditions of each operation. According to Myers and Maulsby [46], *programming by demonstration,* a synonym of *programming by example* (PBE), is defined as follows:

> Programming by examples – when programs are created through the use of examples. The examples serve as placeholders for abstractions. Myers would like to restrict this term to only systems with inferencing and that allow the end user to create real programs, but this would eliminate Pygmalion and SmallStar, which are often classified as PBE (including by their authors), so others prefer the more general definition.

In this definition, the role of abstraction is not clear. It also implies that the result of programming by demonstration (or programming by example) should be a program. Macro recorders represent a primitive kind of programming by demonstration system since they learn programs from demonstrations. Myers [130] proposes a more restrictive definition: he characterizes systems that manipulate variables, iteration, and conditionals as "programmable" and claims that a system must be programmable and use inferencing to be named as programming by example systems. Based on this definition, macro recorders are not programmable and apply no inferencing, therefore they are not programming by example systems. Myers [130] categorizes macro recorder and other systems that use demonstrations are demonstrational interfaces defined as follows:

> Demonstrational interfaces allow the user to perform actions on concrete example objects (often, by direct manipulation), while constructing an abstract program. As defined by Myers, this includes Programming by Example and Programming with Examples, as well as interfaces that do not support programming [46].

Intelligent and adaptive systems are those that use techniques from artificial intelligence, including adaptive interfaces, human-computer dialog modelling, natural language processing, and explanation based systems [185]. The main purpose of intelligent user interface research is to learn about a user's preference and behaviour and adapt them accordingly [44, 185].

Learning by demonstration, or learning by observation, can be viewed as programming by demonstration, except that the outcome of the process is not necessarily a program. Instead, the system generalizes user actions and operations and generates implicit or explicit rules corresponding to operations. Based on the above definitions, learning by demonstration systems can be viewed as demonstrational interfaces in which systems may learn from demonstrations but create no programs.

IntelliPost is designed in a way so that it provides a user interface to a human expert so that it can learn how postprocessing operations should be applied. It lets the user apply postprocessing operations for a presented image during its learn mode, and then it generalizes the actions that are taken in learn mode. In the run mode, it uses generalized implicit postprocessing rules to postprocess a given image. It neither creates a program nor provides the generalized rules to the user. But it is implemented in a way that demonstrational actions are required to teach the system. From that perspective, it is a demonstrational interface that also learns.

2.3 Previous Experimental Systems

2.3.1 Pygmalion

Smith developed the very first system of programming by example, called Pygmalion [168, 169]. Smith created the system as an alternative to non-interactive, abstract programming languages. A screenshot of the system is shown in Figure 2.1. The system was designed to help a regular user write computer programs.

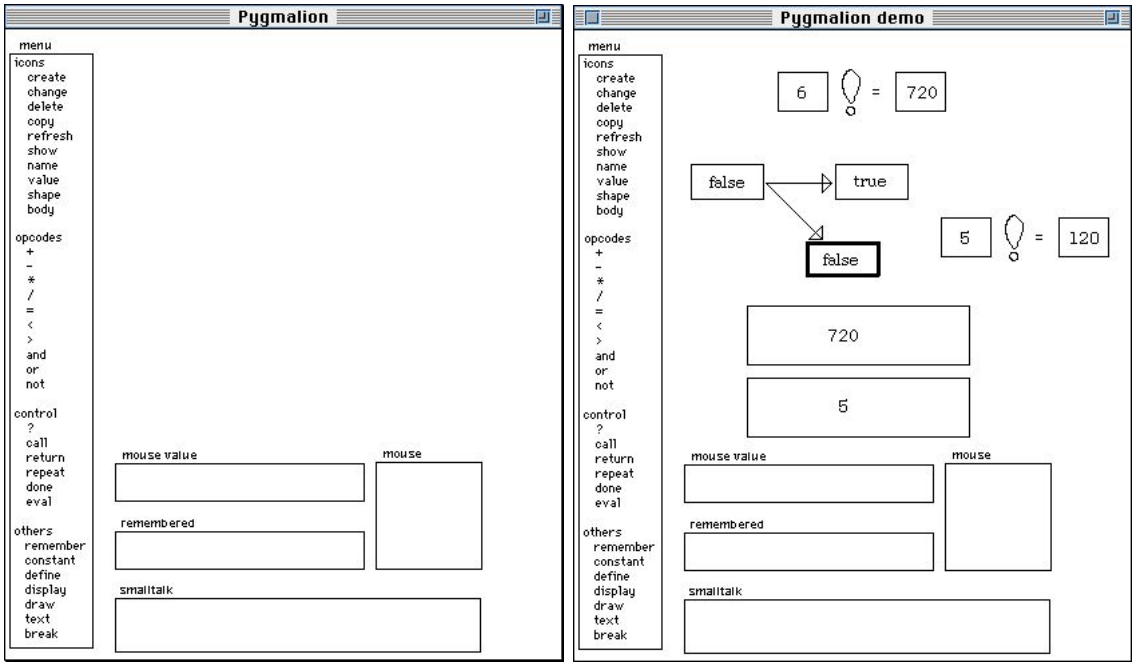


Figure 2.1: Screenshot of Pygmalion. Implementation of factorial procedure (reprinted from [169]).

Users can use the system to construct programs by drawing pictorial, analogical representations called icons. Icons provide both a medium and a programming language for experimenting with ideas. The system uses a "storyboard" technique to represent each function's computation. The first frame of the storyboard represents arguments of the function being defined, and the last frame of the storyboard returns the function's results. In between,

the user demonstrates steps of the function by creating icons on the screen, and pulling old values from previous frame to a new icon. The programming is completed once all sketches are completed. The primary goal of the system was to reduce the distance between the user's mental models and an abstract data model in a computer. The system has proved that it is possible to implement a system that lets an average person write computer programs through the programming by demonstration paradigm. It was developed as an experimental system that provides an alternative approach to classical programing methods. It provides no inferencing and generalization capabilities.

2.3.2 U Editor

Nix developed a text editor based on "editing by example" to automate repetitive textual transformations [133]. The user provides examples of text sequences to be found, and examples of how they should be transformed, and generates procedures to find similar text and make similar changes. Instead of observing sequences of edit commands, it looks at *input-output pairs.* This approach provide two advantages: (1) the user is not required to repeat the same ordering of commands, (2) since the approach does not depend on a particular command set, it provides extensibility and portability to the editing system. The disadvantage is that focusing on input and output pairs ignores a potentially rich source of information in the execution traces. Analyzing these input-output pairs, the system builds *gap programs* in the form of $G \Rightarrow R$ where G is a *gap pattern*, and R is a *gap replacement.* For example, the form of a gap program is $(-1-)q-2---3- \Rightarrow -1---2--3-$ (q indicates space), where gaps are represented by $-n-$ (n is a argument number for a gap program). This gap program finds and transforms all telephone numbers from (540) 231-7272 to 540-231-7272.

2.3.3 Tinker

The Pygmalion system inspired Lieberman to create a system known as Tinker [107]. He took the "example" idea as a method of effective teaching for both the student and the teacher. It should be easier to construct a program by presenting an example to a computer instead of giving abstract program text. Tinker was developed as a programming aid for Lisp developers. It is best suited for developing a graphical application since it shows the end effect of the graphics commands interactively. The system workspace is divided into five windows (shown in Figure 2.2): (1) a window for selection of an operation, (2) a window for defining variables, (3) a window showing the Lisp program that is being developed, (4) a window showing graphical representation of the Lisp program, and (5) a window for the user input. When you define a function in the system, the argument of the function is selected from a object list which has appropriate data description. The system learns by mimicking the role of a student. But the student does not have a capacity of a human student to automatically decide what features of one example may be relevant for the future examples. It is the user's responsibility to indicate which features of the examples are important, and which can be ignored in general. After providing data along with indications of importance, Tinker builds a functional piece of code by applying functions, and building up larger expressions out of these pieces. The pieces are selected and programs are constructed using display menus of commands and program fragments. Tinker keeps track of how the pieces are being fit together, and builds a conventional Lisp function incrementally. The system does not provide any inferencing mechanism; it asks the user to resolve any ambiguities.

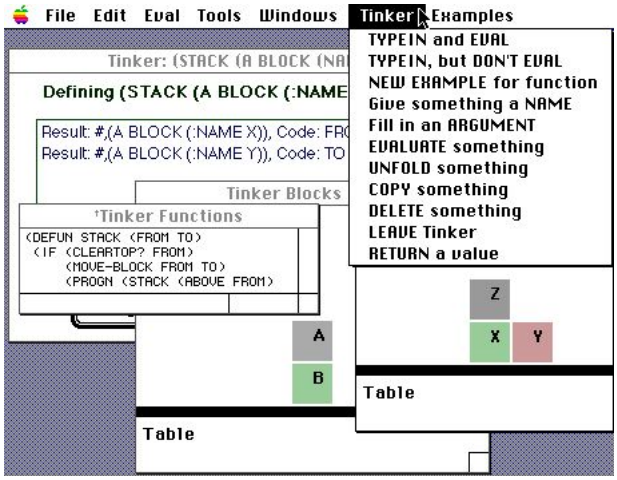


Figure 2.2: Tinker screenshot (reprinted from [107])

2.3.4 Peridot

Peridot was developed as a tool for the implementation of user interfaces [131]. Peridot is an acronym that stands for Programming by Example for Real-time Interface Design Obviating Typing and has been implemented in Interlisp-D. The system lets the user construct a user interface by virtual interaction of devices. Such devices contain typical user interface components such as menus, scroll bars, and other widgets. The system lets the user draw pictures of what the interface should look like and then the user uses the mouse and other input devices to demonstrate how those user interface components should interact and operate. Peridot infers object-to-object relations and lists its inference rules in another window to get the user's confirmation. As result of the confirmation, Peridot produces a Lisp code that implements a user interface that is constructed by the user. It is a rule-based system that contains 60 object-object relationship rules classified by the types of objects that they act upon. This classification mechanism helps speed up the search and provides an extra set of implications for which rule to fire. The rules are ordered so that the most restrictive rules are tested first. Inferences are used in three ways: adding graphical constraints, detecting iterations, and determining how mouse should affect the graphics. Inferencing is made based

on the resulting graphical objects, not on a trace of actions. Peridot applies rule-based inferencing in four ways: inferring the graphics constraints that relate one object to another, inferring when control structures are constructed properly, inferring how control structures are constructed, and inferring the role of the mouse in the user interface.

2.3.5 Metamouse and Turvy

Maulsby's Metamouse [122, 120, 121] was developed for minimally trained regular users who are not programmers to create programs by giving demonstration in a direct manipulation interface. Programs that are generated by Metamouse can be used in other graphical applications (i.e., Apple MacDraw). The program is created by teaching an *instructible agent* how to modify the document. The agent is represented by a turtle named Basil. At the beginning of Metamouse session, the user is informed of the agent's limitations and biases for generalization (see Figure 2.3), and then the user starts giving instructions to draw and modify graphical line and boxes (see Figure 2.3). Meanwhile if Basil recognizes any iteration being performed, it offers help to complete the iteration. This method not only immediately relieves some of the user's burden, but also it guarantees that each iteration of a loop is demonstrated consistently, making it possible to generalize iterations consistently. The agent asks the user for confirmations for predicted operations and explanation of unclear actions when the user performs them. The user incrementally gets the picture of the agent's limitations and its biases on generalization.

The complete system manipulates lines and boxes. It provides a teaching mode to the user. In that mode, a small turtle image jumps to the cursor location between user commands. The user instructs the system to pay attention to touch relations between objects or a specific object in its focus. As objects are modified, the set of touch relationships are high-

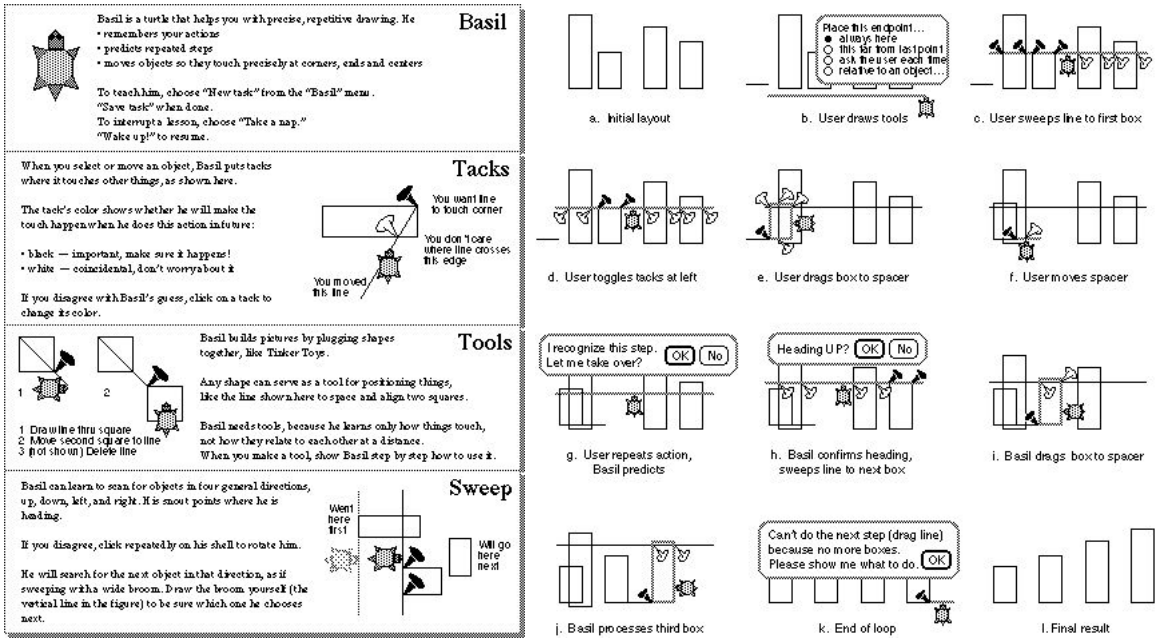


Figure 2.3: Metamouse user manual and teaching the system sort boxes by height (reprinted from [121])

lighted to show whether they are considered important or insignificant. Metamouse infers a procedural construction that contains variables, iteration, and conditionals from execution traces.

The Metamouse system employs mechanisms for both similarity and explanation based learning [191]. During the learning phase, important characteristics of touch relations are recorded. Positive examples are user demonstrations and negative examples are predictions that the user rejects. The control of the operations is represented by production rules. The left hand side of the rule represents a sequence of actions and right hand side of the rule show the next action that will be performed. Negative examples make the system form a new production rule and positive examples reinforce generalized rules by adding a new case that may cover specialization. The rules are used to check each object in a action to see if it has occurred before. If it has occurred, the earlier record will be used, and then the system analyzes whether touch relations are significant. It compares predicted actions to previous

action to check any repetition, and then it performs actions. The Metamouse has not only proved that an instructible agent can be used in such systems successfully, but also has showed that the implementation of such a system could become prohibitively complicated (11,000 lines C++ code).

Maulsby continued his research on problems that Metamouse presented, and has implemented a series of systems. One of them was Turvy [193]. Turvy creates an environment to play *Wizard of Oz* in which a human plays the role of an agent and the computer plays a human. It has a incremental learning mechanism that learns from examples. The user provides an initial example and turvy predicts a subsequent of actions. If the user correct actions that are taken, turvy updates its inference rules based on feedback. The user can emphasize specific points that need special attention, Turvy biases its generalization mechanism to take these specific points into accounts.

2.3.6 TELS

TELS is an another text editing system that uses a demonstrational interface method. Witten and Mo viewed the problem of textual transformation from a different perspective when they developed TELS [192]. TELS builds the procedures through *interaction traces* rather than *input-output pairs* [192]. Witten and Mo claim that synthesizing procedures from traces provides rich information for capturing the exceptions from less regularly structured text. The system provides simple textual transformations that contain a limited number of operations: insert, delete, locate, and select. The higher level commands can be built upon these provided commands. TELS builds procedures from a demonstration session and then uses them in subsequent cases for similar textual transformations. The user gives feedback to the system by indicating steps that should or should not have been performed in the

new example. TELS records operations and their relevant data in every operation's trace, and then the recorded traces are generalized when collapsing multiple traces to the same program step to represent the aim of the operation. If the user provides very few examples to the system, TELS can make overgeneralization and overspecialization mistakes which can be corrected by providing TELS more examples.

2.3.7 Triggers

Triggers [144] developed to demonstrate pixel-data access that treats the screen image as the source for generating descriptions for generalization. Potter claims that using a pixel-based data access method provides rich information that would be otherwise hidden inside an application program and unavailable to the other programming by demonstration systems. The system applies exact pattern matching on screen pixels to infer information that is otherwise unavailable to an external system. The system builds a "trigger" that contains a set of conditions and a set of actions pair as shown in Figure 2.4. Both conditions and actions are recorded in a way that is similar to macro recording. When Triggers is run, the system tests conditions that are part of a trigger. If the condition fails, the action step is not fired. The order of testing condition/action pairs is tested sequentially, and iteratively. For example, triggers are defined for such tasks as surrounding a text label with a rounded rectangle in a graphical illustration program, shortening lines so that the end point intersects an arbitrary shape, and making the text a bold typeface.

The user creates a trigger by showing steps through a sequence of actions in an application, adding special tags that instruct Triggers when the trigger is performed. Once triggers are created, the user can invoke them, iteratively and exhaustively, to execute their operations. The system records three important features: markers, rectangles, and flags. The

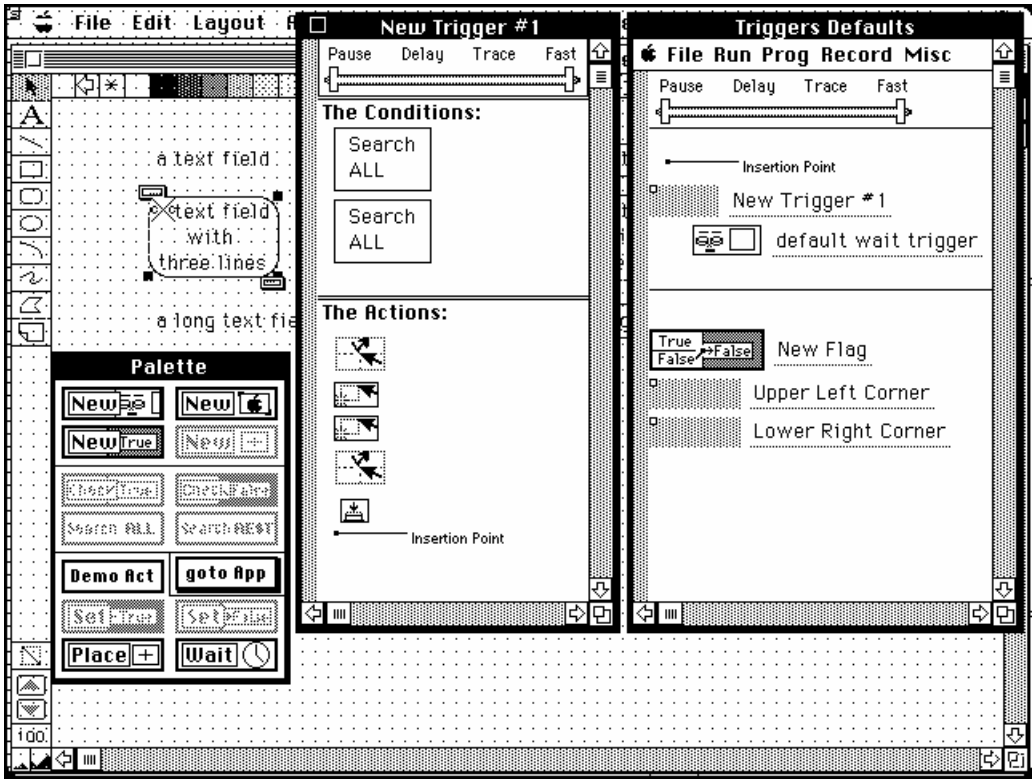


Figure 2.4: Trigger's workspace (reprinted from [145])

markers are used to store the location of a point on the computer screen. A rectangle represents a rectangular area of the computer screen. A flag is used to indicate that a certain action has been already taken to avoid infinitely firing a trigger. The generalization strategy used by Triggers is to record the locations of exact patterns on the application screen. For example, suppose a user creates a macro that modifies a search engine's number of result widgets to 100 instead of 25 as seen in Figure 2.5. The developed macro records the mouse coordinates of the widget and corresponding action which changes the value from 25 to 100. Pixel pattern matching on the screen can provide information to generalize this macro.

Pixel-level data access methods provide a benefit of adapting Trigger to already existing applications just by using their screen images. The system communicates with an external application through interprocess communication protocol. Although the system affects data

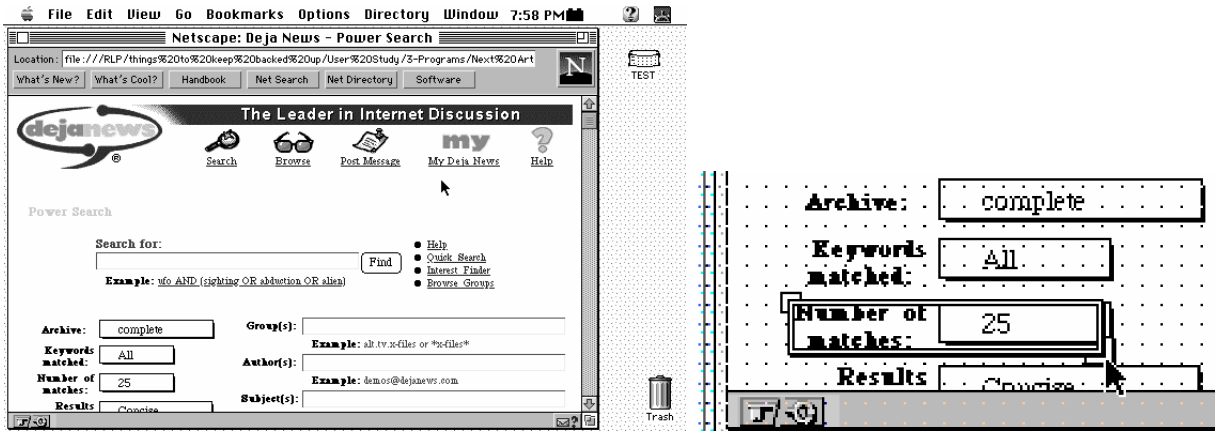


Figure 2.5: Modifying web page (reprinted from [145])

such as text, and graphical objects in an application, it only deals with low-level pixel patterns and screen coordinates. That is the utilization of low-level data to extract high-level meaning. The advantage of this strategy is that low-level data and operations of an applications can map many high-level operations. The disadvantage is that high-level internal information processing and communication are difficult.

2.3.8 Chimera

Kurlander's Chimera [99] is a graphical editing system that provides five tools for automating repetitive tasks in graphical editing applications: graphical search and replace (similar to Unix grep utility), constraint-based search and replace (as shown in Figure 2.6), constraint from multiple snapshots, graphical histories, and graphical macros by example.

Graphical search and replace functionality closely resembles common text editor's search and replace functionality. The user copies a graphical object for searching, and draws another graphical object for replacing ones that are found in an graphical illustration. The user can also indicate which properties are significant for search and replace operations. This tool does not perform any inferencing. The user edits a picture, data descriptions, and

constraints in both search and replace panes, and then the user can specify which geometric features are relevant and their tolerance for search and replace constraints. The search and replace constraints contain geometrical information such as distance, slope, and angle. Constraints from multiple snapshots let the user show constraints by drawing various valid position of objects. At each valid position, the user takes a snapshot, and Chimera defines constraint sets that describe all positions that are valid. Editable graphical histories are similar to Pygmalion's storyboard approach. This provides an optimized series of panes for emphasizing the changes made between panes. Graphical macro by example improves the functionality of editable graphical history by allowing a sequence of command to be selected and encapsulated into a reusable macro. Macro parameter can be provided when macro is executed. Constraints are constructed by analyzing which geometric features of the objects remain the same, if the user moves the parts of the scene later, the corresponding constraints are invoked automatically. Generalization is made by dropping constraints that are not valid for all examples.

2.3.9 Mondrian

Lieberman's Mondrian is an object oriented graphical editing system that learns graphical drawing procedures from demonstration[106]. Mondrian development was inspired by Chimera's editable graphical histories tool. The user can demonstrate a sequence of graphical operations to show how a new procedure should be performed on a concrete example. An agent in the system records steps of a procedure in a symbolic form. The system also keeps track of relationships between graphical objects and interrelations between the interface operations. The agent generalizes a Lisp procedure that can be used later in "similar" examples. Mondrian provides storyboards of icons that show changing states of display during editing. Each iconic image represents a single command. Command icons in a Mondrian

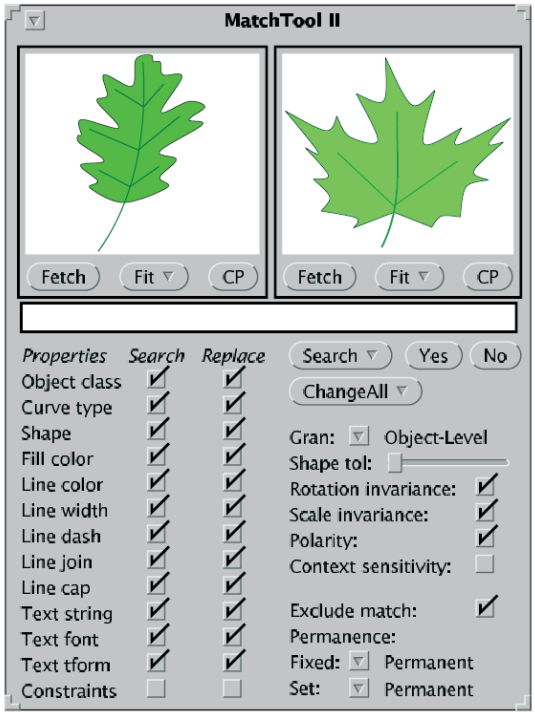


Figure 2.6: Chimera's match tools (reprinted from [99]).

window appear before and after the effect of command as graphical representation to show the effect of executing a command. The basic workspace of the system is shown in Figure 2.7 which shows a drawing of an arch. Even though it is inspired by Chimera, some significant differences exist between Chimera and Metamouse. The order in which generalization advice is given and the representation of the result of a generalization are different. Metamouse gives generalizations advice at the start of the demonstration whereas chimera gives generalizations after the demonstration. Another difference is that Mondrian generates Lisp code as a result of generalization, whereas Chimera does not provide human readable procedural representation of generalization.

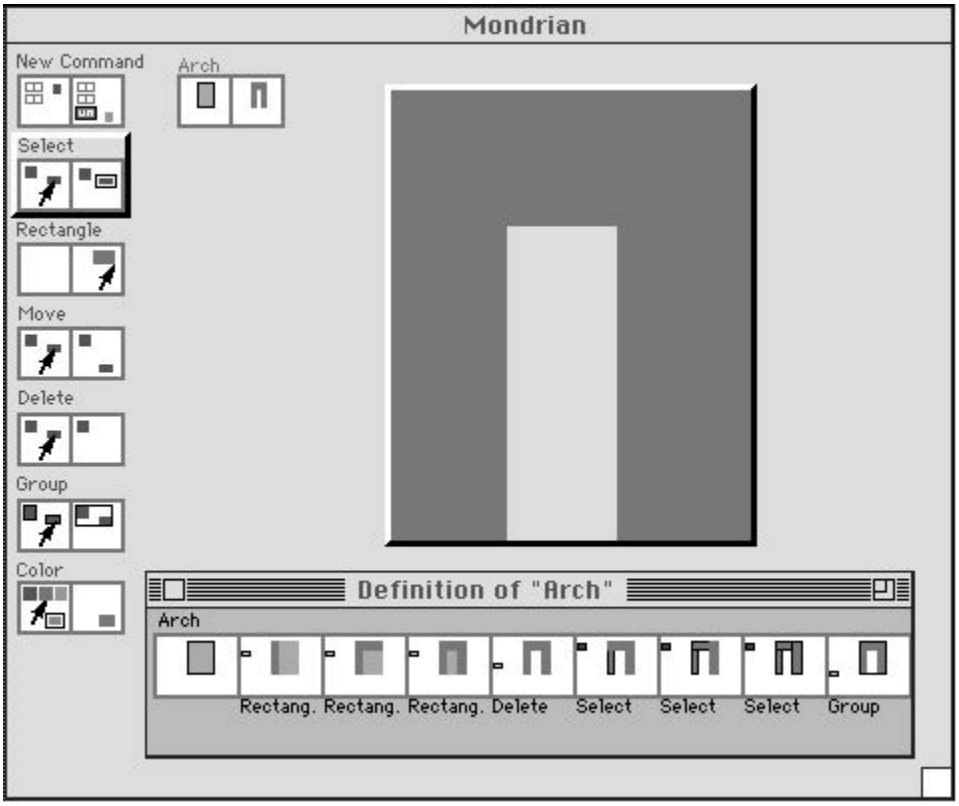


Figure 2.7: Workspace of Mondrian (reprinted from [106]).

2.3.10 SmartEdit

SmartEdit is teachable text editor that was developed by Lau based on the programming by demonstration approach [100]. The system SMARTEdit (Simple Macro Recognition Tool) uses version space algebra [101]. The system treats demonstration phase actions as a partial execution trace of the program that solves the problem at hand. Then it makes a generalization from the execution of traces to the original program. In version space algebra, the system is given a sequence of changes in the application state being observed by system as user performs some tasks on an example, and then the system generalizes given example to a program that is capable of performing the new task by learning the set of functions that transform the initial state to the final state of the given task. The learning mechanism of the

system provides a set of compact representational hypothesis space that is consistent with the observed task. There are two interaction modes available: Recording the user's solution for the next example or running the learned program on an example.

A typical session in SmartEdit is shown in Figure 2.8. In this example, user shows the system how to remove HTML comment lines from the text files. Those comments start with <!– and end with –!>. In learning mode, SmartEdit learns a program that contains a sequence of relatively high level text actions among them: move cursor to a new position, insert a string, delete a string, and manipulate a clipboard.

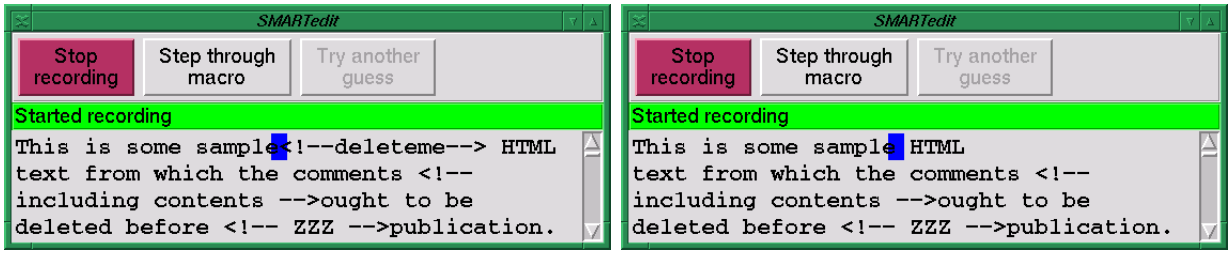


Figure 2.8: SmartEdit screen. User is deleting all HTML comments that begin with <!– and end with –!> (reprinted from [108]).

The user can ask the system to step through a learned macro one action at a time by pressing the "Step through macro" button (see Figure 2.9). The system will guess what likely action to take in this situation along with probability. If the action is not what the user intended, the user can choose "Try another guess" which selects SmartEdit's next most likely action based on its previous observations.

After the user confirms the system's next choice by pressing the "Step through macro" button, the system deletes the extent of the HTML comment (Figure 2.9).

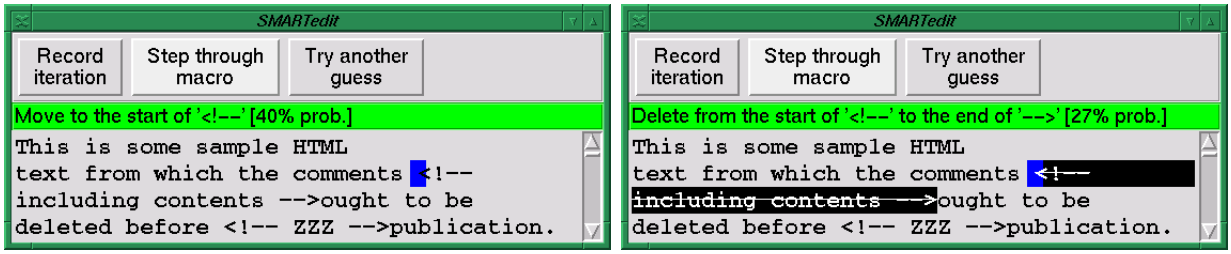


Figure 2.9: SmartEdit performs editing (reprinted from [108]).

2.4 Summary

This chapter has briefly surveyed previously developed experimental systems that are based on demonstrational learning and/or programming. Many programming by demonstration systems have been developed. Although these systems have never had success in commercialization, most have been implemented to explore new learning techniques or interface concepts.

The overview of all surveyed systems is listed in Table 2.1. Systems are categorized as to whether they are text based or graphical based system, perform any inductive learning, and create an human readable program.

Programming by demonstration is a powerful concept for building learning based systems, but it is still the user's responsibility to demonstrate solving a task at hand in a consistent manner. The sequence of demonstration steps is very important for such systems to define clear boundaries for each step of the demonstration. Well defined decision boundaries for an observed task yield better generalization and inferencing capabilities.

The IntelliPost system that is presented in this dissertation is similar to graphical editing systems. But it is significantly different in the way that it analyzes images from the image segmentation point of view and infers what postprocessing operation should be applied for a specific region in an image. It provides a user interface to capture postprocessing rules from

Table 2.1: The summary of surveyed systems.

System Name	Text Based	Graphical	Create Program	Performs Inductive Learning
Pygmalion		√	√	
U Editor	√		√	√
Tinker		√	√	
Peridot		√	√	√
Metamouse		√	√	√
Turvy	√		√	√
TELS	√			√
Triggers		√		
Chimera		√		√
Mondrian		√	√	√
SmartEdit	√			√

the user's demonstration. The system applies generalized postprocessing rules for a new image segmentation problem. If the end result is not satisfactory, the user gives feedback to the system to update its postprocessing rules.

IntelliPost is a demonstrational interface to capture postprocessing rules. But it does not provide any human readable program or rules. It uses these implicit rules for the run mode of the system. Of the systems described above, IntelliPost is most similar Lau's SmartEdit system. Both systems provide both learn mode and run mode operations in the user interface.

Chapter 3

Background

3.1 Introduction

This dissertation claims that the segmentation performance of an ANN based system can be improved by applying learning based postprocessing. IntelliPost is built upon two main components: *decision tree* based learning for its inferencing component and *morphological image processing* for its postprocessing operations library component. As we can see from the system's architecture in Figure 3.1, the postprocessing operation library is used for both the *learn mode* and the *run mode*. In the learn mode, the user uses refinement operations from the operation library to perform postprocessing on an image. In the run mode the system uses the library for performing similar postprocessing operations automatically.

During the learn mode, the system stores information that associates postprocessing operations with regions properties, both geometric and statistical. To prepare for run mode, the system's inference engine reads the stored information and generates postprocessing rules that can be applied automatically. Due to the interactive nature of the postprocessing

system, the system's inference engine should provide its results quickly. Since decision tree induction algorithms have attributes of accuracy, simplicity, generality, and training speed, we have utilized decision tree learning as a major component of its inference engine.

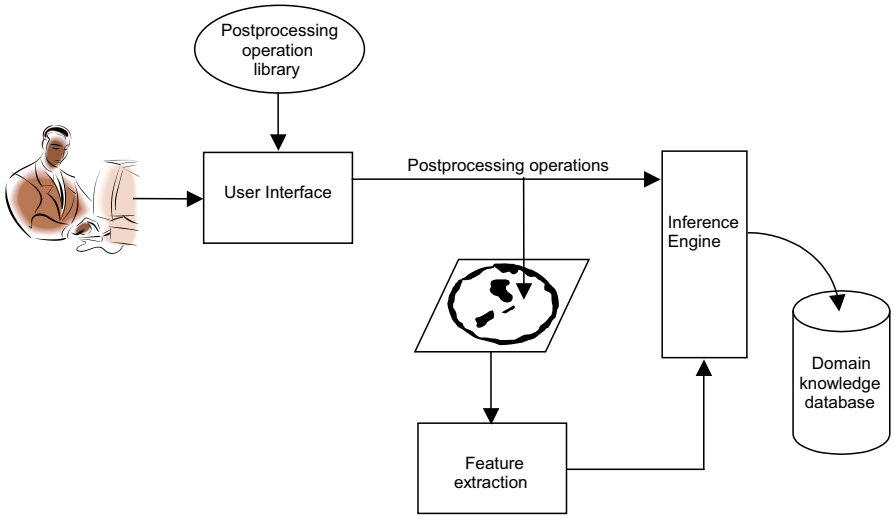


Figure 3.1: Postprocessing system overview

This chapter is structured as follows. The first section presents a brief introduction to the theory of morphological image processing, particularly its basic operators: erosion, dilation, and operators derived from erosion and dilation. The second section briefly presents the theory of decision tree learning methods, and briefly explains different implementations, strengths and weaknesses.

3.2 Morphological Image Analysis

3.2.1 Overview

Mathematical morphology, or simply morphology, is a theory for the analysis of spatial structures[165]. The theory is called morphology just because its goals are to analyze the shape and the form of objects. Mathematical morphology provides powerful image analysis tools[171]. In the area of image processing, it analyzes the shape of a region. A region can be represented as a connected set in a image. The operators in mathematical morphology are defined as set operations and have proved to be powerful for processing sets of binary and gray level images[166]. Mathematical morphology is concerned with image filtering, and with geometric feature extraction by structuring elements. It is a powerful tool for solving problems ranging over the entire image processing area, including pattern recognition, medical imaging, microscopy, inspection, and robot vision [119, 165, 51, 68, 82, 142, 166, 118, 87, 171]. It originated in a seminal work of Minkowski and Hadwidger on geometric measure theory and integral geometry [126, 127, 73] and was introduced to modern world by the work of Matheron and Serra in the Ecole des Mines in Fontainebleau, France [119, 165]. Their work provide not only the formulation of modern concepts of mathematical morphology, but also resulted in the *Texture Analyzer System*, a parallel image processing system based on mathematical morphology [96].

The early use of mathematical morphology was to analyze binary images in the 2D plane, but Stenberg and Serra [165, 175] extended the mathematical morphology theory to gray level images. We should note that we will restrict our discussion of set theory and the theory of mathematical morphology for binary images and binary regions that are defined in discrete space (Z^2).

3.2.2 Image Regions as Set, and Logical Operators

In morphological image processing, the definition of a region is a set (or collection of connected pixels) with either continuous or discrete coordinates. This can be illustrated as in Figure 3.2 which contains two regions or sets, A and B.

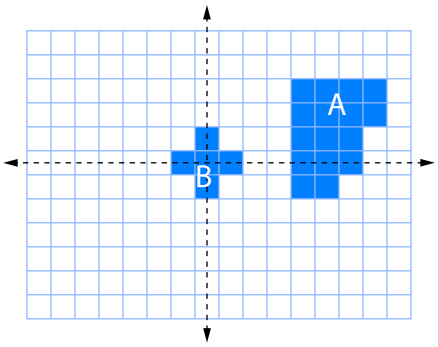


Figure 3.2: A binary image that contains two regions as sets A and B.

Region B in Figure 3.2 is defined as the set {(0,0),(1,0),(-1,0),(0,1),(0,-1)}. The complement of A (or the background of A) is defined as:

$$A^c = \{a \mid a \notin A\} \tag{3.1}$$

If a region A is represented by black pixels in a binary image, then A^c is represented by white pixels. Several common logical operations on image sets are defined in Equation 3.2, and are illustrated in Figure 3.3.

$$\begin{aligned} NOT &\quad C = \overline{A} \\ OR &\quad C = A + B \\ AND &\quad C = A \cdot B \\ XOR &\quad C = A \text{ XOR } B \\ SUB &\quad C = A \backslash B = A - B = A \cdot \overline{B} \end{aligned} \tag{3.2}$$

Both the *translation* of A by vector x, denoted $(A)_x$, and *reflection* of A through the origin, denoted $\hat{A}$ are defined in Equation 3.3, and are illustrated in Figure 3.4. Every pixel of region A is translated by vector x. The *reflection* of a region A about its origin is performed by taking the negation of all pixel coordinates that belong to region A.

$$\begin{aligned} (A)_x &= \{c \mid c = a + x, \text{ for } a \in A\} \\ \hat{A} &= \{x \mid x = -a, \text{ for } a \in A\} \end{aligned} \tag{3.3}$$

In the next section, we will introduce fundamental morphological operators that are the combination of intersection, union, complementation, and translation operators.

3.2.3 Structuring Elements

In morphological image analysis every operator uses a special small set to transform the image under analysis. This small set is called the *structuring element*; it can be of any size and form. The shape of a structuring element is generally chosen based on a priori knowledge

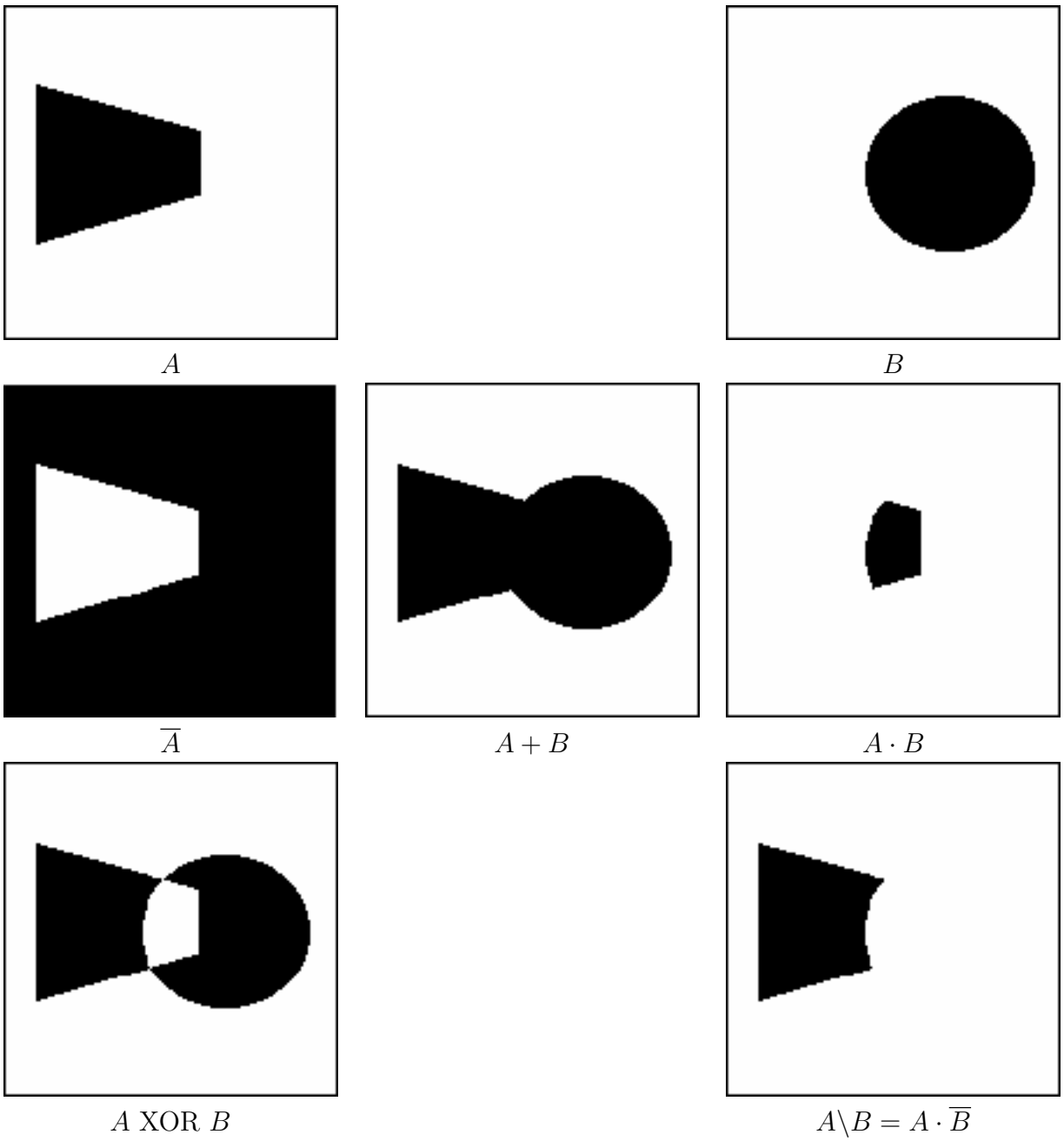


Figure 3.3: Basic binary (Boolean) operations on two images.

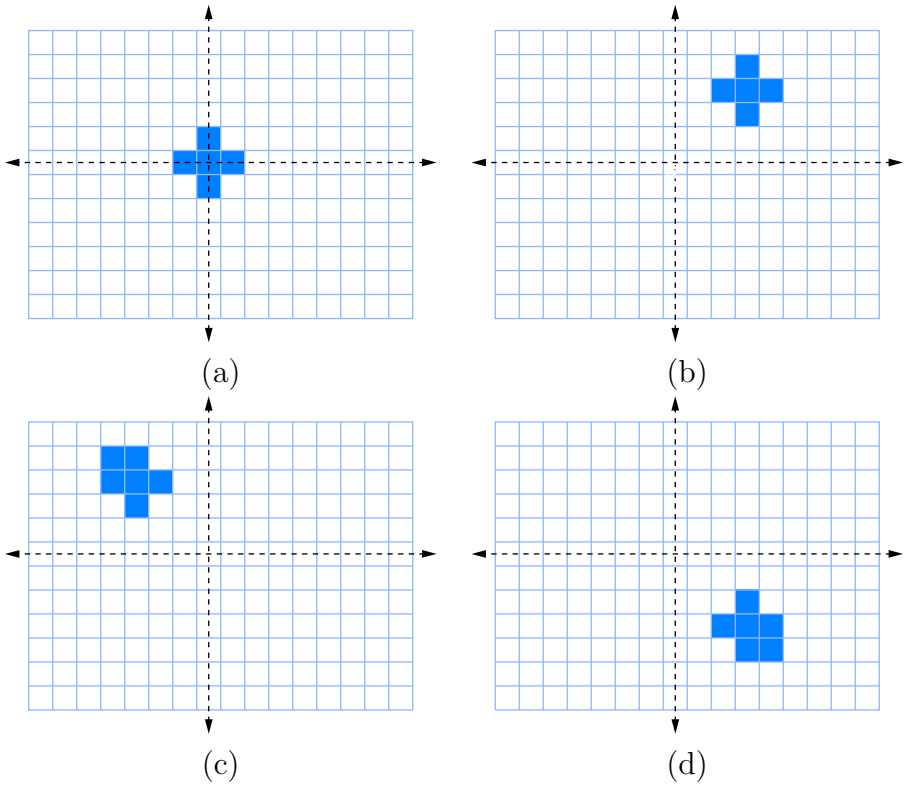
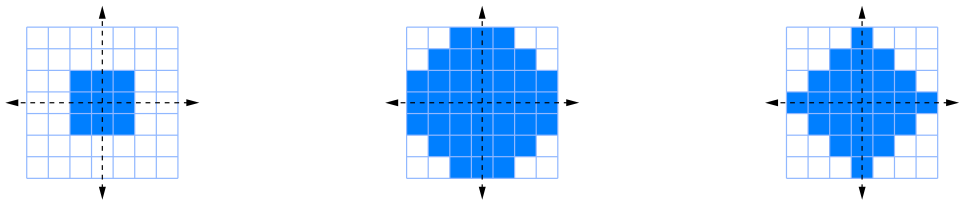


Figure 3.4: Illustration of translation and reflection. (a) Region A. (b) Set A translated by vector x. (c) Set B. (d) The reflection of B.

about the image to be processed. However, the most commonly used structuring elements are a rectangle with specified dimension, a disk with specified diameter, or a diamond with specified size, as shown in Figure 3.5. Another parameter of a structuring element is the location of its *origin.* It is usually the central pixel of a symmetric structuring element, but it can be chosen to be any pixel. If the location of the origin is not at the center, the resulting morphological operations are called *directional morphology.* Directional expansion or shrinkage can be obtained by using directional structuring elements. Using the origin as a handle, translation can place the structuring element anywhere on the image. Structuring elements can be used either for processing or analyzing an image. In the case of processing, for example, a structuring element can be used to enlarge or reduce the size of a region. When analyzing an image, structuring elements can be used to measure the size of a hole in

a region, by checking whether a disk structuring element fits inside the hole, while a larger disk does not.



(a) (b) (c)

Figure 3.5: Common structuring elements: (a) rectangle, (b) disk, and (c) diamond.

3.2.4 Basic Morphological Operations

Dilation

Let A and B be sets in Z^2, and let $\varnothing$ denote the empty set. Then the *dilation* of A by structuring element B, denoted $A \oplus B$, is defined as:

$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \varnothing\} \tag{3.4}$$

That is to say, the dilation of set A by structuring element B can be obtained by taking the reflection of structuring element B about its origin, and then translating the reflected B by x. The dilation of set A by B is the set of all displacements x such that the intersection of $(\hat{B})_x$ and A is nonempty. In other words, dilation by this definition works much like convolution: slide a structuring element to each position in the image and and ask if structuring element "hits" the element of the set A, and take union of all results when the answer is affirmative.

The definition of dilation is not unique. Minkowski [126, 127] has defined dilation in

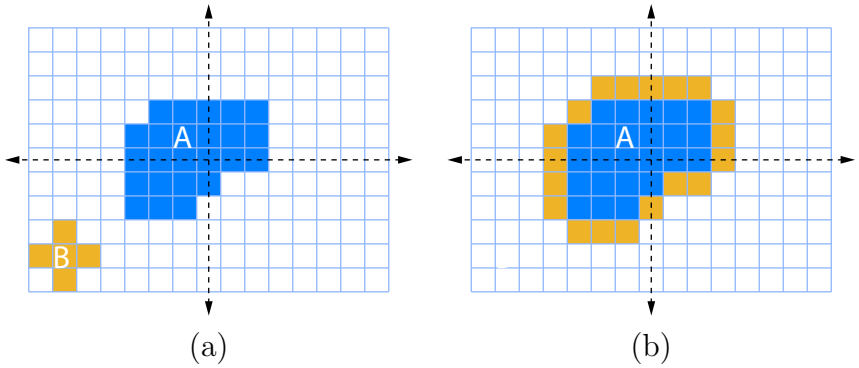


Figure 3.6: The dilation of set A by structuring element B: (a) before dilation, (b) after dilation.

another way which is known as *Minkowsky addition*:

$$A \oplus B = \bigcup_{x \in B} (A)_x \tag{3.5}$$

In other words, the process is to put a copy of B at each pixel in A. As we can see from Figure 3.6, dilation by diamond structuring elements correspond to expansion or swelling of regions in an image.

Erosion

Let A and B be sets in Z^2; the erosion of A by B, denoted by $A \ominus B$, is defined as:

$$A \ominus B = \{x \mid (B)_x \subseteq A\} \tag{3.6}$$

That is the erosion of A by B is the set of all points x such that B, translated by x, is contained in A. In other words, the process of erosion translates the structuring element by translation vector x on set A, and tests if the structuring element fits into the set A. Therefore dilation translates a copy of B at each nonempty member points of set A so that it fits entirely inside of set A. The set of all points x satisfying this condition constitutes

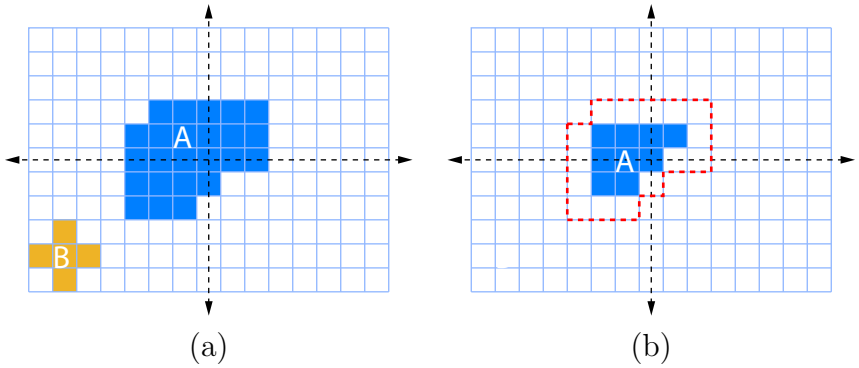


Figure 3.7: The erosion of set A by structuring element B: (a) before erosion, (b) after erosion.

$A \ominus B$.

Minkowsky's erosion definition [126, 127], which is known as *Minkowsky subtraction,* is defined as follows:

$$A \ominus B = \bigcap_{x \in B} (A)_x \tag{3.7}$$

In Minkowsky subtraction, set A is translated by every member point of structuring element B, and the result is obtained by intersecting these translated sets. The process of erosion makes regions in an image "shrinks" or "erodes" as illustrated in Figure 3.7.

Basic Properties of Erosion and Dilation

Commutative–

$$A \oplus B = B \oplus A \tag{3.8}$$

For dilation, the role of structuring element and image can be reversed.

Non-Commutative–

$$A \ominus B \neq B \ominus A \tag{3.9}$$

Associative–

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{3.10}$$

The dilation operation has the associative property, which allows the decomposition of a structuring element into a sequence of simple structuring elements. The decomposition of complex structuring element leads to considerable savings in computational complexity.

Translation Invariance–

$$(A)_x \oplus B \quad = \quad (A \oplus B)_x \tag{3.11}$$

$$A \ominus (B)_x = (A)_x \ominus B = (A \ominus B)_x \tag{3.12}$$

Duality–

$$(A \oplus B)^c = A^c \ominus \hat{B} \tag{3.13}$$

$$(A \ominus B)^c = A^c \oplus \hat{B} \tag{3.14}$$

If A is a region and A^c is the background, Equations 3.13 and 3.14 state that dilation of a region is the same as the erosion of its complement. Similarly, the erosion of a region in an image is the same as the dilation of its complement.

Both erosion and dilation have very important *increasing* properties to extend binary morphology to gray level images.

Increasing in A–

For any structuring element B and two images A_1 and A_2 such that $A_1 \subset A_2$:

$$A_1 \oplus B \subset A_2 \oplus B \tag{3.15}$$

$$A_1 \ominus B \subset A_2 \ominus B \tag{3.16}$$

Decreasing in B–

Let B_1 and B_2 be structuring elements such that $B_1 \subset B_2$:

$$A \ominus B_1 \supset A \ominus B_2 \tag{3.17}$$

Decomposition theorems–

Vincent [184] provides decomposition theorems that make the implementation of basic morphological operators possible in efficient way.

$$\begin{aligned}
Dilation- \quad & A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C) = (B \cup C) \oplus A \\
Erosion- \quad & A \ominus (B \cup C) = (A \ominus B) \cap (A \ominus C) \\
Erosion- \quad & (A \ominus B) \ominus C = A \ominus (B \oplus C) \\
Multiple\,Dilations- \quad & nB = \underbrace{(B \oplus B \oplus B \oplus B \oplus \cdots \oplus B)}_{n\,times}
\end{aligned} \tag{3.18}$$

These theorems are useful in fast implementations of morphological operations since they let a morphological operation with a large structuring element be defined as a sequence of operations with a smaller structuring element.

Opening and Closing

This section introduces two other important operations: opening and closing. Both of these retain important characteristics of an image when filtering [166, 171]. As we have seen from the last section, erosion shrinks an image and dilation swells or expands it. The process of erosion not only removes all pixels in an image that can not contain structuring element, but also it makes all other regions smaller. Opening somewhat reverses the effect of erosion by

dilating the resulting image of erosion.

The *opening* of set A by structuring element set B, denoted $A \circ B$, is defined as:

$$A \circ B = (A \ominus B) \oplus B \tag{3.19}$$

The opening of A by B is therefore erosion of A by B, followed by a dilation of the result by the same structuring element B. If the structuring element fits in A, then the whole structuring element is kept wherever the answer of the test is affirmative. The opening of a binary image by a square structuring element is shown in Figure 3.8. Opening with a disk structuring element eliminates small blobs from an image, breaks narrow isthmuses, breaks narrow bridges between regions, and smooths out contours of regions.

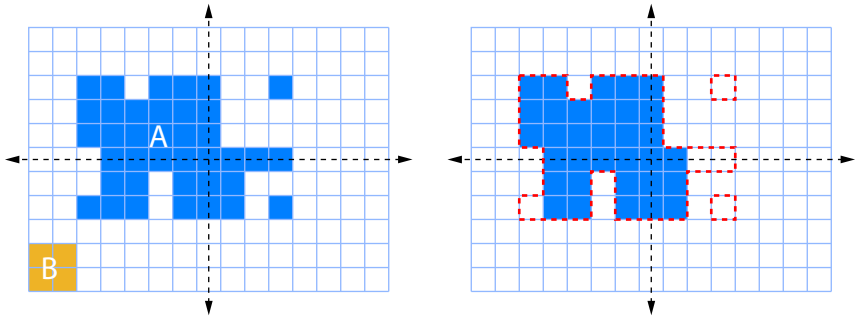


Figure 3.8: Opening of binary image A by structuring element B: (a) before, (b) after (reprinted from [171]).

On the other hand, the *closing* operation smooths contours, merges narrow breaks and long thin gulfs, eliminates small holes, and fill gaps on contours. The closing of set A with structuring element set B, denoted $A \bullet B$, is defined as:

$$A \bullet B = (A \oplus B) \ominus B \tag{3.20}$$

The closing of set A by set B is simply the dilation of A by B, followed by the erosion of

the result by the same structuring element B. In contrast to opening's test of foreground regions that contain the structuring element, the closing operation performs the same test for the background and fills the background region that test affirmative. The effect of a closing operation is illustrated in Figure 3.9. As we can see from Figure 3.9, all background pixels that cannot contain the structuring element are filled by the closing operation.

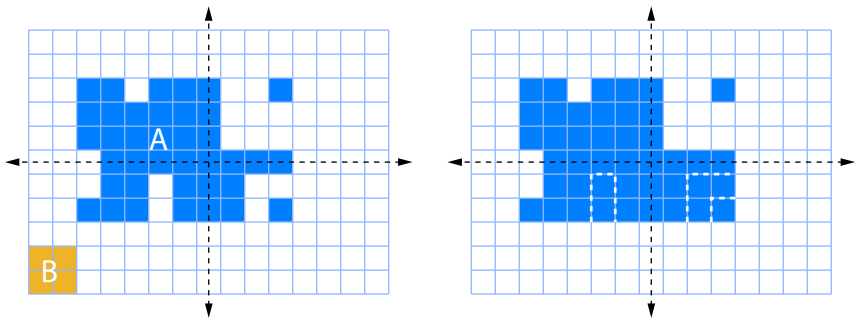


Figure 3.9: Closing of a binary image A by structuring element B: (a) before applying closing operation, (b) after closing. The dashed line shows filled background pixels (reprinted from [171])

Properties of Opening and Closing

Duality–

$$\begin{aligned} (A \bullet B)^c &= A^c \circ B \\ (A \circ B)^c &= A^c \bullet B \end{aligned} \tag{3.21}$$

Opening tests whether the structuring element fits from inside a region. Opening removes region pixels that cannot contain the structuring element. On the other hand, closing behaves in the opposite way since it fills background pixels that can not contain the structuring element. Therefore, opening a region is the same as closing the complement of the same, and then taking the complement of the result.

Translation Invariance–

$$(A)_x \circ B = (A \circ B)_x \tag{3.22}$$

$$(A)_x \bullet B = (A \bullet B)_x \tag{3.23}$$

For the opening of sets A_1, and A_2 such that A_1 is the subset of A_2 ($A_1 \subseteq A_2$) with structuring element B:

Antiextensivity–

$$A \circ B \subseteq A \tag{3.24}$$

Extensivity–

$$A \subseteq A \bullet B \tag{3.25}$$

Increasing monotonicity–

$$A_1 \circ B \subseteq A_2 \circ B \tag{3.26}$$

$$A_1 \bullet B \subseteq A_2 \bullet B \tag{3.27}$$

Idempotence–

$$(A \circ B) \circ B = A \circ B \tag{3.28}$$

$$(A \bullet B) \bullet B = (A \bullet B) \tag{3.29}$$

The idempotence property deserves some discussion. Morphological opening and closing can be seen as ideal bandpass filters [83]. In conventional linear filtering, once an image is filtered by an ideal bandpass filter, consecutive bandpass filtering does not change the result.

In other words, once the result is obtained by a single opening or closing operation, the further iteration of those operations will not effect the end result.

Hit-or-miss

So far, all morphological operations that have been defined use a single structuring element. The hit-or-miss operation uses a special structuring element that contains *two disjoint sets* that share the same origin. The hit-or-miss operation is one of the basic tools for shape detection. During application of a hit-or-miss operation, its structuring element is translated to every possible position on an image. If the first structuring element fits within the foreground, and simultaneously the second structuring element should misses the foreground, then the test is affirmative. The hit-or-miss operation of A by a composite structuring element $B = \{B_{FG}, B_{BG}\}$, denoted $A \otimes B$, is defined as:

$$A \otimes B = (A \ominus B_{FG}) \cap (A^c \ominus B_{BG}) \tag{3.30}$$

where B_{FG} is foreground structuring element, and B_{BG} is background structuring element. This is illustrated in Figure 3.10. In this example, the hit-or-miss operation is used for end-point detection with several composite structuring elements to detect such points.

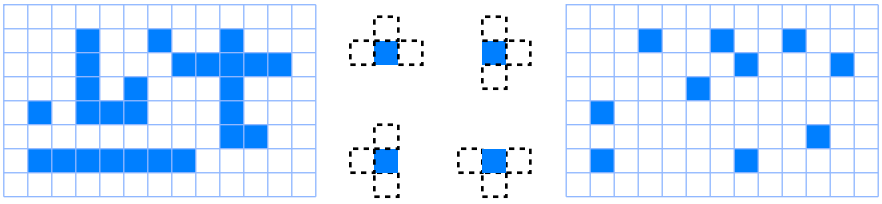


Figure 3.10: End-point detection by a hit-or-miss operation: (a) input image, (b) shows four structuring elements for end-point detection (hatched boxes show B_{BG}), (c) the union of four operations (adapted from [171]).

3.2.5 Gray Level Morphological Processing

Morphological operations that were introduced in the previous section can be extended to gray-level images by using the min or max operations. The extension of gray-level morphology uses *threshold decomposition* in which every gray level is treated as a special binary image for that threshold.

Gray-level morphology has not been used in this thesis. The reader is referred to [165, 173, 174, 175, 83, 86, 166, 171] for more details.

3.3 Decision Trees and Rule Based Classification

3.3.1 The Problem of Classification and Classifiers

A classifier is a system that assigns one of several known class labels to an observed input. For many machine learning algorithms, the goal is to develop rules to be used by a classifier. The problem of classification has been extensively studied. A survey of this field can be found in [190].

Classifiers have been used in different disciplines, such as determining the species of iris plants based on biologist's observations [58] and predicting whether legal contracts are acceptable or not based on their terms [16].

To create a classifier, machine learning algorithms analyze a training data set to determine rules for how classes are assigned, and construct a classifier that can perform similar classifications automatically. The classifier is then tested with a new example whose class is unknown to the system.

For the case of supervised learning, machine learning algorithms are trained on a set of

instances, usually called training samples, each of which is provided with a known class label. After training, the system should be capable of assigning labels to given input samples whose class is unknown. Classifiers can be implemented in many forms but in this thesis, we have used decision trees as classifiers.

3.3.2 Induction of Decision Trees

A decision tree is a graph-theoretic tree in which each interior node represents a decision point, conceptually incorporating an IF-THEN-ELSE statement, and each leaf node represents a final class label that should be assigned. A decision tree can be used to encode knowledge for classifying objects, which are often presented in the form of vectors from a feature space. Each node of the tree defines some test for an *attribute* of an instance, and the branches descending from that node represents the possible values for this attribute.

The problem of determining a class label for a given observation requires inductive inferencing. This is a machine learning process that uses observation to 'guess' class label. This is also true for the induction of decision trees.

To illustrate how a decision tree can be trained, a hypothetical data set is provided in Table 3.1. Each row represents one specific day, Table 3.1 has 14 days (or instances) to describe if the condition of weather for playing golf. The binary-valued "Play" column shows if the condition is good for playing golf which represents class labels ('yes' or 'no' classes). The remaining columns represent features with particular instances shown. For this example. they describe weather condition on a specific day: the humidity, the outlook, the temperature, and if it is windy or not. Figure 3.11 represents a decision tree that has been created using the data in Table 3.1. The classification of a new sample begins at the topmost point in the tree and evaluate the feature of outlook of the given sample. This point,

Table 3.1: Golf anyone? A simple machine learning problem. The data is used for the construction of a decision tree (adapted from [149]).

Day #	Outlook	Temperature	Humidity	Windy?	Play
1	Sunny	85	85	False	No
2	Sunny	80	90	True	No
3	Overcast	83	78	False	Yes
4	Rain	70	96	False	Yes
5	Rain	68	80	False	Yes
6	Rain	65	70	True	No
7	Overcast	64	65	True	Yes
8	Sunny	72	95	False	No
9	Sunny	69	70	False	Yes
10	Rain	75	80	False	Yes
11	Sunny	75	70	True	Yes
12	Overcast	72	90	True	Yes
13	Overcast	81	75	False	Yes
14	Rain	71	80	True	No

and all those in which an attribute is evaluated based on the given data, is called a *node* (circles in Figure 3.11), with the first decision node called the root of the tree. The value of the attribute determines the path that the classifier takes through the tree. Leaf nodes of the tree represent classes (squares in Figure 3.11). For example, if the outlook is sunny, the process of evaluation takes us down to the other node which is humidity, the humidity value from the given sample is tested in the node, and if the humidity is less than or equal to 75%, the decision for playing golf is "yes", if the humidity is greater than 75% then the decision is "no" for playing golf. The question is how to build a decision tree from a set of training data.

3.3.3 Proper Applications of Decision Trees

Even though many algorithms for creating decision trees have been implemented with somewhat different properties and specifications, decision trees are best suited for applications

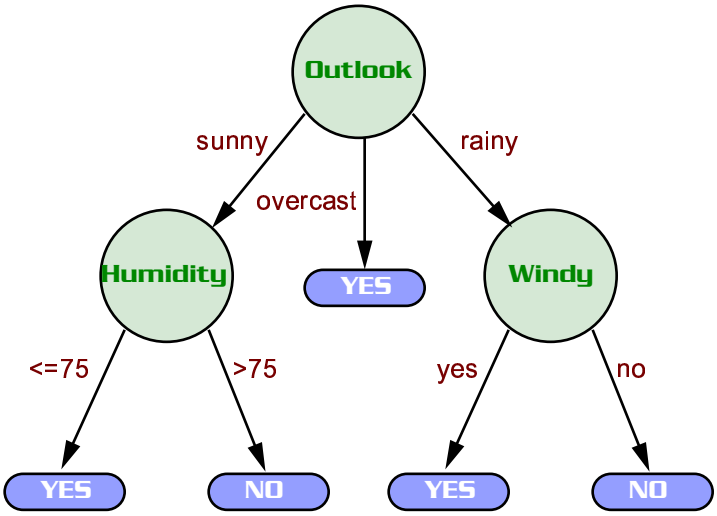


Figure 3.11: Decision tree for the golf problem.

with the following characteristics:

- *Attribute-value description*: instances or objects are expressed by a fixed collection of properties or attributes. This means that each attribute should be discretized or sampled.
- *The output of the decision tree should be predetermined*: The classes which are to be assigned must be known in advance.

Those conditions do not pose any restrictions for most practical applications. Decision trees have been used in such applications as classification of patients by their diseases, customer support by their questions, and the evaluation of mortgage applications by customer's information.

3.3.4 A Classical Algorithm for Building Decision Trees: ID3

Several decision tree learning algorithms have been proposed. The most famous ones among these are ID3, and its new version C4.5 [149, 150], and CART (Classification And Regression

Trees) [26]. The common algorithms that have been implemented in those cases employ exhaustive greedy search from the top down through the spaces of possible decision trees. Since ID3 is the one of the classical decision tree learning algorithm that has been implemented, we will present the concept of decision trees based on the ID3 algorithm.

Quinlan's ID3 [149] builds decision trees in a top-down fashion, and finds which attribute should be selected for the root node by analyzing each instance attribute using a statistical test for how well it classifies instances in the training set. A child node of the root node is then created for each possible value of the selected attribute, and training samples are sorted to the proper child node. The entire process iterates using the training examples associated with each child node to find out which attribute is the best for the child node to test. This process continues until it perfectly classifies all training samples or until all attributes have been used. The simplified version of the ID3 algorithm that is adapted to learn a Boolean valued function is listed in Table 3.2.

As we can see from Table 3.2, the central issue in the algorithm is to choose the "best" attribute for the root node and its child nodes. Intuitively, one can desire to select the attribute that is the most useful for classifying samples. ID3 algorithm uses a statistical property which is called *information gain*. It is used to evaluate how well the attribute under test separates the training samples based on their target classification.

Splitting Criteria

To define information gain accurately, we need to define a measure that is widely used in information theory called *entropy*. Let S be a set of positive and negative samples of some target concept in a 2-class problem. The entropy of set S relative to this simple binary

Table 3.2: ID3 Decision Tree induction algorithm (reprinted from [128]). The best attribute is the one with highest information gain, as defined in Equation 3.31.

ID3(*Examples*, *Target_attribute*, *Attributes*)
Examples are the training examples.
Target_attribute is the attribute whose value is to be predicted by the tree
Attributes is a list of other attributes that may be tested by the learned decision tree
Returns a decision tree that correctly classifies the given *Examples*.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root,* with label = +
- if all *Examples* are negatives, Return the single-node tree *Root*, with label = -
- if Attributes is empty, Return the single-node tree *Root,* with label = most common value of *Target_attributes* in *Examples*
- Otherwise **Begin**
  - $A \leftarrow$the attributes from *Attributes* that best classifies *Examples*
  - The decision attributes for *Root*$\leftarrow A$
  - For each possible value, v_i, of A,
  - Add a new tree branch below *Root,* corresponding to the test $A = v_i$
  - Let $Examples_{v_i}$be the subset of *Examples* that have value of v_i for A
  - if $Examples_{v_i}$ is empty
    - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
    - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, (*Attributes* - $\{A\}$))
- **End**
- Return *Root*

classification is defined as:

$$Entropy(S) = -p_p \log_2 p_p - p_n \log_2 p_n \tag{3.31}$$

where p_p is the proportion of positive samples in S and p_n is the proportion of negative samples in S. In all calculations, $0 \log_2 0$ is defined as 0. Figure 3.12 shows the entropy function in binary classification problem as the ratio of p_p varies between 0 and 1.

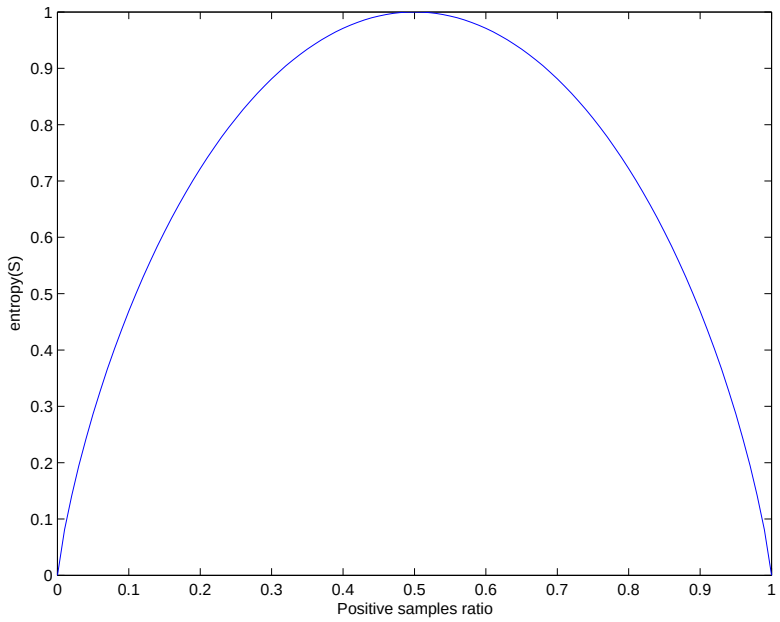


Figure 3.12: The entropy function in binary classification.

Up to this point, we have focused on an entropy that is special case where the target classification is binary. In general, the target attribute can have c different values. Then the entropy with respect to c-wise classification is defined as follows:

$$Entropy(S) = \sum_{i=1}^{N} -p_i \log_2 p_i \tag{3.32}$$

where p_i is the proportion of class i sample in S, and N is the number of classes. From the golf problem, if we want to calculate the entropy of the training set, then the number

of positive and negative samples for playing golf are needed in which they are 9 and 5, respectively. Then the entropy of S relative to the playing decision is

$$Entropy([9+, 5-]) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \tag{3.33}$$

Once the entropy is chosen as a measure of the impurity in a collection of training samples, we can define another measure, which is called *information gain,* that tells us how well an attribute separates the training samples. The information gain, *Gain(S,A)* of an attribute A, relative to collection of samples S, is defined as:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{3.34}$$

where $Values(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v (i.e $S_v = \{s \in S \mid A(s) = v\}$). Again from the golf problem, the information gain for the *Outlook* attribute which can have the values *sunny*, *overcast*, and *rain*. Among the 14 samples, 2 samples are positive and 3 samples are negative decisions

for the attribute *Outlook=sunny*. Similarly the other values for Outlook can be found as:

$$
\begin{aligned}
Values(Outlook) &= sunny, overcast, rain \\
S &= [9+, 5-] \\
S_{sunny} &\leftarrow [2+, 3-] \\
S_{overcast} &\leftarrow [4+, 0-] \\
S_{rain} &\leftarrow [3+, 2-] \\
Gain(S, Outlook) &= Entropy(S) - \sum_{v \in \{sunny, overcast, rain\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= 0.94 - (\frac{5}{14}) Entropy(S_{sunny}) \\
&\quad -(\frac{4}{14}) Entropy(S_{overcast}) - \frac{5}{14} Entropy(S_{rain}) \\
&= 0.94 - 0.347 - 0.347 \\
&= 0.246
\end{aligned}
$$

3.3.5 Problems with Decision Trees

Practical applications of decision trees have some problems. Among the most important are the following: how big the tree should be, handling of continuous attributes, choosing an attribute for a decision node, and computational performance [128].

In general, the algorithm that we presented in the previous section causes the tree to grow until all training samples are perfectly classified. While this is sometimes a reasonable strategy, it can lead to a problem when the training samples are contaminated with noise or when a small set is provided as a training set that poorly represents true target function. The process of perfectly classifying a set of training samples sometimes leads to *overfitting*. When this happens, the learning algorithm simply memorizes all training samples. This may

represent the true function which it tries to approximate. The classification performance of an overtrained algorithm often yields poor performance to unseen test samples.

Overfitting is a major issue in both decision tree learning and other learning algorithms. In the decision tree literature, two methods are available to avoid the overfitting problem: (1) a method to control a tree growing process before classifying all training samples, and (2) a method to let the tree grow to classify all the training samples, and then prune it in the leaf-to-root direction. The second method has been found to be more successful in practical applications. The first one presents the difficulty of estimating the right tree size and deciding when to stop the learning procedure.

No matter which method is used, a primary issue is the choice of criterion to determine the correct final size tree. The most common approaches are: (1) divide all set of samples into training and validation subsets, generate the tree with training set, and then use the validation subset for evaluation of the generated tree; (2) use a statistical measure to evaluate whether expanding (or pruning) of a particular node improves classification accuracy; (3) use an explicit measure of complexity of the tree, halting growth of the tree when this measure is minimized. This method is called *minimum description length.* In practical application, the first method is commonly used.

The other issues of decision tree induction are the handling of continuous-valued attributes, and tolerance of missing attributes in an instance. In case of handling continuous-valued attributes, the implementation of ID3 strictly requires discretized attributes to build a tree. But its new version C4.5 handles continuous-valued attributes with an algorithm that will be described in the next section. Several methods have been proposed to alleviate this restriction [181, 56, 57, 129].

3.3.6 Other Decision Tree Implementations

The previous section has briefly introduced the ID3 algorithm. Since it was one of the first implementations of a decision tree learning method, it has exhibited problems in practical applications. Among them: (1) it does not control the size of the tree, and therefore it can easily overfit training samples, (2) it can not handle continuous variables which is very big problem for practical implementation, and (3) it is not tolerant to a missing attributes of a sample in the training set.

In order to overcome those problems, Quinlan proposed C4.5 [150] which is an extension of ID3. The decision tree grows using a depth-first strategy. Splitting decisions are similar to ID3. Handling of discrete attributes are done the same as in ID3, but in the case of continuous variables, it examines the values of a specific continuous variable in the training set, then it sorts them increasing order for that attribute. Then it identifies the adjacent pairs of samples that differ in their target classifications. Later it generates a set of candidate threshold values by finding middle values of those pairs. Finally, it tests all candidate threshold values against information gain measures and then selects the one that gives the highest information gain. In our golf example, for humidity, if T is the training set, we determine the information for each partition and find best partition at 75. Then the range for this attributes becomes {<=75, >75}. One should note that there is a big computational demand involved in this method.

Another famous algorithm is CART (Classification And Regression Trees) that has been proposed by Breinman [26]. Its algorithm is quite similar to these other decision tree induction algorithms except for its splitting method, controlling the size of tree, and having every node with two branches as oppose to multiple branches in ID3. It is commercially available for data mining applications and for research. It uses a splitting method in each decision

node that is called the Gini index, or "twoing" rule [26].

In recent years, many researchers in the machine learning field have focused on building decision tree based on nonlinear programming. The main focus is to use linear optimization methods to split a given training set instead of using traditional splitting methods. In such methods, a separating plane is used to split training samples in the best possible way. We have used one of these implementations in our system, called OC-SEP (oblique category class separation), tree as proposed by Street [176]. We will present its algorithm in Chapter 4.

3.4 Summary

In this chapter, we have briefly introduced two main areas of study that are used in our system: mathematical morphology, and decision tree induction. In the second section, the theory of decision tree induction has been discussed based on ID3. Although we have focused on ID3, other decision tree algorithms use similar methodologies to construct a tree. The main differences are in: (1) the splitting measure, (2) the choice of top-down vs. bottom-up construction, (3) the pruning method.

Chapter 4

IntelliPost: Intelligent Postprocessing

4.1 System Overview

The IntelliPost system has been developed to improve overall segmentation performance of the complete system that has been described in Chapter 1. The postprocessing system has been designed in such a way that it takes human user experience and/or preference into account for postprocessing, and then stores this information for future use. Initially, the system requires manual postprocessing by a human user. This step is necessary to obtain and store domain knowledge in a knowledge base. The system uses the knowledge base to postprocess other images in a similar way but without a user intervention.

4.2 Modes of Operations

Many supervised machine-learning systems, including the one described here, operate in two different modes: a "learn mode" and a "run mode". During the learn mode, our system

provides a graphical user interface that allows a human user to edit segmented images. The user selects image operators from a menu, designates portions of the image to be processed, and observes the results. This interaction can continue until the user is satisfied with the resulting segmentation for any number of training images.

In the "run mode", a user can load another presegmented image to IntelliPost, and let the system perform postprocessing tasks on it. When the system is in the run mode, IntelliPost uses both a knowledge base and a decision tree based inference engine to postprocess the image automatically. The postprocessing operations that are chosen by IntelliPost should be similar to those performed by the user in the learn mode. In other words, the system should mimic the user actions in the run mode.

We have developed a strategy in which computed region properties, along with other image-related properties, comprise the feature space for decision tree induction. As a simple example, consider region size and radial distance from the center of a log slice as two features that can be computed for a given region in an image. It is possible to map any particular region onto a point in this feature space, and to assign a label indicating a desired action, such as region removal.

4.3 Learn Mode

4.3.1 Overview

In this mode, the user performs manual postprocessing for a given image. The decision about what type of postprocessing operation is suitable for a selected region is made by the user. The user selects from binary morphological tools that were used to implement the postprocessing operation library.

The menu of image-editing operators includes such region tasks as remove, smooth, enlarge, and some others. A user selects those operations interactively to refine a given segmented image. The system observes those actions, and it retains information concerning the regions that were modified. The collected information is stored in a domain knowledge base.

The architecture of learn mode is shown in Figure 4.1. This mode of the system has two main components: the postprocessing operations library that provides region operations to the user to perform refinement on a given image, and a feature extraction module that extracts the relevant information from a region under operation by the user.

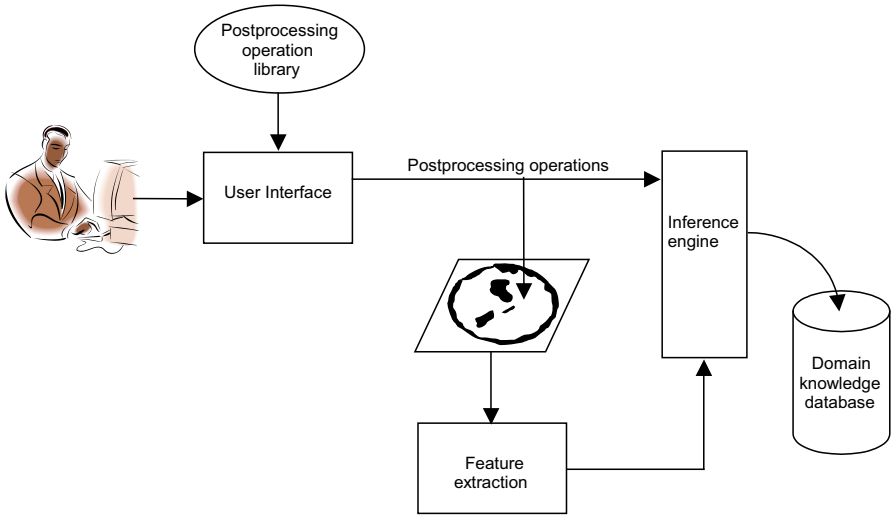


Figure 4.1: System operation during *learn mode*. A human operator edits a segmented image, as the system observes and extracts information to be used later.

4.3.2 Feature Space and Feature Extraction

Instance based learning is one of the commonly used supervised learning methods. In a feature space, each concept or class occupies a subspace, and learning can be viewed as a process of subdividing the feature space. In our application, we assume that the postprocess-

ing problem concerns the use of a decision tree that will be applied to a continuing sequence of cases, in which each new case must be assigned to a predefined postprocessing operation on the basis of observed features.

The input to the decision tree algorithm consists of descriptions of regions from a CT image, each associated with a specific user-selected postprocessing operation. Part of an example feature space is shown in Table 4.1, which includes codes for postprocessing operations and seven feature types. In the feature space, seven geometric feature observations are calculated for each region. Attributes in the feature space were chosen empirically. IntelliPost's feature space can easily be expanded to include other geometric features. To avoid high computational cost, we selected geometric features that are reasonably fast to compute. Since the feature collection is done on-the-fly while the user edits the image, the systems collects the region's attributes and performs necessary refinements.

Table 4.1: An excerpt from the knowledge base that has been used in IntellPost. User-selected operations are indicated in the first column. Initial region types, as assigned by the initial segmentation system, are in the second column. Geometric features are shown in the remaining columns.

Operation Code	Region Type	Area	Radial Distance	Solidity	Major Axis Length	Minor Axis Length	Perimeter
1	1	5	1.027539	1	3.306559	2.129163	14
1	1	14	0.899658	1	5.989203	3.106969	22
1	1	14	0.989989	0.875	7.924809	2.730550	26
2	2	2763	0.602828	0.739957	85.160852	49.409556	387
3	1	22893	0.071006	0.838050	197.766831	169.057192	1223
3	3	38	0.132054	0.826087	22.227154	2.610398	66
4	1	331	0.903929	0.887399	47.154345	9.535321	143
1	2	13	0.472623	1	5.779028	3.098526	22
1	3	19	0.094572	0.76000	13.545129	2.424376	42

The Description of Features

Operation Code

This is the desired class label from the learning system's point of view. The operation code is used to designate what type of operation is to be performed for regions that correspond to the given feature vector. IntelliPost uses this operation code as the class label for its supervised learning algorithm. Each predefined number in this column represents one specific postprocessing operation. When user selects a specific operation in the user interface, the system automatically associates its corresponding code with the new feature vector in the knowledge base. Example operation codes are listed in Table 4.2. The postprocessing operations will be explained later in the chapter.

Table 4.2: Example of codes for postprocessing operations.

Operation Code	Postprocessing Operation
1	Remove
2	Smooth
3	NOOP
4	Enlarge
5	Merge

Region Type

The region type represents the label that was assigned by the initial segmentation system (in our case, an ANN). IntelliPost represents these as different image "layers". When the user refines a presegmented image, region type is stored for each region that is modified. As in the case of operation code, the region type information is encoded as predefined number. The codes are listed in Table 4.3 for general image segmentation. The list can be extended and modified to accommodate other image segmentation applications. For example, one layer

could represent clear wood in wood segmentation but bones in biomedical application.

Table 4.3: Example of codes for region types.

Region Code	Region Type
1	Layer 1
2	Layer 2
3	Layer 3
4	Layer 4
5	Layer 5

Geometric Properties

A binary region can be characterized using a variety of properties that have been developed in the literature [98, 82, 30, 69]. These properties are useful for region classification and provide important resources for comparing and classifying regions in a binary image. A typical binary region is shown in Figure 4.2. The gray shaded area shows the convex hull of the region.

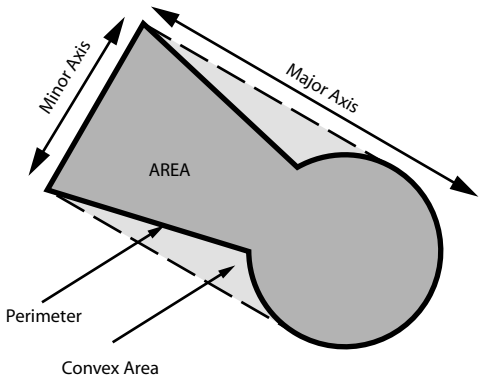


Figure 4.2: Various properties of a binary region: major and minor axes, perimeter, area, convex hull of the region.

Area and Perimeter

The area and perimeter properties are two of the most commonly used for classification problems in a binary image [98, 30]. The area of the region corresponds the total number of foreground pixels in a region.

The perimeter of a binary region is the count of the number of boundary pixels traversed along the region's boundary starting from an arbitrary initial boundary pixel and returning to the initial pixel. The described algorithm works well in 4 connected edges, but a problem emerges when a 8 connected neighborhood is used. The problem occurred mainly when the distance between two adjacent pixels are not positioned diagonally is not constant, but $\sqrt{2}$. To solve this problem, 8-neighborhood chain code is used to represent boundaries of a region, then the perimeter can be estimated as follows [30]:

$$Perimeter = N_e + N_o\sqrt{2} \tag{4.1}$$

where N_e and N_o denote the number of even and odd codes in the chain-coded boundary representation.

Radial Distance

The radial distance property is the Euclidean distance between a region's center of mass and the center of the union of all regions in an image. The easiest way to estimate a region's centroid is the average values of its shape's pixel coordinates. Another method of calculating the centroid is the calculation of moment to obtain the centroid of a binary region. The (i, j)

moment of a region R is defined as:

$$\mu_{ij} = \sum_x \sum_y x^i y^j R(x, y) \tag{4.2}$$

where x and y are the coordinates of the region's pixels. The centroid of a region can be calculated as follows:

$$\begin{aligned} \bar{x} &= \frac{\mu_{10}}{\mu_{00}} \\ \bar{y} &= \frac{\mu_{01}}{\mu_{00}} \end{aligned} \tag{4.3}$$

where ($\bar{x}$,$\bar{y}$) represent the coordinates of the centroid.

The radial distance can be obtained by calculating the Euclidean distance, denoted d_e between the centroid of the union of regions and of a region as follows:

$$d_e(r, u) = \sqrt{(\bar{x}_r - \bar{x}_l)^2 + (\bar{y}_r - \bar{y}_l)^2} \tag{4.4}$$

where r and u represent a region and the union of regions, respectively.

IntelliPost uses the normalized radial distance which is defined as $\frac{d_e(r,u)}{R_u}$, where R_u is the radius of union of regions. Since μ_{00} is the area of a region, R_u can be obtained by $R_u = \sqrt{\frac{\mu_{oo}}{4\pi}}$. The union of regions is assumed to be a rounded shape .

Solidity

The solidity property is defined as the ratio of area measure to the convex hull of the same region. This is illustrated for a star shaped region in Figure 4.3. Soile [170] showed that mathematical morphology can be used to find convex hull of a region. The proposed algo-

rithm offers a trade-off between accuracy of the convex hull and computational efficiency. To avoid computational cost, less accurate convex hull finding can be selected.

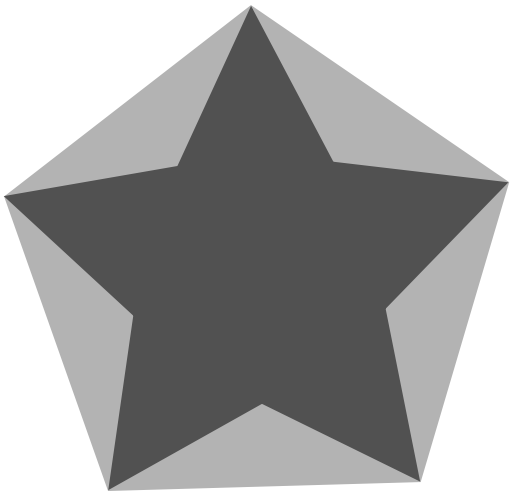


Figure 4.3: The convex hull region of a star. Gray shaded areas show convex hull of the region.

Major and Minor Axis Lengths

Major and minor axis lengths are important features that can be estimated using eigenvalues. To obtain the major and minor axes of a region, a region is represented by a set of points:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\} \tag{4.5}$$

and assume that those points are represented by a random vector $S = [x, y]$ and let C be the covariance matrix of such vector and is defined as follows:

$$C = \begin{pmatrix} \bar{\mu}_{20} & \bar{\mu}_{11} \\ \bar{\mu}_{11} & \bar{\mu}_{02} \end{pmatrix} \tag{4.6}$$

where

$$\begin{aligned} \bar{\mu}_{11} &= \sum_x \sum_y (x-\bar{x})(y-\bar{y}) \\ \bar{\mu}_{20} &= \sum_x \sum_y (x-\bar{x})^2 \\ \bar{\mu}_{02} &= \sum_x \sum_y (y-\bar{y})^2 \end{aligned} \tag{4.7}$$

where $\bar{x}$,and $\bar{y}$ denotes the centroid of x and y coordinates of a region respectively. This is illustrated in Figure 4.4. The eigenvectors of the covariance matrix show the direction of the major and minor axis of a region. The length of those axes can be obtained by taking the square root of eigenvalues of the covariance matrix.

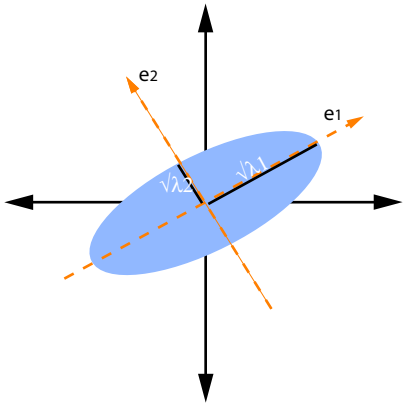


Figure 4.4: The major and minor axes of a ellipsoid. The vectors e_1, and e_2 represent eigenvectors of the covariance matrix.

4.3.3 Postprocessing Operation Library

The postprocessing operation library is another important component of the IntelliPost system. This module is used for both learn mode and run mode. In the learn mode, the user uses the library to refine regions in a presegmented CT image. In the run mode, the inference engine uses the library to perform similar postprocessing operations for a new image. The

postprocessing operation library contains several region operations that have been implemented using mathematical morphology. In the next section, we will explain operations that have been developed for the postprocessing operation library. Those are remove, smooth, enlarge, merge, and NOOP. The set of operations were found to be a reasonable, effective set for region refinement.

Remove

The remove operation eliminates regions from an image. If the user selects a region R_1 as illustrated in Figure 4.5, and the system "removes" by merging its pixels into an adjacent region. Using a region adjacency graph, the algorithm determines the length of each border between R_1 and surrounding regions. For example, the length of the border between R_1 and R_2 is the count of pixels that are shared between these two regions, denoted as $N(R_1, R_2)$. The region that gives maximum border length fills the removed region area. In our example, R_2 fills the area of R_1.

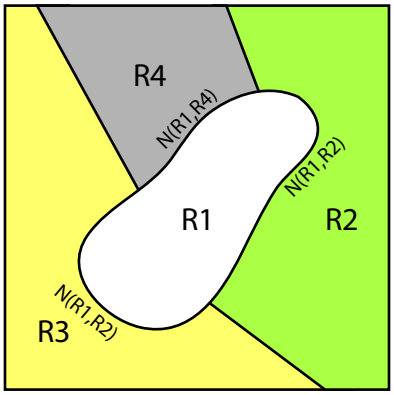


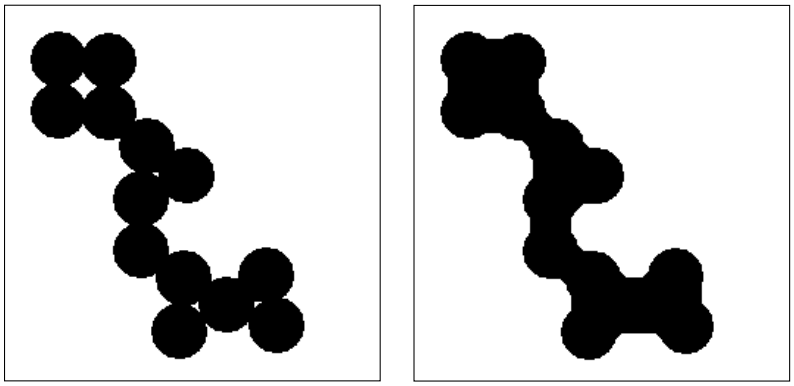
Figure 4.5: The remove operation re-assign a region's pixels to an adjacent region. Here, R1 is combined with R2, because their shared border is the longest.

Smooth

The smooth operation was developed to reduce the curvature of a region's boundary, fill small openings, and connect small gaps. The implementation of the smooth operation uses the morphological closing operation with a disc shaped structuring element. Mathematically, let R_i denote a region that is selected by the user. Then the smooth operation is defined as:

$$R_i^{'} = f(R_i, B) = (R_i \oplus B) \ominus B \tag{4.8}$$

where $R_i^{'}$ is the smoothed region and B is the disc shaped structuring element. An example shown in Figure 4.6.



(a) (b)

Figure 4.6: The smooth operation. (a) Input image. (b) The result smoothing which shows rounded edges, and the removal of small openings.

Enlarge

The enlarge operation is used to swell or expand a region. This operation uses morphological gradients to find the boundary contour of the region. The morphological gradient, known as

a "boundary peeler", is defined as follows:

$$\Psi_{grad}(R) = (R \oplus B) \backslash (R \ominus B) \tag{4.9}$$

where R is the region and B is structuring element. By using erosion and dilation operations and then taking the set difference of the result, it is possible to detect internal and external boundaries of a region. The enlarge operation uses the external morphological gradient operator to obtain the external boundary of a region. This is a variant of Equation 4.9, defined as follows:

$$\Psi^{+}_{grad}(R) = (R \oplus B) \backslash R \tag{4.10}$$

This operation first dilates a region by the structuring element B, and then determines the set difference of this result with the region itself. Once the external boundary pixel set is obtained, the union of the region and the external boundary pixel set gives the expanded version of the region. The effect of the operation is shown in Figure 4.7.

$$R_{enlarged} = R \cup \Psi^{+}_{grad}(R) \tag{4.11}$$

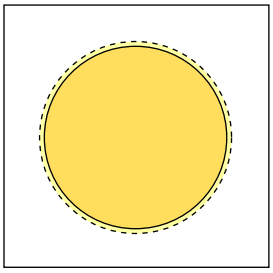


Figure 4.7: The enlargement of a region along its boundary. The dashed lines indicate the enlarged region.

Merge

The merge operation has been implemented to connect selected nearby regions within the same layer. The operation uses directional morphology to connect such regions. The user specifies two regions by mouse, then the system obtains two sets of boundary pixels of the selected regions. Later, it determines two points that give the minimum Euclidean distance between these two contour sets. After these two points have been found, the pair determines a vector that gives the orientation for a line structuring element for directional morphology. The operation finally applies dilation with the constructed structuring element to the smaller region of selected sets iteratively until two regions merge together.

Let R_1 and R_2 denote regions to be merged as illustrated in Figure 4.8. The boundaries of the regions constitute two contour sets:

$$\begin{aligned} C_{R_1} &= (R_1 \oplus B) \backslash R_1 \\ C_{R_2} &= (R_2 \oplus B) \backslash R_2 \end{aligned} \tag{4.12}$$

where C_{R_1} and C_{R_2} denote contour sets of regions R_1 and R_2. Then two sets are used to find a vector $V(p, q)$:

$$V(p, q) = (p \in C_{R_1},\, q \in C_{R_2} \mid \min \|p - q\|) \tag{4.13}$$

where p and q represent pixels in the contour sets in C_{R_1} and C_{R_2}, and $\|\cdot\|$ denotes the Euclidean norm. After finding the vector that gives minimum distance between two contour sets, a directional line structuring element is constructed by using the normalized vector $V(p, q)$. The iterative directional dilation is applied to R_1 until it merges with R_2

$$R_1^{(i+1)} = \{R_1^i \cup (R_1^i \oplus_D B_V) \mid R_1^i \cap R_2 = \varnothing, i = 1, 2, \ldots n\} \tag{4.14}$$

where $\oplus_D$ denote directional dilations which is normal dilation with directional structuring element. Once we obtain the dilated region R_1^n, we can simply take the union of R_1^n and R_2 as:

$$R_{merge} = R_1^n \cup R_2 \tag{4.15}$$

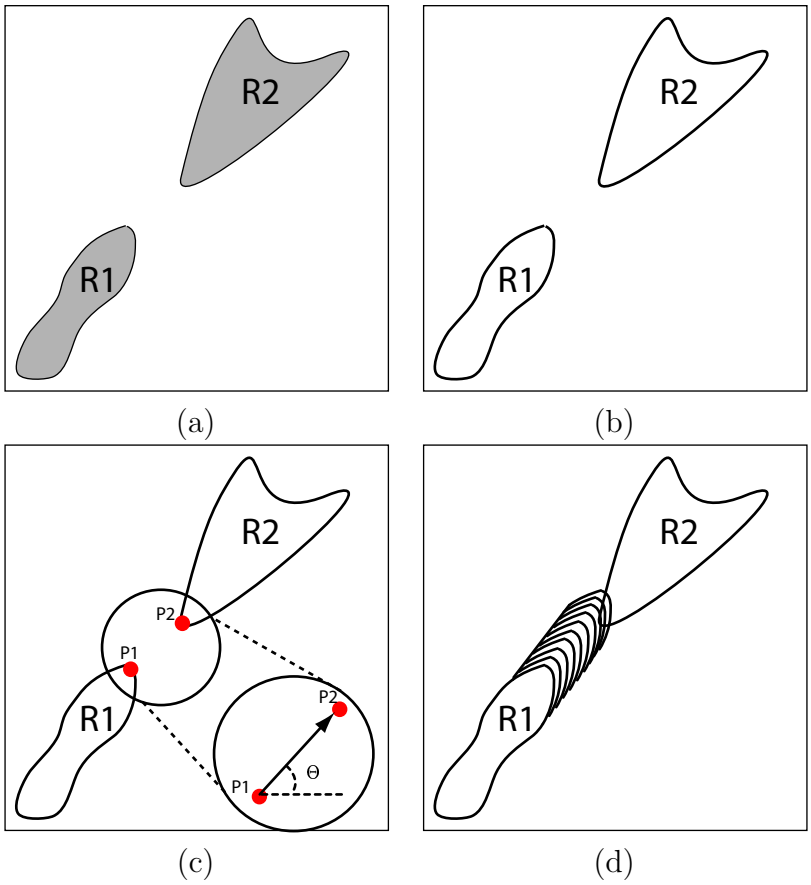


Figure 4.8: Merging two regions. (a) Two regions are selected. (b) Boundary sets are obtained. (c) Two points that give the shortest distance are found, and the directional line structuring is constructed with orientation information. (d) Iterative dilation of R_1 until it merges with R_2.

NOOP

NOOP ("no operation") is necessary for the run mode of the system. The user selects NOOP to indicate which regions are satisfactory and should not be modified. The system does not perform any postprocessing operation for such regions, but it collects geometric features for the knowledge base. In the run mode, the inference engine uses this information to determine satisfactory condition for region refinement. Such regions are processed until they reach a desired condition which need no more refinement.

4.4 Run Mode

4.4.1 Overview

When a user activates the run mode, IntelliPost automatically generates a set of rules based on its stored knowledge. The system will automatically apply its rules to update an image that is loaded by the user. Based on the geometric properties of those regions, the system selects operations and applies them. Ideally, the system will generate a postprocessed output image that is the same as the original human operator would produce.

With a feature-based training set, it is possible to use information-theoretic methods to construct a decision tree that can select an action for any point in the feature space. As mentioned in Chapter 3, many decision tree induction algorithms have been proposed. Among the most common ones are ID3, C4.5, and CART [26, 149, 150], and recent ones are based on linear and nonlinear optimization are MSM-T and OC-SEP [15, 13, 176]. The former group creates a classification tree by repeatedly subdividing the feature space using linear univariate thresholds, and the latter group creates linear separating planes to divide

training samples. The result is a set of separating hyperplanes that may not be parallel to the feature-space axes, with resulting subsets forming a partition of the feature space. In classical decision tree induction algorithms, entropy measures are often used to select which feature variable is to be considered at each node of the decision tree. But in case of linear programming based decision tree induction, separating hyperplanes are used to separate the training examples at each node of the decision tree.

The run mode architecture is shown in Figure 4.9. As illustrated, the inference engine replaces the role of the user. The system extracts geometric features from regions in an image, and then provides feature vectors to an inference engine. Based on postprocessing rules which are constructed using domain knowledge, it decides what region operations should be applied to each region.

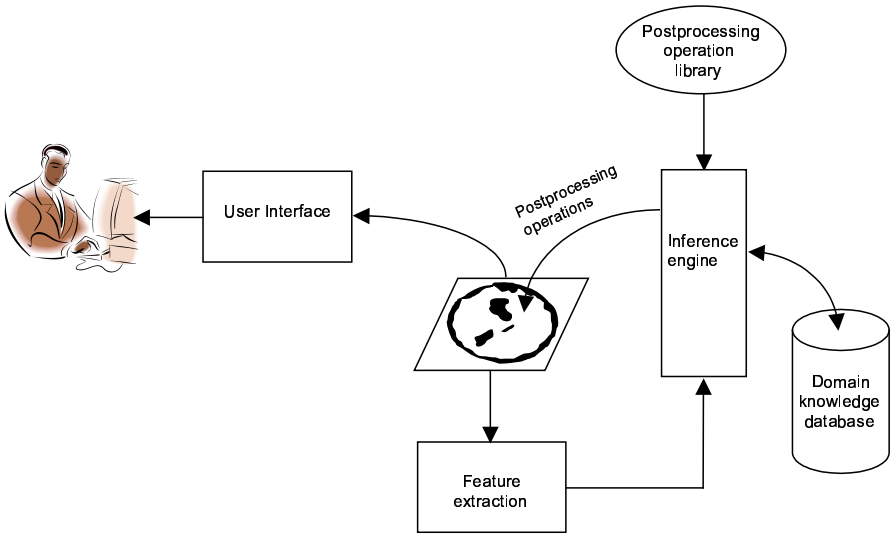


Figure 4.9: System operation during *run mode*. The user provides segmented image, and the system automatically modifies the image in a manner similar to the user's earlier editing steps.

4.4.2 OC-SEP Decision Tree Induction as Inference Engine

The inductive classification algorithm tries to minimize some error metric to construct a classifier that separates unseen test samples with minimum error. Bennett [15] has implemented a decision tree induction algorithm with linear programming. An excellent review of using mathematical programming in learning can be found [25].

As described in the previous chapter, classical decision tree induction algorithms construct a tree based on information gain and entropy measures. The decision trees can be constructed using one or more separating surfaces or hyperplanes in the feature space of a given training examples. Each decision node represents a separating hyperplane. The algorithm divides training examples recursively into the subset of data that is formed in the previous step. This iteration continues until all samples (or nearly all) of the points in a region belong to the same class. Despite the common use of C4.5 and CART, they are limited to separate feature space by axis-parallel separating hyperplanes. This limitation leads to a larger decision tree since it requires several more planes to separate training samples completely. As a result of that, the generalization performance of these trees suffer from because they tend to generate larger decision trees.

Several researchers have addressed this shortcoming. The OC1 family of classifiers have been proposed to offer a number of separating objectives [129] but these are still based on purity based separating criterion. Another method has been proposed by Fayyad and Irani [55]. Their work indicates that purity based separation are not sensitive to class separation because they measure it only indirectly, often causing the decision tree algorithm to generate more nodes and as a result bad generalization performance. Instead of using purity and information gain measures, they have proposed a class of measures based on the orthogonality of the class vectors in the child nodes. The class vector of a set S of training samples is

defined as $[c_1, c_2, \ldots, c_k]$, where c_i is the number of examples of class i in S. Once the training samples are divided into region, their class vector should be as orthogonal as possible relative to the other class vectors. Based on this observation, they have defined several measures called C-SEP (Class SEParation) and proved their effectiveness. But their method still separates training samples by axis-parallel separating planes.

Constructing optimal separating planes has been long studied by researchers [113, 114, 65, 116, 11, 12], and their natural extension to decision tree induction was introduced by Bennett [15]. The introduced decision tree algorithm can handle two-class problems, and most recent one has been proposed by Street [176] that provides an ability to handle multi-class problems.

OC-SEP Decision Tree Induction Algorithm

OC-SEP is based on robust linear mathematical programming [13, 14, 115]. Two-class problems were addressed in those studies. Suppose we are given a training samples that contains two disjoint sets A_1 and A_2 that we would like to separate. The main goal is to find a separating plane which is defined by its normal vector w and the distance from the origin, denoted as θ. A separating plane is defined as:

$$x^T w = \theta \tag{4.16}$$

This lies within the feature space of the given examples such that all points of A_1 lie on one side of the plane ($A_1 w > e\theta$) and the points that belong to A_2 lie on the other side ($A_2 w < e\theta$) where e is a vector of ones of the same dimension as feature space and x is a feature vector. The solution of this problem exist only when these two sets are linearly separable, which in general not the case. Therefore, the distance between the plane and misclassified points is

minimized. This is obtained with the following normalized minimization problem:

$$\min_{w,\theta}(\frac{1}{n_1}\left\|(-A_1w+e(\theta+1))_+\right\|_1+\frac{1}{n_2}\left\|(A_2w-e(\theta-1))_+\right\|)_1 \tag{4.17}$$

where $\|\cdot\|_1$ denotes the 1-norm and $(z)_+$ denotes $((z)_+)_i = \max\{z_i, 0\}, i = 1, \ldots, m$ for $z \in R^m$. n_1 and n_2 denote the number of samples in each set. This formulation can be shown to be equivalent to the following linear problem:

$$\begin{aligned} \min_{w,\theta,y,z} \quad & (\frac{ey}{n_1}+\frac{ez}{n_2}) \\ \text{subject to} \quad & A_1w-e\theta+y \geq e \\ & -A_2w+e\theta+z \geq e \\ & y,z \geq 0 \end{aligned} \tag{4.18}$$

This approach provides several advantages. Some of them are as follows: (1) a solution is found in the linearly separable case; (2) if two sets share the same centroid, the null solution is found which is $w = 0, \theta = 0$; and (3) using 1-norm error gives robustness to the effects of outliers. One should note that this approach is very similar to popular support vector machines [183].

In case of constructing a decision tree, this problem is recursively solved on the subset of the initial training set as done in [15]. But this induction tree algorithm provides a decision tree that is capable of discriminating two-class problems.

In order to handle multi-class problems, OC-SEP provides an algorithm that applies an orthogonality measure instead of using a misclassification minimization method. The basis of orthogonality based separation is the class vector which represents the counts of elements of each class in the given subset of examples. OC-SEP tries to make the corresponding class

vectors induced by a separating hyperplanes at each node as close as possible to orthogonal. In order to utilize continuous optimization methods, distance based approximation is used to estimate a class vector. The value $e^T((A_1w - e\theta)_+)$ is proportional to the total distance to the plane of all points in A_1 that are on the right side ($A_1w > e\theta$) of the plane; the same way, $e^T((-A_1w + e\theta)_+)$ gives a distance measure for the left side of plane. Therefore, this distance measure can be used in robust linear programming problem in place of class vector. To make definition of the problem simpler, lets v_1 represent the distance measure for class 1, which has k components $v_{1,i} = e^T((A_iw - e\theta)_+)$. Using similar notation for the other class vector, the robust linear program becomes:

$$\begin{aligned} \min_{w,\theta} \quad & v_1^T v_2 \\ \text{subject to} \quad & e^T((A_iw - e\theta)_+) = v_{1,i}, i = 1, \ldots, k \\ & e^T((-A_iw - e\theta)_+) = v_{2,i}, i = 1, \ldots, k \end{aligned} \tag{4.19}$$

This mathematical programming approach will find a solution when a separating plane divides training samples entirely. At the same time, the minimum can be found by a plane that lies entirely on one side of all points, for example $v_{1,i} > 0$, $v_{2,i} = 0$. Therefore the optimization problem needs to be modified to alleviate this problem. The modification occurs in the objective function. The new objective tries to minimize the products of elements of both vectors as follows:

$$\begin{aligned} \min_{w,\theta} \quad & v_1^T v_2 + \prod_{i=1}^{k} v_{1,i} + \prod_{i=1}^{k} v_{2,i} \\ \text{subject to} \quad & e^T(A_iw - e\theta)_+ = v_{1,i}, i = 1, \ldots, k \\ & e^T(-A_iw + e\theta)_+ = v_{2,i}, i = 1, \ldots, k \end{aligned} \tag{4.20}$$

If one class has many points, its class vector becomes larger which drives the solution as defined in Equation 4.20 to favor to that class during optimization. In order to eliminate this behaviour, the class vectors should be normalized as $v_{1,i} + v_{2,i} = 1$. The final modification happens in the plus function which not differentiable at zero. To use conventional optimization software, the plus function is replaced with:

$$p(x) = x + \frac{1}{\alpha}\log(1 + \exp(-\alpha x)) \tag{4.21}$$

where α is some predefined constant (IntelliPost uses α= 100,000).

The final version of the mathematical program becomes:

$$\begin{aligned} \min_{w,\theta} \quad & v_1^T v_2 + \prod_{i=1}^{k} v_{1,i} + \prod_{i=1}^{k} v_{2,i} \\ & p(e^T(A_i w - e\theta)) = u_{1,i}, i = 1, \ldots, k \\ & p(e^T(-A_1 w + e\theta)) = u_{2,i}, i = 1, \ldots, k \\ & \frac{u_{1,i}}{u_{1,i} + u_{2,i}} = v_{1,i}, i = 1, \ldots, k \\ & \frac{u_{2,i}}{u_{1,i} + u_{2,i}} = v_{2,i}, i = 1, \ldots, k \end{aligned} \tag{4.22}$$

This optimization problem can be solved using a sequential quadratic programming method [19] as implemented in the MATLAB optimization toolbox [40].

Building a Decision Tree with OC-SEP algorithm

In the IntelliPost implementation, we have used 7 attributes that construct the feature space. In order to illustrate how OC-SEP algorithm learns from training samples, we have used a hypothetical set of training samples which contains 2 dimensional features to make

visualization easier. In this example, OC-SEP constructs a decision tree that generalizes remove, smooth and NOOP operations based on area and radial distance attributes which were described in the previous section. The feature space is shown in Figure 4.10. The task is to build a decision tree to separate these training samples with the OC-SEP algorithm.

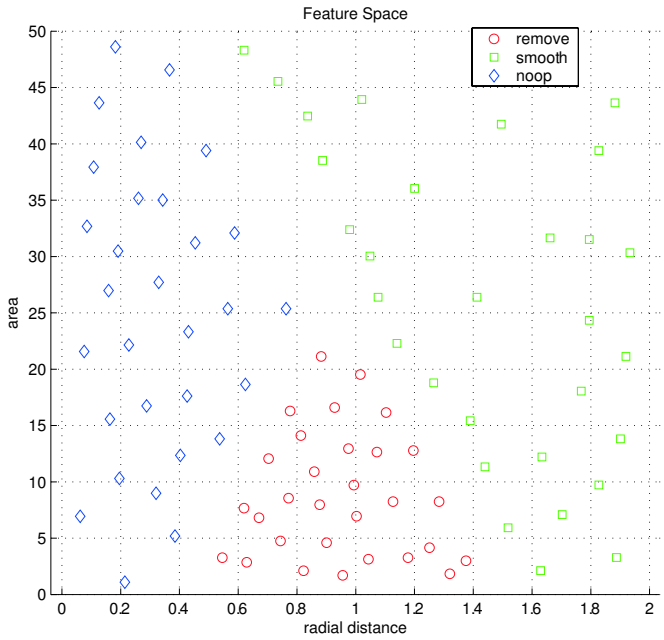


Figure 4.10: Training samples for OC-SEP decision tree algorithm.

After the training samples are given to OC-SEP algorithm, it constructs a data structure with contains decision tree with information concerning separating planes. The first step of the algorithm uses all training samples to solve the optimization problem that has been defined in Equation 4.22. The solution of the problem give the separating plane parameters $w = [0.6487\, 0.0165]$, and $\theta = 1.0302$. These separating planes give a line equation in R^2 space which is shown as P_1 in Figure 4.11a. At this point training samples are divided into two groups: (1) all smooth samples on the right of P_1, and (2) all remove and NOOP samples on the left. The constructed decision tree up to this point is shown in Figure 4.11b.

Because the left side contains both remove samples and NOOP samples, the algorithm

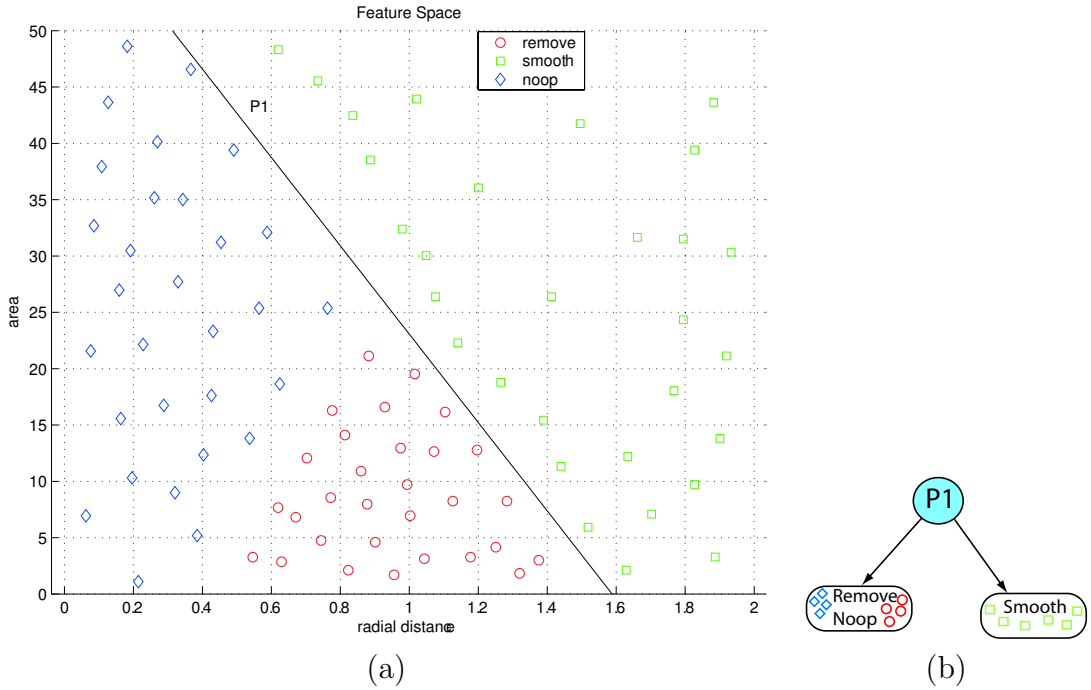


Figure 4.11: The first step of the algorithm. (a) Samples are divided into two partitions, (b) The corresponding decision tree at this step, leaf of the tree represents training samples.

iterates again. The new optimization problem with subdivided training samples is solved based on Equation 4.22. The solution of this iteration yields P_2 which give separating plane parameters $w = [1.91139 \ -0.0229]$, and $\theta = 0.9646$. This solution is illustrated in Figure 4.12a, and the corresponding decision tree is shown in Figure 4.12b. The algorithm stops subdividing feature space at this point since all training samples are separated completely. The final decision tree is shown in Figure 4.12b.

For evaluating a new test vector x with the constructed decision tree, the vector x is presented to the first node of the decision tree, which compares it with P_1. If the test satisfies $x^T w > \theta$, the evaluation path takes right branch of the tree. Otherwise, the left branch will be taken. If the left branch is taken, the test vector will be evaluated with respect to P_2. Depending on the test result, the class label will be assigned accordingly. As we can

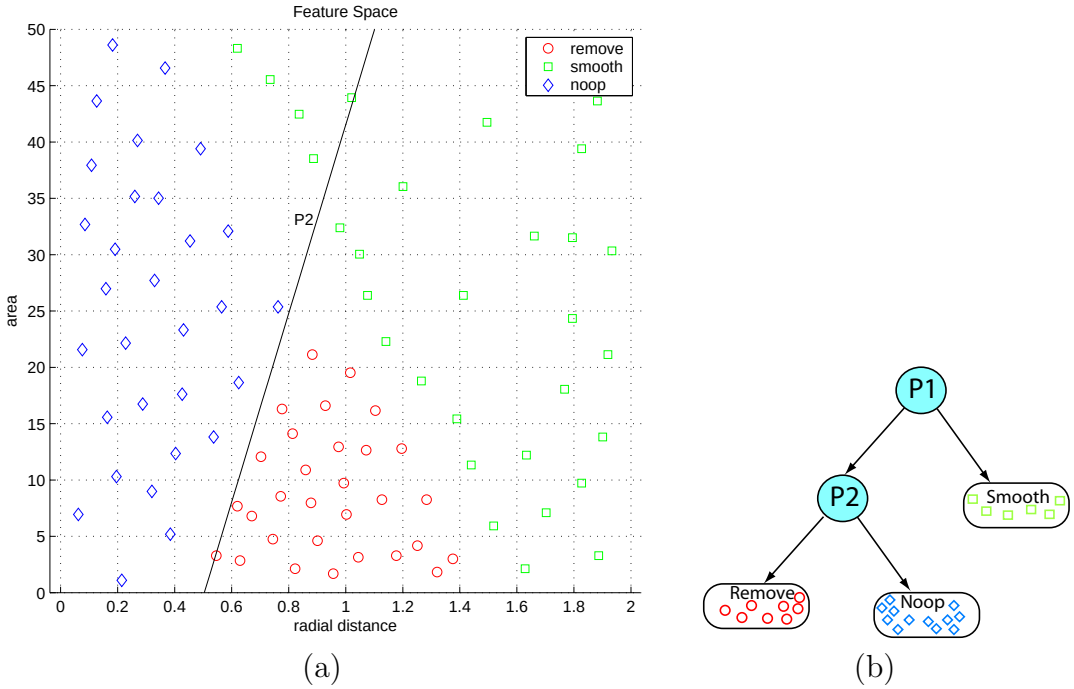


Figure 4.12: The final step of the algorithm. (a) All training samples are separated completely. (b) The final decision tree contains two decision nodes, denoted by P_1 and P_2, and every leaf represents a separate class.

see, the classification rule for each class can be defined as:

if $x^T w_{P1} > \theta_{P1}$ then the class is smooth

else

if $x^T w_{P2} > \theta_{P2}$ then the class is NOOP

else

the class is remove

As we see from the rule, it is not as intuitive as classical decision tree induction. The OC-SEP algorithm provides smaller trees with oblique separating planes, which means better generalization performance.

4.4.3 Region Feature Extraction

After the construction of the decision tree with OC-SEP algorithm is complete, IntelliPost constructs a special data structure that contains all regions' relevant information, which will be used in the region refinement process. Before the system builds the data structure, layer separation takes place to transform the initial segmented image to a series of binary images. In other words, layer separation is nothing but binary thresholding process that separates each defect layer into binary images. Each binary layer contains regions that are regions type. Then IntelliPost applies connected component analysis to identify separate connected sets (regions) in each binary image. All regions in the image are then labeled with a unique number. All regions' geometric feature values and a region adjacency graph are determined and stored in a data structure which is to be used by the inference engine. The structure is then sorted by a region's size from small to large. Region sorting is important because smaller regions tend to be more sensitive to morphological operations such as smooth. If the smooth operation is applied to bigger regions, the result may eliminate smaller regions are inside of the larger region. The process of layer separation and geometric feature extraction are illustrated in Figure 4.13.

Each entity in the structure contains a region's identification number, a region's geometric features that are calculated in a similar way as in the learn mode, and "processed" flag that indicates if the region has been processed. Separately, another matrix is used to store region adjacency information. An example region adjacency graph is shown in Figure 4.14. This information can also be represented as a sparse matrix that contains zeros and ones. Its rows represent region's identification numbers, and its columns contain one where the adjacent regions are present.

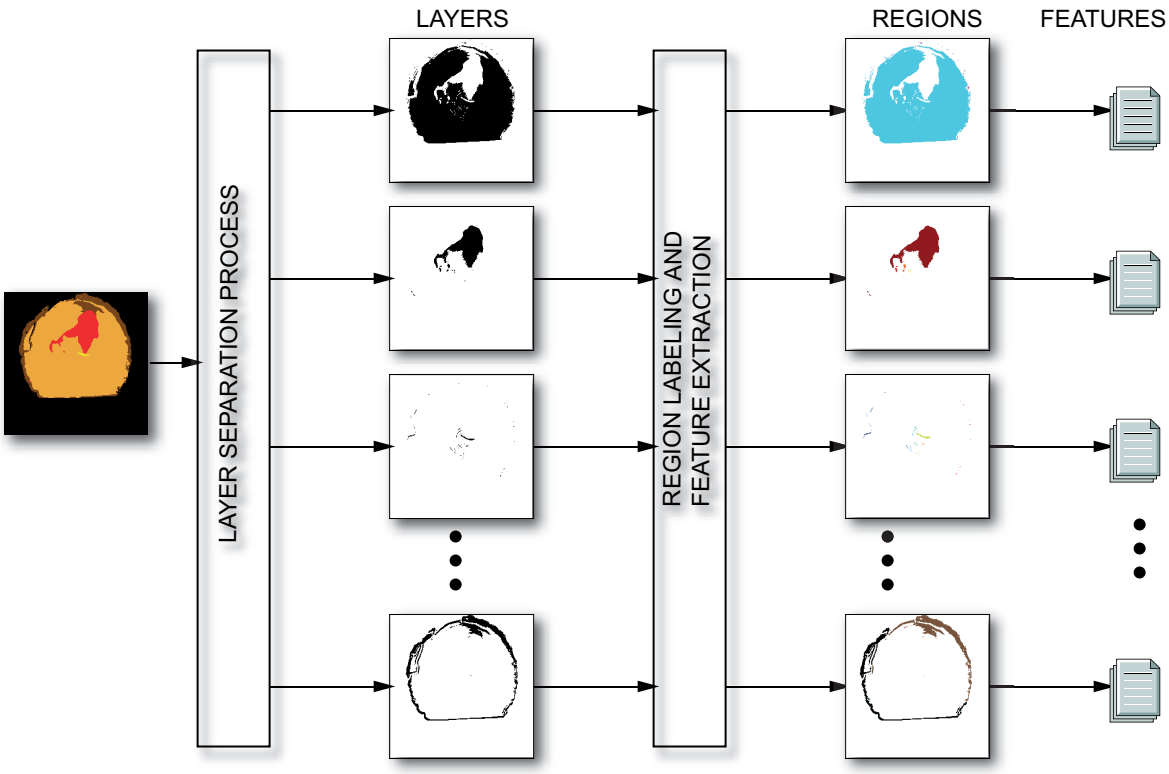
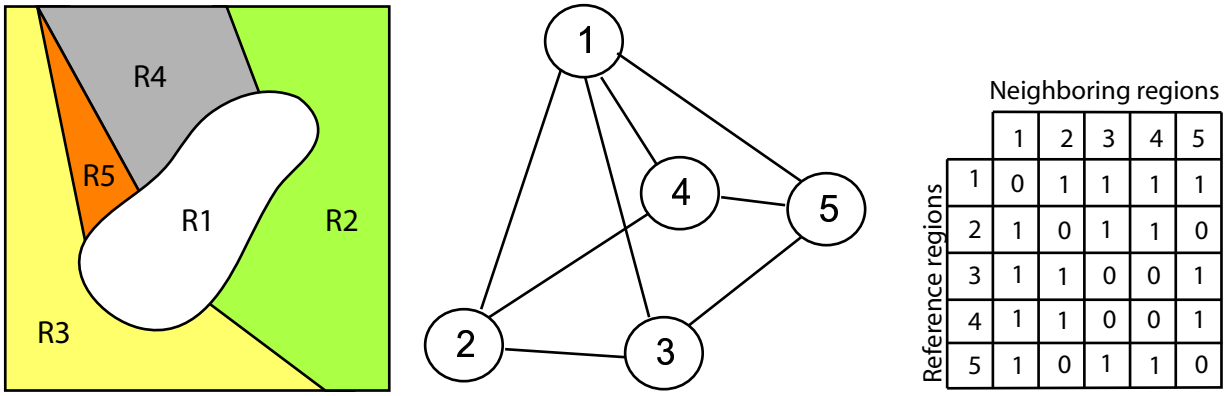


Figure 4.13: Region analysis algorithm. The initial segmented image is separated into binary images (layers). Connected component analysis is used to distinguish separate regions for each layer. Geometric features are determined to populate a data structure for region refinement analysis.

4.4.4 Region Refinement

The final step of the run mode is to analyze all regions in the structure that were obtained in the previous step. Using the knowledge base that was stored during the learn mode, a decision tree is constructed by using the OC-SEP decision tree induction algorithm. As described in the previous section, the OC-SEP decision tree induction algorithm partitions feature space with oblique separating planes. Using oblique separating plane results in a smaller tree, which implies better generalization performance.

After a decision tree has been constructed, IntelliPost analyzes every region in the structure that was constructed in the previous step. In this step, the IntelliPost inference engine



(a) (b) (c)

Figure 4.14: (a) A typical segmented image (b) The region adjacency graph with numbers in graph nodes indicating region identification numbers. (c) The region adjacency matrix, in which each nonzero entry r_{ij} in the matrix indicates that region r_i is adjacent to r_j.

applies the generated decision tree to select an operation. Because regions in the structure are sorted from small to large in size, the order of analyzing region starts with the smallest region to the largest region. The inference engine analyze region features along with the region adjacency information to determine what refinement process should be performed. Finally IntelliPost performs the requested operation on the region under analysis.

Let R_i and V_i denote a region and its feature vector respectively. The refinement process can be represented as follows:

$$\begin{aligned} f &= IE(\mathcal{F}, R_i, V_i) \\ R_i' &= f(R_i) \end{aligned} \tag{4.23}$$

where $\mathcal{F}$ is the function space that contains all possible postprocessing functions and IE is the inferencing process. The output of the inferencing process is a refinement operation f for region R_i. Once the region refinement function f is chosen, the selected refinement function is used to update region R_i to get R_i'. The basic steps of region processing are shown in

Figure 4.15.

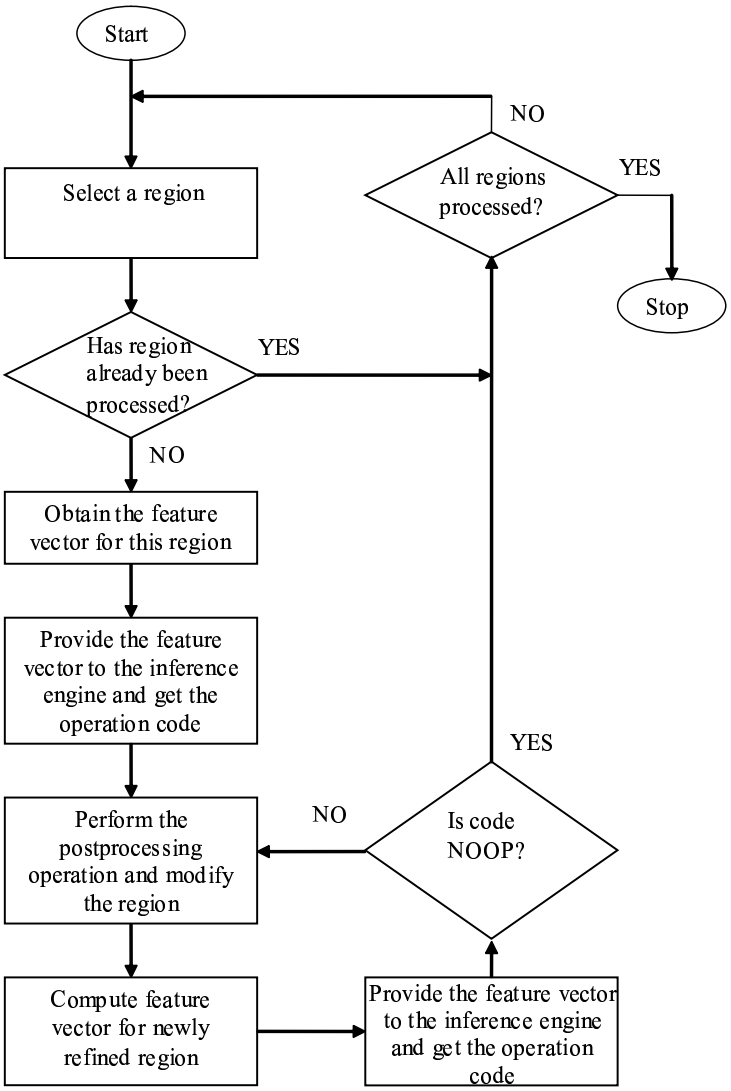


Figure 4.15: Overall region processing algorithm for the run mode of IntelliPost.

As we can see in Figure 4.15, the algorithm starts with the first region in the region structure. In the second step, it checks whether this region was previously processed or not. This test is important for the case of merge operation. If the refinement procedure is asked to merge the current region, it needs another region for merging. Then it searches for the best candidate region nearby to be merged. Then the refinement resumes from where it was before searching another region for merging. There will be at least one region in the structure that

has been already processed (the one that is merged) which needs no refinement anymore. This region should be labeled as processed to avoid any confusion. When the refinement procedure finds regions that are labeled as processed, it skips to the next region in the region data structure.

The inference engine determines what kind of refinement operation should be applied to each presented region. IntelliPost then executes operation on that region. The process of refinement is recursive procedure that refines a region iteratively until the inference engine give NOOP for the region. This iterative mechanism allows the construction of a sequence of refinement operations on a region. Every time each a region is processed, both region adjacency graph and the data structure get updated accordingly. The refinement procedure continues until every region gets analyzed and processed by the inference engine.

After performing region refinement on the individual layers, IntelliPost needs to combine all the processed layers into a final composite result. However, some regions will have grown to overlap other regions. To resolve these conflicts, we have formulated precedence rules covering all region types. The precedence rules determine which defect label is applied when a pixel is assigned two different labels by separate postprocessing operations.

In the case of wood application, we have chosen precedence rules that are shown in Table 4.4. Whenever clear wood overlaps with other defect types, clear wood surrenders its pixel label at that specific point. In other words, defect layers have precedence over the clear wood layer. The first column and the last row of Table 4.4 indicate overlapping pixel layers. IntelliPost uses manually specified precedence table to resolve overlapping layer. This table can be modified and edited by the user based on the type of application and postprocessing requirement.

Table 4.4: Precedence rules for overlapping layers. The first column and last row of the table represent overlapping layer, the rest shows the winner when conflict happens.

Knot	Knot			
Split	Split	Split		
Decay	Decay	Knot	Split	
Bark	Bark	Bark	Bark	Bark
	Clear Wood	**Knot**	**Split**	**Decay**

4.5 Summary

This chapter has presented the architecture of the IntelliPost system. Both learn mode and run mode are similar in how the system components are used, except that the information flow changes direction. The learn mode architecture was designed to obtain a region's relevant information and a region refinement function associated with it. In the run mode, the system extracts the relevant information for each region, and searches for the best refinement function. Inferencing can be seen as search for optimal sequence of refinement operations. Because IntelliPost provides for interaction with a user, training speed is very important for interactivity. That was the main reason for selecting a decision tree based inferencing architecture.

Chapter 5

Results and Discussion

5.1 Overview

The previous chapter has presented the architecture of IntelliPost for both learn mode and run mode. The knowledge and information that are captured in the learn mode are used in run mode to perform similar postprocessing operations. This chapter presents the results of several tests of IntelliPost using a variety of CT/MRI images.

Because IntelliPost is the last component of an overall CT image segmentation system, its performance should be evaluated on the basis of segmentation alone. Although the segmentation problem has been studied for many years, there is no universally accepted method and/or framework to evaluate a segmentation algorithm's performance. Whereas much work as been devoted to the development of segmentation algorithms, considerably less effort has been put into performance evaluation. Many studies have been proposed to fill the void [90, 200, 7, 151, 89]. But it is still a research issue.

This chapter starts by describing the methodology that has been used for evaluating the

performance of IntelliPost. The third section presents information about CT/MRI datasets that have been used to evaluate the overall system. Two different datasets have been used: hardwood log datasets, and medical brain scan MRI/CT datasets. The forth section introduces segmentation performance metrics that were used to find quantitative results for segmentation. Section 5 and 6 provide quantitative results both for hardwood log datasets and for medical CT/MRI datasets. Section 7 concludes the chapter with a brief summary.

5.2 Methodology for Experiments

If we wish to create a postprocessing system for presegmented CT images in a manner similar to that of a human expert, then we should ask a human expert how he or she would analyze a given CT image. This process leads us to create manually segmented images for a given image.

Since we deal with the problem of image segmentation, we need to have a reference image (i.e., ground truth) with which to compare the end result of a segmentation algorithm. For medical image applications, we obtained ground truth information for a publicly available MRI dataset. The corresponding ground truth for a medical CT dataset was not available therefore ground truth images are generated manually. The application of CT imaging for forest products industry is relatively new. Because of that, no publicly available log CT scan datasets with ground truth segmentation were available.

5.2.1 Experimental Evaluation

An overview of the experimental setup is illustrated in Figure 5.1. The input image is processed by an initial segmentation algorithm. The output is a presegmented slice that

needs refinement. Segmentation measures that will be introduced in the next section will be obtained "before" and "after" postprocessing. We will use a reference image for comparison of a presegmented image and the corresponding result image. The segmentation performance measure will be obtained by using these two image pairs. One can see that the reference image play a very crucial role on this experiment. Two USDA Forest Service researchers from the Brooks Forest Products Center of Virginia Tech delineated the boundaries of defects on given CT images for the generation of ground truth datasets.

The ground truth images for the medical datasets were not available for the CT brain scan dataset. We had to follow the same procedure as in the case of log datasets, and performed segmentation manually. In both datasets, the outlined CT images have been used as reference images for comparing with the final segmentation of the system as illustrated in Figure 5.1. For the MRI dataset, the provided ground truth was used for the evaluation.

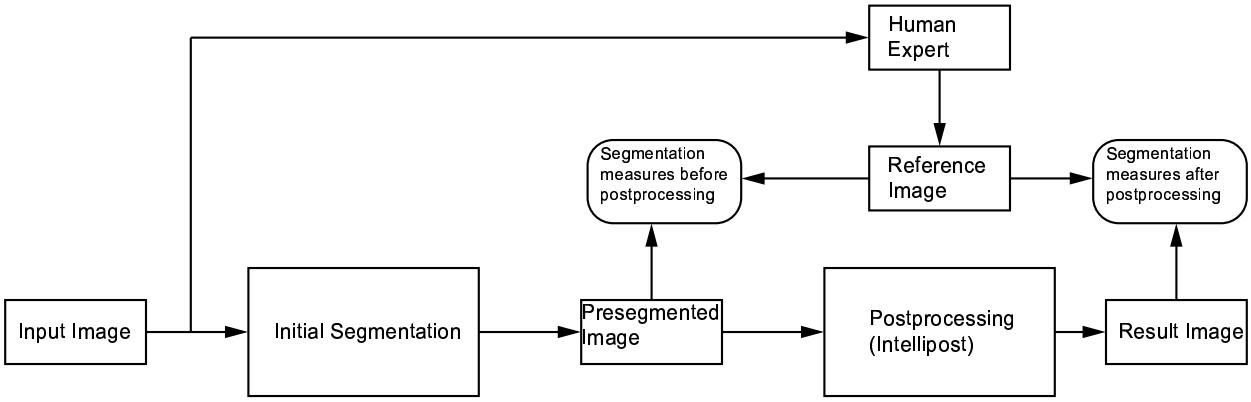


Figure 5.1: Comparing segmentation improvement between presegmented image and the result image. A reference image is generated by a human expert for obtaining segmentation performance measures.

5.2.2 Generation of Ground Truth

In order to obtain a ground truth reference for hardwood log CT image datasets, we have used an off-the-shelf graphical program that lets the user draw an outline of defects on a displayed

CT image. This is saved as a ground truth reference image. We have used Macromedia's Fireworks application for this purpose. This application was specifically designed to create vector graphics for internet applications. It provides easy-to-use contour outlining tools. A typical screenshot of this application is shown in Figure 5.2. Macromedia's Fireworks provides layer functionality to work with different defect layers. The user can enable and disable those layers or control transparency.

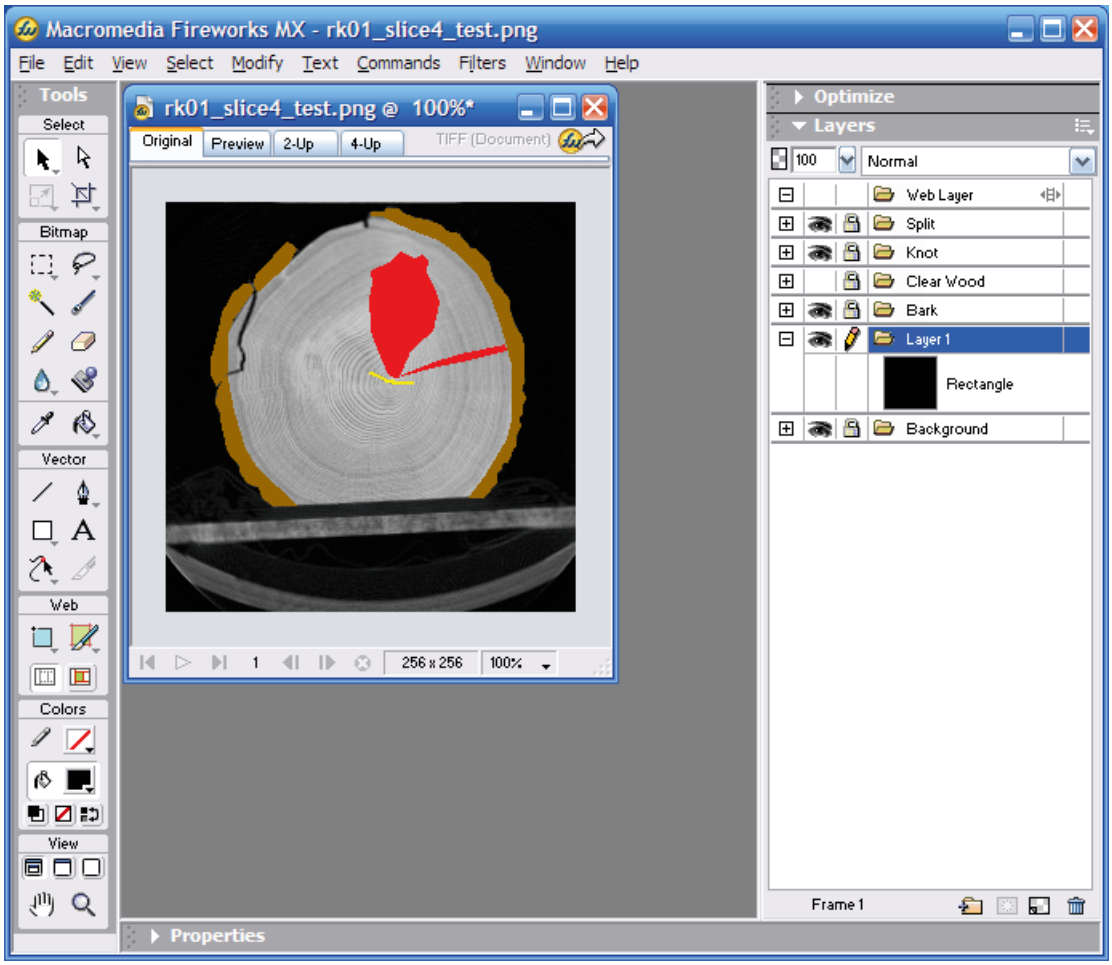


Figure 5.2: Macromedia's Fireworks screenshot for outlining defect's boundaries.

5.3 CT/MRI Image Datasets

We have tested IntelliPost with two different dataset groups: several hardwood log CT image datasets, and medical CT and MRI head scan datasets. The hardwood log CT image datasets were our primary focus in this study. The medical datasets were used to evaluate IntelliPost in biomedical imaging. For the medical dataset, we have used the BrainWeb project dataset which is generated by a MRI simulator. Due to difficulties in obtaining medical datasets for research use, we have chosen the simulation approach.

5.3.1 Hardwood Log Data Sets

Hardwood log CT image datasets were obtained from several sources as listed in Table 5.1. The datasets RK01 and RK12 were obtained from GE Medical when the project was started. The datasets known as 2048, 2049, 2051, and 5357 were obtained from a CT scanner at the Virginia-Maryland Regional College of Veterinary Medicine. The remaining datasets were provided by Forintek Canada Inc.

In Table 5.1, 'Study Number' indicates a unique code that helps to identify each log section. 'Number of Slices' indicates how many images are in that log dataset. 'Slice Thickness' shows spatial distance between voxel centers in two consecutive slices for the cases of RK01 and RK12 datasets, and spatial thickness of CT images (i.e, the voxel size in z direction) for the remaining datasets. Finally 'CT Image Size' shows the number of rows and columns of each CT slice. In each study, we have partitioned the dataset into two groups: a training dataset, and a testing dataset.

Table 5.1: Summary of hardwood log datasets.

Study Number	Data Source	Species	Number of Slices	Slice Thickness	CT Image Size	Pixel Size
rk01	GE Medical	Red Oak	45	1 mm	256×256	1 mm × 1 mm
rk12	GE Medical	Red Oak	10	2.5 mm	256×256	2.5 mm × 2.5 mm
2048	Veterinary Medicine	Red Oak	18	10 mm	512×512	0.93 mm × 0.93 mm
2049	Veterinary Medicine	Red Oak	23	10 mm	512×512	0.93 mm × 0.93 mm
2051	Veterinary Medicine	Red Oak	17	10 mm	512×512	0.93 mm × 0.93 mm
5357	Veterinary Medicine	Red Oak	36	10 mm	512×512	0.93 mm × 0.93 mm
567b	Forintek	Sugar Maple	103	10 mm	512×512	0.65 mm × 0.65 mm
578a	Forintek	Sugar Maple	97	10 mm	512×512	0.65 mm × 0.65 mm
bille3-1	Forintek	Sugar Maple	89	10 mm	512×512	0.93 mm × 0.93 mm
bille3-2	Forintek	Sugar Maple	100	10 mm	512×512	0.93 mm × 0.93 mm
bille4-1	Forintek	Sugar Maple	55	10 mm	512×512	0.96 mm × 0.96 mm
bille5-1	Forintek	Sugar Maple	105	10 mm	512×512	0.93 mm × 0.93 mm
bille6-1	Forintek	Sugar Maple	100	10 mm	512×512	0.93 mm × 0.93 mm
bille6-2	Forintek	Sugar Maple	97	10 mm	512×512	0.93 mm × 0.93 mm
bille7-2	Forintek	Sugar Maple	100	10 mm	512×512	0.72 mm × 0.72 mm

5.3.2 Medical Brain CT/MRI Scan Data Sets

We have used two medical image datasets to test IntelliPost's performance in other application domain. The CT dataset was obtained from the Washington University School of Medicine in St. Louis, Missouri which was available for research and instructional purposes. An MRI dataset was obtained from BrainWeb project web site[1], along with its ground truth. The two datasets are summarized in Table 5.2. To test the system, we have split the datasets equally: training and testing set. In each case, we have used the training set to train both the ANN and IntelliPost. After training step was complete, images from test sets were segmented by the ANN. Then segmented images were postprocessed by IntelliPost. Postprocessed images were compared against ground truth images to obtain quantitative results for segmentation improvement.

[1]BrainWeb Project Web Site: http://www.bic.mni.mcgill.ca/brainweb/

Table 5.2: Summary of medical CT and MRI datasets.

Study Number	Data Source	Number of Slices	Slice Thickness	CT/MRI Image Size	Voxel Size
algotech-23 (CT)	Univ. of Washington School of Medicine	163	1.30 mm	340×340	0.61 mm × 0.61 mm
BrainWeb MRI Dataset	McGill University	181	1 mm	181×217	1 mm × 1 mm

5.4 Segmentation Performance Metrics

5.4.1 Overview

Evaluation of segmentation performance is difficult because this tends to vary based on the application domain [80]. Performance measures have been described in a few studies [198, 103, 90, 7, 151, 89], but this is still an open research issue and there is no widely accepted framework for evaluation of segmentation algorithms.

Several evaluation and comparison methods have been proposed in different research areas: edge detection schemes [54], global thresholding methods [102], optical flow estimation and segmentation [9], stereo matching [21], and shape from shading [199]. Generally, attempts are divided into two major categories: analytical and empirical methods [200]. Analytical methods look at an algorithm's principles and properties. Empirical methods indirectly evaluate segmentation algorithms by measuring the quality of segmentation results. Zhang [200] divides empirical methods into "goodness" methods and "discrepancy" methods. Goodness methods look for some desirable properties of segmented images, such as intra-region uniformity [188], inter-region contrast [103], and region shape [156]. On the other hands, discrepancy methods measure differences to a predefined reference segmentation (ground truth). Quantitative discrepancy measures could be further subdivided into two groups: image based discrepancy metrics, and region based discrepancy metrics.

5.4.2 Confusion Matrix Analysis

The most common empirical discrepancy analysis method is probably confusion matrix analysis [198]. This method presents quantitative discrepancy results between empirical classification results and the ideal case.

The size of a confusion matrix is determined by the number of classes in a classification problem. A confusion matrix C of dimension $N \times N$ can be constructed as:

$$C = \begin{bmatrix} c_{11} & \cdots & c_{1N} \\ \vdots & c_{ij} & \vdots \\ c_{N1} & \cdots & c_{NN} \end{bmatrix} \tag{5.1}$$

where rows of the matrix represent classes for ideal case, and columns of the matrix represent the predicted class. Each entry c_{ij} represents the number of class i samples that were classified as class j by the classification algorithm. The diagonal of the matrix represents the number of correct classifications, and off-diagonal elements represent the number of samples that are incorrectly classified. For an ideal classifier, the matrix is diagonal.

Three error measures can be defined for each class k, which can all be used to describe class detection accuracy of a segmentation algorithm. The false negative rate is the proportion of class k samples that are incorrectly classified. This is known as Type I error in the statistical literature. The false negative rate is defined as:

$$M_I^k = \frac{(\sum_{i=1}^{N} C_{ki}) - C_{kk}}{\sum_{i=1}^{N} C_{ki}} \tag{5.2}$$

where the numerator represents the number of samples of class k that are incorrectly classified, and the denominator represents the total number of samples of class k.

In a similar way, the false positive rate, which is known as a Type II error, is defined as:

$$M_{II}^{k} = \frac{(\sum_{i=1}^{N} C_{ki}) - C_{kk}}{(\sum_{i=1}^{N} \sum_{j=1}^{N} C_{ij}) - \sum_{i=1}^{N} C_{ik}} \tag{5.3}$$

where the numerator represents the number of samples of other classes that are incorrectly classified as class k. The denominator is the total number of samples of other classes.

The true positive rate is the number of samples of class k that are correctly classified as class k. The true positive rate is defined as:

$$M_{III}^{k} = \frac{C_{kk}}{\sum_{i=1}^{N} C_{ki}} \tag{5.4}$$

where C_{kk} represents the number of samples that are correctly classified and the denominator represents the number of samples of class k.

Single summary statistics that can be derived from a multiclass confusion matrix analysis is defined as:

$$P = \frac{Tr(C)}{\sum_{ij} C_{ij}} \tag{5.5}$$

where $Tr(.)$ denotes trace of a confusion matrix sum of diagonal elements. In Equation 5.2, Equation 5.3, Equation 5.4, and Equation 5.5, each class is weighted equally.

5.4.3 ROC Analysis

Receiver Operating Characteristics (ROC) analysis is a classical method in signal detection theory [70, 182, 177]. The use of ROC has been extended to statistics, to comparison of medical diagnostic techniques in medicine [77, 91, 31], and to machine learning as an alternative method for comparing learning systems [147]. It is gradually gaining popularity in the com-

puter vision and image analysis community for comparative evaluation of algorithms such as color models [5], edge detectors [1, 24], and appearance identification [52]. Receiver operating characteristics plots denotes a coordinate system used for visualizing the performance of a classifier, where the true positive rate (TPR) (sensitivity) is plotted on the vertical axis and the false positive rate (FPR) (1-specificity) on the horizontal axis. This allows for a 2-dimensional comparison of classifiers. ROC graphs separate classification performance from class distribution or error cost and they provide a valuable visualization technique for evaluating the behaviour of a classifier. Using ROC analysis, the operating point of a classifier can be selected to obtain a good trade-off of classification performance. In general, every classifier has some parameters to be adjusted, and by varying these parameters, classification performance is changed in which the operating condition is represented as a point in a ROC curve. By changing these parameters, a curve is produced, which illustrates the classification error trade offs or a trade-off between true positive rate and false positive rate.

Even though ROC analysis is intended for two-class classification problems, it can be easily extended to the multiclass case by separately taking each class as positive and the rest as the negative class.

5.4.4 Area Similarity Measure

Let A_S be a set of points on segmented image region, and A_G be a set of points on a ground truth region. A common method currently used in magnetic resonance image segmentation is comparing the area between two region R_S and R_G [7],

$$S_{area}(R_S, R_G) = \frac{2n(A_S \cap A_G)}{n(A_S) + n(A_G)} \tag{5.6}$$

where $n(A)$ represents the number of pixels in A. Based on observation in [7], $S_{area} > 0.7$ indicates good segmentation performance.

Another area based similarity measure that is commonly used in the literature is known as area overlapping index. The overlapping index is defined as:

$$overlap = \frac{n(GT \cap AS)}{n(GT \cup AS)} \tag{5.7}$$

where GT, and AS denote the size of region in a ground truth, automatic segmentation, respectively. $n(.)$ denotes the number of pixels. Overlapping index varies in the range of between 0 and 1. The lesser the less similar regions are. Ideally, if two regions are identical, overlapping index becomes one.

5.4.5 Shape Similarity Measure

The area similarity measure provides a quantitative results about how similar two regions in terms of their sizes (or areas) and their spatial locations are. But it may be less informative with respect to details on the shapes of the two contours. Two different pairs of regions may result in the same area similarity values. An example of this is illustrated in Figure 5.3. Recent work [143] has introduced another shape similarity measure, which is a modification of the chamfer matching method. This is based on earlier work by [36] and modified slightly.

Let C_S be a set of points on the contour of a region from a segmented image, and C_G be the set of points on the contour from a ground truth region. The goal is to obtain a similarity measure $S_{shape} \in [0, 1]$ that quantitatively measures the similarity between two contours. Obtaining the similarity measure requires several steps. First, the signed Euclidean distance

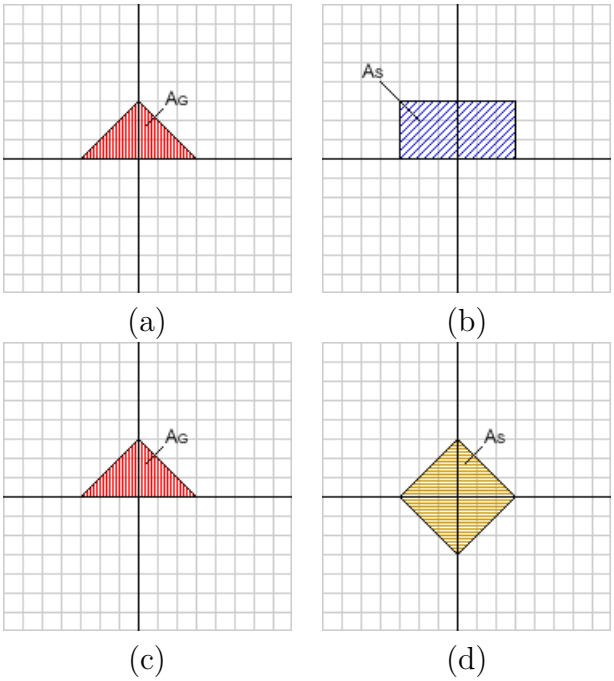


Figure 5.3: Two different regions that give the same area similarity measure with respect to ground truth. (a) Ground truth region. (b) Rectangular region that yields $S_{area} = \frac{2}{3}$. (c) Ground truth. (d) a different region that gives the same area similarity measure.

transform is applied to both contours. The signed distance transform is defined as:

$$D(q) = \begin{cases} -\min_{p \in C} \|p - q\| & \text{if } p \text{ is inside } C \\ \min_{p \in C} \|p - q\| & \text{if } p \text{ is outside } C \end{cases} \tag{5.8}$$

where q is a pixel coordinate in the image domain, and $p \in C$ represents a pixel on the contour C, and $\|\cdot\|$ is the Euclidean norm. By applying signed distance map to both contours, D_S and D_G distance maps are obtained respectively. Second, phase map calculation is performed as follows:

$$\Phi(x, y) = \tan^{-1} \frac{\nabla_y D(x, y)}{\nabla_x D(x, y)} \tag{5.9}$$

where $\nabla_x D(x, y)$ and $\nabla_y D(x, y)$ represents the x and y components of the gradient of the signed distance map $D(x, y)$, respectively. After phase transform is performed, we compute

normalized phase similarity between two contours as follows:

$$S_{phase} = \frac{|\Phi_S - \Phi_G - \pi|}{\pi} \tag{5.10}$$

where Φ_S and Φ_G are phase maps of contours C_S and C_G respectively. The value of S_{phase} lies in the range [0,1], where the value 1 indicates that both contours have the same phase and a value of 0 indicates the maximum phase difference of π. Finally the shape similarity measure is obtained by taking weighted sum of the phase similarity measure along C_S against C_G as:

$$S_{shape}(C_G, C_S) = \frac{1}{n(C_S)} \sum_{(x,y)\in C_S} \Gamma_G(x,y) \cdot S_{phase}(x,y) \tag{5.11}$$

where $n(C_S)$ indicates the number of pixels on the contour C_S, and $\Gamma_G(x,y)$ denotes weighting function that is derived from D_G as:

$$\Gamma_G(x,y) = \exp\{-\frac{D_G^2(x,y)}{\sigma^2}\} \tag{5.12}$$

where σ^2 is a positive constant.

5.5 Results for Hardwood Log Datasets

5.5.1 Confusion Matrix Analysis Results

This section describes the segmentation performance of IntelliPost on the CT log datasets. Each dataset has been divided into two equal-sized partitions: a training partition and a test partition. Each dataset' training partition was used to train the ANN module and IntelliPost independently of other training set. To train the ANN, the back-propagation algorithm was

applied using the entire training set. The ANN was slightly overtrained in each case to see the effect of postprocessing better.

Since the entire system was implemented in MATLAB, the training of the ANN system was performed in MATLAB environment as well. We used neural network toolbox to implement the ANN module. For all datasets, the size of sample window was kept the same which was 5×5. In back-propagation algorithm, the learning rate was set to 0.2 and momentum constant was set to 0.8 for all datasets. Those parameter did not change throughout the experiment. The goal for training was set to "mean square error" function and the value of mean square error was set to 0.02.

The trained ANN module was used to segment images from the other set in training partition. Segmented images was stored to train IntelliPost. A USDA Forest Service researcher from the Brooks Forest Product Center at Virginia Tech used IntelliPost to postprocess segmented images which were segmented by the ANN module. During the training of IntelliPost, he mostly used 'remove', 'smooth', 'enlarge', 'noop' operation to carry out refinement on segmented images. The 'merge' operation was rarely used due to limited case faced training dataset. He found out that 'remove' and 'smooth' operations the most commonly used operation than others. After training IntelliPost was complete, images from test partition were postprocessed by IntelliPost, and the resultant images were saved as postprocessed images. In each case, two images ("before" and "after" postprocessing) were compared against the corresponding ground truth image at the pixel level.

Out of this analysis, two confusion matrices have been obtained each image slice in the corresponding dataset. We have chosen one representative image slice per dataset to illustrate the performance of the system. The confusion matrices for the GE Medical and Veterinary Medicine datasets are listed in Table 5.3 through Table 5.14. As we see from those results, the overall segmentation accuracy for each defect class has been improved by IntelliPost in

general. Background class in confusion matrices were eliminated since background separation was performed in the preprocessing module. IntelliPost does not have any control over background thresholding process therefore including background class in confusion matrix does not give any relevant information regarding to segmentation improvement with respect to background.

In a few cases, the ANN does not detect a region where one exists. The corresponding ground truth image contains that region. This is known as undersegmentation, as we see in the 2048 dataset (see Figure 5.9). The knot defects in the 10 o'clock and 2 o'clock directions have not been detected by the ANN algorithm, whereas they appear in the corresponding ground truth image. IntelliPost cannot do anything in this situation because the image under analysis does not contain such regions. Such cases limit the ability of IntelliPost to improve segmentation performance.

The false positive and true positive rates were obtained using Equation 5.3 and Equation 5.4. The results are listed in Table 5.15 through Table 5.17. Intuitively we would like to see that true positive rates increase as well as false positive rates decrease. For cases of $\emptyset$ detected pixels and $\emptyset$ ground truth pixels, the true positive rate of 1 has been assumed. If a postprocessed image contained just one pixel of a defect class that does not exist in the ground truth image, then the true positive rate would be zero. As we can see from those results, the overall segmentation performance has been improved since the true positive rates increase, and false positive rates decrease in general.

Visual results for RK01, RK12, 2048, 2049, 2051, and 5357 are presented in Figure 5.7 through Figure 5.12. The result for RK12, which is shown in Figure 5.8, deserves additional discussion. This image has a large decay region that contains a large split defect. Because of the nature of split defects, individual pixels are often low enough in density to be classified as voids during the early background thresholding step. In a typical CT image, splits may

therefore appear as actual split defects, or as background pixels that are ignored during classification and postprocessing steps. In the figure, pixels classified as split defects are visible as a narrow, linear region with values near the low end of the clear wood density values. The physical split happens to be narrower that the size of a pixel, so the CT values represent an average density for the void and the surrounding wood.

The ANN segmentation module bas been trained to detect low-density, "subresolution" splits. However, large voids are eliminated by background thresholding and cannot be detected by the ANN segmentation module. A more complete analysis of split defects can be performed [159], but is outside the scope of this dissertation.

The reference image associated with this specific slice of RK12 was outlined in the light of the aforementioned limitation. Because the ANN module does not process the larger voids, these were also ignored by IntelliPost. The corresponding background pixels were ignored in the confusion matrix analysis.

In a similar way, the confusion matrices for the Forintek datasets are listed in Table 5.19 through Table 5.23. The number of defect types is more than the previous case. The improvement is not as dramatic as for the previous case, and sometimes the segmentation performance decreases (for example, the bille3-1 dataset). Their corresponding true positive and false positive rates are listed in Table 5.25 through Table 5.27. Visual results showed (see Figure 5.13 through Figure 5.15) that IntelliPost provided less improvement for the Forintek datasets.

Overall correct segmentation rates are listed in Table 5.29 and Table 5.30 for hardwood log datasets. The results indicates that IntelliPost improves overall segmentation accuracy. The average segmentation improvement was 1.92% for RK01, RK12, 2048, 2049, 2051, and 5357 datasets. For the case of Forintek dataset, the average improvement was 9.45% for the

selected slices in the dataset. Postprocessed region size can have a large effect on the single segmentation accuracy metric (Equation 5.5). This can occur, for example, if a big region is initially segmented incorrectly, but is then corrected by IntelliPost. An example of this is shown in the case of 567b of the Forintek dataset, where IntelliPost changes a heartwood region to sapwood by merging those pixels into a larger sapwood region. Such postprocessing operations improves segmentation accuracy dramatically.

Confusion matrix analysis did not consider a separate "background" class, since segmentation of background is controlled by Otsu thresholding in the preprocessing module. IntelliPost had no ability to change the thresholding parameter. Confusion matrices were obtained through the comparison of foreground pixels of segmented images versus foreground pixels of the corresponding ground truth images.

Table 5.3: Confusion matrices for slice 3 from dataset RK01. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	KN	SP	DC	BR		CW	KN	SP	DC	BR
RK01	3	CW	21556	642	117	119	857	CW	21621	707	16	52	897
		KN	111	1336	0	0	19	KN	86	1365	0	0	15
		SP	17	0	41	0	0	SP	17	0	41	0	0
		DC	0	0	0	0	0	DC	0	0	0	0	0
		BR	1329	1	11	100	2300	BR	1295	0	0	0	2442

Table 5.4: Improvement in confusion matrix after postprocessing for slice 3 from dataset RK01. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	**KN**	**SP**	**DC**	**BR**
RK01	3	**CW**	65	65	-101	-67	40
		KN	-25	29	0	0	-4
		SP	0	0	0	0	0
		DC	0	0	0	0	0
		BR	-34	-1	-11	-95	142

Table 5.5: Confusion matrices for slice 5 from dataset RK12. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	**KN**	**SP**	**DC**	**BR**		**CW**	**KN**	**SP**	**DC**	**BR**
RK12	5	**CW**	17381	367	30	366	644	**CW**	17448	346	0	319	675
		KN	266	54	35	21	0	**KN**	362	0	0	14	0
		SP	0	0	0	0	0	**SP**	0	0	0	0	0
		DC	718	0	240	7180	9	**DC**	817	0	73	7257	0
		BR	365	9	58	91	2762	**BR**	282	0	0	18	2985

Table 5.6: Improvement in confusion matrix after postprocessing for slice 5 from dataset RK12. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	**KN**	**SP**	**DC**	**BR**
RK12	5	**CW**	67	-21	-30	-47	31
		KN	96	-54	-35	-7	0
		SP	0	0	0	0	0
		DC	99	0	-167	77	-9
		BR	-83	-9	-58	-73	223

Table 5.7: Confusion matrices for slice 15 from dataset 2048. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	KN	SP	DC	BR		CW	KN	SP	DC	BR
2048	15	CW	85928	2951	0	20	81	CW	85881	3067	0	0	32
		KN	2481	5714	0	0	65	KN	2546	5714	0	0	0
		SP	0	0	0	3	0	SP	0	0	0	0	0
		DC	0	0	0	0	0	DC	0	0	0	0	0
		BR	4411	0	0	917	11624	BR	4127	0	0	0	12718

Table 5.8: Improvement in confusion after postprocessing for slice 15 from dataset 2048. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	KN	SP	DC	BR
2048	15	CW	-47	116	0	-20	-49
		KN	65	0	0	0	-65
		SP	0	0	0	-3	0
		DC	0	0	0	0	0
		BR	-284	0	0	-917	1094

Table 5.9: Confusion matrices for slice 5 from dataset 2049. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	KN	SP	DC	BR		CW	KN	SP	DC	BR
2049	5	CW	63050	3897	0	24	400	CW	63292	3794	0	0	285
		KN	439	2345	0	0	25	KN	437	2372	0	0	0
		SP	0	0	0	0	0	SP	0	0	0	0	0
		DC	0	0	0	0	0	DC	0	0	0	0	0
		BR	2455	1	0	562	8384	BR	2093	0	0	0	9309

Table 5.10: Improvement in confusion matrix after postprocessing for slice 5 from dataset 2049. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	KN	SP	DC	BR
2049	5	CW	242	-103	0	-24	-115
		KN	-2	27	0	0	-25
		SP	0	0	0	0	0
		DC	0	0	0	0	0
		BR	-362	-1	0	-562	925

Table 5.11: Confusion matrices for slice 1 from dataset 2051. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	KN	SP	DC	BR		CW	KN	SP	DC	BR
2051	1	CW	53092	548	0	31	691	CW	54221	80	0	0	61
		KN	783	2111	0	0	79	KN	867	2106	0	0	0
		SP	0	0	0	0	0	SP	0	0	0	0	0
		DC	0	0	0	0	0	DC	0	0	0	0	0
		BR	1985	0	0	581	6936	BR	1600	0	0	0	7902

Table 5.12: Improvement in confusion matrix after postprocessing for slice 1 from dataset 2051. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	KN	SP	DC	BR
2051	1	CW	1129	-468	0	-31	-630
		KN	84	-5	0	0	-79
		SP	0	0	0	0	0
		DC	0	0	0	0	0
		BR	-385	0	0	-581	966

Table 5.13: Confusion matrices for slice 3 from dataset 5357. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing Confusion Matrix						After Postprocessing Confusion Matrix					
			CW	KN	SP	DC	BR		CW	KN	SP	DC	BR
5357	3	CW	36591	451	125	0	1232	CW	37445	424	12	0	518
		KN	296	3424	1	0	270	KN	322	3430	0	0	239
		SP	27	0	52	0	0	SP	27	0	52	0	0
		DC	0	0	0	0	0	DC	0	0	0	0	0
		BR	954	4	0	0	7900	BR	198	1	0	0	8659

Table 5.14: Improvement in confusion matrix after postprocessing for slice 3 from dataset 5357. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix					
			CW	KN	SP	DC	BR
5357	3	CW	854	-27	-113	0	-714
		KN	26	6	-1	0	-31
		SP	0	0	0	0	0
		DC	0	0	0	0	0
		BR	-756	-3	0	0	759

Table 5.15: True positive rates for selected images in datasets RK01, RK12, 2048, 2049, 2051, and 5357. The true positive rates generally increase for each class after postprocessing. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing					After Postprocessing				
		True Positive Rate					True Positive Rate				
		CW	KN	SP	DC	BR	CW	KN	SP	DC	BR
RK01	3	0.9255	0.9113	0.7069	1.0000	0.6148	0.9282	0.9311	0.7069	1.0000	0.6526
RK12	5	0.9251	0.1436	1.0000	0.8813	0.8408	0.9287	0.0000	1.0000	0.8908	0.9087
2048	15	0.9657	0.6918	0.0000	1.0000	0.6857	0.9652	0.6918	1.0000	1.0000	0.7550
2049	5	0.9359	0.8348	1.0000	1.0000	0.7353	0.9395	0.8444	1.0000	1.0000	0.8164
2051	1	0.9766	0.7101	1.0000	1.0000	0.7300	0.9974	0.7084	1.0000	1.0000	0.8316
5357	3	0.9529	0.8579	0.6582	1.0000	0.8918	0.9752	0.8594	0.6582	1.0000	0.9775

Table 5.16: Improvement in true positive rates (TPR) for selected images in datasets RK01, RK12, 2048, 2049, 2051, and 5357. Positive numbers indicate improvement by IntelliPost.

Dataset	Slice Number	Improvement in TPR				
		CW	KN	SP	DC	BR
RK01	3	0.0027	0.0198	0.0000	0.0000	0.0378
RK12	5	0.0036	-0.1436	0.0000	-0.0733	0.0679
2048	15	-0.0005	0.0000	1.0000	0.0000	0.0693
2049	5	0.0036	0.0096	0.0000	0.0000	0.0811
2051	1	0.0208	-0.0017	0.0000	0.0000	0.1016
5357	3	0.0223	0.0015	0.0000	0.0000	0.0857

Table 5.17: False positive rates for selected images in datasets RK01, RK12, 2048, 2049, 2051, and 5357. The false positive rates decrease after postprocessing, as desired. There are five classes: clear wood (CW), knot (KN), split (SP), decay (DC), and bark (BR).

Dataset	Slice Number	Before Postprocessing					After Postprocessing				
		False Positive Rate					False Positive Rate				
		CW	KN	SP	DC	BR	CW	KN	SP	DC	BR
RK01	3	0.3295	0.0048	0.0006	0.0000	0.0581	0.3175	0.0037	0.0006	0.0000	0.0524
RK12	5	0.1192	0.0107	0.0000	0.0431	0.0191	0.1135	0.0124	0.0000	0.0396	0.0110
2048	15	0.1210	0.0240	0.0000	0.0000	0.0548	0.1234	0.0241	0.0000	0.0000	0.0424
2049	5	0.3041	0.0059	0.0000	0.0000	0.0430	0.2870	0.0055	0.0000	0.0000	0.0298
2051	1	0.1018	0.0135	0.0000	0.0000	0.0448	0.0113	0.0136	0.0000	0.0000	0.0279
5357	3	0.1399	0.0120	0.0005	0.0000	0.0226	0.0738	0.0119	0.0005	0.0000	0.0047

Table 5.18: Increase in false positive rates (FPR) for selected images in datasets RK01, RK12, 2048, 2049, 2051, and 5357. Negative numbers are desired, and indicate improvement by IntelliPost.

Dataset	Slice Number	Improvement in FPR				
		CW	KN	SP	DC	BR
RK01	3	-0.0120	-0.0011	0.0000	0.0000	-0.0057
RK12	5	-0.0057	0.0017	0.0000	-0.0035	-0.0081
2048	15	0.0024	0.0001	0.0000	0.0000	-0.0124
2049	5	-0.0171	-0.0004	0.0000	0.0000	-0.0132
2051	1	-0.0905	0.0001	0.0000	0.0000	-0.0169
5357	3	-0.0661	-0.0001	0.0000	0.0000	-0.0179

Table 5.19: Confusion matrices for slice 7370557 from Forintek dataset bille3-1. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are eight classes: knot (KN), live bark (LB), decay (DC), split (SP), sapwood (SW), hardwood (HW), dead knot (DK), and dead bark (DB).

Dataset	Slice Number	Before Postprocessing Confusion Matrix

	KN	LB	DC	SP	SW	HW	DK	DB
KN	1936	1317	0	0	144	5	0	0
LB	13	7323	0	0	848	0	0	566
DC	0	0	1261	28	0	79	0	0
SP	0	0	0	0	0	0	0	0
SW	106	874	1189	136	89660	1185	0	173
HW	11	0	401	135	43	10575	0	0
DK	0	0	0	0	0	0	0	0
DB	0	76	0	0	370	0	0	5284

bille3-1	7370557	After Postprocessing Confusion Matrix

	KN	LB	DC	SP	SW	HW	DK	DB
KN	1518	0	0	0	1876	9	0	0
LB	0	7502	0	0	326	0	0	922
DC	0	0	1308	0	0	60	0	0
SP	0	0	0	0	0	0	0	0
SW	61	241	0	0	91305	1563	0	164
HW	0	0	311	0	72	10782	0	0
DK	0	0	0	0	0	0	0	0
DB	0	147	0	0	12	0	0	5570

Table 5.20: Improvement in confusion matrix after postprocessing for slice 7370557 from Forintek dataset bille3-1. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix							
			KN	**LB**	**DC**	**SP**	**SW**	**HW**	**DK**	**DB**
bille3-1	7370557	**KN**	-418	-1317	0	0	1732	4	0	0
		LB	-13	179	0	0	-522	0	0	356
		DC	0	0	47	-28	0	-19	0	0
		SP	0	0	0	0	0	0	0	0
		SW	-45	-633	-1189	-136	1645	378	0	-9
		HW	-11	0	-90	-135	29	207	0	0
		DK	0	0	0	0	0	0	0	0
		DB	0	71	0	0	-358	0	0	286

Table 5.21: Confusion matrices for slice 4127003 from Forintek dataset 567b. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are eight classes: knot (KN), live bark (LB), decay (DC), split (SP), sapwood (SW), hardwood (HW), dead knot (DK), and dead bark (DB). Every pixel in the ground truth image belongs to sapwood (SW) class as we see one row in confusion matrix.

Dataset	Slice Number	Before Postprocessing Confusion Matrix

	KN	LB	DC	SP	SW	HW	DK	DB
KN	0	0	0	0	0	0	0	0
LB	0	0	0	0	0	0	0	0
DC	0	0	0	0	0	0	0	0
SP	0	0	0	0	0	0	0	0
SW	708	4935	2792	3374	126995	13193	0	2786
HW	0	0	0	0	0	0	0	0
DK	0	0	0	0	0	0	0	0
DB	0	0	0	0	0	0	0	0

Dataset	Slice Number	After Postprocessing Confusion Matrix

	KN	LB	DC	SP	SW	HW	DK	DB
KN	0	0	0	0	0	0	0	0
LB	0	0	0	0	0	0	0	0
DC	0	0	0	0	0	0	0	0
SP	0	0	0	0	0	0	0	0
SW	0	237	0	0	151735	0	0	2805
HW	0	0	0	0	0	0	0	0
DK	0	0	0	0	0	0	0	0
DB	0	0	0	0	0	0	0	0

Table 5.22: Improvement in confusion matrix after postprocessing for slice 4127003 from Forintek dataset 567b. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix							
			KN	**LB**	**DC**	**SP**	**SW**	**HW**	**DK**	**DB**
567b	4127003	**KN**	0	0	0	0	0	0	0	0
		LB	0	0	0	0	0	0	0	0
		DC	0	0	0	0	0	0	0	0
		SP	0	0	0	0	0	0	0	0
		SW	-708	-4698	-2792	-3374	24740	-13193	0	19
		HW	0	0	0	0	0	0	0	0
		DK	0	0	0	0	0	0	0	0
		DB	0	0	0	0	0	0	0	0

Table 5.23: Confusion matrices for slice 4133900 from Forintek dataset 578a. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are eight classes: knot (KN), live bark (LB), decay (DC), split (SP), sapwood (SW), hardwood (HW), dead knot (DK), and dead bark (DB).

Dataset	Slice Number	Before Postprocessing Confusion Matrix							
			KN	LB	DC	SP	SW	HW	DK	DB
578a	4133900	KN	0	0	0	0	0	0	0	0
		LB	0	0	0	0	0	0	0	0
		DC	0	0	3536	429	9	250	0	0
		SP	0	0	0	0	0	0	0	0
		SW	150	95	4096	529	129862	1135	0	3827
		HW	190	0	326	368	78	6515	0	0
		DK	0	0	0	0	0	0	0	0
		DB	0	0	0	0	0	0	0	0
		After Postprocessing Confusion Matrix								
578a	4133900		KN	LB	DC	SP	SW	HW	DK	DB
		KN	0	0	0	0	0	0	0	0
		LB	0	0	0	0	0	0	0	0
		DC	0	0	3981	2	0	241	0	0
		SP	0	0	0	0	0	0	0	0
		SW	0	0	35	0	139206	1277	0	46
		HW	0	0	514	0	115	6848	0	0
		DK	0	0	0	0	0	0	0	0
		DB	0	0	0	0	0	0	0	0

Table 5.24: Improvement in confusion matrix after postprocessing for slice 4133900 from Forintek dataset 578a. The difference between postprocessed and initial segmentation as compared with ground truth. In most cases, diagonal elements increase and off-diagonal elements decrease as desired.

Dataset	Slice Number	Improvement in Confusion Matrix							
			KN	LB	DC	SP	SW	HW	DK	DB
578a	4133900	KN	0	0	0	0	0	0	0	0
		LB	0	0	0	0	0	0	0	0
		DC	0	0	445	-427	-9	-9	0	0
		SP	0	0	0	0	0	0	0	0
		SW	-150	-95	-4061	-529	9344	142	0	-3781
		HW	-190	0	188	-368	37	333	0	0
		DK	0	0	0	0	0	0	0	0
		DB	0	0	0	0	0	0	0	0

Table 5.25: True positive rates for Forintek datasets. Overall true positive rates increase except knot (KN) class for bille3-1. There are eight classes: knot (KN), live bark (LB), decay (DC), split (SP), sapwood (SW), hardwood (HW), dead knot (DK), and dead bark (DB).

Dataset	Slice Number	Before Postprocessing							
		True Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	0.5691	0.8369	0.9218	1.0000	0.9607	0.9472	1.0000	0.9222
567b	4127003	1.0000	1.0000	1.0000	1.0000	0.8205	1.0000	1.0000	1.0000
578a	4133900	1.0000	1.0000	0.8371	1.0000	0.9296	0.8713	1.0000	1.0000
		After Postprocessing							
		True Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	0.4461	0.8574	0.9561	1.0000	0.9783	0.9657	1.0000	0.9722
567b	4127003	1.0000	1.0000	1.0000	1.0000	0.9803	1.0000	1.0000	1.0000
578a	4133900	1.0000	1.0000	0.9425	1.0000	0.9903	0.9159	1.0000	1.0000

Table 5.26: Improvement in true positive rates for Forintek datasets. Increase in true positive rates is desired therefore positive numbers show improvement in segmentation but negative numbers indicate decrease in true positive detection rates.

Dataset	Slice Number	Improvement in True Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	-0.1230	0.0205	0.0343	0.0000	0.0176	0.0185	0.0000	0.0500
567b	4127003	0.0000	0.0000	0.0000	0.0000	0.1598	0.0000	0.0000	0.0000
578a	4133900	0.0000	0.0000	0.1054	0.0000	0.0607	0.0446	0.0000	0.0000

Table 5.27: False positive rates for Forintek dataset. NAN represents "not a number". Since denominator of false positive rates represents the total number of other classes, which is zero, we can not find a numeric value for that specific case.

Dataset	Slice Number	Before Postprocessing							
		False Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	0.0122	0.0124	0.0009	0.0000	0.1204	0.0052	0.0000	0.0038
567b	4127003	0.0000	0.0000	0.0000	0.0000	NAN	0.0000	0.0000	0.0000
578a	4133900	0.0000	0.0000	0.0047	0.0000	0.8403	0.0067	0.0000	0.0000
		After Postprocessing							
		False Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	0.0157	0.0109	0.0005	0.0000	0.0667	0.0034	0.0000	0.0013
567b	4127003	0.0000	0.0009	0.0000	0.0000	NAN	0.0000	0.0000	0.0138
578a	4133900	0.0000	0.0000	0.0016	0.0000	0.1161	0.0043	0.0000	0.0000

Table 5.28: Improvement in false positive rates for Forintek datasets. Decrease in false positive rates is desired therefore we would like to see negative numbers as improvement but positive numbers show decrease in segmentation performance. NAN represents "not a number". Since denominator of false positive rates represents the total number of other classes, which is zero, we can not find a numeric value for that specific case.

Dataset	Slice Number	Improvement in False Positive Rate							
		KN	LB	DC	SP	SW	HW	DK	DB
BILLE3-1	7370557	0.0035	-0.0015	-0.0004	0.0000	-0.0537	-0.0018	0.0000	-0.0025
567b	4127003	0.0000	0.0000	0.0000	0.0000	NAN	0.0000	0.0000	0.0000
578a	4133900	0.0000	0.0000	-0.0031	0.0000	-0.7242	-0.0024	0.0000	0.0000

Table 5.29: Overall correct segmentation rates for selected images in datasets RK01, RK12, 2048, 2049, 2051, and 5357.

Dataset	Slice Number	Before Postprocessing	After Postprocessing	Improvement in segmentation
		Overall Correct Segmentation Rates	Overall Correct Segmentation Rates	Percent
RK01	3	0.8836	0.8918	0.93 %
RK12	5	0.8948	0.9050	1.14 %
2048	15	0.9043	0.9143	1.10 %
2049	5	0.9044	0.9190	1.61 %
2051	1	0.9297	0.9610	3.37 %
5357	3	0.9345	0.9660	3.37 %

Table 5.30: Overall correct segmentation rates for selected images in datasets Forintek.

Dataset	Slice Number	Before Postprocessing	After Postprocessing	Improvement in segmentation
		Overall Correct Segmentation Rates	Overall Correct Segmentation Rates	Percent
BILLE3-1	7370557	0.9378	0.9534	1.66 %
567b	4127003	0.8205	0.9803	19.47 %
578a	4133900	0.9189	0.9854	7.24 %

5.5.2 ROC Analysis

Because IntelliPost is a learning system that learns postprocessing rules from a user, we would like to evaluate IntelliPost's learning capability through ROC analysis. In this experiment, we used the RK01 dataset because we have obtained 13 slices as ground truth for this dataset. We have used slices that were not used for training IntelliPost.

The experiment contains seven steps. For each step, the number of postprocessing operations was incrementally increased to test how quickly the system learns desired postprocessing

rules from observation of human segmentation activities. Table 5.31 shows which slices have been used in this step and number of operations that were performed to postprocess the corresponding slice set. For each step, the training set is a superset of the previous step. After each training step completed, IntelliPost was tested with slices which have the corresponding ground truth slices for comparison. All confusion matrices for each step and their corresponding true positive and false positive rates were obtained (see Table 5.32). ROC curves for each defect class are illustrated in Figure 5.4 through Figure 5.6.

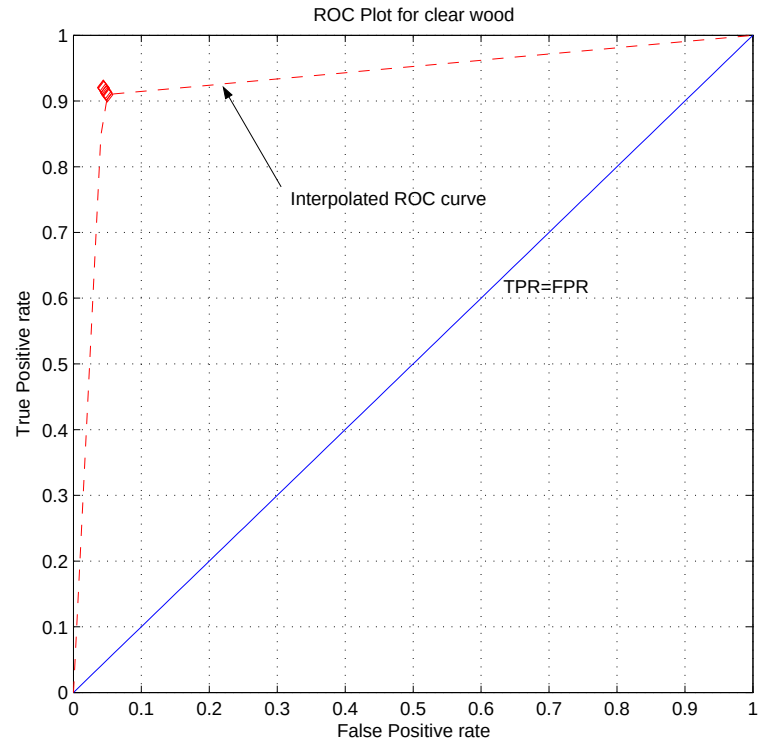
The results show that IntelliPost gradually learns. The ROC graph of clear wood (see Figure 5.4) shows that clear wood type is not sensitive to postprocessing steps. The other defect types show the similar results as clear wood but the split class is the most sensitive to the number of postprocessing operation steps since the number of pixels are few compares to other classes. The true positive and false positive rates put emphasis on the number of pixels therefore split type is the most sensitive among the other classes. The problem with the ROC analysis is that it takes the total error made by a classifier, from each class weighted equally. When the classes are not balanced, as in case of split versus other, the dominant class is easier to detect, and performance reported by ROC analysis can be exaggerated by unbalanced class distribution.

Table 5.31: Summary of training steps for ROC analysis. Slice number indicates which slices were used in the corresponding training step. The last column shows the number of postprocessing operations that were performed in that step. It is monotonically increasing.

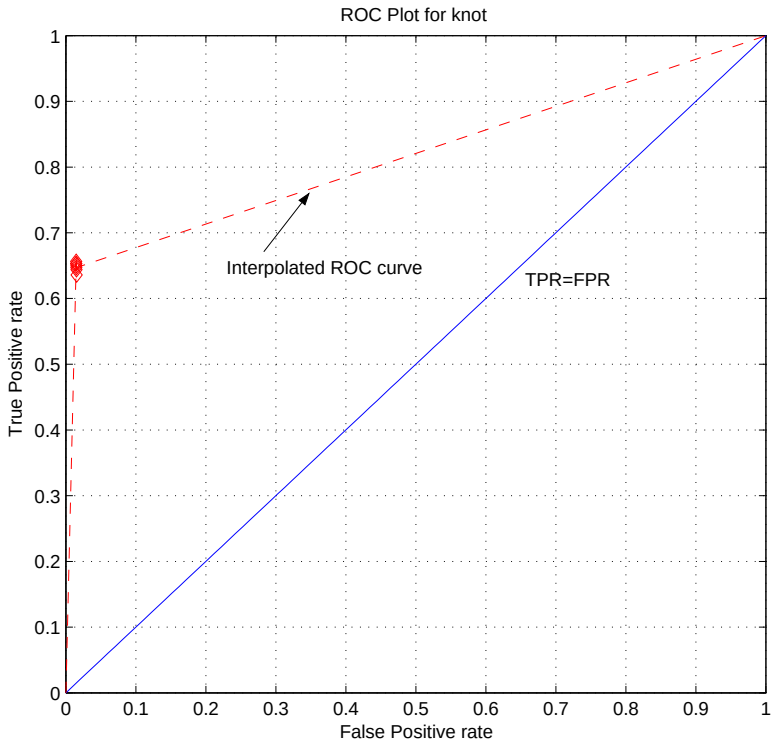
Training Step	Slice number used	The number of postprocessing operations
1	5,16,27,37	48 Operations
2	8,17,28,38	54 Operations
3	9,19,30,39	60 Operations
4	11,21,32,40	67 Operations
5	12,22,33,41	74 Operations
6	13,23,35,42	79 Operations
7	14,25,36,43	85 Operations

Table 5.32: True positive rates (TPR) and false positive rates (FPR) are listed for each defect type.

Step Number	CW		KN		SP		DC		BR	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
1	0.9098	0.0499	0.6531	0.0148	0.4752	0.0006	0.9231	0.0003	0.7140	0.0113
2	0.9149	0.0472	0.6358	0.0155	0.5025	0.0005	0.9231	0.0003	0.7057	0.0117
3	0.9123	0.0485	0.6564	0.0147	0.6564	0.0006	0.9231	0.0003	0.7197	0.0112
4	0.9201	0.0443	0.6498	0.0151	0.4452	0.0006	0.9231	0.0003	0.6706	0.0126
5	0.9190	0.0448	0.6438	0.0152	0.5077	0.0005	0.9231	0.0003	0.6763	0.0125
6	0.9121	0.0487	0.6528	0.0148	0.5077	0.0005	0.9231	0.0003	0.6922	0.0119
7	0.9207	0.0439	0.6470	0.0150	0.5106	0.0005	0.9231	0.0003	0.6666	0.0128

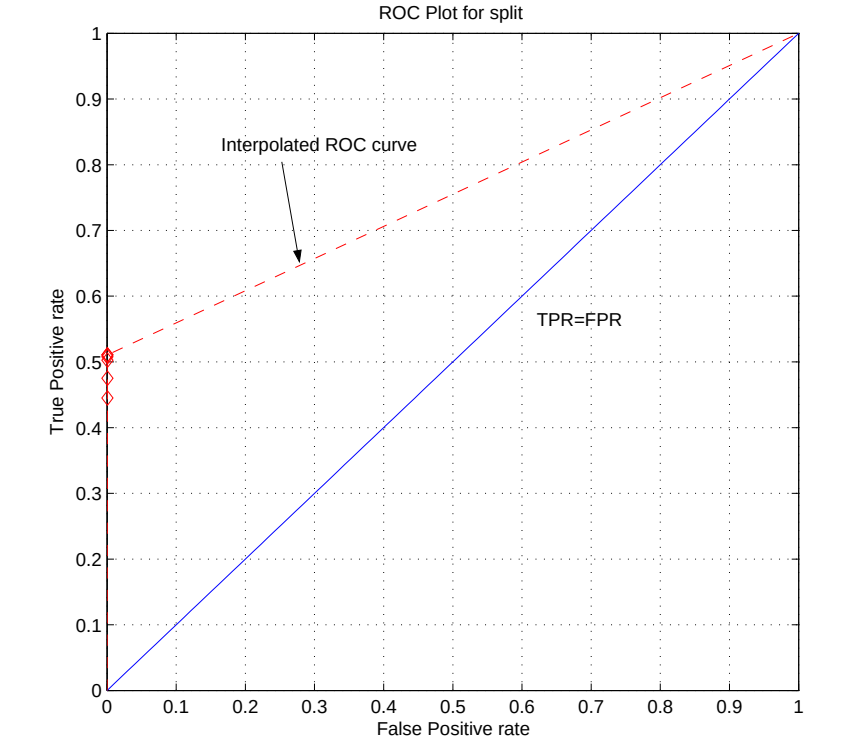


(a)

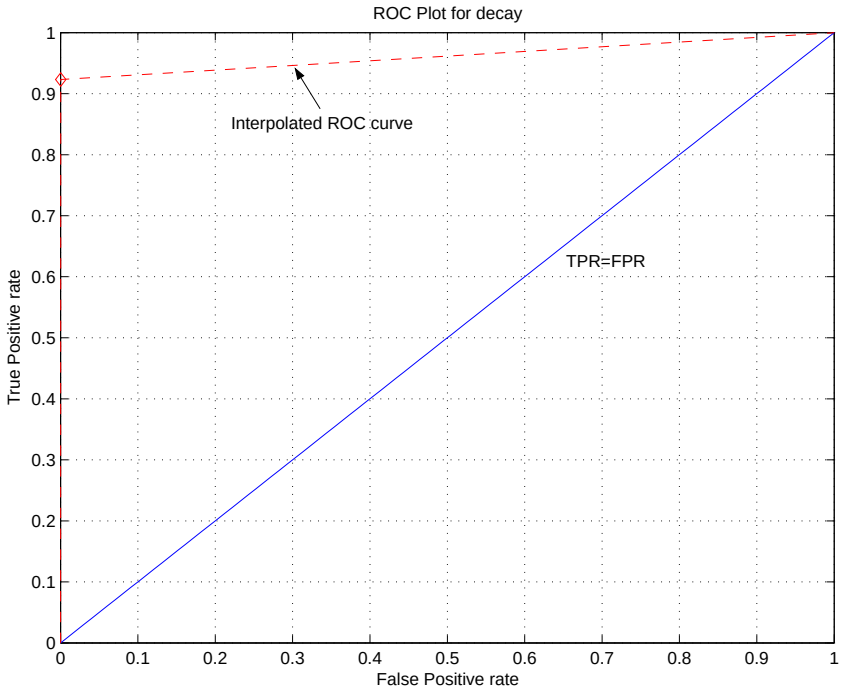


(b)

Figure 5.4: ROC points for each defect type. (a) ROC points for clear wood. (b) ROC points for knot. Each point represents one training step and the corresponding classification performance.



(a)



(b)

Figure 5.5: ROC points for each defect type. (a) ROC points for split. (b) ROC points for decay. Each point represents one training step and the corresponding classification performance.

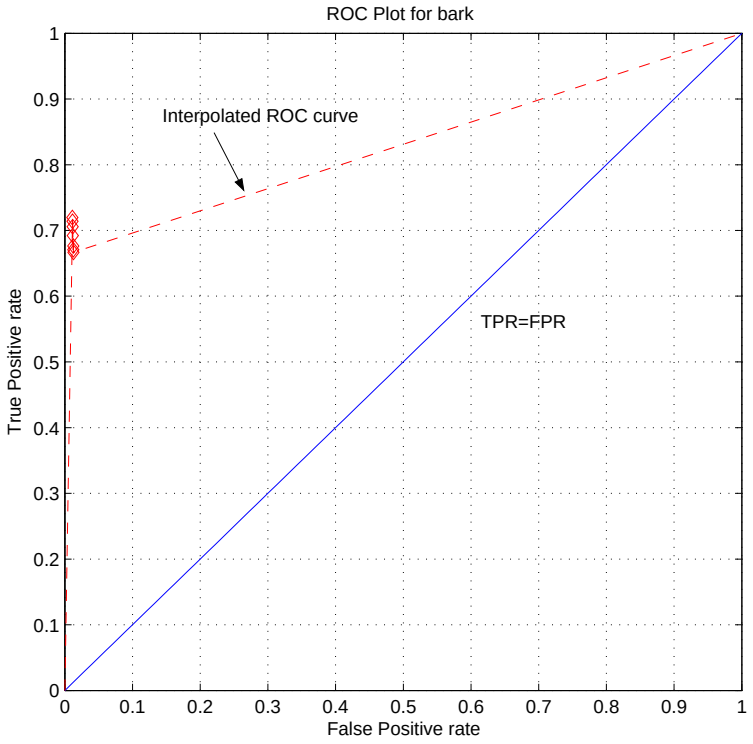


Figure 5.6: ROC points for bark. Each point represents one training step and the corresponding classification performance.

5.5.3 Region Based Analysis Results

So far, we have seen that IntelliPost achieves an improvement in overall segmentation in pixel-wise comparison with ground truth. In order to see how regions in the final images are similar to the corresponding regions in ground truth images, a region based similarity analysis is needed. Region based similarity results were obtained by using area similarity and shape similarity measures.

To obtain similarity measures of two corresponding regions, two regions needs to be chosen: one from postprocessed image and one from the associated ground truth image. Such regions are picked manually by using mouse and then area similarity measure [7, 208], shape similarity measure [143], and overlapping index [94], using the same notation as Equation

5.6, that is defined in Equation 5.7 were obtained. The results are shown in Table 5.33.

Table 5.33: Region based similarity measures are listed as the area similarity measure (ASM), the shape similarity measure (SSM), and overlap index. Region based similarity values are shown before and after postprocessing as compared with ground truth. Each similarity measures range from 0 to 1. The value of 0 indicates the most dissimilar and 1.0 indicates the most similar (ideal case). The same slices were analyzed at region level as in the case of confusion matrix analysis.

			Before Postprocessing			After Postprocessing		
Dataset	**Region Number**	**Type**	**ASM**	**SSM**	**Overlap**	**ASM**	**SSM**	**Overlap**
RK01	1	KN	0.82	0.86	0.70	0.81	0.81	0.69
	2	CW	0.93	0.62	0.87	0.94	0.73	0.89
	3	SP	0.48	0.94	0.32	0.49	0.96	0.32
	4	BR	0.63	0.93	0.46	0.65	0.94	0.48
RK12	1	BR	0.58	0.67	0.40	0.77	0.73	0.63
	2	DC	0.91	0.62	0.83	0.93	0.75	0.87
	3	CW	0.94	0.82	0.89	0.95	0.92	0.91
2048	1	KN	0.70	0.32	0.54	0.68	0.32	0.52
	2	CW	0.94	0.67	0.90	0.94	0.70	0.90
	3	BR	0.72	0.72	0.56	0.76	0.72	0.61
2049	1	KN	0.51	0.35	0.34	0.48	0.31	0.32
	2	KN	0.71	0.85	0.55	0.71	0.83	0.56
	3	BR	0.76	0.80	0.62	0.68	0.54	0.51
	4	CW	0.94	0.57	0.90	0.95	0.73	0.90
2051	1	KN	0.87	0.84	0.78	0.87	0.84	0.78
	2	CW	0.96	0.70	0.93	0.97	0.92	0.95
	3	BR	0.72	0.77	0.56	0.65	0.55	0.49
578A	1	SW	0.95	0.45	0.91	0.98	0.83	0.97
	2	HW	0.89	0.85	0.81	0.91	0.90	0.83
	3	DC	0.17	0.63	0.09	0.71	0.45	0.55
BILLE3-1	1	DC	0.84	0.80	0.72	0.87	0.91	0.77
	2	HW	0.80	0.60	0.67	0.80	0.61	0.67
	3	KN	0.70	0.75	0.54	0.64	0.82	0.47
	4	SW	0.96	0.73	0.93	0.96	0.85	0.93
	5	LB	0.89	0.98	0.80	0.90	0.99	0.82
	6	DB	0.78	0.93	0.64	0.78	0.94	0.65

5.6 Result for CT/MRI Medical Datasets

In this section, we present the results obtained with IntelliPost for the medical CT/MRI scan datasets, as described in section 5.3. Both CT and MRI dataset were divided into two parts: training set and test set. The ANN segmentation module was trained by using images from training set. Then segmented images were generated by the ANN. The segmented images were used to train IntelliPost. After the training, the subset of test set was used to evaluate IntelliPost for the medical dataset. Again, ground truth was obtained by manual segmentation as described earlier in the chapter for CT dataset. Ground truth images for MRI dataset was available from BrainWeb project site therefore we obtained the corresponding ground truth for MRI dataset. One representative slice was selected from each dataset to illustrate improvement in segmentation (see Figure 5.16 and Figure 5.17). Their confusion matrices are shown in Table 5.34 through Table 5.37. Confusion matrices show improvement on final segmentation since diagonal elements increase and off-diagonal element decrease in most cases. True positive rates and false positive rates for each class type are shown in Table 5.38 through Table 5.41. Overall improvement in correct segmentation rate are listed in Table 5.42. As we see from overall segmentation accuracy results, the average improvements for algotech-23 and BrainWeb datasets (for the corresponding images) were 4.22% and 0.33%, respectively.

Table 5.34: Confusion matrices for slice CT.29017.1 from dataset algotech-23. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are three classes: skin (SN), skull (SK), and brain (BRN).

Dataset	Slice Number	Before Postprocessing Confusion Matrix				After Postprocessing Confusion Matrix			
			BRN	SK	SN		SN	SK	BRN
algotech-23	CT.29017.1	BRN	29582	714	80	SN	29662	714	0
		SK	447	6719	34	SK	256	6719	225
		SN	1666	692	5158	BRN	0	692	6824

Table 5.35: Improvement in confusion matrix after postprocessing for slice CT.29017.1 from dataset algotech-23. The difference between postprocessed and initial segmentation as compared with ground truth.

Dataset	Slice Number	Improvement in Confusion Matrix			
			SN	SK	BRN
algotech-23	CT.29017.1	SN	80	0	-80
		SK	-191	0	191
		BRN	-1666	0	1666

Table 5.36: Confusion matrices for slice 96 from brainweb dataset. Classification accuracy values are shown before and after postprocessing as compared with ground truth. There are nine classes: cerebral spinal fluid (CSF), gray matter (GM), white matter (WM), fat (FT), muscle/skin (MSK), skin (SN), skull (SK), glial matter (GLM), and connective (CN).

Dataset	Slice Number	Before Postprocessing Confusion Matrix								
			CSF	**GM**	**WM**	**FT**	**MSK**	**SN**	**SK**	**GLM**	**CN**
BrainWeb	96	**CSF**	2596	151	0	0	0	0	94	0	0
		GM	380	5864	412	0	62	0	0	0	3
		WM	0	447	8676	0	0	0	0	0	0
		FT	0	0	73	228	0	0	0	0	78
		MSK	162	262	6	0	1453	121	0	0	42
		SN	521	25	0	0	25	1722	558	0	4
		SK	96	0	0	0	0	64	1978	0	0
		GLM	87	87	39	0	0	0	0	0	0
		CN	0	139	94	26	334	0	0	0	510
		After Postprocessing Confusion Matrix									
			CSF	**GM**	**WM**	**FT**	**MSK**	**SN**	**SK**	**GLM**	**CN**
		CSF	2596	151	0	0	0	0	94	0	0
		GM	403	5906	412	0	0	0	0	0	0
		WM	0	447	8676	0	0	0	0	0	0
		FT	0	0	0	234	67	0	0	0	78
		MSK	21	10	0	6	1500	107	200	0	202
		SN	90	0	0	2	54	1539	1165	0	5
		SK	36	0	0	0	0	0	2102	0	0
		GLM	87	87	39	0	0	0	0	0	0
		CN	0	15	18	49	458	2	12	0	549

Table 5.37: Improvement in confusion matrix after postprocessing for slice 96 from brainweb dataset. The difference between postprocessed and initial segmentation as compared with ground truth.

Dataset	Slice Number	Improvement in Confusion Matrix								
			CSF	**GM**	**WM**	**FT**	**MSK**	**SN**	**SK**	**GLM**	**CN**
BrainWeb	96	**CSF**	0	0	0	0	0	0	0	0	0
		GM	23	42	0	0	-62	0	0	0	-3
		WM	0	0	0	0	0	0	0	0	0
		FT	0	0	-73	6	67	0	0	0	0
		MSK	-141	-252	-6	6	47	-14	200	0	160
		SN	-431	-25	0	2	29	-183	607	0	1
		SK	60	0	0	0	0	-64	124	0	0
		GLM	0	0	0	0	0	0	0	0	0
		CN	0	-124	-76	23	124	2	12	0	39

Table 5.38: True positive rates for medical datasets. There are nine classes: Cerebral Spinal Fluid (CSF), gray matter (GM), white matter (WM) or brain (BRN), fat (FT), muscle/skin (MSK), skin (SN), skull (SK), glial matter (GLM), and connective (CN).

Dataset	Slice Number	Before Postprocessing								
		True Positive Rate								
		CSF	**GM**	**WM (BRN)**	**FT**	**MSK**	**SN**	**SK**	**GLM**	**CN**
algotech-23	CT.29017.1	-	-	0.6863	-	-	0.9739	0.9332	-	-
BrainWeb	96	0.9138	0.8725	0.9510	0.6016	0.7102	0.6032	0.9252	0.0000	0.4624
		After Postprocessing								
		True Positive Rate								
		CSF	**GM**	**WM (BRN)**	**FT**	**MSK**	**SN**	**SK**	**GLM**	**CSF**
algotech-23	CT.29017.1	-	-	0.9079	-	-	0.9765	0.9332	-	-
BrainWeb	96	0.9138	0.8787	0.9510	0.6174	0.7331	0.5391	0.9832	0.0000	0.4977

Table 5.39: Improvement in true positive rates for medical datasets. Positive numbers are desired.

Dataset	Slice Number	Improvement in True Positive Rate								
		CSF	GM	WM (BRN)	FT	MSK	SN	SK	GLM	CN
algotech-23	CT.29017.1	-	-	0.2216	-	-	0.0026	0.0000	-	-
BrainWeb	96	0.0000	0.0062	0.0000	0.0158	0.0229	-0.0641	0.0580	0.0000	0.0353

Table 5.40: False positive rates for medical datasets. There are nine classes: cerebral spinal fluid (CSF), gray matter (GM), white matter (WM) or brain (BRN), fat (FT), muscle/skin (MSK), skin (SN), skull (SK), glial matter (GLM), and connective (CN).

Dataset	Slice Number	Before Postprocessing								
		False Positive Rate								
		CSF	GM	WM (BRN)	FT	MSK	SN	SK	GLM	CN
algotech-23	CT.29017.1	-	-	0.0628	-	-	0.0540	0.0127	-	-
BrainWeb	96	0.0100	0.0414	0.0244	0.0056	0.0234	0.0461	0.0063	0.0078	0.0225
		After Postprocessing								
		False Positive Rate								
		CSF	GM	WM (BRN)	FT	MSK	SN	SK	GLM	CN
algotech-23	CT.29017.1	-	-	0.0184	-	-	0.0485	0.0127	-	-
BrainWeb	96	0.0100	0.0394	0.0244	0.0054	0.0215	0.0536	0.0014	0.0078	0.0211

Table 5.41: Improvement in false positive rates for medical datasets. Negative numbers are desired.

Dataset	Slice Number	Improvement in False Positive Rate								
		CSF	GM	WM (BRN)	FT	MSK	SN	SK	GLM	CN
algotech-23	CT.29017.1	-	-	0.2216	-	-	0.0026	0.0000	-	-
BrainWeb	96	0.0000	-0.0002	0.0000	-0.0002	-0.0019	0.0075	-0.0049	0.0000	-0.0014

Table 5.42: Overall correct segmentation rates for selected images in medical datasets.

Dataset	Slice Number	Before Postprocessing	After Postprocessing	Improvement in segmentation
		Overall Correct Segmentation Rates	Overall Correct Segmentation Rates	Percent
algotech-23	CT.29017.1	0.9194	0.9582	4.22 %
BrainWeb	96	0.8398	0.8426	0.33 %

5.7 Summary

In this chapter, we have investigated and evaluated the segmentation performance of IntelliPost. The results showed that overall segmentation performance was improved by IntelliPost, at both the image level and the region level in general. Because IntelliPost is both a learning system and a region refinement/analysis system, we have tested the system's learning capability through the ROC analysis. The results of ROC curves show that IntelliPost gradually captures postprocessing rules from a user, and applies obtained rules for the similar cases.

The system performs better when the output of an ANN segmentation algorithm is oversegmented. As the some of the results showed, there is no way to recover undersegmented regions by using the present architecture of overall segmentation system. In other words, IntelliPost assumes all defects regions are detected by an ANN algorithm and there is no need to go back to the original CT/MRI data to analyze such regions. The analysis of undersegmented regions requires some other segmentation methods to improve accuracy of segmentation.

As we see from the results, IntelliPost improves segmentation accuracy in general but not dramatic. The current architecture of the overall system does not let IntelliPost go back the original data and/or modify segmentation algorithm of the ANN module. The ANN

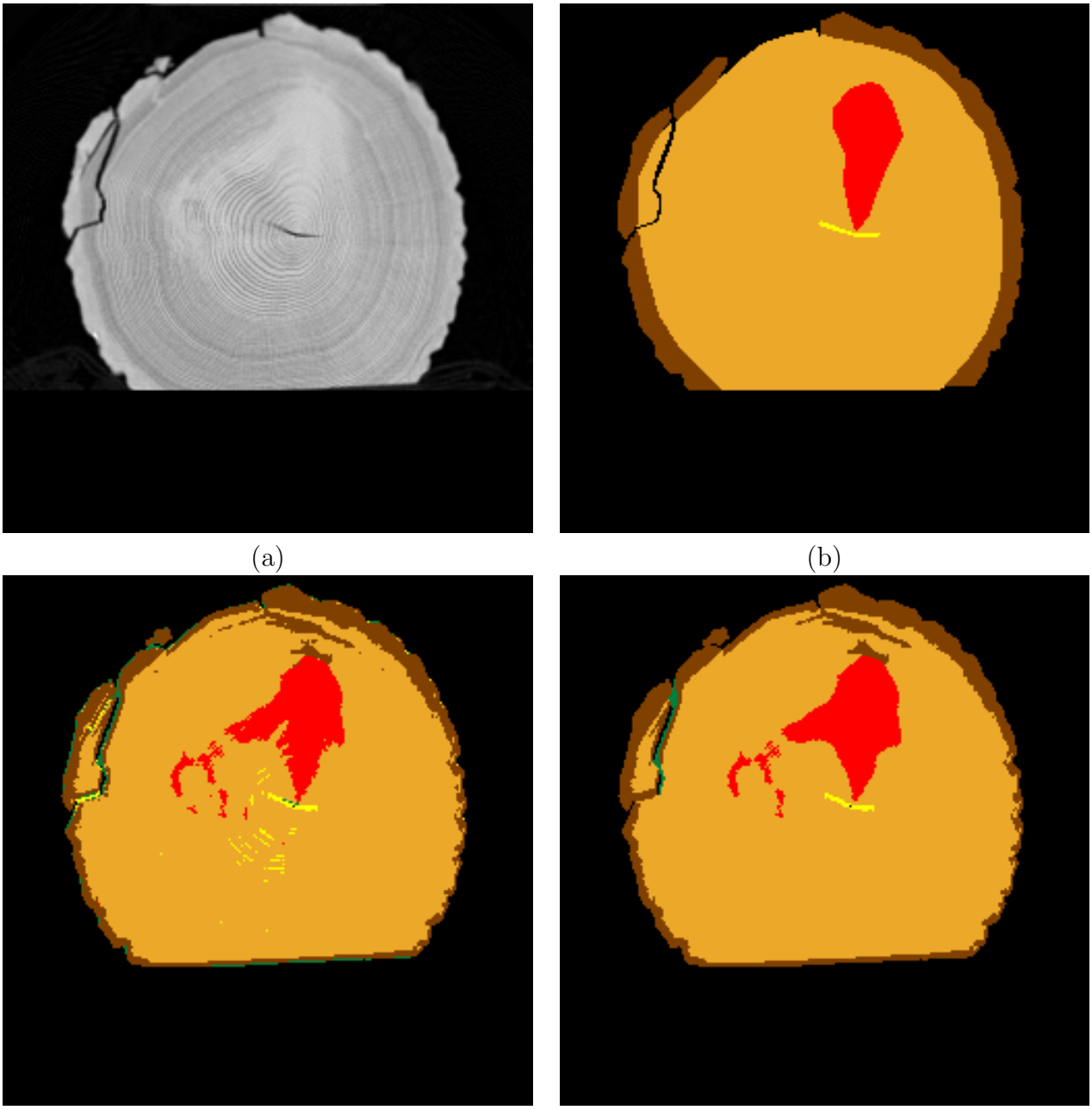
segmentation module does not provide any mechanism to manipulate the result of the ANN. Therefore IntelliPost can do nothing but relying on the ANN result. IntelliPost has to accept whatever the ANN gives for refinement. The overall system's ability is dramatically reduced by the open loop architecture of the overall system.

Another drawback of ANN approach, the ANN module needs to be trained every time when a new dataset is provided. Training process requires collection of samples by expert and retraining the ANN network from scratch. Once the ANN segmentation module is trained, the ANN segmentation module provides no functionality to adjust its segmentation response therefore it behaves as a fixed segmentation module.

We expected that IntelliPost would give more dramatic improvement in segmentation accuracy but the current architecture of the ANN module that was described above and its limited postprocessing operation library caused moderate but not dramatic segmentation improvements.

IntelliPost could be improved dramatically by expanding the operation library, and improving the ability of inference engine. The decision tree approach for inference engine has pitfall to get over training. Fuzzy logic based inference engine could be used as replacement for inferencing. By using fuzzy logic approach, the inferencing process for refinement operation could be as close as possible to a human user's decision making process. The current inferencing mechanism is close to classification approach that heavily depends on the feature space.

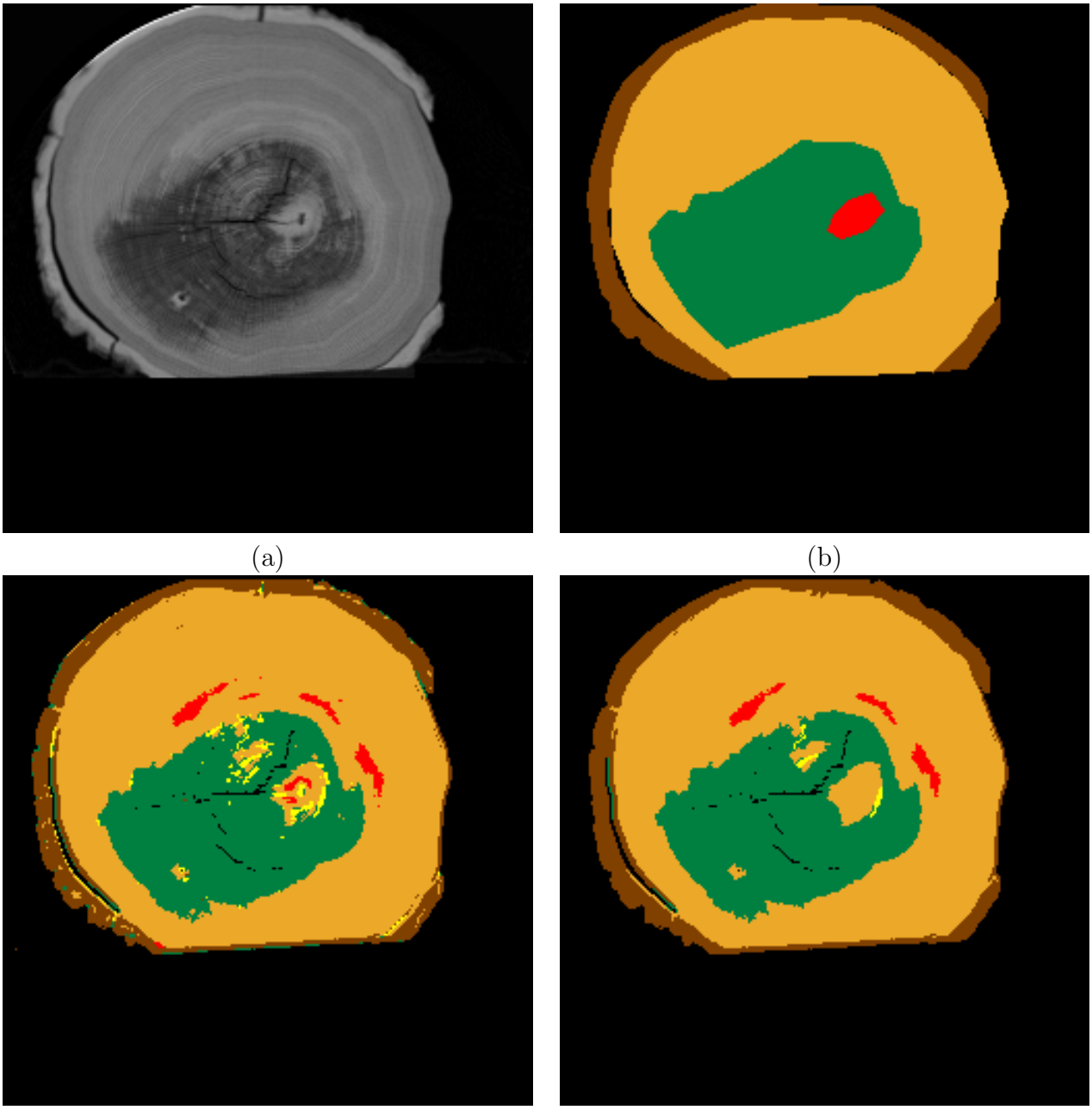
Results from medical CT dataset were provided to show the applicability and usability of the overall system to medical applications. Medical result shows promising results. Such segmentation concept could be used in medical application. IntelliPost and segmentation module still requires improvement on segmentation accuracy.



(a) (b)

(c) (d)

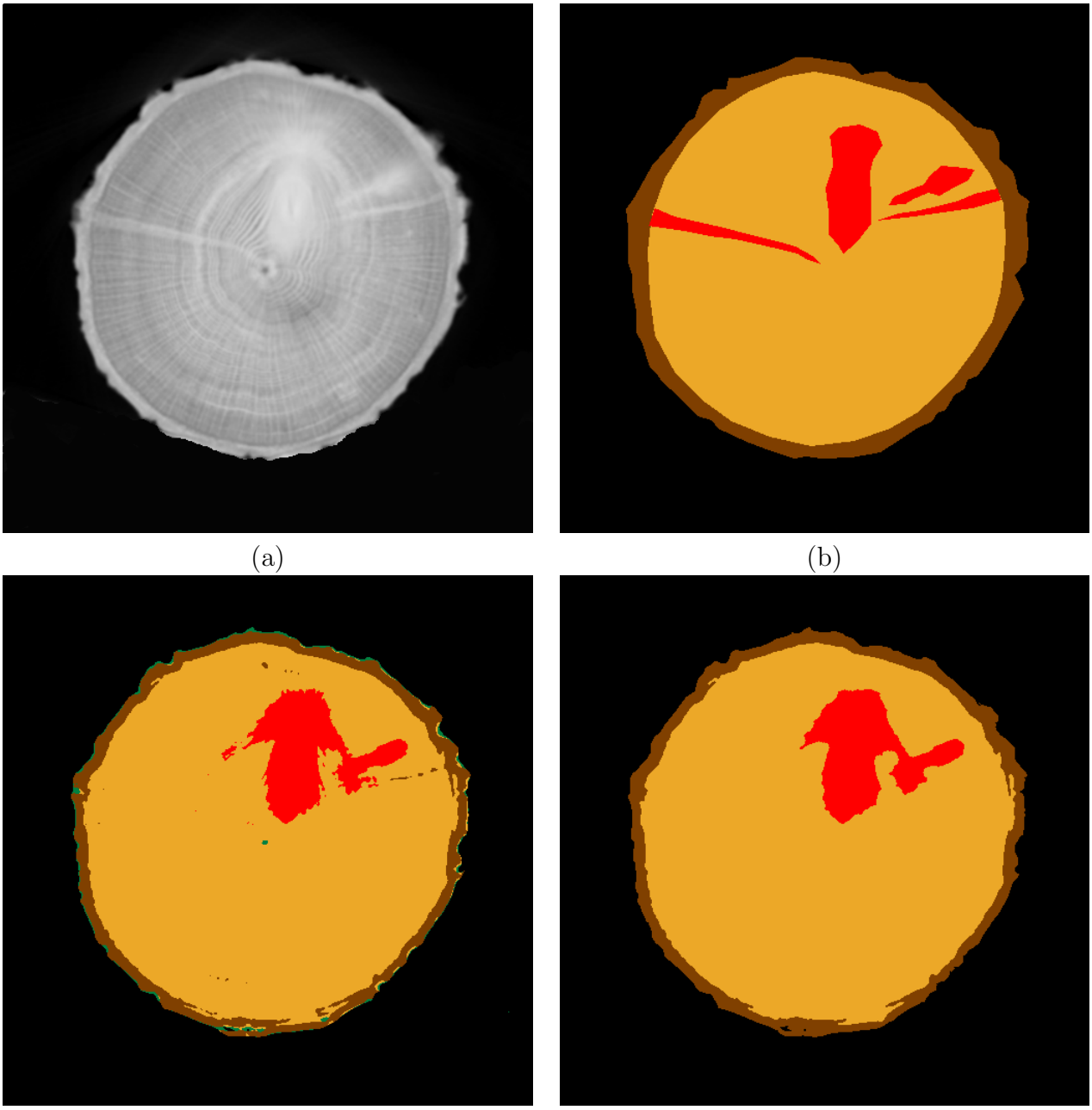
Figure 5.7: Visual comparison of RK01 dataset slice number 3. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



(a) (b)

(c) (d)

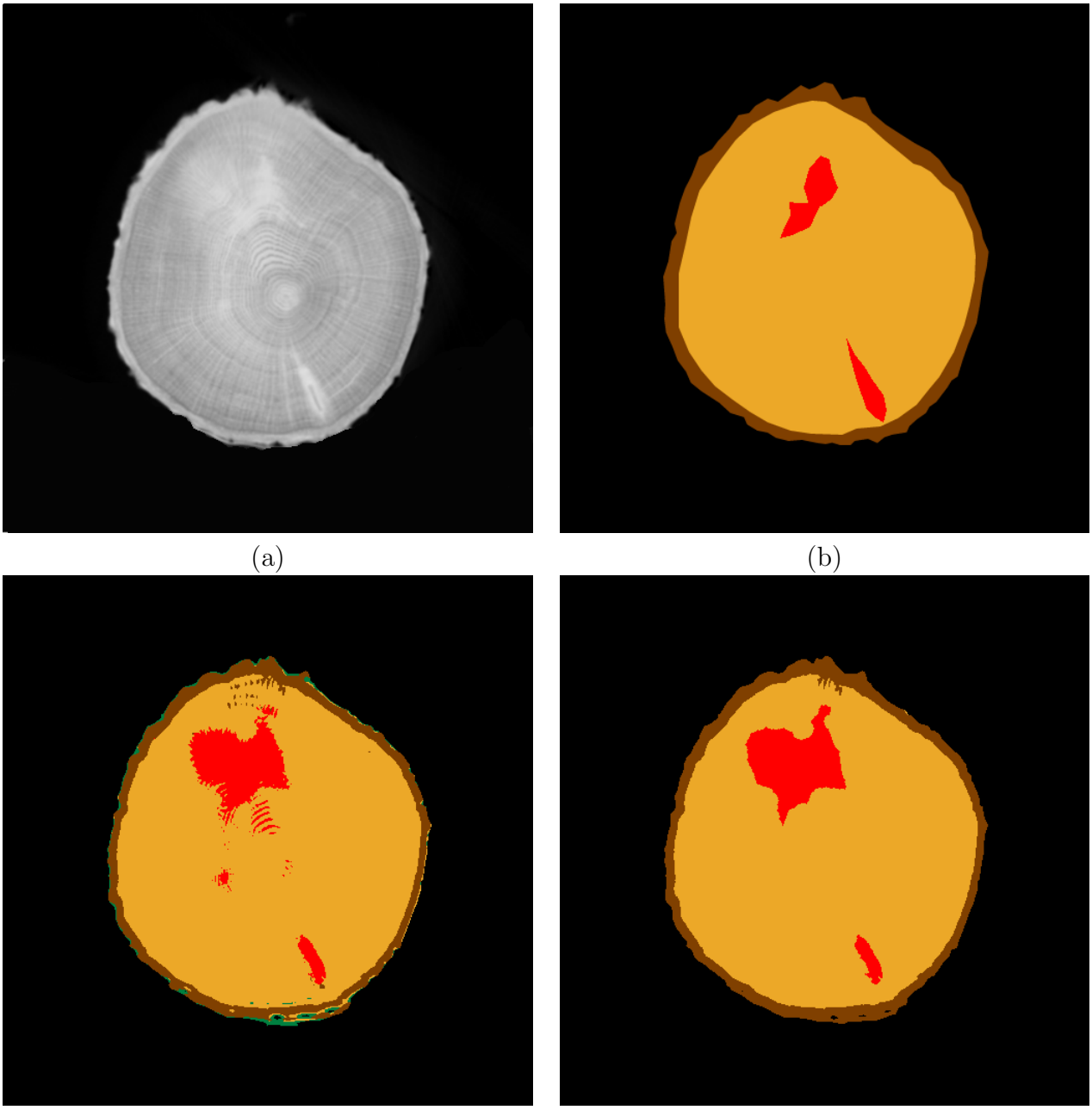
Figure 5.8: Visual comparison of RK12 dataset slice number 5. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



(a) (b)

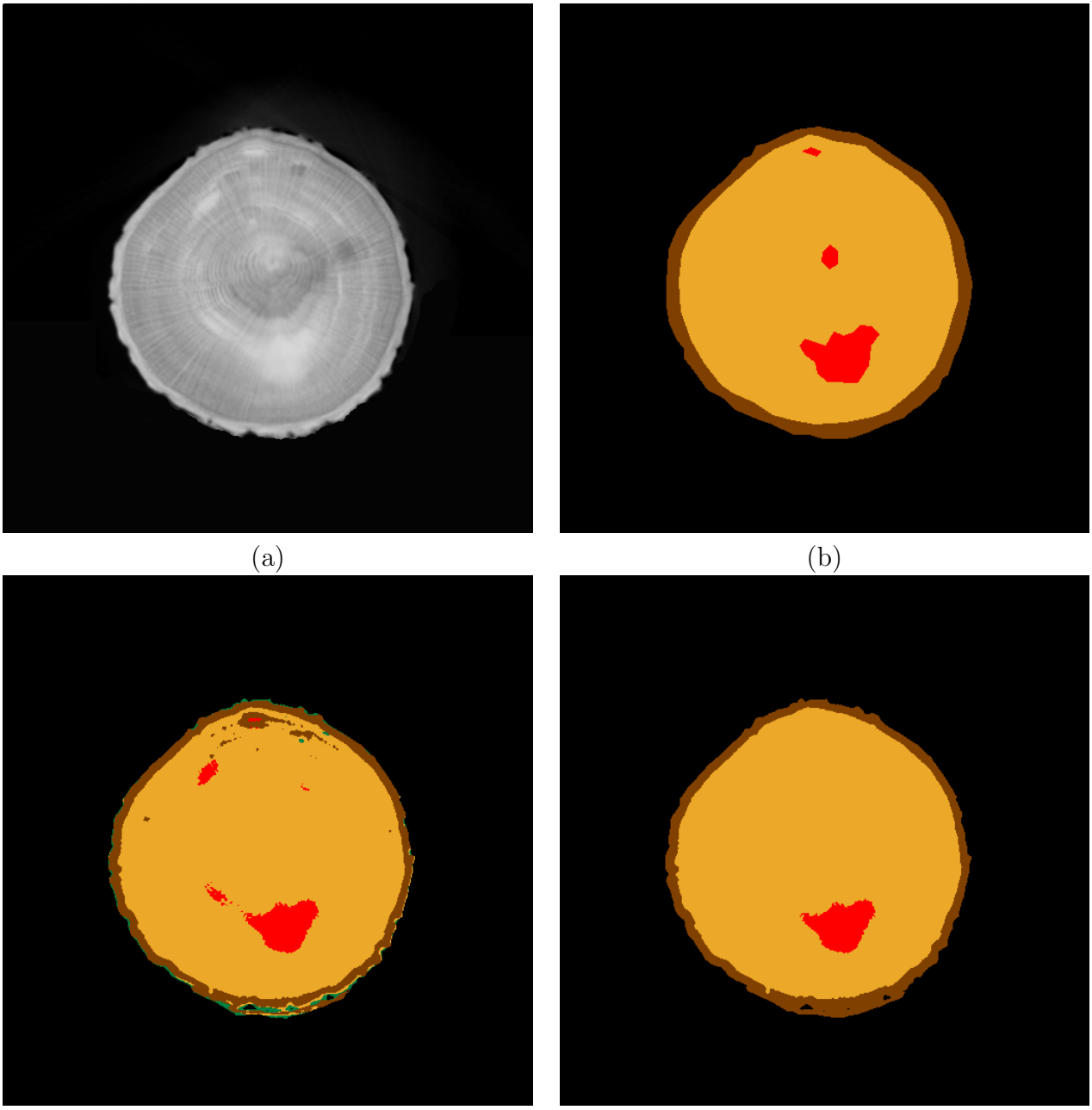
(c) (d)

Figure 5.9: Visual comparison of 2048 dataset slice number 15. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Pixels from the stands supporting the log were included in the corresponding class in confusion matrices. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



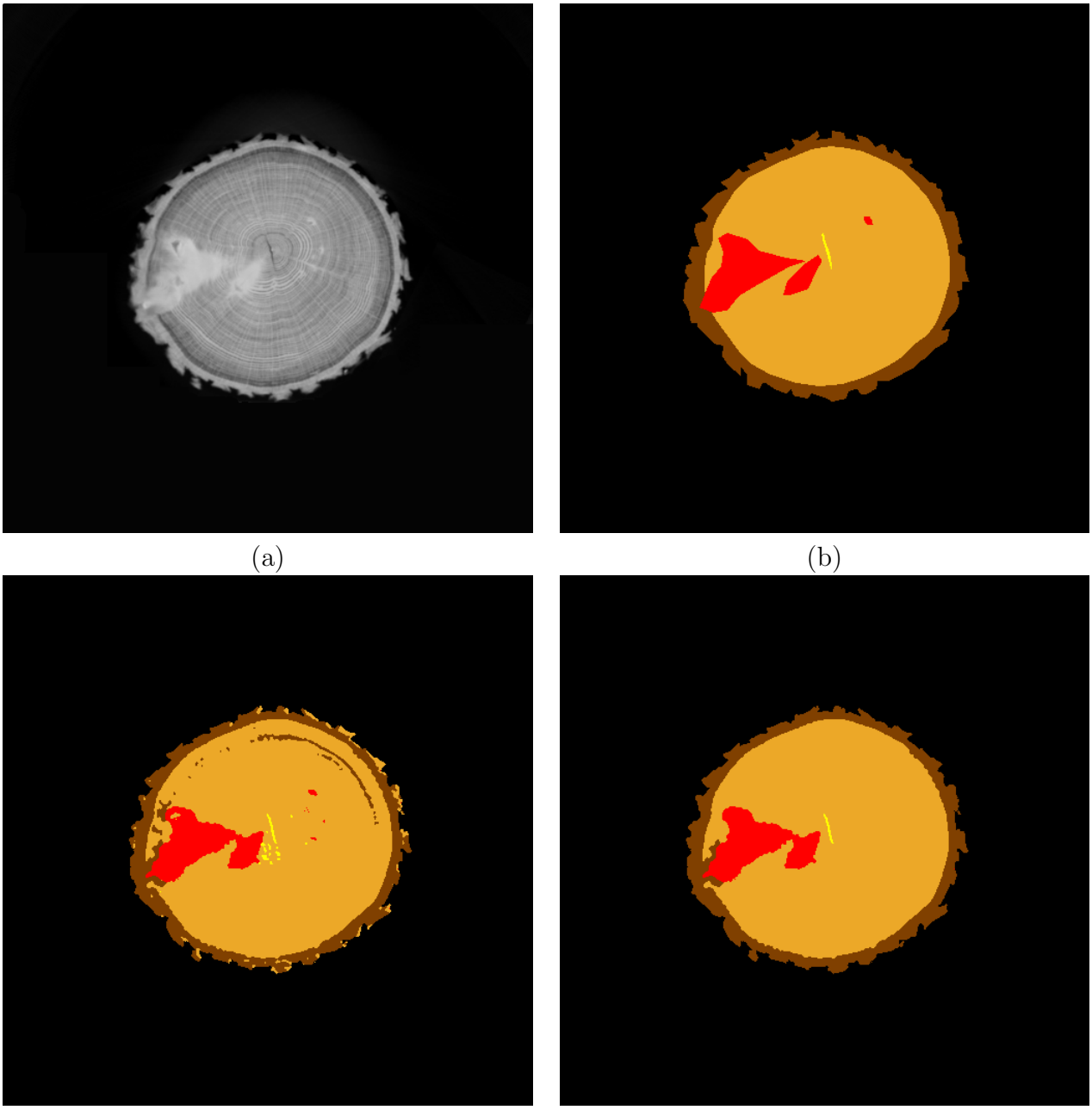
(a) (b) (c) (d)

Figure 5.10: Visual comparison of 2049 dataset slice number 5. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Pixels from the stands supporting the log were included in the corresponding class in confusion matrices. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



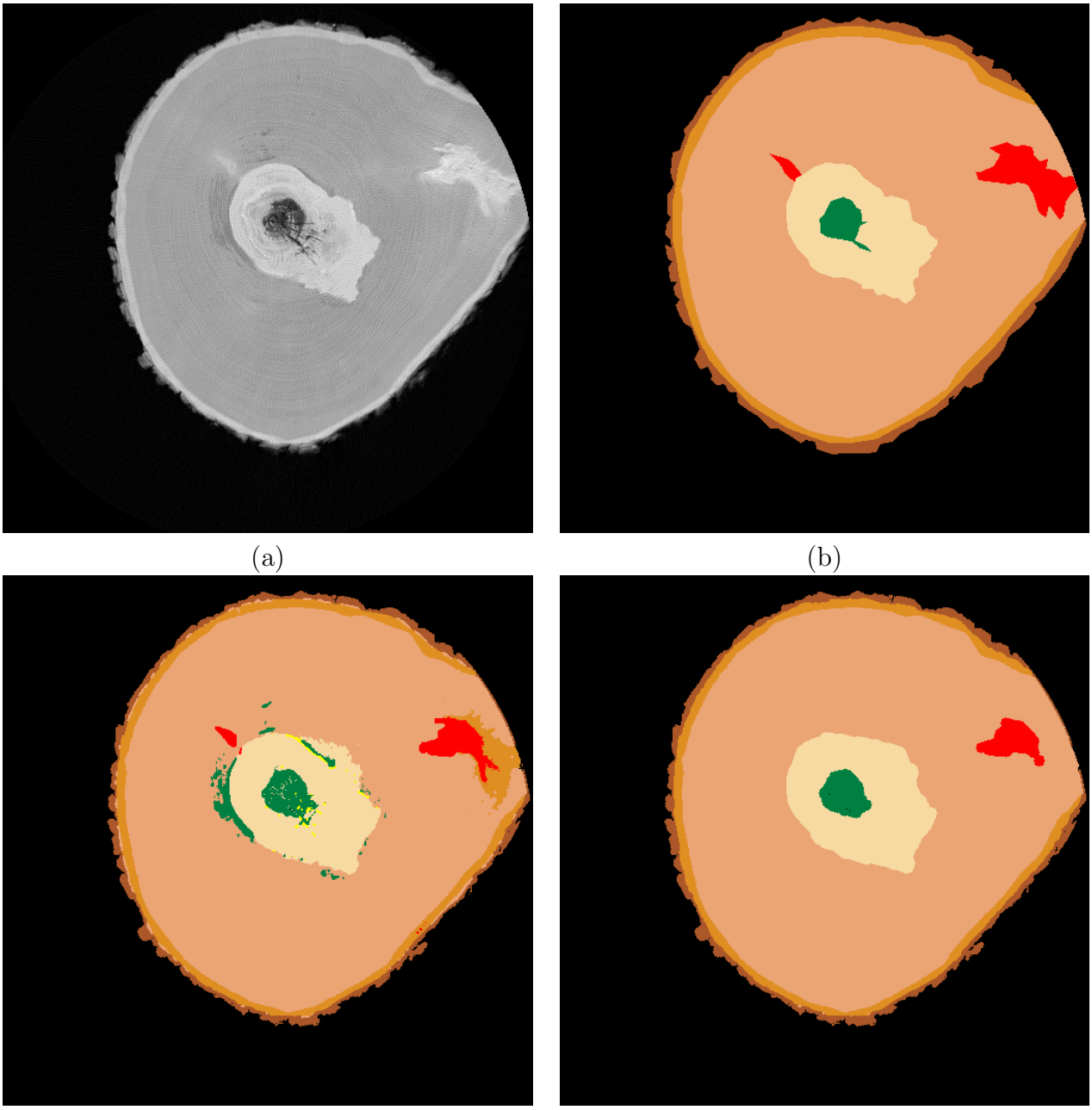
(a) (b) (c) (d)

Figure 5.11: Visual comparison of 2051 dataset slice number 1. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Pixels from the stands supporting the log were included in the corresponding class in confusion matrices. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



(a) (b) (c) (d)

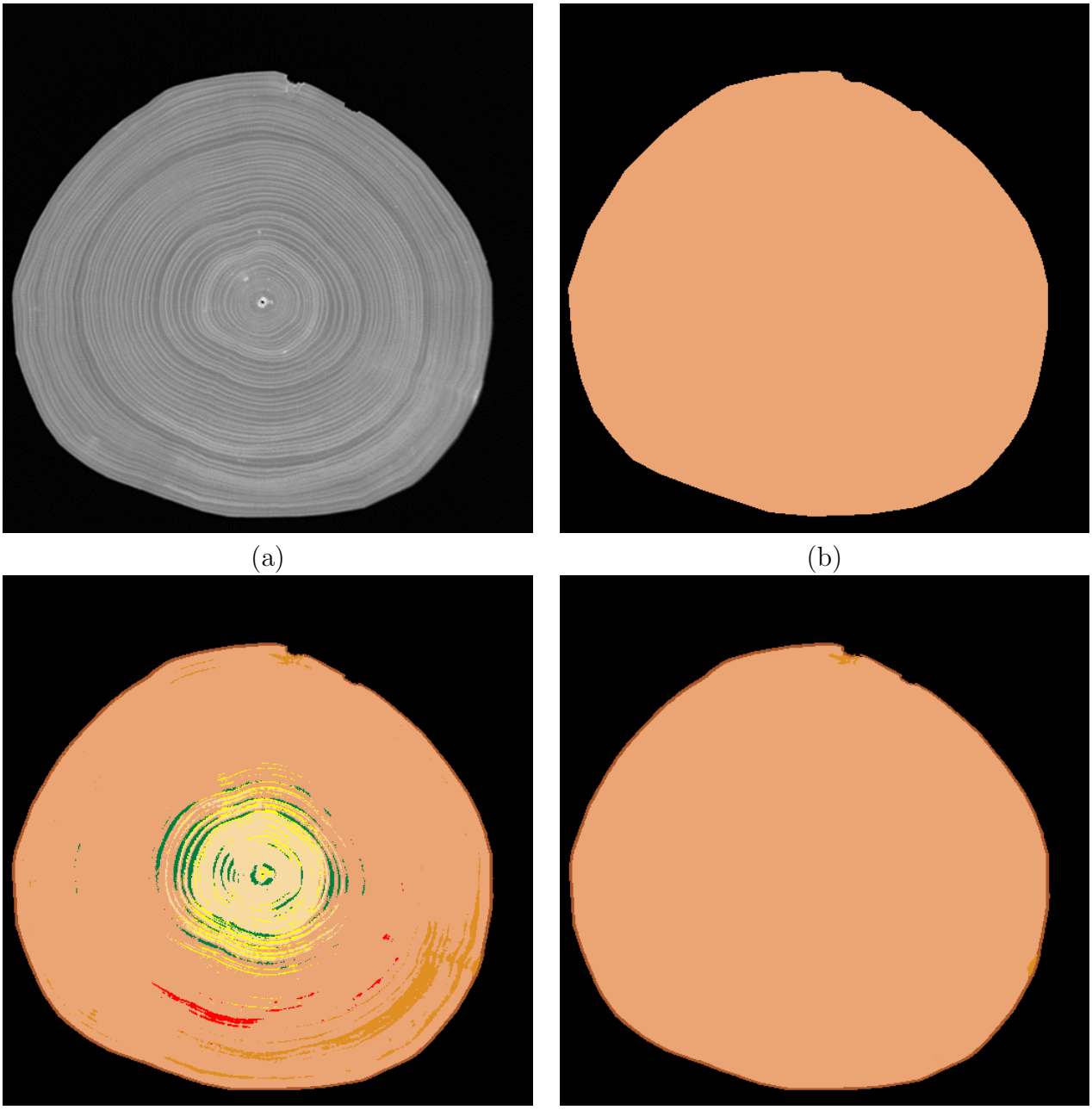
Figure 5.12: Visual comparison of 5357 dataset slice number 3. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Pixels from the stands supporting the log were included in the corresponding class in confusion matrices. Tan color is clear wood, brown is bark, green is decay, and red is knot defect type.



(a) (b)

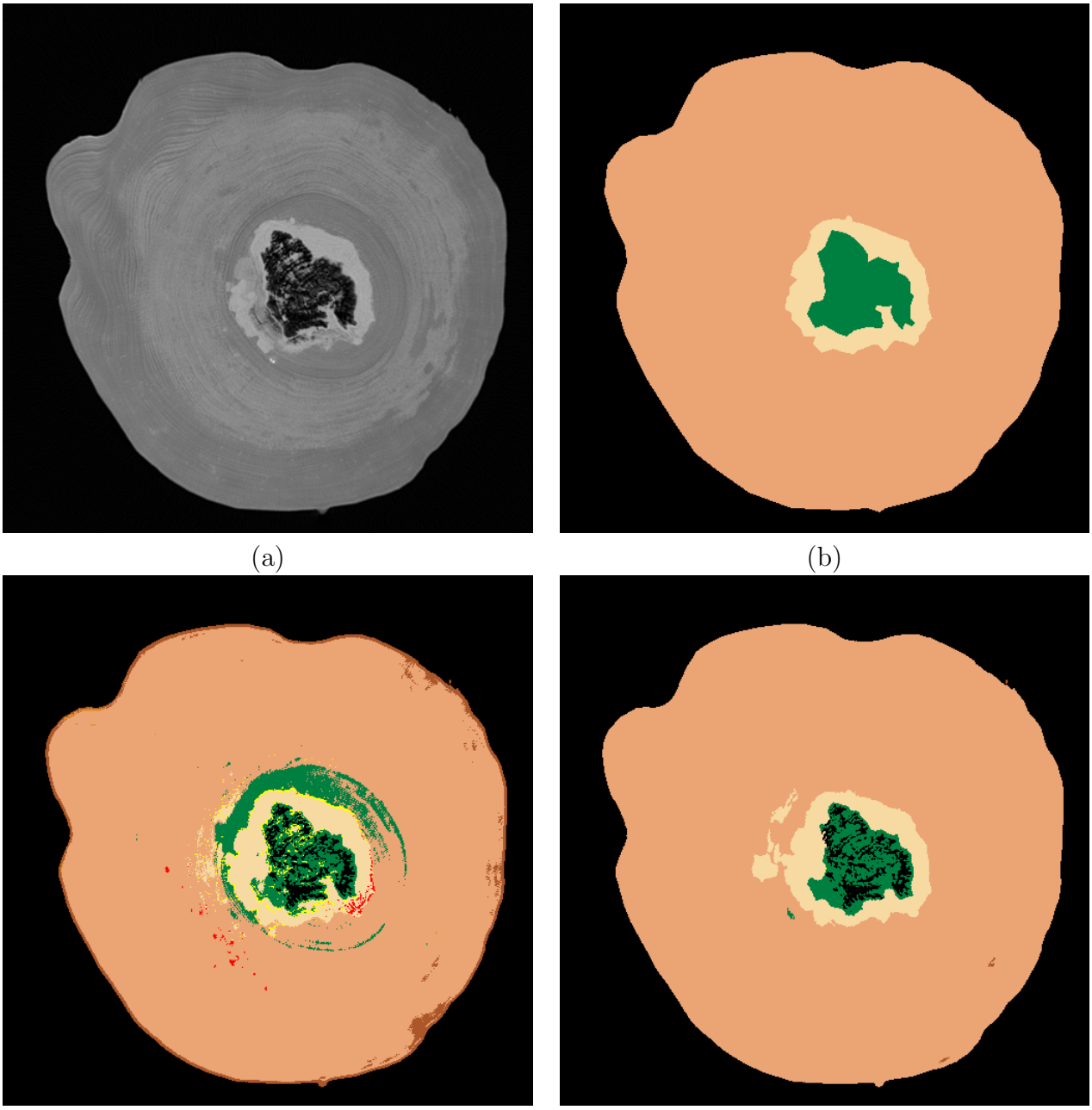
(c) (d)

Figure 5.13: Visual comparison of bille3-1 dataset slice number 7370557. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Light tan color represents sapwood, beige color represents heartwood, yellow represents split defects, green represents decay, red represents knot, and dark tan color represents live bark.



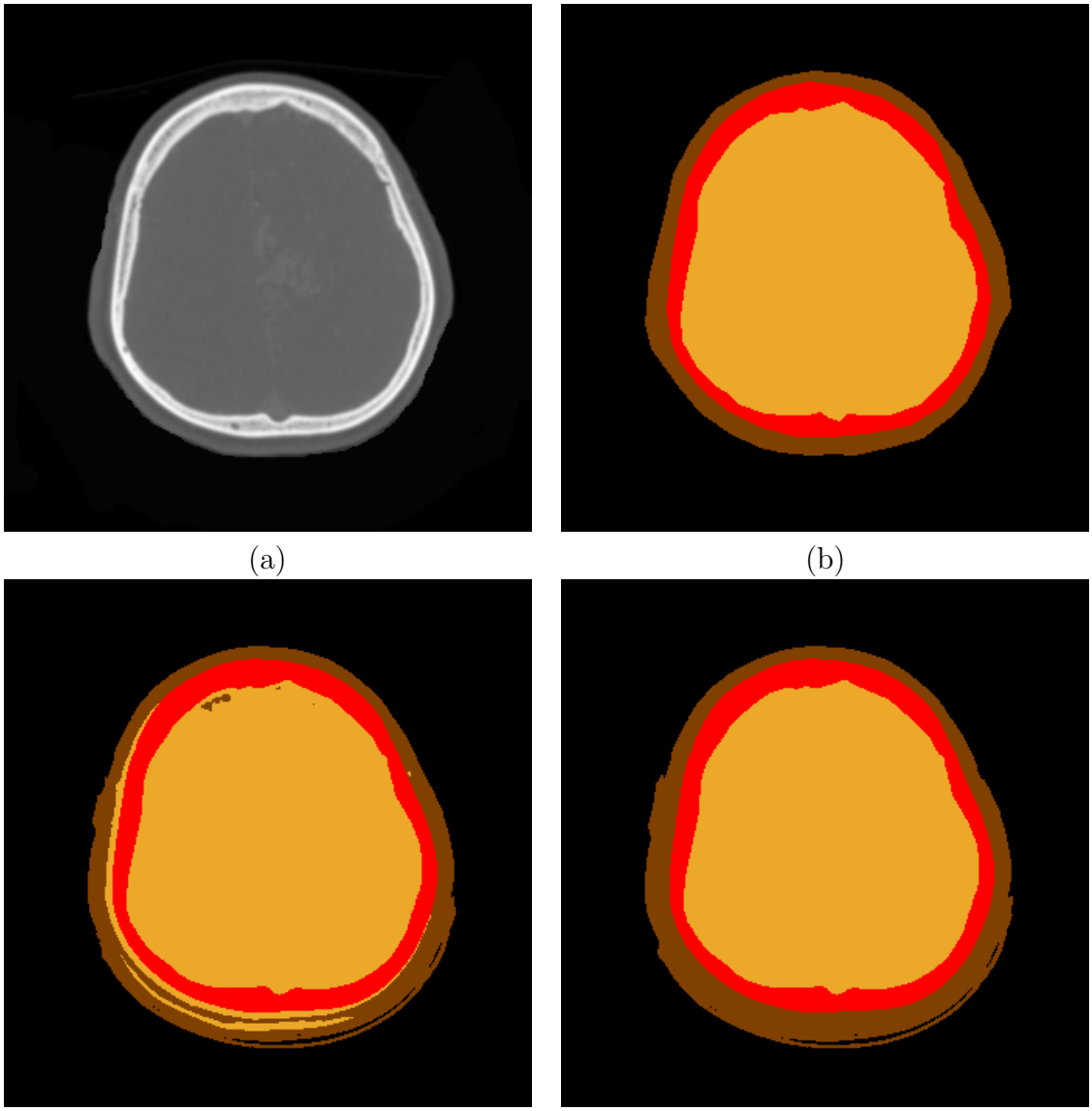
(a) (b) (c) (d)

Figure 5.14: Visual comparison of 567b dataset slice number 4127003. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Light tan color represents sapwood, beige color represents heartwood, yellow represents split defects, green represents decay, red represents knot, and dark tan color represents live bark.



(a) (b) (c) (d)

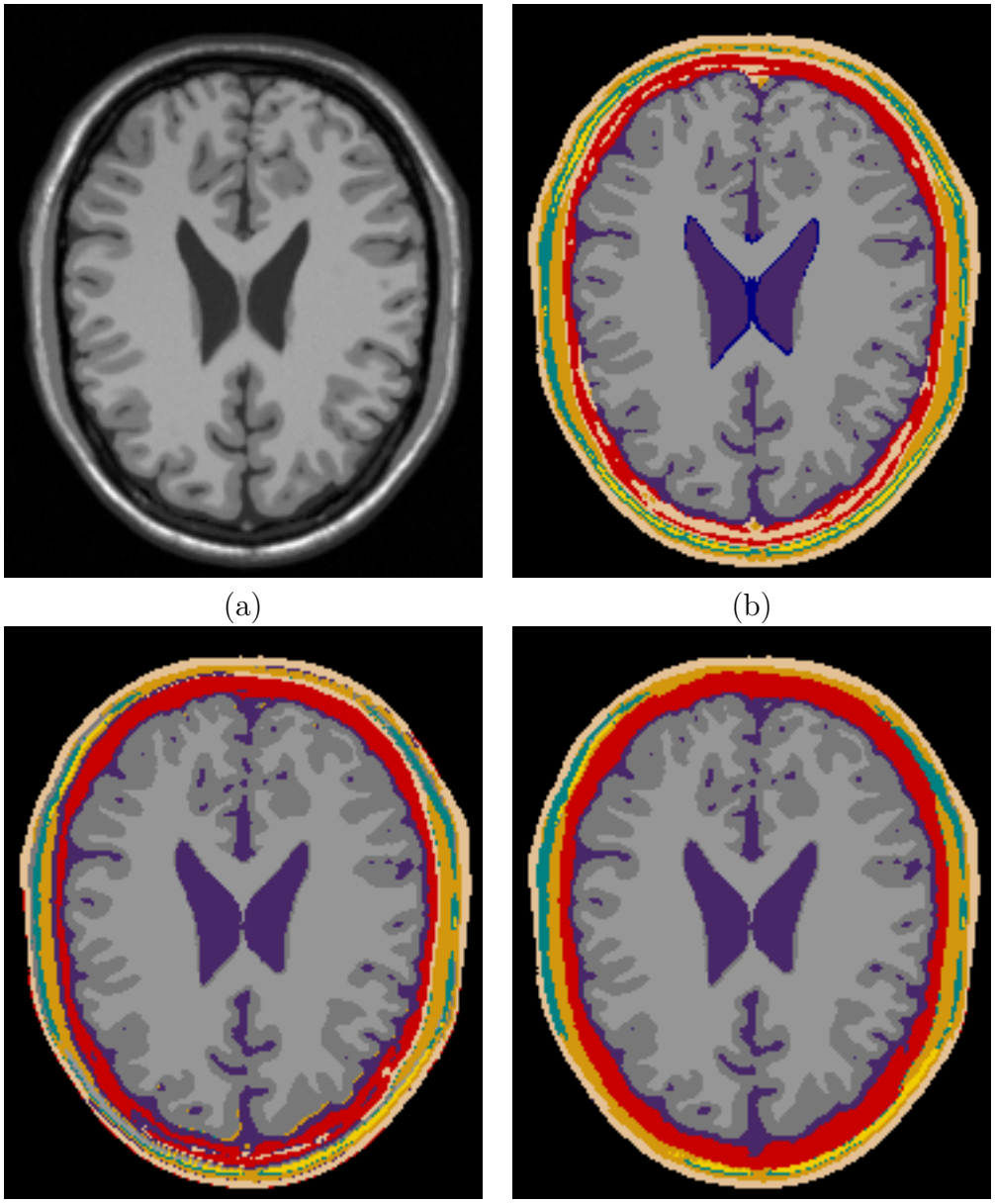
Figure 5.15: Visual comparison of 578a dataset slice number 4133900. (a) Original slice. (b) Manual segmentation (ground truth). (c) The result of ANN segmentation. (d) IntelliPost result. Light tan color represents sapwood, beige color represents heartwood, yellow represents split defects, green represents decay, red represents knot, and dark tan color represents live bark.



(a) (b)

(c) (d)

Figure 5.16: Visual comparison for medical CT dataset. (a) Original CT slice. (b) Ground truth.The corresponding ground truth. (c) Initial segmentation from the ANN. (d) The final result after postprocessing. Pixels from the stands supporting the head were included in the corresponding class in confusion matrices. Brown represents skin, red represents skull, and tan color represents brain.



(a) (b)

(c) (d)

Figure 5.17: Visual comparison for medical MRI dataset. (a) Original MRI slice. (b) The corresponding ground truth. (c) Initial segmentation from the ANN. (d) The final result after postprocessing.

Chapter 6

Summary and Conclusion

6.1 Summary

Intellipost was built using two different areas of research: machine learning and image segmentation. One aspect of this study was to explore the possibility of using demonstrational learning methods in the area of image analysis. In any machine learning system, capturing and formulating a human's expertise have been, and will continue to be, a challenging issue. The demonstrational learning methods provide some promising approaches to capture such expertise from a human user and use it in an automated system.

This thesis has explored the potential of using learning by demonstration for a problem related to image segmentation. We have developed a learning system that captures expertise from a human user. The captured knowledge is used to refine regions in CT/MRI segmentation problems.

The other aspect of this study was to improve image segmentation of CT/MRI images through region analysis and refinement. A CT/MRI image segmentation, the partitioning of

an CT/MRI image slice into semantically meaningful regions, is an important prerequisite for many applications. The processing of CT images of hardwood logs is one example among those applications. This is a difficult task because the natural shape of wood defects varies dramatically from one log to another.

IntelliPost extracts domain specific information through a demonstrational learning method that provides an environment for a human expert to demonstrate his/her expertise on presented cases. Through the learning by demonstration concept, the difficulty of obtaining domain knowledge through formal interaction between a domain expert and an expert system designer is reduced. Using the demonstrational learning method also eliminates any possible communication error between the expert system designer and the domain expert. When the user demonstrates his/her actions on a presented example, the system captures relevant information about the given problem.

The system uses decision tree induction as a major component of its inferencing engine. It provides two modes of operation: learn mode and run mode. During the learn mode, the user refines a segmented image using operations in the postprocessing operation library. Simultaneously, the system extracts high level information from regions that are being postprocessed by the user, and stores the extracted information into the domain knowledge base for later use. In the run mode, the system constructs postprocessing rules by using an OC-SEP decision tree induction algorithm. Once the decision tree is constructed, the system constructs a scene description structure that contains every region's geometric features in a given image along with their high level information. Region adjacency graph is also constructed for the presented image. Finally all regions are analyzed and processed based on their geometric features.

During run mode, IntelliPost picks regions in the order of smaller to larger. Their corresponding geometric features were obtained from previously built the scene description struc-

ture. The feature vector was constructed using obtained information for that region. The feature vector was given to the constructed decision tree to get proper refinement operation for the region. The refinement decision is executed by IntelliPost by using postprocessing operation library. This process continues until all regions in a image are analyzed and processed by IntelliPost. Results showed that improvements in segmentation accuracy for hardwood log datasets were 1.92% for the read oak (combined GE Medical and VetMed) datasets and 9.45% for the Forintek datasets. For the case of medical datasets, improvement for algotech-23 and BrainWeb datasets were 4.22% and 0.33%, respectively.

6.2 Conclusion

This thesis has introduced a new approach for refining segmented images. We have developed an experimental system that observes the actions of a human operator who interactively edits a set of test images. The system then applies automated inferencing techniques to develop its own refinement rules based on those actions. After this learning process, the system is capable of automatically applying similar refinement steps to other images.

The system does not simply memorize a sequence of operations by the user, as is often used for robotic teach pendants. Instead, the system develops more general rules based on labeled region properties, such as size, elongation, defect type, and position in the image. Although this approach has been developed particularly for use with CT image slices of hardwood logs, it is sufficiently general that it can be used for other applications, such as medical image analysis or aerial image segmentation.

6.3 Future Directions

This study explored the possibility of using learning by demonstration method for the image segmentation problem. During the course of the research, we have learned that it was very difficult to implement such a system due to the modelling the human behaviour and inconsistency in human actions. Modeling the action by formal way requires extensive effort by the system to understand meaning of a action.

There are several ways to improve the overall segmentation performance. For example, the one way is to replace the ANN by more adaptive segmentation algorithm. By replacing the segmentation module with more adaptive one could alleviate training requirement for segmentation module. Such replacement also helps improving the usability of the overall system and gives a mechanism to interact with the postprocessing module. Interaction between postprocessing module and segmentation module is essential way to improve overall image segmentation accuracy. For example, postprocessing module can analyze such region based on their high level relevant information and then send request to image segmentation module to resegment region that is not uniform enough. Two ways information passing could boost the system ability to give better segmentation results. Unfortunately, the current architecture does not let going back to segmentation module to resegment images.

Another way to improve the system is to expand IntelliPost's operation library. IntelliPost provide very limited operations for postprocessing. The expansion of the library will vastly increase capability and usability of IntelliPost.

Bibliography

[1] Abdou, I. E., "Quantitative methods of edge detection," Tech. Rep. 830, University of California Image Processing Institute, Los Angeles, CA, July 1978.

[2] Abutaleb, A. S., "Automatic thresholding of gray level pictures using two-dimensional entropy," *Computer Vision, Graphics, and Image Processing*, vol. 47, pp. 22–32, 1989.

[3] Adams, R. and Bischof, L., "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 641–647, June 1994.

[4] Ahuja, N. and Rosenfeld, A., "A note on the use of second order gray level statistics for threshold selection," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 8, pp. 895–898, 1978.

[5] Alexander, D. and Buxton, B., "Modelling of single mode distributions of colour data using directional statistics," in *Proceeding of Computer Vision and Pattern Recognition*, pp. 319–324, 1997.

[6] Askar, M. and Derin, H., "A recursive algorithm for the bayes solution of the smooting problem," *IEEE Transactions on Automatic Control*, vol. AC-26, pp. 558–561, 1981.

[7] Atkins, M. S. and Mackiewich, B. T., "Fully automatic segmentation of the brain in MRI," *IEEE Transactions in Medical Imaging*, vol. 17, pp. 98–107, February 1998.

[8] Babaguchi, N., Yamada, K., Kise, K., and Tezuka, Y., "Connectionist model binarization," *Proceeding of 10th International Conference on Pattern Recognition*, vol. 90, pp. 51–56, 1990.

[9] Barron, J. L., Fleet, D. J., and Beauchemin, S. S., "Performance of optical-flow techniques," *International Journal of Computer Vision*, vol. 12, pp. 43–77, 1994.

[10] Basu, S., "Image segmentation by semantic method," *Pattern Recognition*, vol. 20, no. 5, pp. 497–511, 1987.

[11] Bennett, K. P. and Bredensteiner, E. J., "Geometry in learning," in *Mathematical Association of America* (Gorini, C., ed.), (Washington, DC), pp. 132–145, 1996.

[12] Bennett, K. P. and Bredensteiner, E. J., "A parametric optimization method for machine learning," *INFORMS Journal on Computing*, vol. 3, no. 9, pp. 311–318, 1997.

[13] Bennett, K. P. and Mangasarian, O. L., "Multicategory separation via linear programming," Tech. Rep. 1127, Computer Science Department, University of Wisconsin, Madison, WI, 1992.

[14] Bennett, K. P. and Mangasarian, O. L., "Robust linear programming discrimination of two linearly inseperable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34, 1992.

[15] Bennett, K., "Decision tree construction via linear programming," in *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference* (Evans, M., ed.), (Utica, IL), pp. 97–101, 1992.

[16] Bergadano, F., Matwin, S., Michalski, R. S., and Zhang, J., "Measuring quality of concept descriptions," in *Proceedings of the European Working Session on Learning*, pp. 1–14, 1988.

[17] Beulieu, J. and Goldberg, M., "Hierarchy in picture segmentation: A stepwise optimization approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 150–163, February 1989.

[18] Bhandarkar, S., Faust, T., and Tang, M., "A system for detection of internal log defects by computer analysis of axial CT images," *IEEE International Workshop Applied Computer Vision*, pp. 258–263, December 1996.

[19] Biggs, M. C., "Constrained minimization using recursive quadratic programming," in *Toward Global Optimization* (Dixon, L. C. and Szergo, G. P., eds.), (North-Holland), pp. 341–349, 1975.

[20] Blanz, W., Cox, C., and Gish, S., "Connectionist architectures in low level image segmentation," in *Handbook of Pattern Recognition and Computer Vision*, p. Chapter V:5, 1997.

[21] Bolles, R. C., Baker, H. H., and Hannah, J. M., "The JISCT stereo evaluation," in *Proceedings of Image Understanding Workshop*, (Washington, D.C), pp. 263–274, 1994.

[22] Bonnin, P., Blanc Talon, J., Hayot, J., and Zavidovique, B., "A new edge point/region cooperative segmentation deduced from a 3d scene reconstruction application," *SPIE Applications of Digital Image Processing XII*, vol. 1153, pp. 579–591, 1989.

[23] Boukharouba, S., Rebordao, J. M., and Wendel, P. L., "An amplitude sedimentation method based on the distribution function of an image," *Computer Vision Graphics, and Image Processing*, vol. 29, pp. 47–59, 1985.

[24] Bowyer, K., Kranenburg, C., and Dougherty, S., "Edge detector evaluation using empirical ROC curves," in *Proceeding of Computer Vision and Pattern Recognition*, pp. 1354–1359, 1999.

[25] Bradley, P. S., Fayyad, U. M., and Mangasarian, O. L., "Mathematical programming for data mining: Formulation and challenges," *INFORMS Journal on Computing*, vol. 3, no. 11, pp. 217–238, 2004.

[26] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, P. J., *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[27] Burgess, A., "Potential application of imaging techniques to wood products," in *Proceedings of 1st International Conference on Scanning Technology in Sawmilling* (Szymani, R., ed.), vol. 7, (San Francisco, CA), pp. 1–13, October 10-12, 1985.

[28] Campbell, N. W., Thomas, B. T., and Troscianko, T., "Automatic segmentation and classification of outdoor images using neural networks," *International Journal of Neural Systems*, vol. 8, no. 1, pp. 137–144, 1997.

[29] Canny, J., "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679–698, November 1986.

[30] Castelman, K. R., *Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[31] Centor, R. M., "Signal detectability: The use of ROC curves and their analysis," *Medical Decision Making*, pp. 102–106, 1991.

[32] Chanda, B., Chaudhuri, B. B., and Majumder, D., "On image enhancement and threshold selection using the gray level co-occurence matrix," *Pattern Recognition Letters*, pp. 243–251, 1985.

[33] Chang, S. J. and Guddanti, S., "Application of high speed image processing in hardwood sawing research," in *Proceedings of 5th International Conference on Scanning Technology and Process Control for the Wood Products Industry*, (Atlanta, GA), October 25-27, 1993.

[34] Chang, S. J. and Olson, J. R., "Nuclear magnetic resonance imaging of hardwood logs," in *Proceedings of 2nd International Conference on Scanning Technology in Sawmilling* (Szymani, R., ed.), vol. 7, (San Francisco, CA), October 1-2, 1987.

[35] Chang, Y. and Li, X., "Adaptive image region-growing," *IEEE Transactions on Image Processing*, vol. 3, pp. 868–872, November 1994.

[36] Cheng, J. C. and Moura, M. F., "Capture and representation of human walking in live video sequences," *IEEE Transactions on Multimedia*, vol. 1, pp. 144–156, June 1999.

[37] Cheriet, M., Said, J. N., and Suen, C. Y., "A recursive thresholding technique for image segmentation," *IEEE Transaction on Image Processing*, vol. 7, no. 6, pp. 918–920, 1998.

[38] Chow, C. K. and Kaneko, T., "Automatic boundary detection of the left-ventricle from cineangiograms," *Computed Biomedical Research*, vol. 5, pp. 388–410, 1972.

[39] Chu, C. and Aggarwal, J., "The integration of image segmentation maps using region and edge information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 1241–1252, December 1993.

[40] Coleman, T., Branch, M. A., and Grace, A., *Optimization Toolbox for Use with MATLAB*. Natick, MA: The Mathworks, Inc., 1999.

[41] Comer, M. and Delp, E., "Segmentation of textured images using a multiresolution gaussian autoregressive model," *IEEE Transacations on Image Processing*, vol. 8, pp. 408–420, March 1999.

[42] Comer, M. and Delp, E., "The em/mpm algorithm for segmentation of textured images: Analysis and further experimental results," *IEEE Transacations on Image Processing*, vol. 9, pp. 1731–1744, October 2000.

[43] Cortes, C. and Hertz, J. A., "A network system for image segmentation," *Proceedings of International Conference on Neural Network*, vol. 1, pp. 121–125, 1989.

[44] Crow, D. N. and Smith, B. M., "DB habits: Comparing minimal knowledge and knowledge-based approaches to pattern recognition in the domain of user-computer interaction," in *Pattern Recognition and Neural Networks in Human-computer Interaction* (Beale, R. and Finlay, J., eds.), (Chichester, UK), pp. 39–63, 1992.

[45] Cufi, X., Munoz, X., Freixenet, J., and Marti, J., "A concurrent region growing algorithm guided by circumscribed contours," in *International Conference on Pattern Recognition*, vol. 1, (Barcelona, Spain), pp. 432–435, 2000.

[46] Cypher, A., ed., *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press, 1993.

[47] Davis, L., "A survey of edge detection techniques," *Computer Vision, Graphics, and Image Processing*, vol. 4, pp. 248–270, September 1975.

[48] Deravi, F. and Pal, S. K., "Gray level thresholding using second-order statistics," *Pattern Recognition Letters*, pp. 417–422, 1983.

[49] Derin, H., Elliott, H., Cristi, R., and Geman, D., "Bayes smoothing algorithms for segmentation of binary images modeled by markov random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 707–720, November 1984.

[50] Dondes, P. A. and Rosenfeld, A., "Pixel classification on gray level and local "busyness"," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, pp. 79–84, 1982.

[51] Dougherty, E. R. and Astola, J., *Introduction to Non-linear Image Processing*. Bellingham, WA: SPIE, 1994.

[52] Edwards, G. J., Taylor, C. J., and Cootes, T. F., "Improving identification performance by integrating evidence from sequences," in *Proceeding of Computer Vision and Pattern Recognition*, pp. 1486–1491, 1999.

[53] Falah, R., Bolon, P., and Cocquerez, J., "A region-region and region-edge cooperative approach of image segmentation," *International Conference on Image Processing*, vol. 3, pp. 470–474, October 1994.

[54] Farm, J. R. and Deutsch, E. W., "On the quantitative evaluation of edge detection schemes and their comparison with human performance," *IEEE Transaction on Computer*, vol. 24, pp. 616–628, June 1975.

[55] Fayyad, U. M. and Irani, K., "The attribute selection problem in decision tree generation," in *Proceedings of the 11th National Conference on Artificial Intelligence*, (San Jose, CA), pp. 322–327, MIT Press, 1992.

[56] Fayyad, U. M. and Irani, K. B., "On the handling of continous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, pp. 87–102, 1992.

[57] Fayyad, U. M. and Irani, K. B., "Multi-interval discretization of continous-valued attributes for classification learning," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (Bajcsy, R., ed.), (Amherst, MA), pp. 1022–1027, Morgan Kaufmann Publishing Inc., 1993.

[58] Fisher, R. A., "The use of multiple measurements in taxomonic problems," *Annual Eugenics*, vol. II, no. 7, pp. 179–188, 1936.

[59] Freixenet, J., Munoz, X., Raba, D., Marti, J., and Cufi, X., "Yet another survey on image segmentation: Region and boundary information integration," in *European Conference on Computer Vision* (Heyden, E., ed.), pp. 408–422, 2002.

[60] Fu, K. S. and Mui, J. K., "A survey on image segmentation," *Pattern Recognition*, vol. 13, no. 1, pp. 3–16, 1981.

[61] Funt, B. V. and Bryant, E., "Detection of internal log defects by automatic interpretation of computer tomography images," *Forest Products Journal*, vol. 37, no. 1, pp. 56–62, 1987.

[62] Gambotto, J., "A new approach to combining region growing and edge detection," *Pattern Recognition Letters*, vol. 14, pp. 869–875, 1993.

[63] Geman, S. and Geman, D., "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, November 1984.

[64] Ghosh, A., Pal, N. R., and Pal, S. K., "Image segmentation using neural networks," *Biological Cybernetics*, vol. 66, no. 2, pp. 151–158, 1991.

[65] Glover, F., "Improved linear programming models for discriminant analysis," *Decision Sciences*, vol. 21, pp. 771–785, 1990.

[66] Gokmen, M. and Li, C., "Edge detection and surface reconstruction using refined regularization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 492–499, May 1993.

[67] Gonzales, R. C. and Wintz, P., *Digital Image Processing*. Massachusetts: Addison-Wesley, 1987.

[68] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*. New York: Addison-Wesley, 1992.

[69] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*. Upper Saddle River, NJ: Addison-Wesley, 2nd ed., 2002.

[70] Green, D. M. and Swets, J. A., *Signal Detection Theory and Psychophysics*. John Wiley and Sons, 1966.

[71] Haddon, J. and Boyce, J., "Image segmentation by unifying region and boundary information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 929–948, October 1990.

[72] Haddon, J. and Boyce, J., "Image segmentation by unifying region and boundary information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 929–948, October 1990.

[73] Hadwiger, H., *Vorlesungen Über Inhalt, Oberflœche und Isoperimetrie*. Berlin: Springer-Verlag, 1957.

[74] Hagman, P. O. G. and Grundberg, S. A., "Classification of Scots pine (*pinus sylvestris*) knots in density images from CT scanned logs," *Hols als Roh- und Werkstoff*, vol. 53, pp. 75–81, 1995.

[75] Hall, E. L., *Computer Image Processing and Recognition*. New York: Academic Press, 1979.

[76] Han, W. and Birkeland, R., "Ultrasonic scanning for internal log defects," in *Proceedings of the 4th International Conference on Scanning Technology in the Wood Industry*, (San Francisco, CA), Miller-Freeman Publishing Company, 1991.

[77] Hanley, J. A. and McNeil, B. J., "The meaning and use of the area under a reciever operating characteristics (ROC) curve," *Radiology*, vol. 143, pp. 29,36, April 1982.

[78] Hansen, F. and Elliott, H., "Image segmentation using simple markov field models," *Computer Vision, Graphics, and Image Processing*, vol. 20, pp. 101–132, 1982.

[79] Hansen, M. and Higgins, W., "Relaxation methods for supervised image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 949–962, September 1997.

[80] Haralick, R. M., "Performance characterization in computer vision," *Computer Vision, Graphics, and Image Processing*, vol. 60, pp. 245–249, September 1994.

[81] Haralick, R. M. and Shapiro, L. G., "Survey: Image segmentation techniques," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 100–132, January 1985.

[82] Haralick, R. M. and Shapiro, L. G., *Computer and Robot Vision*. New York: Addison-Wesley, 1992.

[83] Haralick, R. M., Sternberg, S. R., and Zhuang, X., "Image analysis using mathematical morphology," *IEEE Transaction on Pattern Analysis Machine Intelligence*, vol. 9, no. 4, pp. 532–550, 1987.

[84] Haralick, R., "Digital step edges from zero-crossings of second directional derivatives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 58–68, January 1984.

[85] Haralick, R. and Kelly, G., "Pattern recognition with measurement space and spatial clustering for multiple images," *Proceedings of IEEE*, vol. 57, pp. 654–665, April 1969.

[86] Heijmans, H., "Theoretical aspects of gray-level morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 568–582, 1991.

[87] Heijmans, H. J. A. M. and Roerdink, J. B. T. M., eds., *Mathematical Morphology and its Applications to Image and Signal Processing*. Boston: Kluwer Academic Publishers, 1998.

[88] Hojjatoleslami, S. and Kittler, J., "Region growing: A new approach," *IEEE Transacations on Image Processing*, vol. 7, pp. 1079–1084, July 1998.

[89] Hoveer, A., Baptiste, G., Jiang, X., Flynn, P. J., Bunke, H., Goldgof, D. B., Bowyer, K., Eggert, D. W., Fitzgibbon, A., and Fisher, R. B., "An experimental comparison of range image segmentation algorithms," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, pp. 673–689, July 1998.

[90] Huang, Q. and Dom, B., "Quantitative methods of evaluating image segmentation," *IEEE International Conference on Image Processing*, pp. 53–56, December 1995.

[91] Hunink, M. G. M., Deslegte, R. G. M., and Hoogesteger, M. F., "ROC analysis of the clinical, CT and MRI diagnosis of orbital space occupying lesions," *ORBIT-The International Journal on Orbital Disorders, Oculoplastic and Lacrimal Surgery*, vol. 8, September 1989.

[92] Jacobus, C. and Chien, R., "Two new edge detectors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, pp. 581–592, September 1981.

[93] Jain, A., "Advances in mathematical models for image processing," *Proceedings of IEEE*, vol. 69, pp. 502–528, May 1981.

[94] Kelemen, A., Szekely, G., and Gerig, G., "Elastic model-based segmentation of 3D neuroradiology data sets," *IEEE Transaction on Medical Imaging*, vol. 18, pp. 828–839, October 1999.

[95] Kittler, J., Eggleton, J., Illingworth, J., and Paler, K., "An averaging edge detector," *Pattern Recognition Letters*, vol. 6, no. 1, pp. 27–32, 1987.

[96] Klein, J. and Serra, J., "The texture analyzer," *Journal of Microscopy*, vol. 95, 1972.

[97] Kohler, R., "A segmentation system based on thresholding," *Computer Vision Graphics, and Image Processing*, vol. 15, pp. 319–338, 1981.

[98] Kulpa, Z., "Area and perimeter measurement of blobs in discrete binary pictures," *Computer Vision, Graphics and Image Processing*, vol. 6, pp. 434–454, 1977.

[99] Kurlander, D., *Graphical Editing by Example*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 1993.

[100] Lau, T., *Programming by Demontration: a Machine Learning Approach*. PhD thesis, Department of Computer Science, University of Washington, Seattle, WA, 2001.

[101] Lau, T., Domingos, P., and Weld, D. S., "Version space algebra and its application to programming by demonstration," in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 527–534, June 2000.

[102] Lee, S. U., Chung, S. Y., and Park, R. H., "A comparative performance study of several global thresholding techniques for segmentation," *Computer Vision, Graphics, and Image Processing*, vol. 52, pp. 171–190, 1990.

[103] Levine, M. and Nazif, A. M., "Dynamic measurement of computer generated image segmentations," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 155–164, March 1985.

[104] Li, L., Gong, J., and Chen, W., "Gray-level thresholding based on fisher linear projection of two-dimensional histogram," *Pattern Recognition*, vol. 30, no. 5, pp. 743–749, 1997.

[105] Li, P., Abbott, A., and Schmoldt, D. L., "Automated analysis of CT images for the inspection of hardwood logs," in *Proceedings of International Conference on Neural Networks*, (Washington, DC), June 1996.

[106] Lieberman, H., "Mondrian: A teachable graphical editor," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 341–358, Cambridge, MA: MIT Press, 1993.

[107] Lieberman, H., "Tinker: A programming by demonstration system for beginning programmers," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 49–64, Cambridge, MA: MIT Press, 1993.

[108] Lieberman, H., *Your Wish is My Command: Programming By Example*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2001.

[109] Lloyd, D. E., "Automatic target classification using moment invariants of image shapes," Tech. Rep. RAE IDN AW126, Farnborough, UK., 1985.

[110] Lu, S. and Xu, H., "Textured image segmentation using autoregressive model and artificial neural-network," *Pattern Recognition*, vol. 28, pp. 1807–1817, December 1995.

[111] Lunscher, W. and Beddoes, M., "Optimal edge detector design i: Parameter selection and noise effects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 164–177, March 1986.

[112] Lunscher, W. and Beddoes, M., "Optimal edge detector design ii: Coefficient quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 178–187, March 1986.

[113] Mangasarian, O. L., "Linear and nonlinear separation of patterns by linear programming," *Operations Research*, vol. 13, pp. 444–452, 1965.

[114] Mangasarian, O. L., "Multi-surface method of pattern separation," *IEEE Transactions on Information Theory*, vol. IT-14, pp. 801–807, 1968.

[115] Mangasarian, O. L., "Mathematical programming in neural networks," *ORSA Journal on Computing*, vol. 5, pp. 349–360, 1993.

[116] Mangasarian, O. L., "Misclassification minimization," *Journal of Global Optimization*, vol. 5, pp. 309–323, 1994.

[117] Mantas, J., "Methodologies in pattern recognition and image analysis: a brief survey," *Pattern Recognition*, vol. 20, pp. 1–6, 1987.

[118] Maragos, P., Schafer, R. W., and Butt, M. A., eds., *Mathematical Morphology and its Applications to Image and Signal Processing*. Boston, MA: Kluwer Academic Publishers, 1996.

[119] Matheron, G., *Random Sets and Integral Geometry*. New York: John Wiley, 1975.

[120] Maulsby, D., "Inducing procedures interactively: Adventures with metamouse," Master's thesis, Department of Computer Science, University of Calgary, Calgary, December 1988.

[121] Maulsby, D. and Witten, I. H., "Metamouse: An instructible agent for programming by demonstration," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 155–181, Cambridge, MA: MIT Press, 1993.

[122] Maulsby, D. L., Witten, I. H., and Kittlitz, K. A., "Metamouse: specifying graphical procedures by example," in *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 127–136, ACM Press, 1989.

[123] McClelland, J. and Rumelhart, D., *Parallel Distributed Processing*. Cambridge,MA: MIT Press, 1986.

[124] McMillin, C. W., "Applications of automatic image analysis to wood science," *Wood Science*, vol. 14, no. 3, pp. 97–105, 1982.

[125] Mehnert, A. and Jackway, P., "An improved seeded region growing algorithm," *Pattern Recognition Letters*, vol. 18, pp. 1065–1071, October 1997.

[126] Minkowski, H., "Volumen und oberflache," *Mathematische Annalen*, vol. 57, pp. 447–495, 1903.

[127] Minkowski, H., *Gesammelte Abhandlungen*. Leipzig-Berlin: Teubner Verlag, 1911.

[128] Mitchell, T. M., *Machine Learning*. Boston, MA: McGraw-Hill, 1997.

[129] Murthy, S. K., Kasif, S., and Salzberg, S., "A system for induction of oblique decision trees," *Journal of Artificial Intelligence Research*, vol. 2, pp. 1–33, 1994.

[130] Myers, B. A., "Demonstrational interfaces: A step beyond direct manipulation," *Computer*, vol. 25, pp. 61–73, August, 1992.

[131] Myers, B. A., "Peridot: Creating user interfaces by demonstration," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 125–153, Cambridge, MA: MIT Press, 1993.

[132] Nakagawa, Y. and Rosenfeld, A., "A note on the use of local min and max operation in digital picture processing," *IEEE Transaction of System, Man, and Cybernetics*, vol. SMC-8, pp. 632–635, 1978.

[133] Nix, R. P., *Editing by Example*. PhD thesis, Department of Computer Science, Yale University, New Haven, CT, 1983.

[134] Occeña, L. G., "Computer integrated manufacturing issues related to the hardwood log sawmill," *Journal of Forest Engineering*, vol. 3, no. 1, pp. 39–45, 1991.

[135] Otsu, N., "A threshold selection method from gray level histograms," *IEEE Transactions on Systems Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[136] Pal, N. R. and Pal, S. K., "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, pp. 1277–1294, September 1993.

[137] Pal, N. and Pal, S. K., "Entropic thresholding," *Signal Processing*, vol. 16, pp. 97–108, 1989.

[138] Papamarkos, N., Strouthopoulos, C., and Andreadis, I., "Multithresholding of color and gray-level images through a neural network technique," *Image and Vision Computing*, vol. 18, pp. 213–222, February 2000.

[139] Pavlidis, T. and Liow, Y., "Integrating region growing and edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 225–233, March 1990.

[140] Peli, T. and Malah, D., "A study of edge detection algorithms," *Computer Vision, Graphics, and Image Processing*, vol. 20, no. 1, pp. 1–21, 1982.

[141] Perez, A. and Gonzalez, R. C., "An iterative thresholding algorithm for image segmentation," *IEEE Pattern Analysis and Machine Intelligence*, vol. 9, pp. 742–751, 1987.

[142] Pitas, I. and Venetsanopoulos, A. N., *Nonlinear Digital Filters: Principles and Applications*. Boston,MA: Kluwer Academic Publishers, 1990.

[143] Pluempitiwiriyawej, C., Moura, J. M. F., Wu, Y. L., and Ho, C., "Cardiac MR image segmentation: Quality assessment of STACS," *IEEE International Symposium on Bioimaging*, April 2004.

[144] Potter, R., "TRIGGERS: Guiding automation with pixels to achieve data access," in *Watch What I Do – Programming by Demonstration* (Cypher, A., ed.), pp. 361–380, Cambridge, MA: MIT Press, 1993.

[145] Potter, R. and Shneiderman, B., "Pixel data access for end-user programming and graphical macros," Tech. Rep. CS-TR-4019, Department of Computer Science, University of Maryland, College Park, MD, 1999.

[146] Prager, J., "Extracting and labeling boundary segments in natural scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, pp. 16–26, January 1980.

[147] Provost, F., Fawcett, T., and Kohavi, R., "The case against accuracy estimation for comparing induction algorithms," in *Proceedings 15th International Conference on Machine Learning*, (San Francisco, CA), pp. 445–453, Morgan Kaufmann Publishers Inc., 1998.

[148] Pun, T., "A new method ofr gray level picture thresholding using the entropy of the histogram," *Signal Processing*, pp. 210–239, 1980.

[149] Quinlan, J. R., "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[150] Quinlan, J. R., *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufman, 1993.

[151] Rees, G., Greenway, P., and Morray, D., "Metrics for image segmentation," in *Proceedings of SPIE, Visual Information Processing VII* (Park, S. K. and Juday, R. D., eds.), vol. 3387, (Orlando, FL), pp. 199–210, April 1998.

[152] Richards, D. B., Adkins, W. K., Hallock, H., and Bulgrin, E. H., "Simulation of hardwood log sawing," Tech. Rep. FPL-355, USDA Forest Products Research Lab, Madison, WI, 1979.

[153] Rosenfeld, A., "Survey, image analysis and computer vision," *Computer Vision Graphics, and Image Processing*, vol. 46, pp. 196–250, 1989.

[154] Rosenfeld, A., Hummel, R., and Zucker, S., "Scene labeling by relaxation operations," *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 6, pp. 420–433, June 1976.

[155] Rosenfeld, A. and Kak, A. C., *Digital Picture Processing*. New York: Academic Press, 1982.

[156] Sahoo, P. K., Soltani, S., Wong, A. K. C., and Chen, Y. C., "A survey of thresholding techniques," *Computer Vision Graphics, and Image Processing*, vol. 41, pp. 233–260, February 1988.

[157] Sarigul, E., Abbott, A. L., and Schmoldt, D. L., "Nondestructive rule based defect detection and identification system in CT images of hardwood logs," in *Review of Progress in Quantitative Nondestructive Evaluation* (Thompson, D. O. and Chimenti, D. E., eds.), vol. 20, pp. 1936–1943, 2000.

[158] Sarigul, E., Abbott, A. L., and Schmoldt, D. L., "Rule driven defect detection in CT images of hardwood logs," in *4th International Conference on Image Processing and Scanning of Wood*, (Mountain Lake, VA), pp. 37–51, 2000.

[159] Sarigul, E., Abbott, A. L., and Schmoldt, D. L., "Rule-driven defect detection in CT images of hardwood logs," *Computers and Electronics in Agriculture (COMPAG)*, 2001.

[160] Sarigul, E., Abbott, A. L., and Schmoldt, D. L., "Interactive machine learning for postprocessing CT images of hardwood logs," in *5th International Conference on Image Processing and Scanning of Wood*, (Wien, Austuria), March 2003.

[161] Sarigul, E., Abbott, A. L., and Schmoldt, D. L., "Progress in analysis of computed tomography (CT) of hardwood logs for defect detection," in *10th International Conference on Scanning technology and Process Optimization for the Wood Industry* (Szymani, R., ed.), (Seattle, WA), November 2003.

[162] Sato, M., Lakare, S., Wan, M., and Kaufman, A., "A gradient magnitude based region growing algorithm for accurate segmentation," in *International Conference on Image Processing*, vol. 3, pp. 448–451, 2000.

[163] Schmoldt, D. L., He, J., and Abbott, A. L., "Automated labeling of log features in CT imagery of multiple hardwood species," *Wood and Fiber Science*, vol. 32, no. 3, pp. 287–300, 2000.

[164] Schmoldt, D. L., Li, P., and Abbott, A. L., "Machine vision using artificial neural networks and 3D pixel neighborhoods," *Computers and Electronics in Agriculture (COMPAG)*, vol. 16, no. 3, pp. 255–271, 1997.

[165] Serra, J., *Image Analysis and Mathematical Morphology*. London: Academic Press, 1982.

[166] Serra, J. and Soille, P., eds., *Mathematical Morphology and its Applications to Image Processing*. Boston, MA: Kluwer Academic Publishers, 1994.

[167] Singh, S. and Al-Mansoori, R., "Identification of regions of interest in digital mammograms," *Journal of Intelligent Systems*, vol. 10, no. 2, pp. 183–217, 2000.

[168] Smith, D. C., *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. Birkhauser, Basel, 1977.

[169] Smith, D. C., "Pygmalion: An executable electronic blackboard," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 19–47, Cambridge, MA: MIT Press, 1993.

[170] Soile, P., "Grey scale convex hulls: Definition, implementation, and application," in *Proceedings of International Symposium on Mathematical Morphology*, pp. 83–90, 1998.

[171] Soille, P., *Morphological Image Analysis: Principles and Applications*. Berlin: Springer, 2003.

[172] Som, S., Wells, P., and Davis, J., "Automated feature extraction of wood from tomographic images," *2nd International Conference on Automation, Robotics and Computer Vision*, September 15-18, 1992.

[173] Sternberg, S. R., "Biomedical image processing," *Computer*, vol. 16, January 1983.

[174] Sternberg, S. R., "Overview of image algebra and related issues," in *Integrated Technology for Parallel Image Processing* (Levialdi, S., ed.), (London), Academic Press, 1985.

[175] Sternberg, S. R., "Grayscale morphology," *Computer Graphics and Image Processing*, vol. 35, pp. 333–335, 1986.

[176] Street, W. N., "Oblique multicategory decision trees using nonlinear programming," *INFORMS Journal on Computing*, vol. 4, no. 5, 2004.

[177] Swets, J. A. and Pickett, R. M., *Evaluation of diagnostic Systems: Methods from Signal Detection Theory*. New York: Academic Press, 1988.

[178] Taxt, T., Flynn, P. J., and Jain, A. K., "Segmentation of document images," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 11, no. 12, pp. 1322–1329, 1989.

[179] Taylor, F. W., Wagner, J. F. G., McMillin, C. W., Morgan, I. L., and Hopkins, F. F., "Locating knots by industrial tomography - a feasibility study," *Forest Products Journal*, vol. 34, no. 5, pp. 42–46, 1984.

[180] Tsolakides, J. A., "A simulation model for log yield study," *Forest Products Journal*, vol. 19, no. 7, pp. 21–26, 1969.

[181] Utgoff, P. and Brodley, C. E., "Linear machine decision trees," Tech. Rep. 91-10, University of Massachusetts, Amherst, MA, 1991.

[182] van Trees, H. L., *Detection, Estimation and Modulation Theory, Part I*. John Wiley and Sons, 1968.

[183] Vapnik, V., *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[184] Vincent, L., "Morphological transformations of binary images with arbitrary structuring elements," *Signal Processing*, vol. 22, no. 1, pp. 3–23, 1991.

[185] Wærn, A., "What is an intelligent interface?," Tech. Rep. 03-1997, Swedish Institute of Computer Science, Kista, Sweden, 1997.

[186] Wagner, F. G., Harless, T. E. G., Stelle, P. H., Taylor, F. W., Yadama, V., and McMillin, C. W., "Management of wood products: Technology for the 90's," *Proceedings of Process Control*, pp. 77–88, 1990.

[187] Wang, S. and Haralick, R. M., "Automatic multithreshold selection," *Computer Vision Graphics, and Image Processing*, vol. 25, pp. 46–67, 1984.

[188] Weszka, J. S. and A.Rosenfeld, "Threshold evaluation techniques," *IEEE Transaction on System Man, and Cybernetics*, vol. 8, pp. 622–629, 1978.

[189] Weszka, J., "Survey of threshold selection techniques," *Computer Vision, Graphics and Image Processing*, pp. 259–265, 1978.

[190] Witten, I. H. and Frank, E., *Data Mining.* San Francisco: Morgan Kaufmann Publisher Inc., 2000.

[191] Witten, I. H. and MacDonald, B., "Using concept learning for knowledge acquisition," *International Journal of Man-Machine Studies*, vol. 29, pp. 171–196, August 1988.

[192] Witten, I. H. and Mo, D. H., "Tels: Learning text editing task from examples," in *Watch What I Do: Programming by Demonstration* (Cypher, A., ed.), pp. 183–203, Cambridge, MA: MIT Press, 1993.

[193] Witten, I. H. and Mo, D. H., "The turvy experience: simulating an instructible user interface," in *Watch What I Do: Programming by Demonstration* (A.Cypher, ed.), pp. 239–269, Cambridge, MA: MIT Press, 1993.

[194] Wong, A. K. C. and Sahoo, P. K., "A grey level threshold selection method based on maximum entropy principle," *IEEE Transaction on Systems, Man, and Cybernetics*, pp. 866–871, 1989.

[195] Wu, A. Y., Hong, T., and Rosenfeld, A., "Threshold selection using quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 4, pp. 90–94, 1982.

[196] Xiaohan, Y., Yla-Jaaski, J., and Baozong, Y., "A new algorithm for texture segmentation based on edge detection," *Pattern Recognition*, vol. 24, no. 11, pp. 1105–1112, 1991.

[197] Yanowitz, S. D. and Bruckstein, A. M., "A new method for image segmentation," *Computer Vision Graphics and Image Processing*, vol. 46, pp. 82–95, 1989.

[198] Yasnoff, W. A., Mui, W. A., and Bacus, J. W., "Error measures in scene segmentation," *Pattern Recognition*, vol. 9, no. 4, pp. 217–217, 1977.

[199] Zhang, R., Tsai, P., Cryer, J., and Shah, M., "Analysis of shape from shading techniques," *Computer Vision and Pattern Recognition*, pp. 377–384, 1994.

[200] Zhang, Y. J., "A survey on evaluation methods for image segmentation," *Pattern Recognition*, vol. 29, pp. 1335–1346, December 1996.

[201] Zhou, Y., Venkateswar, V., and Chellappa, R., "Edge detection and linear feature extraction using a 2-d random field model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 84–95, January 1989.

[202] Zhu, D., *A Feasibility Study on Using CT Image Analysis for Hardwood Log Inspection*. PhD thesis, Department of Electrical Engineering and Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1993.

[203] Zhu, D., Conners, R. W., and Araman, P., "3-D signal processing in a computer vision system," *IEEE International Conference on Systems Engineering*, pp. 457–460, August 1-3, 1991.

[204] Zhu, D., Conners, R. W., and Beex, A. A., "Stochastic field-based object recognition in computer vision," *Proceedings of SPIE - The International Society of Optical Engineering*, vol. 1569, pp. 174–181, July 21-26, 1991.

[205] Zhu, D., Conners, R. W., Lamb, F. M., Schmoldt, D., and Araman, P., "A computer vision system for locating and identifying internal log defects using CT imagery," in *Proceedings of 4th International Conference on Scanning Technology in Sawmilling* (Szymani, R., ed.), (San Francisco, CA), October 28-31, 1991.

[206] Zhu, D., Conners, R. W., Schmoldt, D., and Araman, P., "CT image sequence analysis for object recognition - rule-based 3D computer vision system," *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 173 – 178, October 13-16, 1991.

[207] Zhu, D. P., Conners, R. W., and Schmoldt, D. L., "Nondestructive evaluation of hardwood logs using automated interpretation of CT images," in *Review of Progress in Quantitative Nondestructive Evaluation* (Thompson, D. O. and Chimenti, D., eds.), vol. 12, (New York), pp. 2257–2264, Plenum Press, 1993.

[208] Zijdenbos, A., Dawant, B., and Margolin, R., "Morphometric analysis of white matter leisons in MR images: Method and validation," *IEEE Transcation on Medical Imaging*, vol. 13, pp. 716–724, December 1994.

Vita

Erol Sarigul was born in Erzincan, Turkey in 1969. After his high school education at Erzincan, he went to Dokuz Eylul University, in Izmir (Turkey) for a Bachelor of Engineering degree in Electronics and Communication Engineering. After graduating with distinction of 3rd in class rank in 1992, he worked in industry as technical computer consultant for providing technical support for contracted companies. Due to a desire to have good graduate studies in US, he came to Virginia Tech to obtain a Master of Science degree in Electrical Engineering. He graduated with an M.S. in December 1998. Due to the interest on Image and Signal Processing strongly suggested him to pursue his Ph.D.