COMPUTERIZED IMAGE ANALYSIS OF SEED BORNE FUNGUS

P-2173

A PROJECT REPORT

Submitted by

A. EMILY	71204205007
G. SOWMYA RAO	71204205051
Y. SPOORTHI	71204205052

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2008

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report "**COMPUTERIZED IMAGE ANALYSIS OF SEED BORNE FUNGUS**" is the bonafide work of

A. EMILY	71204205007
G. SOWMYA RAO	71204205051
Y. SPOORTHI	71204205052

who carried out the project work under my supervision.

[illegible]

SIGNATURE

Dr. S. Thangasamy

HEAD OF THE DEPARTMENT

Dept. of Information Technology,
Kumaraguru College of Technology,
Technology,
Coimbatore - 641006

S. Vani

SIGNATURE

Mrs. S. Vani

SUPERVISOR

Lecturer

Dept. of Information Technology,
Kumaraguru College of

Coimbatore - 641006

The candidates with University Register Numbers 71204205007, 71204205051, 71204205052 were examined by us in the project viva-voce examination held on 24.04.2008

[illegible]

INTERNAL EXAMINER

[illegible] 24/04/08

EXTERNAL EXAMINER

DECLARATION

We,

A. EMILY	**71204205007**
G. SOWMYA RAO	**71204205051**
Y. SPOORTHI	**71204205052**

Declare that the project entitled "**COMPUTERIZED IMAGE ANALYSIS OF SEED BORNE FUNGUS**", submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of **Mrs. S. Vani, M.E.**, Lecturer, Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore

Date: 9/4/08

Emily. A

[A.Emily]

G. Sowmya Rao

[G.Sowmya Rao]

Spoorthi

[Y.Spoorthi]

Project Guided by

S. Vani

[Mrs. S.Vani]

ACKNOWLEDGEMENT

We are extremely thankful to **Dr. Joseph V Thanikal, B.E., M.E., Ph.D., PDF., CEPIT,** Principal, Kumaraguru College of Technology, Coimbatore, for permitting us undertake a project work.

We sincerely thank **Dr. S. Thangasamy, B.E.(Hons), Ph.D.,** Head of the Department, Information Technology, Kumaraguru College of Technology, Coimbatore, for the immense support he has provided throughout our project.

We are grateful to our project coordinator **Prof. K.R.Baskaran, B.E., M.S.,** Assistant Professor, Department of Information Technology, for his constant support and encouragement.

We would like to express our heartfelt gratitude to our guide, **Mrs. S.Vani, M.E.,** Lecturer, Department of Information Technology, for her everlasting counseling and untiring help throughout our project.

We like to express our special thanks to **Mr. B.N.Sriharan, DME (Sandwich),B.B.A.,G.I.E.,** Department of Mechanical Engineering, other staff members and lab technicians of the Department of Information Technology for their continuous support.

We are grateful to **Dr. Valluva Paridasan, Ph.D,** Controller of Examination and **Mr. Manickam Rajesh, 2nd year Ph.D student** from Tamil Nadu Agricultural University, Coimbatore for their extended help and support.

We would be failing in our duty if we do not express our gratitude to all those authors who have contributed abundant literature through various research papers, books, etc. in many national and international journals and conferences.

Finally we thank **the Almighty, our parents and family members** for the blessings they have showered upon us.

ABSTRACT

Fungi are important plant pathogens. Their wide diversity and variability make the identification of different genus of fungi a difficult task for non-specialized technicians and research workers. The proposed system enables the identification of different genus of fungi such as Helminthosporium, Alternaria and Curvularia. The identification is done by means of computerized image analysis, which helps in exactly determining the elaborate morphological traits in microscopic pictures. The system proposed here is aimed at designing an automated procedure for determining various morphological parameters. These parameters are used to identify the different genus of fungi. The various morphological parameters that are calculated are size, area, perimeter, maximum diameter D_{max} (i.e. length), minimum diameter D_{min}, circular form factor F_{circle}, F_{shape} and D_{circle}.

The images of the fungus which affects the cereals like rice, wheat etc. are given as input to the system through Visual Basic. The input image is then processed using MATLAB to calculate the specified parameters and the genus of the fungus is identified by training the system using Neural Networks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

F_{circle}	-	Circular Form Factor
D_{max}	-	Maximum Diameter
D_{min}	-	Minimum Diameter
A	-	Area
P	-	Perimeter

LIST OF ABBREVIATIONS

MATLAB	-	Matrix Laboratory
COM	-	Component Object Model
VB	-	Visual Basic
ANN	-	Artificial Neural Network
NN	-	Neural Network

INTRODUCTION

1. INTRODUCTION

Image Processing is a powerful tool that has been applied in many domains such as intelligent remote sensing via satellite, medical image analysis, radar, sonar, robotics and automated inspection. Image information can play a crucial role in identifying the different genera of fungus in the agricultural domain. These fungi infect the cereals like rice, wheat and cause diseases in them. They also cause disorders in humans, when these affected cereals are consumed. Also when we import these cereals we need to check at the quarantine whether the cereals are infected by any fungus. Hence there is a necessity to identify the infected cereals.

1.1. PROPOSED SYSTEM

1.1.1. OVERVIEW

The input to the proposed system is the image of a fungus. This image is at first enhanced by equalizing the background illumination. From this enhanced image, the background is eliminated to obtain the object of interest i.e. the fungus. The extra noise surrounding the fungus is removed. Then the features of the fungus such as area, perimeter, length, breadth etc. are extracted. These extracted features serve as a set of parameters to be given as inputs to the Neural Networks so as to train the system. The output of Neural Networks helps in identifying the genus of the fungus and thereby classifying it suitably.

The system obtains the image of fungus as an input using Visual Basic Interface. The input image is then processed using MATLAB in order to extract the features and then the genus of the fungus is identified by training the system using Neural Networks and the results are displayed to the user via Visual Basic Interface.

The block diagram of the Image Analysis system is shown below

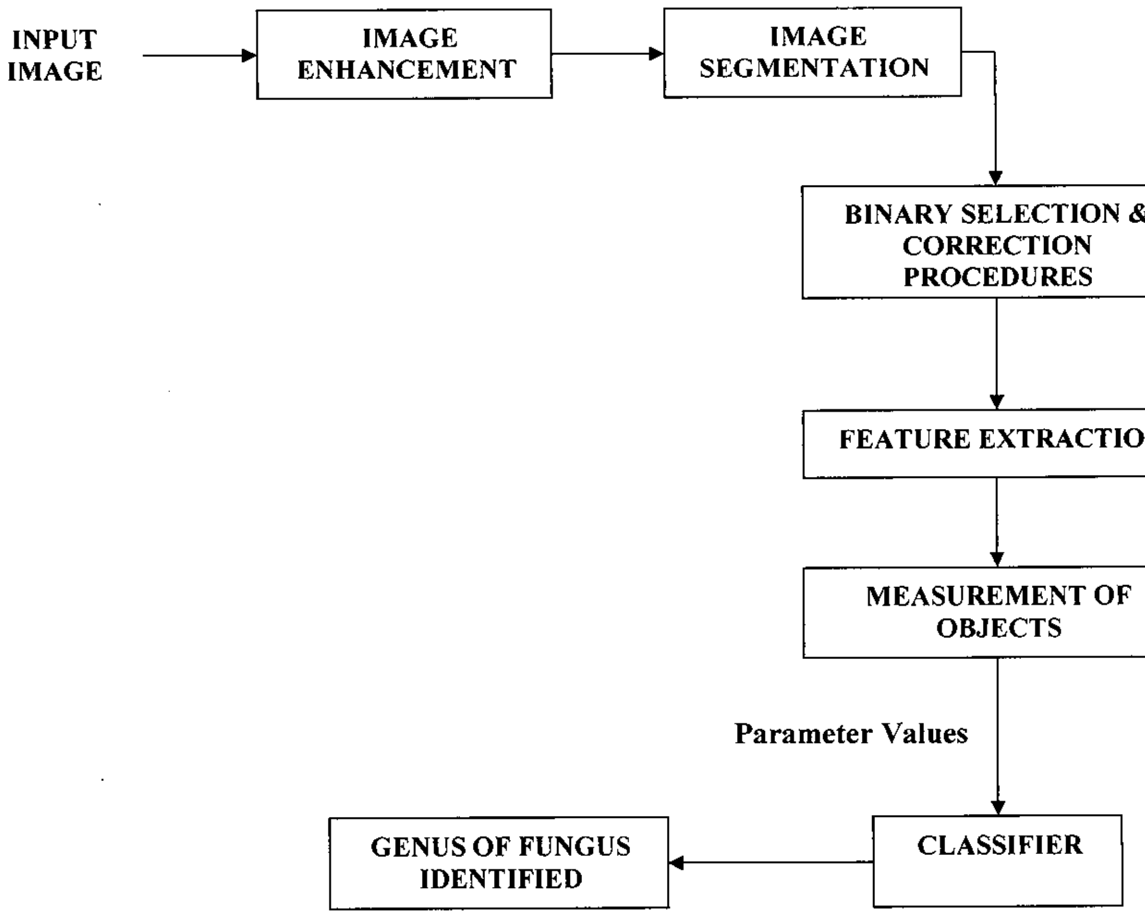


Figure 1.1.1. (a) Block Diagram of Image Analysis System

SYSTEM REQUIREMENT ANALYSIS

2. SYSTEM REQUIREMENTS ANALYSIS

2.1. SYSTEM REQUIREMENTS

2.1.1 HARDWARE REQUIREMENTS

Processor	: Pentium IV 3.3 GHz
RAM	: 512 MB
Hard Disk	: 40 GB
Monitor	: SVGA color
Microscope	: Optika Microscope, N-400T (Made in Italy), Halogen lamp 6v 20w
Camera	: Nikon E995 (Made in Japan)

2.2.2. SOFTWARE SPECIFICATION

Platform	: Windows 2000, XP
Software	: MATLAB 7.2 Microsoft Visual Basic 6.0 Macromedia Flash 6

2.2. SYSTEM ANALYSIS

The primary goal of this phase is to create a detailed Functional Specification defining the full set of system capabilities to be implemented, along with accompanying data and process models illustrating the information to be managed and the process to be supported by the new system.

All analysis methods are related by a set of operational principles:

- The information domain of a problem must be represented and understood.
- The functions that the software is to perform must be defined.
- The behavior of the software must be defined.

- The model that depicts information function and behavior must be portioned.
- The analysis process should move from essential information towards implementation detail.

By applying these principles we approach a problem systematically. The information domain is examined so that the functions may be understood completely. Partitioning is applied to reduce complexity.

2.3. FEASIBILITY ANALYSIS

All projects are feasible-given unlimited resources and infinite time. But the development of computer-based system is likely to be plagued by scarcity of resources and limited time. A system which is ill conceived, if recognized early will avert month or years of effort, thousands of dollars, professionals embarrassment etc.

Estimation of resources, cost and schedule for a software development effort requires experience, access to good historical information and the courage to commit to quantitative measures when qualitative data is all that exists. Estimation carries inherent risk and it is this that leads to uncertainty.

While discussing about feasibility we need to concentrate on the points given below

- Economic Feasibility
- Technical Feasibility
- Information and Data Feasibility
- Control and Security Feasibility
- Behavioral Feasibility

2.4. SOFTWARE OVERVIEW

2.4.1. MATLAB

MATLAB is a high-performance language for technical computing. The name MATLAB stands for matrix laboratory. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning.

MATLAB features a family of add-on application-specific solutions called ***toolboxes.*** Very important to most users of MATLAB, toolboxes allow one to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include many domains out of which the proposed system uses Image Processing toolbox and Neural Networks

2.4.1.1. IMAGE PROCESSING TOOLBOX

The Image Processing Toolbox is a collection of functions that extend the capability of the MATLAB numeric computing environment. The toolbox supports a wide range of image processing operations

- Spatial image transformations
- Morphological operations
- Neighborhood and block operations
- Linear filtering and filter design
- Transforms Image analysis and enhancement
- Deblurring
- Region of interest operations

2.4.1.1.1. REPRESENTING IMAGES IN MATLAB

The basic data structure in MATLAB is the array, an ordered set of real or complex elements. This object is naturally suited to the representation of images, real-valued ordered sets of color or intensity data. MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as RGB, require a three-dimensional array, where the first plane in the third dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. This convention makes working with images in MATLAB similar to working with any other type of matrix data, and makes the full power of MATLAB available for image processing applications

The Image Processing Toolbox supports four basic types of images:

- Indexed images
- Intensity images
- Binary images
- RGB images

MATLAB provides commands for reading, writing and displaying several types of graphics file format images. As with MATLAB generated images, once a graphics file format image is displayed, it becomes a Handle Graphics Image object.

MATLAB supports the following graphics file formats:

- BMP (Microsoft Windows Bitmap)
- HDF (Hierarchical Data Format)

- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- XWD (X Window Dump)

2.4.1.2. MATLAB FUNCTIONS

The MATLAB functions made use of are as follows:

1. imread

Read image from graphics file

Syntax

A = imread(filename,fmt)

Description

The imread function reads an image from any supported graphics image file format, in any of the supported bit depths. Most image file formats use 8 bits to store pixel values. When these are read into memory, MATLAB stores them as class uint8. For file formats that support 16-bit data, PNG and TIFF, MATLAB stores the images as class uint16. imread infers the file format to use from the contents of the file. The file format can also be specified as an argument to imread.

2. rgb2gray

Convert an RGB image or colormap to grayscale

Syntax

I = rgb2gray(RGB)

Description

rgb2gray converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

I=rgb2gray(RGB) converts the true-color image RGB to the grayscale intensity image I.

3. strel

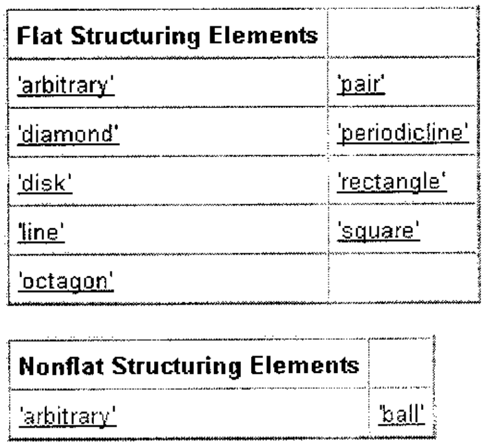
Create morphological structuring element

Syntax

SE = strel(shape,parameters)

Description

SE = strel(shape,parameters) creates a structuring element, SE, of the type specified by shape. This table lists all the supported shapes. Depending on shape, strel can take additional parameters.



Flat Structuring Elements	
'arbitrary'	'pair'
'diamond'	'periodicline'
'disk'	'rectangle'
'line'	'square'
'octagon'	

Nonflat Structuring Elements	
'arbitrary'	'ball'

Table 2.4.1.2. (a) Flat and Non flat structuring Elements

SE = strel('diamond',R) creates a flat, diamond-shaped structuring element, where R specifies the distance from the structuring element origin to the points of the diamond. R must be a nonnegative integer scalar.

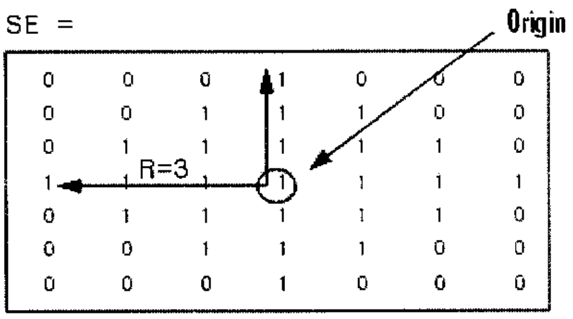


Figure 2.4.1.2. (a) Structuring Element

4. imopen

Morphological opening of an image is an erosion followed by a dilation, using the same structuring element for both operations.

Syntax

C=imopen(A, B)

Description

In C=imopen(A, B), A is a binary image and B is a matrix of 0's and 1's that specifies the structuring element. Morphological opening removes completely regions of an object that cannot contain the structuring element, smoothes object contours, breaks thin connections and removes thin protrusions.

5. imclose

Morphological closing of an image is dilation followed by erosion, using the same structuring element for both operations

Syntax

C=imclose(A, B)

Description

In C=imclose(A, B), A is a binary image and B is a matrix of 0's and 1's that specifies the structuring element. Morphological closing tends to smooth the contours of objects. It generally joins narrow breaks, fills long thin gulfs and fills holes smaller than the structuring element.

6. imsubtract

Subtract one image from another, or subtract a constant from an image

Syntax

Z = imsubtract(X,Y)

Description

Z = imsubtract(X,Y) subtracts each element in array Y from the corresponding element in array X and returns the difference in the corresponding element of the output array Z. X and Y are real, nonsparse numeric arrays of the same size and class, or Y is a double scalar. The array returned, Z, has the same size and class as X.

If X is an integer array, then elements of the output that exceed the range of the integer type are truncated, and fractional values are rounded.

If X and Y are double arrays, then you can use the expression X-Y instead of this function.

7. imadjust

Adjust image intensity values or colormap.

Syntax

J = imadjust(I,[low_in; high_in],[low_out; high_out])

Description

J = imadjust(I,[low_in; high_in],[low_out; high_out]) maps the values in intensity image I to new values in J such that values between low_in and high_in map to values between low_out and high_out. Values below low_in and above high_in are clipped; that is, values below low_in map to low_out, and those above high_in map to high_out. You can use an empty matrix ([]) for [low_in high_in] or for [low_out high_out] to specify the default of [0 1].

8. stretchlim

Find limits to contrast stretch an image

Syntax

LOW_HIGH = stretchlim(I,TOL)

LOW_HIGH = stretchlim(RGB,TOL)

Description

LOW_HIGH = stretchlim(I,TOL) returns a pair of intensities that can be used by imadjust to increase the contrast of an image.

TOL = [LOW_FRACT HIGH_FRACT] specifies the fraction of the image to saturate at low and high intensities.

If TOL is a scalar, TOL = LOW_FRACT, and HIGH_FRACT = 1 - LOW_FRACT, which saturates equal fractions at low and high intensities. If you omit the argument, TOL defaults to [0.01 0.99], saturating 2%.

If TOL = 0, LOW_HIGH = [min(I(:)) max(I(:))].

LOW_HIGH = stretchlim(RGB,TOL) returns a 2-by-3 matrix of intensity pairs to saturate each plane of the RGB image. TOL specifies the same fractions of saturation for each plane.

9. graythresh

Compute global image threshold using Otsu's method

Syntax

level = graythresh(I)

Description

level = graythresh(I) computes a global threshold (level) that can be used to convert an intensity image to a binary image with im2bw. level is a normalized intensity value that lies in the range [0, 1]. The graythresh function uses Otsu's method, which chooses the threshold to minimize the intraclass variance of the black and white pixels. Multidimensional arrays are converted automatically to 2-D arrays using reshape. The graythresh function ignores any nonzero imaginary part of I.

10. im2bw

Convert an image to a binary image, based on threshold

Syntax

BW = im2bw(I,level)

Description

im2bw produces binary images from indexed, intensity, or RGB images. To do this, it converts the input image to grayscale format (if it is not already an intensity image), and then uses thresholding to convert this grayscale image to binary. The output binary image BW has values of 0 (black) for all pixels in the input image with luminance less than level and 1 (white) for all other pixels. (Note that you specify level in the range [0,1], regardless of the class of the input image.) .BW = im2bw(I,level) converts the intensity image I to black and white.

11. medfilt2

Perform two-dimensional median filtering

Syntax

B = medfilt2(A,[m n])

B = medfilt2(A)

B = medfilt2(A,'indexed',...)

Description

Median filtering is a nonlinear operation often used in image processing to reduce "salt and pepper" noise. Median filtering is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges.

B = medfilt2(A,[m n]) performs median filtering of the matrix A in two dimensions. Each output pixel contains the median value in the m-by-n neighborhood around the corresponding pixel in the input image. medfilt2 pads the image with 0's on the edges, so the median values for the points within [m n]/2 of the edges might appear distorted.

B = medfilt2(A) performs median filtering of the matrix A using the default 3-by-3 neighborhood.

B = medfilt2(A,'indexed',...) processes A as an indexed image, padding with 0's if the class of A is uint8, or 1's if the class of A is double.

12. <u>bwareaopen</u>

Binary area open; remove small objects

Syntax

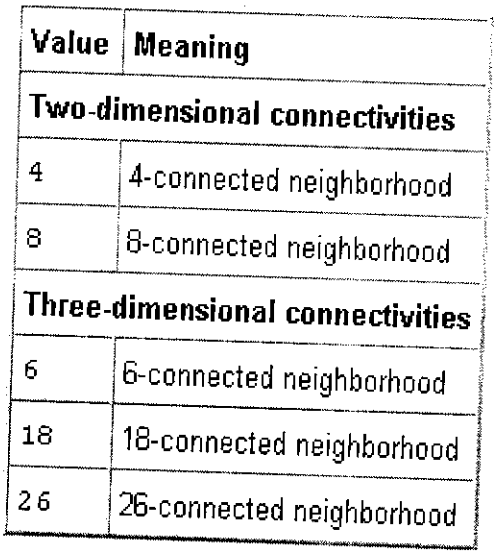
BW2 = bwareaopen(BW,P)

BW2 = bwareaopen(BW,P,CONN)

Description

BW2 = bwareaopen(BW,P) removes from a binary image all connected components (objects) that have fewer than P pixels, producing another binary image, BW2. The default connectivity is 8 for two dimensions, 26 for three dimensions, and conndef(ndims(BW),'maximal') for higher dimensions.

BW2 = bwareaopen(BW,P,CONN) specifies the desired connectivity. CONN can have any of the following scalar values.



Value	Meaning
Two-dimensional connectivities	
4	4-connected neighborhood
8	8-connected neighborhood
Three-dimensional connectivities	
6	6-connected neighborhood
18	18-connected neighborhood
26	26-connected neighborhood

Table 2.4.1.2. (b) CONN Values

13. <u>imclearborder</u>

Suppress light structures connected to image border

Syntax

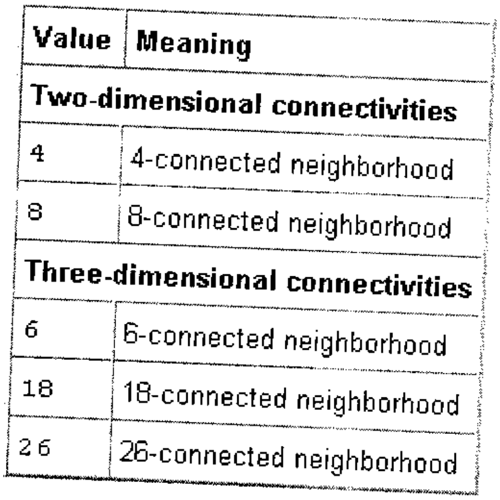
IM2 = imclearborder(IM)

IM2 = imclearborder(IM,CONN)

Description

IM2 = imclearborder(IM) suppresses structures that are lighter than their surroundings and that are connected to the image border. IM can be an intensity or binary image. The output image, IM2, is intensity or binary, respectively. The default connectivity is 8 for two dimensions, 26 for three dimensions, and conndef(ndims(BW),'maximal') for higher dimensions. For intensity images, imclearborder tends to reduce the overall intensity level in addition to suppressing border structures.

IM2 = imclearborder(IM,CONN) specifies the desired connectivity. CONN can have any of the following scalar values.



Value	Meaning
Two-dimensional connectivities	
4	4-connected neighborhood
8	8-connected neighborhood
Three-dimensional connectivities	
6	6-connected neighborhood
18	18-connected neighborhood
26	26-connected neighborhood

Table 2.4.1.2. (c) CONN Values

14. <u>bwlabel</u>

Label connected components in a binary image

Syntax

L = bwlabel(BW,n)

[L,num] = bwlabel(BW,n)

Description

L = bwlabel(BW,n) returns a matrix L, of the same size as BW, containing labels for the connected objects in BW. n can have a value of either 4 or 8, where 4 specifies 4-connected objects and 8 specifies 8-connected objects; if the argument is omitted, it defaults to 8. The elements of L are integer values greater than or equal to 0. The pixels labeled 0 are the background. The pixels labeled 1 make up one object, the pixels labeled 2 make up a second object, and so on.

[L,num] = bwlabel(BW,n) returns in num the number of connected objects found in BW.

15. regionprops

Measure properties of image regions

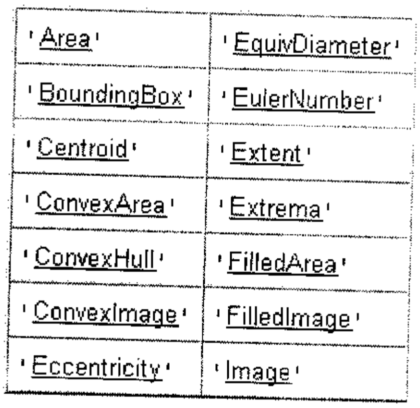
Syntax

STATS = regionprops(L,properties)

Description

STATS = regionprops(L,properties) measures a set of properties for each labeled region in the label matrix L. Positive integer elements of L correspond to different regions. For example, the set of elements of L equal to 1 corresponds to region 1; the set of elements of L equal to 2 corresponds to region 2; and so on. The return value STATS is a structure array of length max(L(:)). The fields of the structure array denote different measurements for each region, as specified by properties.

'properties' can be a comma-separated list of strings, a cell array containing strings, the single string 'all', or the string 'basic'. This table lists the set of valid property strings. Property strings are case insensitive and can be abbreviated.



'Area'	'EquivDiameter'
'BoundingBox'	'EulerNumber'
'Centroid'	'Extent'
'ConvexArea'	'Extrema'
'ConvexHull'	'FilledArea'
'ConvexImage'	'FilledImage'
'Eccentricity'	'Image'

Table 2.4.1.2. (d) Property Strings

If 'properties' is the string 'all', then all the preceding measurements are computed. If 'properties' is not specified or if it is the string 'basic', then these measurements are computed: 'Area', 'Centroid', and 'BoundingBox'.

16. find

Find indices and values of nonzero elements

Syntax

indices = find(X)

Description

indices = find(X) returns the linear indices corresponding to the nonzero entries of the array X. If none are found, find returns an empty matrix. In general, find(X) regards X as X(:), which is the long column vector formed by concatenating the columns of X.

17. ismember

Detect members of a specific set

Syntax

tf = ismember(A, S)

tf = ismember(A, S, 'rows')

[tf, loc] = ismember(A, S, ...)

Description

tf = ismember(A, S) returns a vector the same length as A, containing logical true (1) where the elements of A are in the set S, and logical false (0) elsewhere. In set theory terms, k is 1 where A S. A and S can be cell arrays of strings.

tf = ismember(A, S, 'rows'), when A and S are matrices with the same number of columns, returns a vector containing 1 where the rows of A are also rows of S and 0 otherwise. You cannot use this Syntax if A or S is a cell array of strings.

[tf, loc] = ismember(A, S, ...) returns index vector loc containing the highest index in S for each element in A that is a member of S. For those elements of A that do not occur in S, ismember returns 0.

18. <u>bwboundaries</u>

Trace region boundaries in a binary image

Syntax

B = bwboundaries(BW,CONN,options)

Description

B = bwboundaries(BW) traces the exterior boundaries of objects, as well as boundaries of holes inside these objects, in the binary image BW. bwboundaries also descends into the outermost objects (parents) and traces their children (objects completely enclosed by the parents). BW must be a binary image where nonzero pixels belong to an object and 0 pixels constitute the background. The following figure illustrates these components.

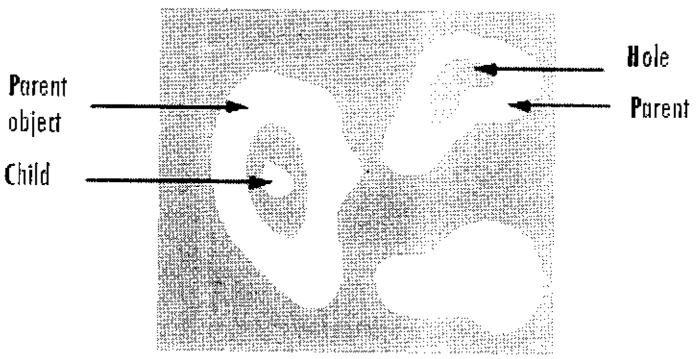
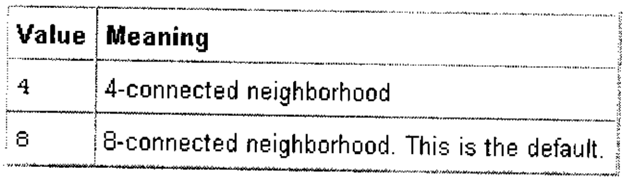


Figure 2.4.1.2. (b) Parent and Child Object and Holes

bwboundaries returns B, a P-by-1 cell array, where P is the number of objects and holes. Each cell in the cell array contains a Q-by-2 matrix. Each row in the matrix contains the row and column coordinates of a boundary pixel. Q is the number of boundary pixels for the corresponding region.

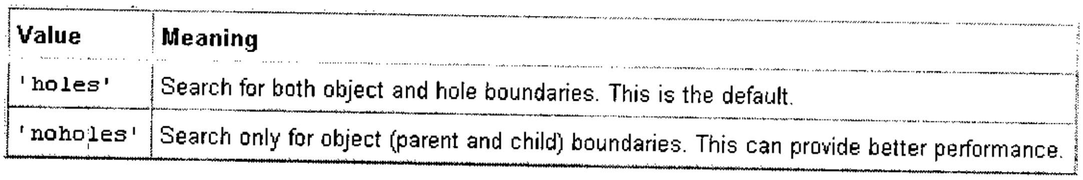
The connectivity is specified which can be used when tracing parent and child boundaries. CONN can have either of the following scalar values.



Value	Meaning
4	4-connected neighborhood
8	8-connected neighborhood. This is the default.

Table 2.4.1.2. (e) CONN Values

'Options' is an optional argument, where options can have either of the following values:



Value	Meaning
'holes'	Search for both object and hole boundaries. This is the default.
'noholes'	Search only for object (parent and child) boundaries. This can provide better performance.

Table 2.4.1.2. (f) Option Values

19. <u>label2rgb</u>

Convert label matrix into RGB image

Syntax

RGB = label2rgb(L,map,zerocolor)

Description

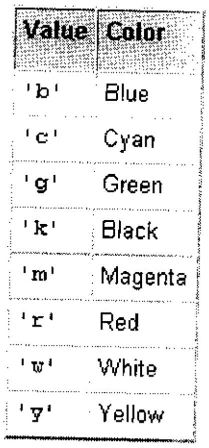
label2rgb converts a label matrix L, such as those returned by bwlabel or watershed, into an RGB color image for the purpose of visualizing the labeled regions. The label2rgb function determines the color to assign to each object based on the number of objects in the label matrix and range of colors in the colormap. The label2rgb function picks colors from the entire range.

map defines the colormap map to be used in the RGB image. map can have any of the following values:

- n-by-3 colormap matrix
- String containing the name of a MATLAB colormap function, such as 'jet' or 'gray' (See colormap for a list of supported colormaps.)
- Function handle of a colormap function, such as @jet or @gray.If you do not specify map, the default value is 'jet'.

RGB = label2rgb(L,map,zerocolor) defines the RGB color of the elements labeled 0 (zero) in the input label matrix L. As the value of zerocolor, specify an RGB triple or one of the strings listed in this table.

If you do not specify zerocolor, the default value for zero-labeled elements is [1 1 1] (white).



Value	Color
'b'	Blue
'c'	Cyan
'g'	Green
'k'	Black
'm'	Magenta
'r'	Red
'w'	White
'y'	Yellow

Table 2.4.1.2. (g) Zerocolor Values

20. <u>hold</u>

Retain current graph in figure

Syntax

hold on

Description

hold on retains the current plot and certain axes properties so that subsequent graphing commands add to the existing graph.

21. <u>diff</u>

Differences and approximate derivatives

Syntax

Y = diff(X)

Description

Y = diff(X) calculates differences between adjacent elements of X. If X is a vector, then diff(X) returns a vector, one element shorter than X, of differences between adjacent elements:

[X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)]

If X is a matrix, then diff(X) returns a matrix of row differences:

[X(2:m,:)-X(1:m-1,:)]

In general, diff(X) returns the differences calculated along the first non-singleton (size(X,dim) > 1) dimension of X.

22. <u>round</u>

Symbolic matrix element-wise round

Syntax

round(X)

Description

Y = round(X) rounds the elements of X to the nearest integers. Values halfway between two integers are rounded away from zero.

23. <u>sum</u>

Sum of array elements

Syntax

B = sum(A)

Description

B = sum(A) returns sums along different dimensions of an array.If A is a vector, sum(A) returns the sum of the elements. If A is a matrix, sum(A) treats the columns of A as vectors, returning a row vector of the sums of each column. If A is a multidimensional array, sum(A) treats the values along the first non-singleton dimension as vectors, returning an array of row vectors.

24. <u>min</u>

Minimum elements of an array

Syntax

C = min(A)

Description

C = min(A) returns the smallest elements along different dimensions of an array.If A is a vector, min(A) returns the smallest element in A.If A is a matrix, min(A) treats the columns of A as vectors, returning a row vector containing the minimum element from each column. If A is a multi-dimensional array, min operates along the first nonsingleton dimension.

25. <u>max</u>

Maximum elements of an array

Syntax

C = max(A)

Description

C = max(A) returns the largest elements along different dimensions of an array. If A is a vector, max(A) returns the largest element in A. If A is a matrix, max(A) treats the columns of A as vectors, returning a row vector containing the maximum element from each column. If A is a multi-dimensional array, max(A) treats the values along the first non-singleton dimension as vectors, returning the maximum value of each vector.

26. abs

Absolute value and complex magnitude

Syntax

Y = abs(X)

Description

abs(X) returns an array Y such that each element of Y is the absolute value of the corresponding element of X. If X is complex, abs(X) returns the complex modulus (magnitude), which is the same as sqrt(real(X).^2 + imag(X).^2)

2.4.1.3. NEURAL NETWORK TOOLBOX

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements.

Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. Such a situation is shown below. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are used to train a network. This is called supervised learning.

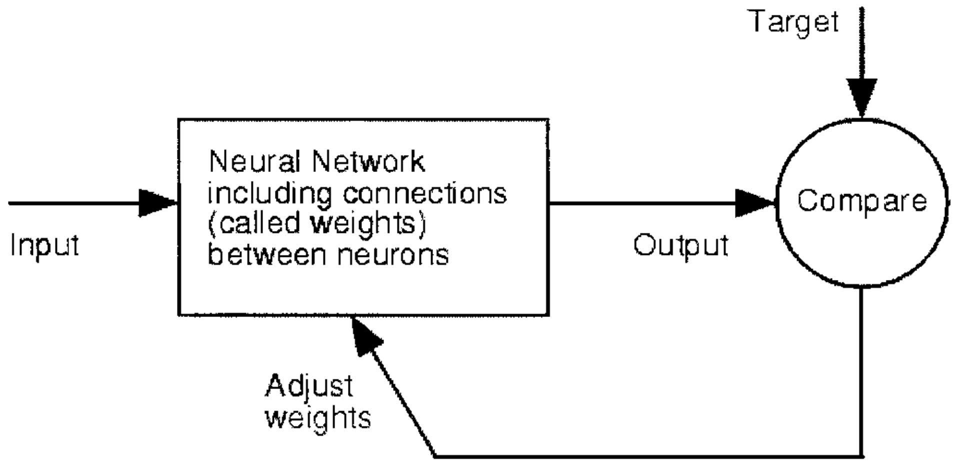

Target
Neural Network including connections (called weights) between neurons
Input
Output
Compare
Adjust weights

Figure 2.4.1.3. (a) Neural Network trained to produce a specific target output for a particular input

The supervised training methods are commonly used, but other networks can be obtained from unsupervised training techniques or from direct design methods. Unsupervised networks can be used, for instance, to identify groups of data. Certain kinds of linear networks and Hopfield networks are designed directly.

Batch training of a network proceeds by making weight and bias changes based on an entire set (batch) of input vectors. Incremental training changes the weights and biases of a network as needed after presentation of each individual input vector. Incremental training is sometimes referred to as "on line" or "adaptive" training.

Neural networks can be trained to solve problems that are difficult for conventional computers or human beings.

2.4.1.3.1. NEURON MODEL

SIMPLE NEURON

A neuron with a single scalar input and no bias appears on the left below.

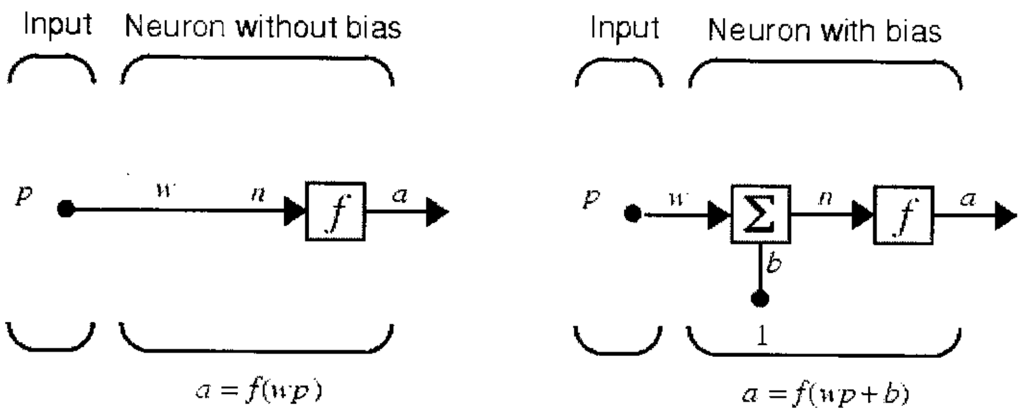


Figure 2.4.1.3. (b) Single scalar input neuron without bias and with bias

The scalar input p is transmitted through a connection that multiplies its strength by the scalar weight w, to form the product wp, again a scalar. Here the weighted input wp is the only argument of the transfer function f, which produces the scalar output a.

The neuron on the right has a scalar bias, b. The bias may be viewed as simply being added to the product wp as shown by the summing junction or as shifting the function f to the left by an amount b. The bias is much like a weight, except that it has a constant input of 1.

The transfer function net input n, again a scalar, is the sum of the weighted input wp and the bias b. This sum is the argument of the transfer function f. Note that w and b are both adjustable scalar parameters of the neuron.

The central idea of neural networks is that such parameters can be adjusted so that the network exhibits some desired or interesting behavior. Thus, we can train the network to do a particular job by adjusting the weight or bias parameters, or perhaps the network itself will adjust these parameters to achieve some desired end.

2.4.1.3.2. NETWORK ARCHITECTURES

Two or more of the neurons can be combined in a layer, and a particular network could contain one or more such layers. First consider a single layer of neurons.

A LAYER OF NEURONS

A one-layer network with R input elements and S neurons are shown.

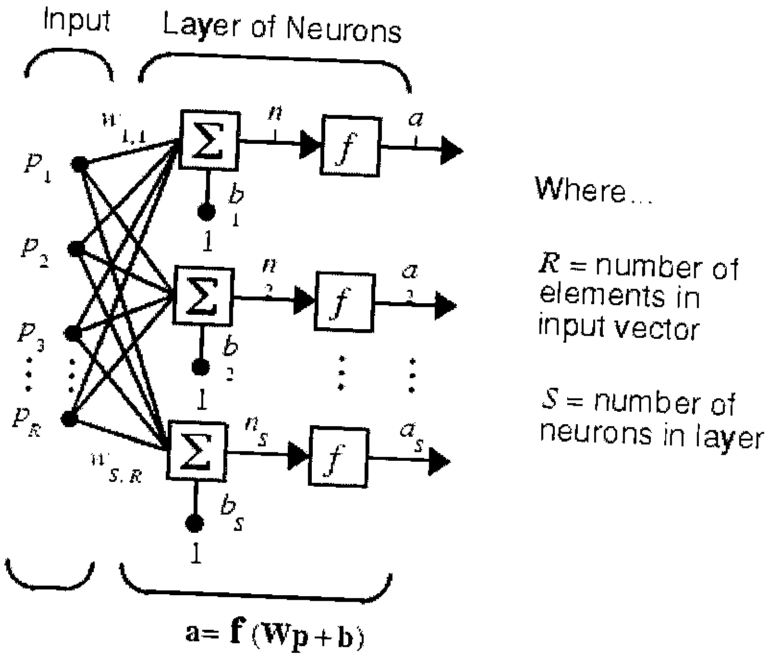

Input
Layer of Neurons
$w_{1,1}$
p_1
p_2
p_3
p_R
$w_{S,R}$
Σ
n_1
n_2
n_S
f
a_1
a_2
a_S
b_1
b_2
b_S
1
Where...
R = number of elements in input vector
S = number of neurons in layer
$\mathbf{a} = \mathbf{f}(\mathbf{Wp}+\mathbf{b})$

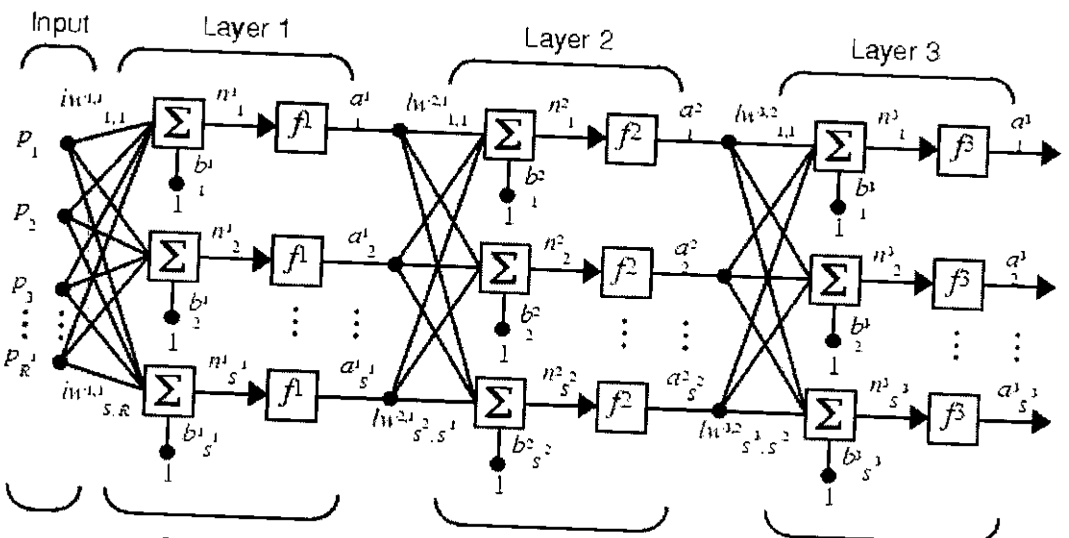
Figure 2.4.1.3. (c) One-layer network with R input elements and S neurons

In this network, each element of the input vector p is connected to each neuron input through the weight matrix W. The ith neuron has a summer that gathers its weighted inputs and bias to form its own scalar output n(i). The various n(i) taken together form an S-element net input vector n. Finally, the neuron layer outputs form a column vector a.

It is common for the number of inputs to a layer to be different from the number of neurons (i.e., R ≠ S). A layer is not constrained to have the number of its inputs equal to the number of its neurons.

MULTIPLE LAYERS OF NEURONS

A network can have several layers. Each layer has a weight matrix W, a bias vector b, and an output vector a. To distinguish between the weight matrices, output vectors, etc., for each of these layers, the number of the layer is appended as a superscript to the variable of interest.



$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p} + \mathbf{b}^1)$ $\quad$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1 + \mathbf{b}^2)$ $\quad$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2 + \mathbf{b}^3)$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\,\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

Figure 2.4.1.3. (d) Muliple layers of neurons

The network shown above has R1 inputs, S1 neurons in the first layer, S2 neurons in the second layer, etc. It is common for different layers to have different numbers of neurons. A constant input 1 is fed to the biases for each neuron.

Note that the outputs of each intermediate layer are the inputs to the following layer. Thus layer 2 can be analyzed as a one-layer network with S1 inputs, S2 neurons, and an S2xS1 weight matrix W2. The input to layer 2 is a1; the output is a2. Now that we have identified all the vectors and matrices of layer 2, we can treat it as a single-layer network on its own. This approach can be taken with any layer of the network.

The layers of a multilayer network play different roles. A layer that produces the network output is called an output layer. All other layers are called hidden layers. The three-layer network shown earlier has one output layer (layer 3) and two hidden layers (layer 1 and layer 2).

Multiple-layer networks are quite powerful. For instance, a network of two layers, where the first layer is sigmoid and the second layer is linear, can be trained to approximate any function (with a finite number of discontinuities) arbitrarily well. This kind of two-layer network is used extensively in Backpropagation

2.4.1.3.3. BACKPROPAGATION

INTRODUCTION

Backpropagation was created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors, or classify input vectors in an appropriate way as defined by you. Networks with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities.

Standard backpropagation is a gradient descent algorithm, as is the Widrow-Hoff learning rule, in which the network weights are moved along the negative of the gradient of the performance function. The term backpropagation refers to the manner in which the gradient is computed for nonlinear multilayer networks. Typically, a new input leads to an output similar to the correct output for input vectors used in training that are similar to the new input being presented. This generalization property makes it possible to train a network on a representative set of input/target pairs and get good results without training the network on all possible input/output pairs.

There are generally four steps in the training process:

1. Assemble the training data
2. Create the network object
3. Train the network
4. Simulate the network response to new inputs

FUNDAMENTALS

ARCHITECTURE

This section presents the architecture of the network that is most commonly used with the backpropagation algorithm - the multilayer feedforward network.

NEURON MODEL (TANSIG, LOGSIG, PURELIN)

An elementary neuron with R inputs is shown below. Each input is weighted with an appropriate w. The sum of the weighted inputs and the bias forms the input to the transfer function f. Neurons may use any differentiable transfer function f to generate their output.

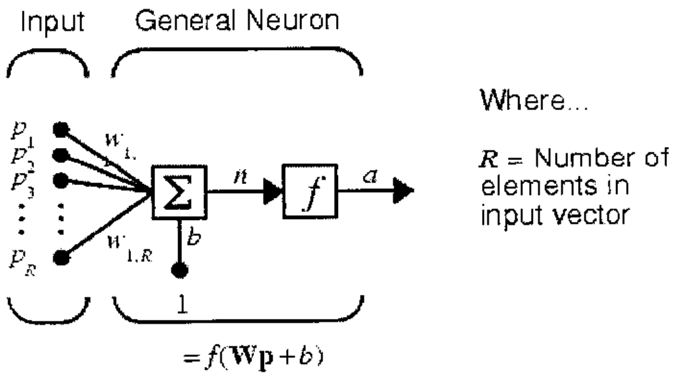
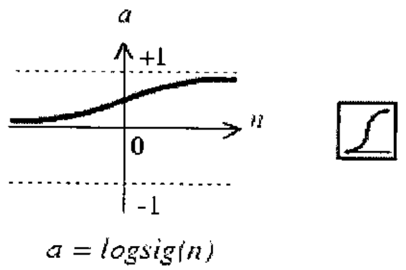


Figure 2.4.1.3. (e) Neuron Model

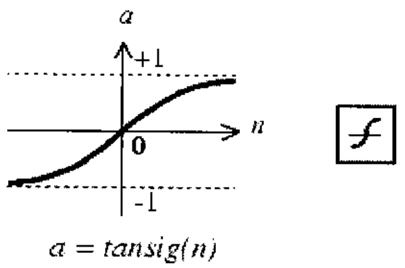
Multilayer networks often use the log-sigmoid transfer function logsig.



Log-Sigmoid Transfer Function

Figure 2.4.1.3. (f) Representation of Log-Sigmoid Transfer Function

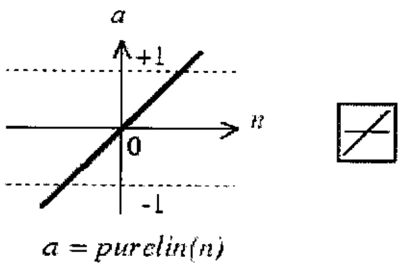
The function logsig generates outputs between 0 and 1 as the neuron's net input goes from negative to positive infinity. Alternatively, multilayer networks may use the tan-sigmoid transfer function tansig.



Tan-Sigmoid Transfer Function

Figure 2.4.1.3. (g) Representation of Tan-Sigmoid Transfer Function

Occasionally, the linear transfer function purelin is used in backpropagation networks.



Linear Transfer Function

Figure 2.4.1.3. (h) Representation of Linear Transfer Function

If the last layer of a multilayer network has sigmoid neurons, then the outputs of the network are limited to a small range. If linear output neurons are used the network outputs can take on any value.

FEEDFORWARD NETWORK

A single-layer network of S logsig neurons having R inputs is shown below in full detail on the left and with a layer diagram on the right.

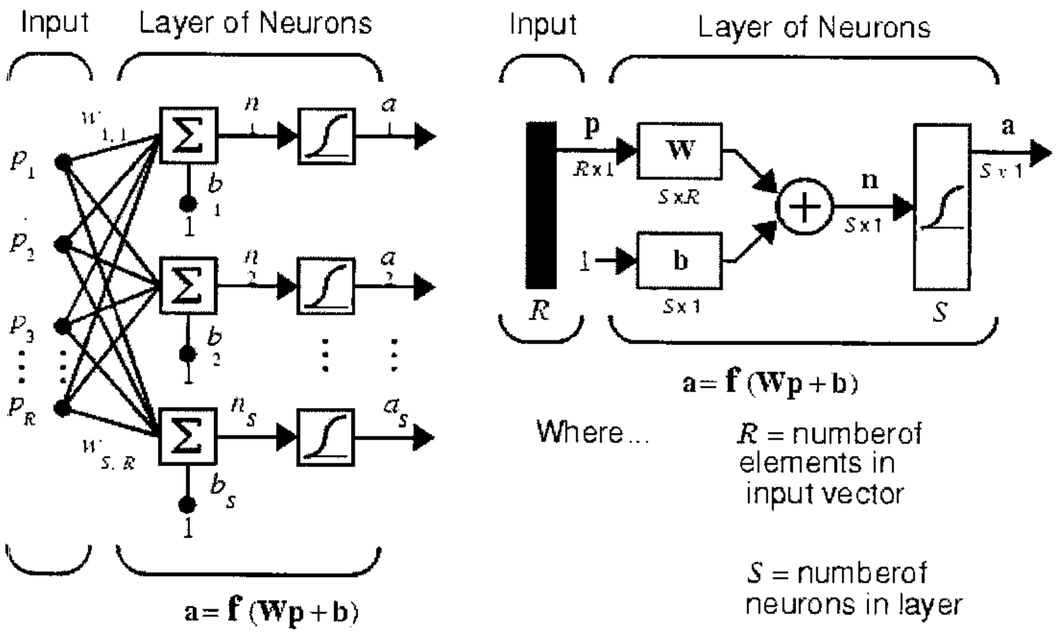


Figure 2.4.1.3. (i) Single-layer network of S logsig neurons having R inputs and layer diagram

Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. The linear output layer lets the network produce values outside the range -1 to +1.

- Menus and Toolbars
- Project Explorer
- Property Sheets

2.4.2.1. MATLAB-VISUAL BASIC CONNECTIVITY

The *Component Object Model*, or COM, provides a framework for integrating reusable, binary software components into an application. Because components are implemented with compiled code, the source code may be written in any of the many programming languages that support COM.

The MATLAB COM Builder is an extension to the MATLAB Compiler that enables customers to automatically convert MATLAB applications to Component Object Model (COM) objects. Developers can do modeling and analysis in MATLAB and convert the models to ready-to-use COM objects. These objects can be immediately integrated with any COM-based application such as Visual Basic.

BUILDING STEPS

To create a simple COM component in MATLAB, you must pass four steps:

- Creating a project
- Managing m-files
- Building project
- Packaging and distributing the component

A project consists of all the elements necessary to build an application using the MATLAB COM Builder. COM Builder components are COM objects accessible through Visual Basic, Visual C++, Delphi, Power Builder or any other language that supports COM. Each COM object exposes one or more classes. Each class contains a set of functions called methods, corresponding to the original MATLAB functions included in the component's project. When creating a component, you must additionally provide one or more class names. The component name represents the name of the DLL file to be created. A class name denotes the name of the class that performs a call on a specific method at run-time. The relationship between component name and class name, and which methods (MATLAB functions) go into a particular class, are purely organizational. As a general rule, when compiling many MATLAB functions, it helps to determine a scheme of function categories and to create a separate class for each category. The name of each class should be descriptive of what the class does.

The step by step procedure is as follows:

1) When creating a new project, enter the MATLAB `comtool` command. In this case the MATLAB COM Builder GUI will appear.
2) Next, select File→New Project. The 'New Project Settings' dialog box opens in which the component name is entered.
3) The component name is name of the DLL file that is built by the build process. The component name must differ from any m-file added later to the project. One can add more classes to the project when entering the class name and pressing "Add >>" button. And click ok

MANAGING M-FILES

After creating the project, it is time to add m-files related to the project. You can do this through "Add File" button. Editing or removing a file is very simple. Just select the file and press the "Edit" or "Remove" button! Every object that is added to the project is displayed in project tree.

BUILDING PROJECT

By collecting all of the necessary information, we can build the project. For this reason, select "COM Object" from the Build menu or simply click on the "Build" button in the right pane. The build status pane will show the status of build process. If any error occurs during the build process, the status pane will show it.

PACKAGING AND DISTRIBUTING THE COMPONENT

Once you have successfully compiled your models and tested the COM object, you are ready to package the component for distribution to your end users. Choose Component, Package Component to create a self-extracting executable files.

IMPORTING THE COM COMPONENT

The procedure to include the component in Visual Basic is as follows: Start Visual Basic 6.0. Open a new project and select "References" from Project menu. Add the component, which was built using the COM tool in MATLAB.

DETAILS OF METHODOLOGY EMPLOYED

3. DETAILS OF METHODOLOGY EMPLOYED

3.1. PROPOSED SYSTEM – PROCESS FLOW CHART

The computerized image analysis system of seed borne fungus obtains the image of fungus as an input. The input image is then processed and the features extraction is done. The final step is identification of the genus of the fungus. The above process is depicted in form of a flowchart

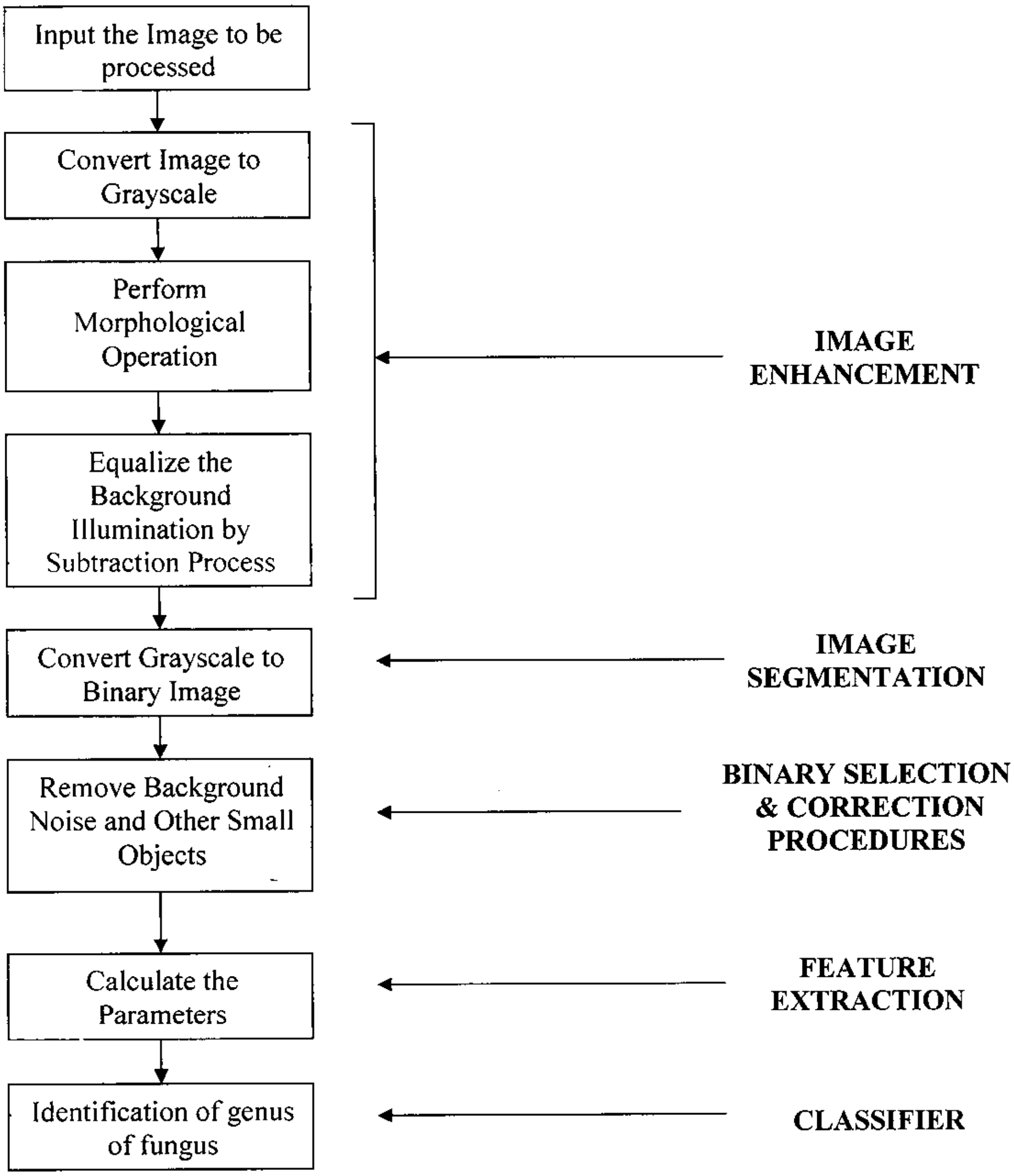

Input the Image to be processed
Convert Image to Grayscale
Perform Morphological Operation
Equalize the Background Illumination by Subtraction Process
Convert Grayscale to Binary Image
Remove Background Noise and Other Small Objects
Calculate the Parameters
Identification of genus of fungus
IMAGE ENHANCEMENT
IMAGE SEGMENTATION
BINARY SELECTION & CORRECTION PROCEDURES
FEATURE EXTRACTION
CLASSIFIER

Figure 3.1.1. Process Flow Chart

3.2. MODULES INVOVED

The following are the modules in the proposed system:

- Image Acquisition
- Image Enhancement
- Image Segmentation
- Binary selection and correction procedures
- Feature Extraction - Recognizing spores as vectorized objects
- Measurement of Objects
- Classifier - Identification of genus of fungi

The acquisition of the image samples was done as follows: The reference cultures of different genera of fungi were obtained from the Tamil Nadu Agricultural University (TNAU), Coimbatore and they are as mentioned below:

- *Helminthosporium*
- *Alternaria*
- *Curvularia*

3.2.1. IMAGE ACQUISITION

In this phase, for acquiring the image the following camera and microscope specifications were used.

- Microscope - Optika Microscope, N-400T (Made in Italy), Halogen lamp 6v 20w
- Camera - Nikon E995 (Made in Japan)
- Magnification – 100 x 4 = 400 magnification.

All images are of the dimension 2048 X 1536 and the storage format of the images is in the jpeg format

3.2.2. IMAGE ENHANCEMENT

The aim of image enhancement is to improve the interpretability or perception of information in images for human viewers, or to provide `better' input for other automated image processing techniques. Image enhancement techniques can be divided into two broad categories:

1. Spatial domain methods, which operate directly on pixels, and
2. Frequency domain methods, which operate on the Fourier Transform of an image.

Image enhancement techniques are used to emphasize and sharpen image features for display and analysis. Image enhancement is the process of applying these techniques to facilitate the development of a solution to a computer-imaging problem. Consequently, the enhancement methods are application specific and are often developed empirically. The type of techniques includes point operations, where each pixel is modified according to a particular equation that is not dependent on other pixel values; mask operations, where each pixel is modified according to the values of the pixel's neighbors (using convolution masks); or global operations, where all the pixel values in the image (or sub-image) are taken into consideration. Spatial domain processing methods include all three types, but frequency domain operations, by nature of the frequency transforms, are global operations. Of course, frequency domain operations can become "mask operations," based only on a local neighborhood, by performing the transform on small image blocks instead of the entire image. Enhancement is also used as a preprocessing step in applications where human viewing of an image is required before further processing. Image enhancement is used for post-processing to generate a visually desirable image. Overall, image enhancement methods are used to make images look better.

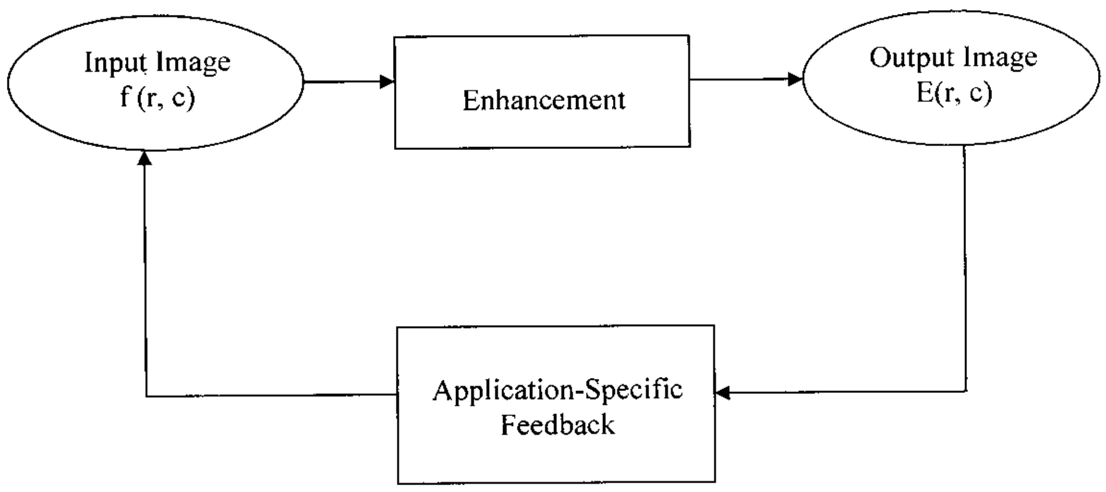


Figure 3.2.2.1. Image Enhancement

3.2.3. IMAGE SEGMENTATION

Image segmentation is the actual binarization process, extracting vectorized binary objects from the image. The recognized objects can be manipulated by corrective and selective functions. Segmentation refers to distinguishing objects from background. Four popular approaches used are:

- Threshold techniques
- Edge-based methods
- Region-based techniques
- Connectivity-preserving relaxation methods

Segmentation is the isolation of scene components that are likely connected and at different depths. The goal is to find scene regions that correspond to "salient" objects or object parts. Since salience depends on intention, segmentation is not well defined outside a particular visual task context, but in many tasks important 3-D objects cohere in depth and are separated in depth from other objects and from the background. Segmentation means to divide up the image into a patchwork of regions, each of which is "homogeneous", that is, the "same" in some sense like intensity, texture, color and so on. The various methods involved in segmentation can be broadly classified as:

- Local Methods
- Global Methods

Some of the local methods are:

- Watershed Segmentation
- Edge based methods followed by region growing
- Simple thresholding

Some of the global methods are:

- K-means clustering
- Expectation maximization
- Normalized graph cuts
- Greedy graph cuts

3.2.4. BINARY SELECTION AND CORRECTION PROCEDURES

An image condition is judged based on a mean value and a standard deviation of a feature quantity of an image, and image correction information in the image condition is created based on the mean value and the standard deviation, to thereby correct the image, based on the created image correction information. In this manner, parameters for the image correction processing are determined automatically and highly accurately corresponding to the feature quantity of the image, and the image correction processing is performed based on the parameters. The recognized objects are manipulated by corrective and selective functions. The binary corrective functions are applied to connect broken outlines and to remove boundary objects.

3.2.5. FEATURE EXTRACTION - RECOGNIZING SPORES AS VECTORISED OBJECTS

A feature is a cluster of points or a boundary/region of pixels that satisfy a set of pre-defined criteria. The criteria can be based on any quantities, such as shape, spectral similarity, orientation, and spatial distribution. Feature extraction is the process of studying and locating areas and objects on the ground and deriving useful information from images. Feature extraction is the operation to extract various image features for identifying or interpreting meaningful physical objects from images.

Features are classified into three types.

- **Spectral features:** color, tone, ratio, spectral index
- **Geometric features:** edges
- **Textural features:** pattern, homogeneity, spatial frequency

Position, outline and average for each spore are determined. Fine tuning of optimization and segmentation procedures for each spore separately facilitates full recognition of the inner structure. The vectorized objects considered are: size, shape, area, length, width, length/width, circular form factor, convexity and curvature. Each of these parameters helps in uniquely identifying the genus of fungi.

3.2.6. MEASUREMENT OF OBJECTS

Objects are prepared for measurement by reduction of contours to lines of one pixel width. The measurement is done based on size, area, perimeter, length, width, length/width, F_{Shape}, D_{max}, D_{min}, F_{circle}, and D_{circle}.

3.2.6.1. Calculation of F_{circle}

The circular form factor (F_{circle}) relates the perimeter of a circle of the same area to the real perimeter of the object, returning 1.0 for a circular object and approximately zero for oblong objects.

$$F_{circle} = (4 * \pi * A) / P^2$$

Where: A = area of the object

P = perimeter of the object

3.2.6.2. Calculation of D_{max}

Algorithm

```
STEP 1:   Check the orientation of the image.
STEP 2:   For vertical objects do the following
            (i)   Perform horizontal scanning at the top of the image to find the endpoint $A_V$(colht1,rowht1)
            (ii)  Perform horizontal scanning at the bottom of the image to find the endpoint $B_V$ (colhb1,rowhb1)
            (iii) Calculate the distance between the two points $A_V$ and $B_V$ to find $\mathbf{D_{max}}$ using the formula
                  $\mathbf{D_{max}=\sqrt{((colht1-colhb1)^2 + (rowht1-rowhb1)^2)}}$
                  Go to step 6.
STEP 3:   For horizontal objects do the following
            (i)   Perform vertical scanning at the left of the image to find the endpoint $A_H$(colvt1,rowvt1)
            (ii)  Perform vertical scanning at the right of the image to find the endpoint $B_H$(colvb1,rowvb1)
            (iii) Calculate the distance between the two points $A_V$ and $B_V$ to find $\mathbf{D_{max}}$ using the formula
```

```
                $D_{max}= \sqrt{((colvt1-colvb1)^2 + (rowvt1-rowvb1)^2)}$
                Go to step 6.
STEP 4:   For inclined objects do the following
            (i)   Repeat the steps 2 and 3 at each end of the image and trace the coordinates at each end
            (ii)  Average is calculated for the traced points at each end
            (iii) Two end points are obtained namely $A_I$(AcDmax,ArDmax)and (BcDmax,BrDmax)
            (iv)  Distance is calculated using the formula
                  $D_{max} = \sqrt{((x1-x2)^2 + (y1-y2)^2)}$. Go to step 6
STEP 5:   In case of acute angle do the following
            (i)   Reflect the image
            (ii)  Then repeat the steps 4 to find the value of $D_{max}$.
STEP 6:   End
```

3.2.6.3. Calculation of D_{min}

Algorithm

```
STEP 1:   Check the orientation of the image.
STEP 2:   For vertical objects do the following
            (i)   From the endpoints ($A_V$ and $B_V$) obtained during the calculation of $D_{max}$ procedure, calculate the midpoint of these endpoints.
            (ii)  Perform horizontal scanning along the midpoint and find the endpoints of the object i.e. one on the left (minL) and other on the right (minR) of the image.
```

(iii) Calculate the distance between the two points 'minL (xv1, yv1)' and 'minR (xv2, yv2)' to find $\mathbf{D_{min}}$ using the formula.

$\mathbf{D_{min}} = \mathbf{\sqrt{((yv1-yv2)^2 + (xv1-xv2)^2)}}$.Go to step 6

<u>STEP 3</u>: For horizontal objects do the following

(i) From the endpoints (A_H and B_H) obtained during the calculation of D_{max} procedure, calculate the midpoint of these endpoints.

(ii) Perform vertical scanning along the midpoint and find the endpoints of the object i.e. one at the top (minT) and other at the bottom (minB) of the image.

(iii) Calculate the distance between the two points 'minT (xh1, yh1)' and 'minB(xh2,yh2)' to find $\mathbf{D_{min}}$ using the formula.

$\mathbf{D_{min}} = \mathbf{\sqrt{((yh1-yh2)^2 + (xh1-xh2)^2)}}$.Goto step 6

<u>STEP 4</u>: For inclined objects do the following

(i) From the endpoints (A_I and B_I) obtained during the calculation of D_{max} procedure, calculate the midpoint of these endpoints.

(ii) Perform horizontal scanning along the midpoint that is obtained in the previous step and find the endpoints of the object i.e. one on the left (minL(xv1,yv1)) and other on the right (minR(xv2,yv2)) of the image.

(iii) Perform vertical scanning along the midpoint that is obtained before and find the endpoints of the object i.e. one at the top 'minT (xh1, yh1)' and other at the bottom 'minB (xh2, yh2)' of the image.

(iv) Now find the midpoint between the points 'minL' and 'minB' say 'Admin' and also find the midpoint between the points 'minT' and 'minR' say 'Bdmin'.

(v) Distance is calculated between 'Admin' and 'Bdmin' using the formula.

$\mathbf{D_{min} = \sqrt{((x1-x2)^2 + (y1-y2)^2)}}$. Go to step 6

<u>STEP 5</u>: In case of acute angle do the following

(i) Reflect the image

(ii) Then repeat the steps 4 to find the value of $\mathbf{D_{min}}$.

<u>STEP 6</u>: End

3.2.6.4. Calculation of F_{shape}:

$F_{shape} = D_{max}/D_{min}$

3.2.6.5. Calculation of D_{circle}:

$D_{circle} = (2*\sqrt{Area})/\pi$

3.2.7. CLASSIFIER - IDENTIFICATION OF GENUS OF FUNGUS

Unique identification of genus of fungi is done with the help of the parameters like area, circumference and curvature as they give a complete morphological description of the object. An artificial neural network (ANN) was used to perform our classification task. There are many different types of ANNs; the most widely used is the Back Propagation ANN. This type of ANN is excellent for performing classification tasks. We have used a feed-forward ANN with one hidden layer, and standard back propagation as the training algorithm. The transfer function used is *purelin* function. The number of neurons present in Layer 1(Hidden Layer) is 12 and the number of neurons present in Layer 2(Output Layer) is 1. The parameters obtained from the above phase were given as inputs to this network and the system was trained to achieve the target that was set. The performance graph that was obtained showed a performance value of 0.0269751.The Simulink model was designed using the trained network, which enabled the identification of genus of fungus.

SYSTEM DESIGN

4. SYSTEM DESIGN

4.1. INPUT DESIGN

Once the analysis of the system has been done, it would be necessary to identify the data that is required to produce the required output. Input design features can ensure reliability of the system and generate correct reports from the accurate data. The input design also determines whether the user can interact with the system efficiently. The user interface design is dialog based, thus making it comfortable for the user to selects an image and provides it as an input to the image analyzer part. The input image that is acquired from the camera is pre-processed using MATLAB. This pre-processed image is given as the input to the image analyzer. The image may be of any of the formats like ".jpg", ".bmp", etc.

The output of this code is used an input to the Neural Network Model. This model takes input from the MATLAB file and writes output to another MATLAB file. Meanwhile, the COM object created using MATLAB, works in Visual Basic to transfer input to the M-File and vice-versa.

4.2. PROCESS DESIGN

4.2.1. SEQUENCE DIAGRAM

Sequence diagrams model the flow of logic within the system in a visual manner, enabling the user both to document and validate the logic, and are commonly used for both analysis and design purposes. The sequence diagram shown here is for positive scenario depicting the complete process.

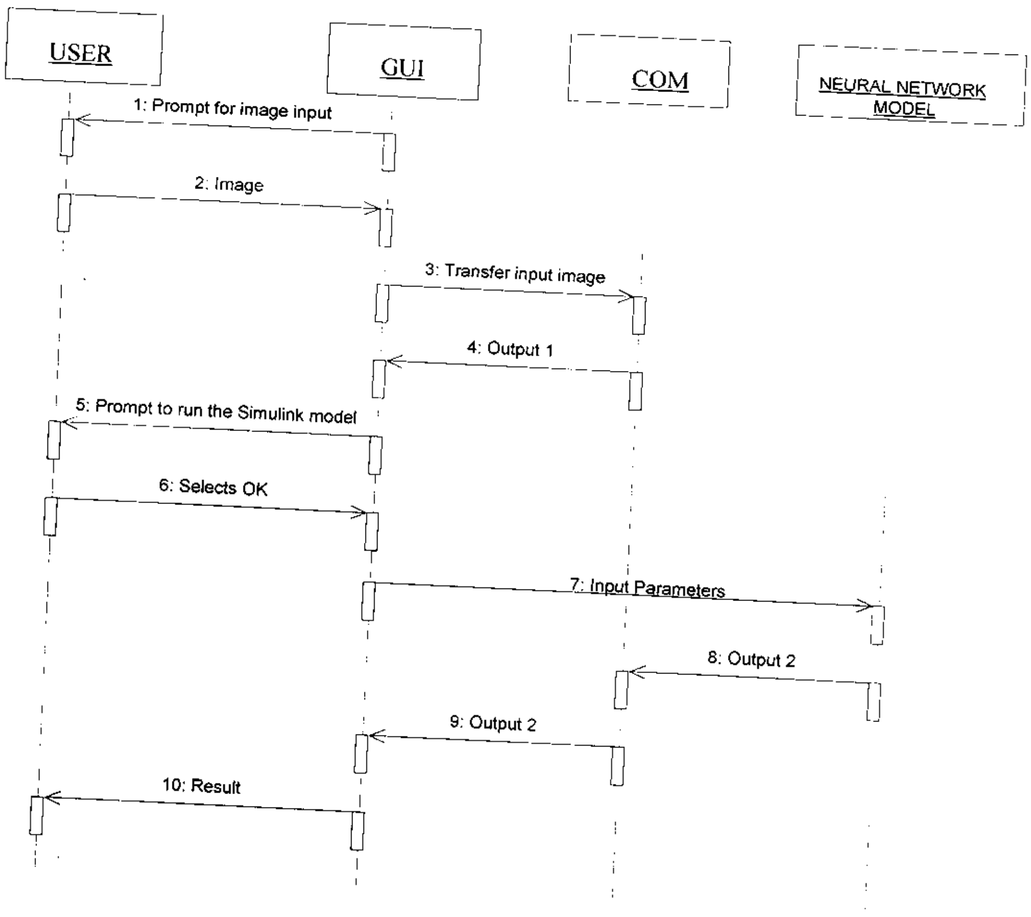


Figure 4.2.1.1. Sequence Diagram

4.2.2. COLLABORATION DIAGRAM

A collaboration diagram shows the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction. The collaboration diagram shown here is for positive scenario depicting the complete process.

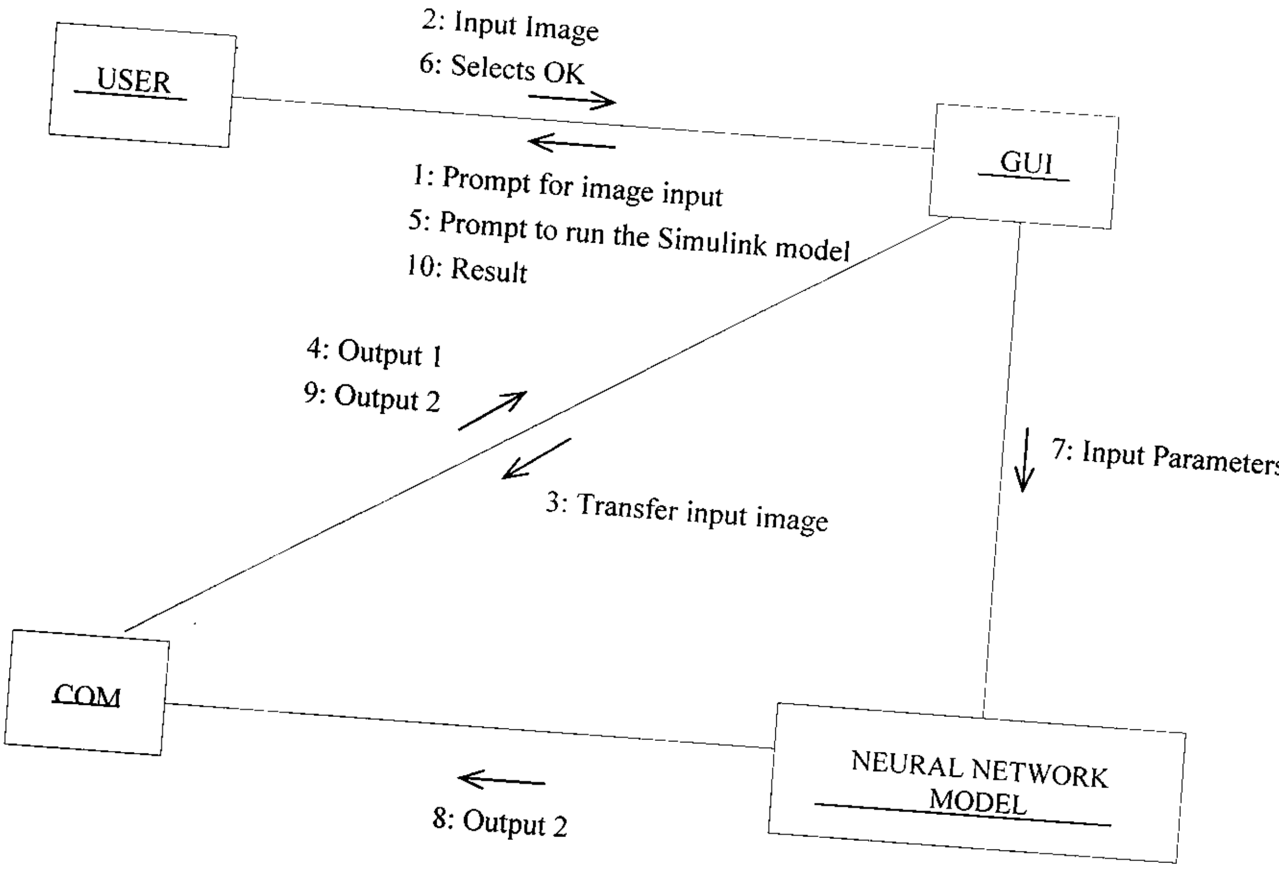


Figure 4.2.2.1. Collaboration Diagram

4.3. OUTPUT DESIGN

Computer output is the most important and direct source of information to the user. Efficient, intelligible output design should improve the system relationship with the user and help in decision making. After processing the image, the result is sent to the Visual Basic module. Both, the Visual Basic module and the MATLAB module are integrated using the COM object. This COM object helps to transfer the output code to Visual Basic as well as the output from Neural Network model. Finally the result is viewed as a solution from the Visual Basic module.

TESTING

5. TESTING

Testing presents an interesting anomaly for the software engineers, who by their nature are constructive people. Testing requires that the developer discard preconceived notions of the "correctness" of software just developed and then work hard to design test cases to "break" the software.

The goal of testing is to find errors and a good test is one that has a high probability of finding an error. Therefore, a software engineer should design and implement a computer-based system or a product with "testability" in mind. At the same time, the tests themselves must exhibit a set of characteristics that achieve the goal of finding the most errors with a minimum of effort.

A good test has the following characteristics:

- A good test has a high probability of finding an error
- A good test is not redundant
- A good test should be "best of breed"
- A good test should be neither too simple nor too complex

5.1. UNIT TESTING

It focuses verification effort on the smallest unit of software design, the module. It is also known as module testing. The modules are tested separately. The testing is carried out usually during programming stage itself.

Each and every module is tested separately to check if its intended functionality is met. Some unit testing tasks performed are listed below:

- Check whether the background is eliminated properly
- Check whether the image is segmented properly

- Check whether the different species are identified properly
- See if the mat files are called properly and executed when required
- Check whether the input parameters and output parameters are written to the respective files and invoked properly during execution.

5.2. VALIDATION TESTING

Validation is a process of finding if we are building the right product.

The software product should functionally do what it is supposed to. Validation is done during or at the end of the development process in order to determine whether the product satisfies specified requirements.

After the validation test has been conducted, one of the two possible conditions exists:

- The functions and the performance characteristics confirm to the specification and are accepted.
- Deviation from the specification is uncovered and the deficiency list is created.

5.3. OUTPUT TESTING

After performing the validation testing, the next step is the output testing of the proposed system since no system is useful if it does not produce the required output in the specific format. The outputs generated and displayed by the system under consideration are tested by the users about the formats required by them.

- Verify whether the simulation takes place properly.

- Verify whether the system is able to give the final result approximately to the value of target, which had been set while training the network using Neural Network.

5.4. INTEGRATION TESTING

It is the testing performed to detect errors on interconnection between modules. Here all the modules in VB and MATLAB are integrated to form the entire application and tested to ensure that they work in synchronization and without interference from each other.

5.5. SYSTEM TESTING

The system is tested against the software requirements specification to see if all the requirements are met and if the system performs as per the client's expectations. The system is tested as a whole to check for its functionality. Non-functional requirements like performance considerations and platform support are checked as a whole.

RESULTS AND DISCUSSIONS

6. RESULTS AND DISCUSSIONS

6.1. Input Sample: Genus of Fungus – *Helminthosporium*

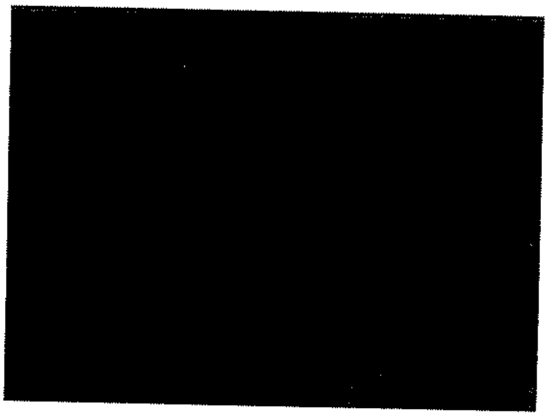


Figure 6.1.1. ***Helminthosporium*** **Fungus**

A set of eight samples of this genus were processed using the above algorithms and formulas. The following values are obtained:

Samples	**Perimeter**	**Area**	F_{circle}	D_{max}	D_{min}	F_{shape}	D_{circle}	**NN Output**
H1	4330	324556	0.2175	1410	249	5.6627	362.6811	1.142
H2	3732	292013	0.2635	1472	221	6.6606	344.0180	1.182
H3	3720	312027	0.2833	1411	262	5.3855	355.6118	1.379
H4	4711	499444	0.2828	1737	332	5.2319	449.9078	0.7574
H5	4746	463605	0.2586	1679	320	5.2469	433.4651	0.8337
H6	4708	499071	0.2829	1730	336	5.1488	449.7398	0.7597
H7	4665	494868	0.2858	1738	319	5.4483	447.8420	0.7581
H8	3845	339794	0.2888	1333	310	4.3000	371.0975	1.294

Table 6.1.1. Parameter Values of *Helminthosporium* Samples

From the results obtained, if the NN Output value lies between 0.5 and 1.5, the fungus belongs to the genus *"Helminthosporium"*.

6.2. Input Sample: Genus of Fungus – *Alternaria*

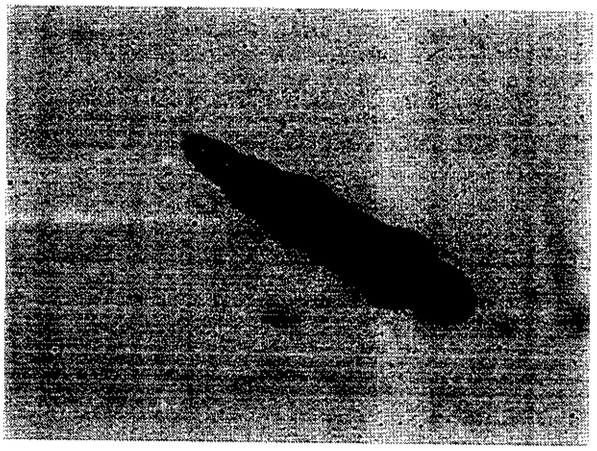


Figure 6.2.1 *Alternaria* Fungus

A set of eight samples of this genus were processed using the above algorithms and formulas. The following values are obtained

Samples	Perimeter	Area	F_{circle}	D_{max}	D_{min}	F_{shape}	D_{circle}	NN Output
A1	3078	185655	0.2463	1224	220	5.5636	274.3048	1.943
A2	3111	185650	0.2410	1229	212	5.7972	274.3011	1.870
A3	2936	238913	0.3483	1154	290	3.9793	311.1716	1.894
A4	3175	230766	0.2877	1143	301	3.7973	305.8201	1.875
A5	3280	249518	0.2914	1150	330	3.4848	318.0029	1.754
A6	2999	239327	0.3344	1153	314	3.6720	311.4411	1.868
A7	3064	173881	0.2327	1182	189	6.2540	265.4643	1.733
A8	3042	174247	0.2366	1185	204	5.8088	265.7436	1.898

Table 6.2.1. Parameter Values of *Alternaria* Samples

From the results obtained, if the NN Output value lies between 1.5 and 2.5, the fungus belongs to the genus *"Alternaria"*.

6.3 Input Sample: Genus of Fungus – *Curvularia*

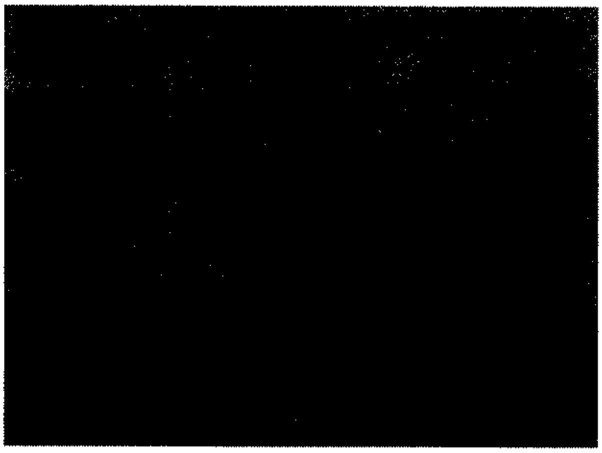


Figure 6.3.1. *Curvularia* Fungus

A set of eight samples of this genus were processed using the above algorithms and formulas. The following values are obtained

amples	Perimeter	Area	F_{circle}	D_{max}	D_{min}	F_{shape}	D_{circle}	NN Output
C1	2228	90892	0.2301	541	213	2.5399	191.9300	2.935
C2	2517	79760	0.1582	462	205	2.2537	179.7930	3.01
C3	1918	83916	0.2867	476	207	2.2995	184.4177	3.105
C4	2473	86986	0.1787	496	207	2.3961	187.7607	2.935
C5	2383	86585	0.1916	504	217	2.3226	187.3275	2.942
C6	2303	80992	0.1919	456	228	2.000	181.1762	2.992
C7	2043	81038	0.2440	454	229	1.9825	181.2277	3.051
C8	3192	82362	0.1016	500	196	2.5510	182.7021	2.877

Table 6.3.1. Parameter Values of *Curvularia* Samples

From the results obtained, if the NN Output value lies between 2.5 and 3.5, the fungus belongs to the genus *"Curvularia"*.

CONCLUSION

7. CONCLUSION

This project has demonstrated the usefulness of modern technology in identifying the genus to which the fungus belongs. The following Image Processing phases were applied to identify the genus:

1) Image Enhancement
2) Image Segmentation
3) Binary Selection and Correction procedures
4) Recognizing spores as vectorized objects
5) Measurement of objects
6) Classifier
7) Identification of genus

In order to employ this system, the training process is done with a set of images belonging to different genus. The system has been tested with three different genera of fungus namely *Helminthosporium, Alternaria* and *Curvularia.* The results of this test indicate that this system could identify the genus to which the fungus belongs with a higher level of accuracy.

FUTURE ENHANCEMENTS

8. FUTURE ENHANCEMENTS

The work presented in this project opens up lot of areas for future research. The following are some of the enhancements that can be done in the future:

- Currently, this system focuses only on three different genera of fungus namely *Helminthosporium, Alternaria* and *Curvularia*. This can be extended by considering many other new genera of fungus.
- The image samples can be provided by direct acquisition of input sample images via Internet.

APPENDICES

9. APPENDICES

9.1. Sample Code

9.1.1. MATLAB Code

ProcessImage.m

```
function [Areav,Perimeter,Dmax,Dmin,FCircle,FShape,DCircle] =
                                                    ProcessImage()

global img;

% [fname,pname]=uigetfile('*.jpg','Select the Image');

% Reading The Image
I=imread(img);
% figure(1);
% imshow(I)

% Converting into Grayscale
Igray=rgb2gray(I);
% figure(1);
% imshow(Igray)

% Using Structuring element in morphological opening
se=strel('diamond',300);
background=imopen(Igray,se);
% figure(4);
% imshow(background)
bk=imclose(background,se);
% figure(5);
```

```
    % imshow(bk)
    % Eliminating the Background
    Object = imsubtract(Igray,bk);

    % Removing the extra noise by setting a threshold values
    Iadjust = imadjust(Object, stretchlim(Object), [0 1]);
    % figure(6);
    % imshow(Iadjust)
    for i=1:1:1536
      for j=1:1:2048
        if Iadjust(i,j) > 100
          Iadjust(i,j)=0;
        else
          Iadjust(i,j)=255;
        end
      end
    end
    % figure(2);
    % imshow(Iadjust);

    % Creating a Binary Image
    level = graythresh(Iadjust);
    Ibw = im2bw(Iadjust, level);
    % figure(3);
    % imshow(Ibw)
    % Removing Extra noise
    Inoiserem=medfilt2(Ibw,'symmetric');
    % figure(9);
```

```
% imshow(Inoiserem)

% Clearing the objects in the border
Imarearem=bwareaopen(Inoiserem,8000,8);
Imborrem=imclearborder(Imarearem,8);
% figure(10);
% imshow(Imborrem)

% Labeling the objects in the Image
L1=bwlabel(Imborrem,4);

% Measuring the Image Properties
stats=regionprops(L1,'all');

% Eliminating more labeled components. Only one label component
% should be present and that is the fungus image
idx = find([stats.Area] > 70000);
Imunwrem = ismember(L1,idx);
% figure(4);
% imshow(Imunwrem)

% Finding the Boundary Coordinates
[B,L2] = bwboundaries(Imunwrem,'noholes');
L2=bwlabel(Imunwrem,4);

% Displaying the Label Matrix in RGB
% figure(12);
label2rgb(L2,@jet,'k');
```

```
 % Imselectimage=bwselect;


 % Plotting the boundary pixels
 % hold on
 % for k = 1:length(B)
 %   boundary = B{k};
 %   plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
 % end


 % Calculating the Parameters---1) AREA
 %                               2) PERIMETER
 %                               3) FCIRCLE
 %                               4) ORIENTATION
 for k = 1:length(B)
  boundary = B{k};
  delta_sq = diff(boundary).^2;
  Perimeter = round(sum(sqrt(sum(delta_sq,2))));
  Areav = stats(k).Area;
  Orientation=stats(k).Orientation;
  FCircle = 4*pi*Areav/Perimeter^2;
end


% To Reflect the Image by Calulating the Orientation Angle
% Negative -- Dont Reflect....Otherwise reflect
if Orientation < 0
  for i=1:1:1536
     for j=1:1:2048
        Imcomrem(i,j)=Imunwrem(i,j);
        end
      end
  else
      for i=1:1:1536
         for j=1:1:2048
            Imcomrem(i,2048+1-j)=Imunwrem(i,j);
         end
      end
  end
  % figure(13);
  % imshow(Imcomrem)

  % Horizontal Scanning at the top of the image to find the (colht1,rowht1)
  for i=1:1:1536
     colht1=min(find(Imcomrem(i,:)));
     if colht1~=0
        rowht1=i;
        break;
     end
  end

  % Veritical Scanning at the top of the image to find the (colvt1,rowvt1)
  for j=1:1:2048
    rowvt1 = min(find(Imcomrem(:,j)));
    if rowvt1~=0
       colvt1=j;
       break;
    end
    end


    % Horizontal Scanning at the bottom of the image to find the
    (colhb2,rowhb2)
    for i=1536:-1:1
       colhb2=min(find(Imcomrem(i,:)));
       if colhb2~=0
          rowhb2=i;
          break;
       end
    end


   % Vertical Scanning at the bottom of the image to find the
   (colvb2,rowvb2)
   for j=2048:-1:1
     rowvb2 = min(find(Imcomrem(:,j)));
     if rowvb2~=0
        colvb2=j;
        break;
     end
   end


 % Checking Whether verical scanning is to be taken or not
 % If YES set the flag value to 1 and set (colvb2,rowvb2) to 0
 % If NO set the flag value to 0
 if Orientation > 0  && Orientation < 10
         flagbout=1;
 elseif Orientation > 80  && Orientation < 90
```

```
        flagbout=2;
    else
        flagbout=0;
    end

    % Calculating the endpoints A(ca,ra) and B(cb,rb) of the Image to
    % calculate Dmax
    if flagbout==1
        ArDmax=rowvt1;
        AcDmax=colvt1;
        BrDmax=rowvb2;
        BcDmax=colvb2;
    elseif flagbout==2
        ArDmax=rowht1;
        AcDmax=colht1;
        BrDmax=rowhb2;
        BcDmax=colhb2;
    else
    % figure(5);
    % imshow(Imcomrem); hold on;
        endpt=(colht1-colvt1);
        boundary1 = bwtraceboundary(Imcomrem, [rowht1, colht1], 'S',8,
                                                endpt, 'counterclockwise');
    % plot(1+boundary1(:,2),1+boundary1(:,1),'g','LineWidth',4);

        endpb=(colvb2-colhb2);
        boundary2 = bwtraceboundary(Imcomrem, [rowhb2, colhb2], 'S',8,
                                                endpb, 'counterclockwise');
```

```
 % plot(1+boundary2(:,2),1+boundary2(:,1),'g','LineWidth',4);
    ArDmax=round(mean2(boundary1(:,1)));
    AcDmax=round(mean2(boundary1(:,2)));
    BrDmax=round(mean2(boundary2(:,1)));
    BcDmax=round(mean2(boundary2(:,2)));
 end

 % Parameter---- 5) DMAX
 Dmax=round(((ArDmax - BrDmax)^2 + (AcDmax - BcDmax)^2)^(1/2));

 % Calculating the parameter DMIN
 % STEP:1 Calculate the midpoints of 'A' and 'B'. The midpoint
 % "(ABmidc,ABmidr)"
 ABmidr=round((ArDmax+BrDmax)/2);
 ABmidc=round((AcDmax+BcDmax)/2);

 % STEP:2 Perform horizontal scanning along the 'ABmidr' and find the
 % first on pixel value(column)
 chlr=min(find(Imcomrem(ABmidr,:)));

 % STEP:3 Perform horizontal scanning along the 'ABmidr' and find the
 % last on pixel value(column)
 chrl=max(find(Imcomrem(ABmidr,:)));

 % STEP:4 Find the midpoint of output values obtained in 'STEP:2' and
 % 'STEP:3'. The midpoint  "(midc,ABmidr)"
 midc=round((chlr+chrl)/2);
```

```
% STEP:5 Perform vertical scanning along the 'midc' and find the first on
% pixel value(row)
rvtb=min(find(Imcomrem(:,midc)));

% STEP:6 Perform vertical scanning along the 'midc' and find the last on
% pixel value(row)
rvbt=max(find(Imcomrem(:,midc)));

% STEP:7 Find the midpoint of output values obtained in 'STEP:2' and
% 'STEP:5'.
ArDmin=round((ABmidr+rvbt)/2);
AcDmin=round((chlr+midc)/2);

% STEP:8 Find the midpoint of output values obtained in 'STEP:3' and
% 'STEP:6'.
BrDmin=round((rvtb+ABmidr)/2);
BcDmin=round((midc+chrl)/2);

% STEP:9 Parameter--- 6)DMIN
if  abs(Orientation) > 0 && abs(Orientation) < 10
% Distance Between the points of 'STEP:5' and 'STEP:6'
   Dmin=round(((midc - midc)^2 + (rvtb - rvbt)^2)^(1/2));

elseif abs(Orientation) > 80 && abs(Orientation) < 90
% Distance Between the points of 'STEP:2' and 'STEP:3'
   Dmin=round(((ABmidr - ABmidr)^2 + (chlr - chrl)^2)^(1/2));
else
% Distance Between the midpoints of 'STEP:7' and 'STEP:8'
```

```
   Dmin=round(((ArDmin - BrDmin)^2 + (AcDmin - BcDmin)^2)^(1/2));
end


% Parameter--- 7) FSHAPE
%                   8) DCIRCLE
FShape=Dmax/Dmin;
DCircle= (2*((Areav)^(1/2)))/pi;
```

vbtomat.m

```
function vbtomat()
   global Areav;
   global Perimeter;
   global FCircle;
   global Dmax;
   global Dmin;
   global FShape;
   global DCircle;

  inp(1,1)=0;
  inp(2,1)=Perimeter;
  inp(3,1)=Areav;
  inp(4,1)=FCircle;
  inp(5,1)=Dmax;
  inp(6,1)=Dmin;
  inp(7,1)=FShape;
  inp(8,1)=DCircle;
  inpath='E:\MatlabR2006a\work\InputParameters.mat';
  save(inpath,'inp');
  load('E:\MatlabR2006a\work\InputParameters.mat');
  load('E:\MatlabR2006a\work\network2.mat');
```

CreateSimulinkModel.m

```
function CreateSimulinkModel()
    sim('simumodel.mdl');
```

Resultret.m

```
function Resultret()
  load('E:\MatlabR2006a\work\FinalResult.mat');
  global result;
  result=simout(2,1);
```

CloseClear.m

```
function CloseClear()
   close all;
   clear all;
```

9.1.2. VB Code

frmmain.frm

```
Public flag As Boolean
Public img As Variant

Private Sub mniexit_Click()
        Unload Me
End Sub
```

```
Private Sub mniopen_Click()
        CommonDialog1.Filter = "JPEGImageFiles | *.jpg"
        CommonDialog1.ShowOpen
        img = CommonDialog1.FileName
        If img = "" Then
           MsgBox "Select an Image", vbInformation, "Alert"
        Else
           flag - CommonDialog1.CancelError
           frmprocess.Show
        End If
        flag = CommonDialog1.CancelError
        frmprocess.Show
End Sub
```

frmprocess.frm

```
Public object As New Seed.Seedclass
Dim Matlab As MLApp.MLApp
Public Areav As Variant
Public Perimeter As Variant
Public Dmax As Variant
Public Dmin As Variant
Public FCircle As Variant
Public FShape As Variant
Public DCircle As Variant
Public resultvb As Variant
Dim a As Double
Dim counter As Integer
```

```
    Private Sub cmdprocess_Click()
        frmprocess.Caption = "Processing Image......"
        If cmdprocess.Enabled = True Then
            a = Shell("E:\progresspro.exe", vbNormalFocus)
            Call object.ProcessImage(7, Areav, Perimeter, Dmax, Dmin,
                                        FCircle, FShape, DCircle)
            Timer1.Enabled = True
        End If
    End Sub
    Private Sub Form_Load()
        Set Matlab = New MLApp.MLApp
        OLE1.Visible = False
        If frmmain.flag = False Then
            Image1.Picture = LoadPicture(frmmain.img)
        End If
        object.img = frmmain.img
End Sub
Private Sub Timer1_Timer()
    frmprocess.Caption = "Process Completed."
    Label2.Visible = True
    counter = counter + 1
    If counter = 20 Then
        Timer1.Enabled = False
        Unload Me
        frmresult.Show
    End If
nd Sub
```

frmresult.frm

```
Dim Matlab As MLApp.MLApp

Private Sub cmdrefresh_Click()
        frmprocess.object.CloseClear
        txtarea.Text = ""
        txtperi.Text = ""
        txtfcircle.Text = ""
        txtdmax.Text = ""
        txtdmin.Text = ""
        txtfshape.Text = ""
        txtdcircle.Text = ""
        txtresult.Text = ""
End Sub

Private Sub Form_Load()
        Set Matlab = New MLApp.MLApp
        Frame1.Visible = True
        cmdrefresh.Visible = True
        txtarea.Text = frmprocess.Areav
        txtperi.Text = frmprocess.Perimeter
        txtfcircle.Text = frmprocess.FCircle
        txtdmax.Text = frmprocess.Dmax
        txtdmin.Text = frmprocess.Dmin
        txtfshape.Text = frmprocess.FShape
        txtdcircle.Text = frmprocess.DCircle
        frmprocess.object.Areav = frmprocess.Areav
        frmprocess.object.Perimeter = frmprocess.Perimeter
        frmprocess.object.Dmax = frmprocess.Dmax
        frmprocess.object.Dmin = frmprocess.Dmin
        frmprocess.object.FCircle = frmprocess.FCircle
        frmprocess.object.FShape = frmprocess.FShape
        frmprocess.object.DCircle = frmprocess.DCircle
        Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
        Call frmprocess.object.vbtomat

        Matlab.Execute ("cd E:\MatlabR2006a\work")
        Matlab.Visible = 0
        Matlab.Execute ("CreateSimulinkModel")

        Call frmprocess.object.Resultret
        resultvb = frmprocess.object.result
        txtresult = resultvb

        resultdisplay.Visible = True
        resultdisplay.Caption = "From the result obtained....."
                imagegenus.Visible = True
        If resultvb < 1.5 And resultvb > 0 Then
                imagegenus1.Caption = "The Fungus belongs to the Genus:
                                                    HELMINTHOSPORIUM"
        ElseIf resultvb >= 1.5 And resultvb < 2.5 Then
                imagegenus1.Caption = "The Fungus belongs to the Genus:
                                                          ALTERNARIA"
        ElseIf resultvb >= 2.5 And resultvb < 3.5 Then
                imagegenus1.Caption = "The Fungus belongs to the Genus:
                                                        CURVULARIA"
        Else
                imagegenus1.Caption = "The image belongs to some other
                        genus other than Helminthosporium, Alternaria and
                                                        Curvularia"
        End If
        Timer1.Enabled = False
End Sub
```

frmhome.frm

```
Private Sub cmdgo_Click()
        frmmain.Show
        Unload Me
End Sub

Private Sub Form_Load()
        WebBrowser1.Navigate "E:\Computerized Image Analysis in Seed
                                        Borne Fungus\home.htm"
End Sub
```

9.2. Sample Output

9.2.1. VB Screen Shots

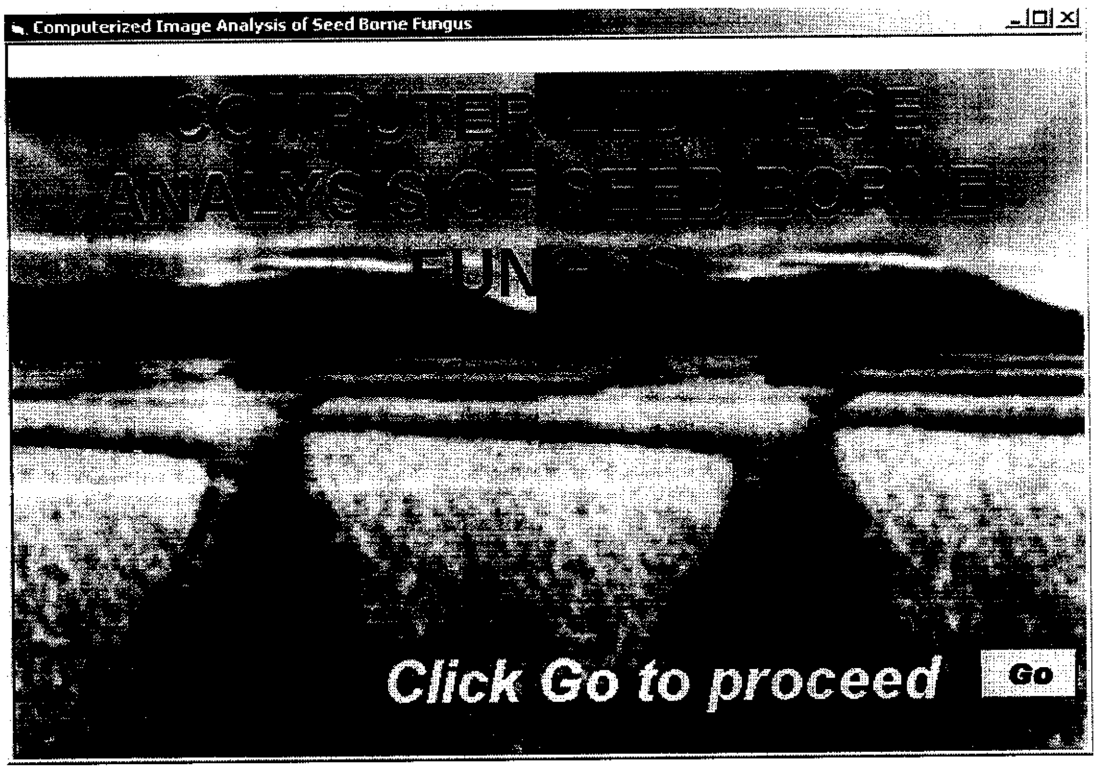

Computerized Image Analysis of Seed Borne Fungus
COMPUTERIZED IMAGE ANALYSIS OF SEED BORNE FUNGUS
Click Go to proceed
Go

Figure 9.2.1.1. Starting Screen of Image Analysis System

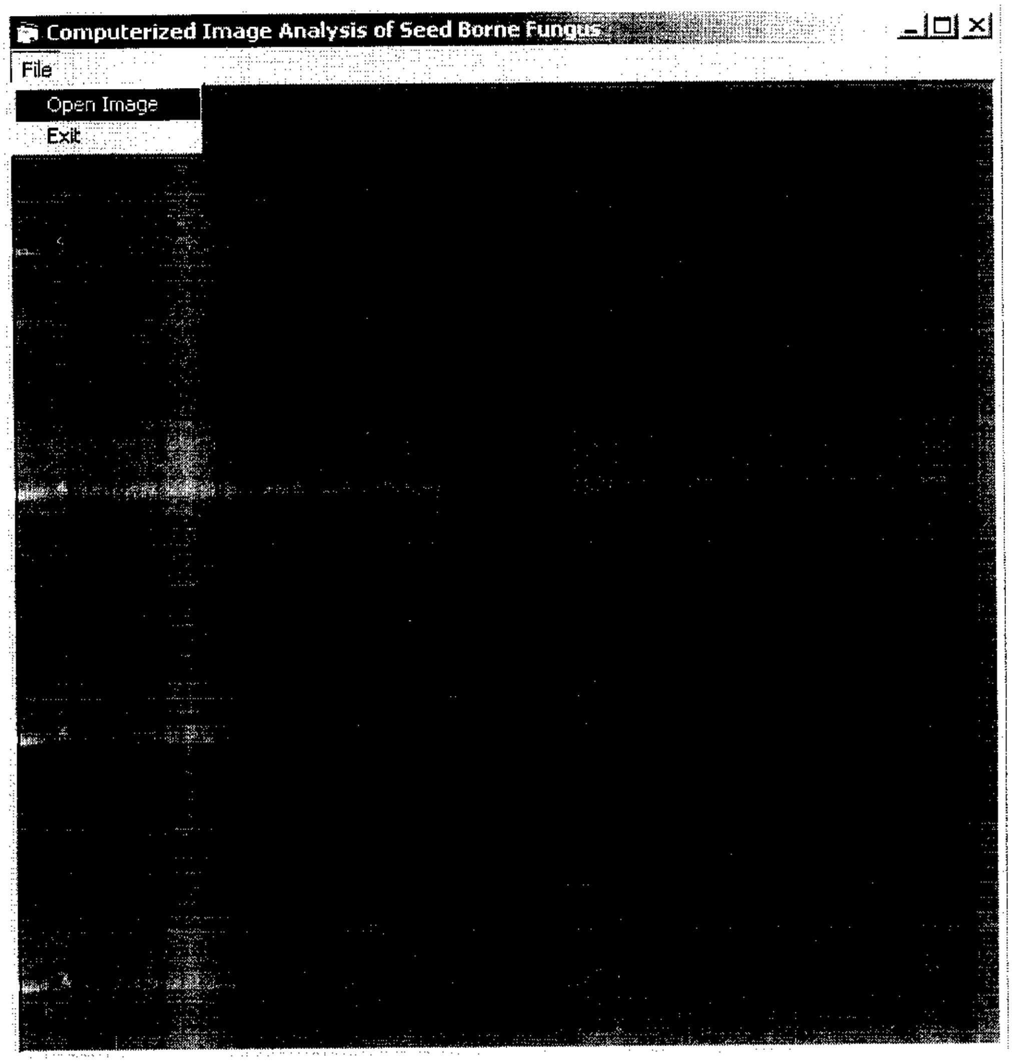
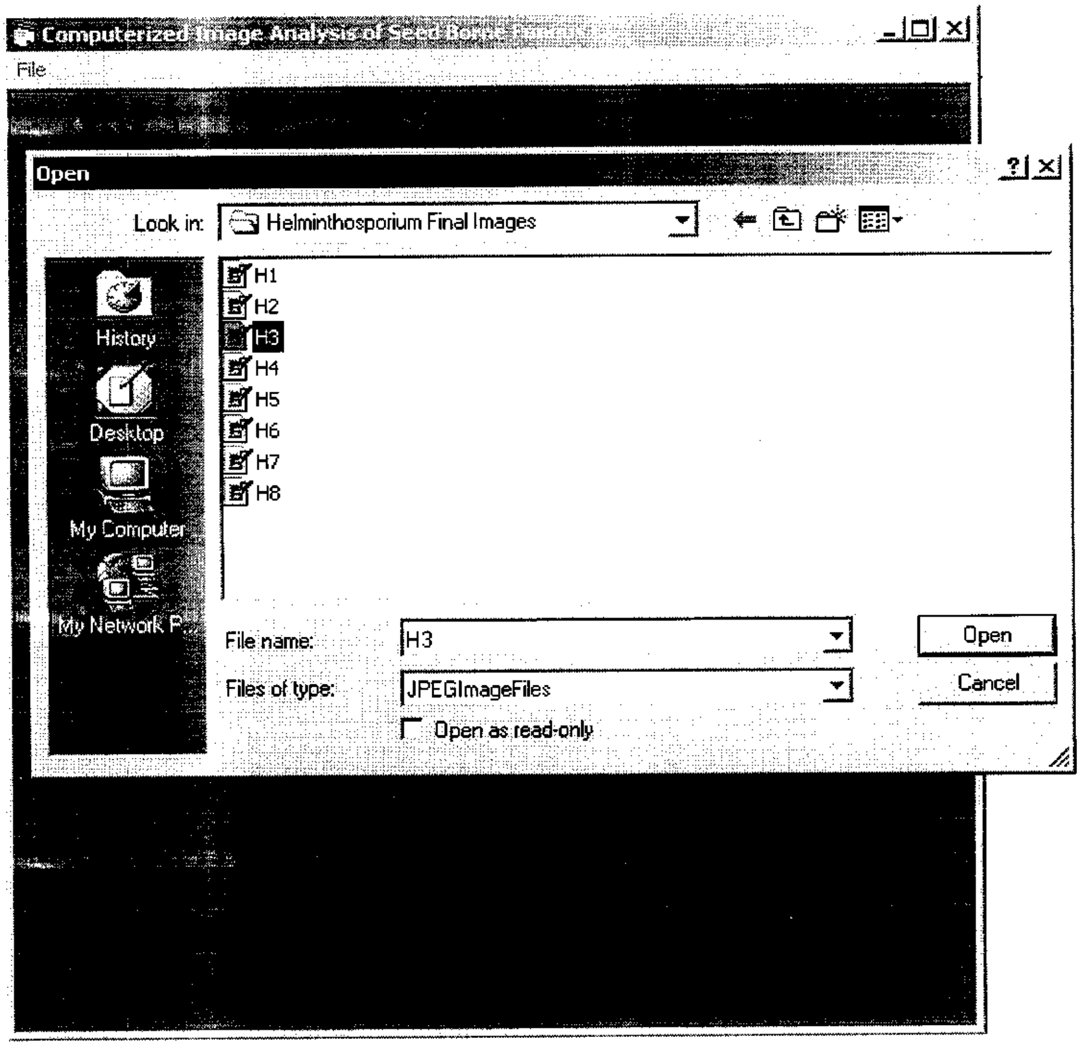


Figure 9.2.1.2. Screen shot of Image Analysis System (i)



File name: H3

Files of type: JPEGImageFiles

Figure 9.2.1.3. Screen shot of Image Analysis System (ii)

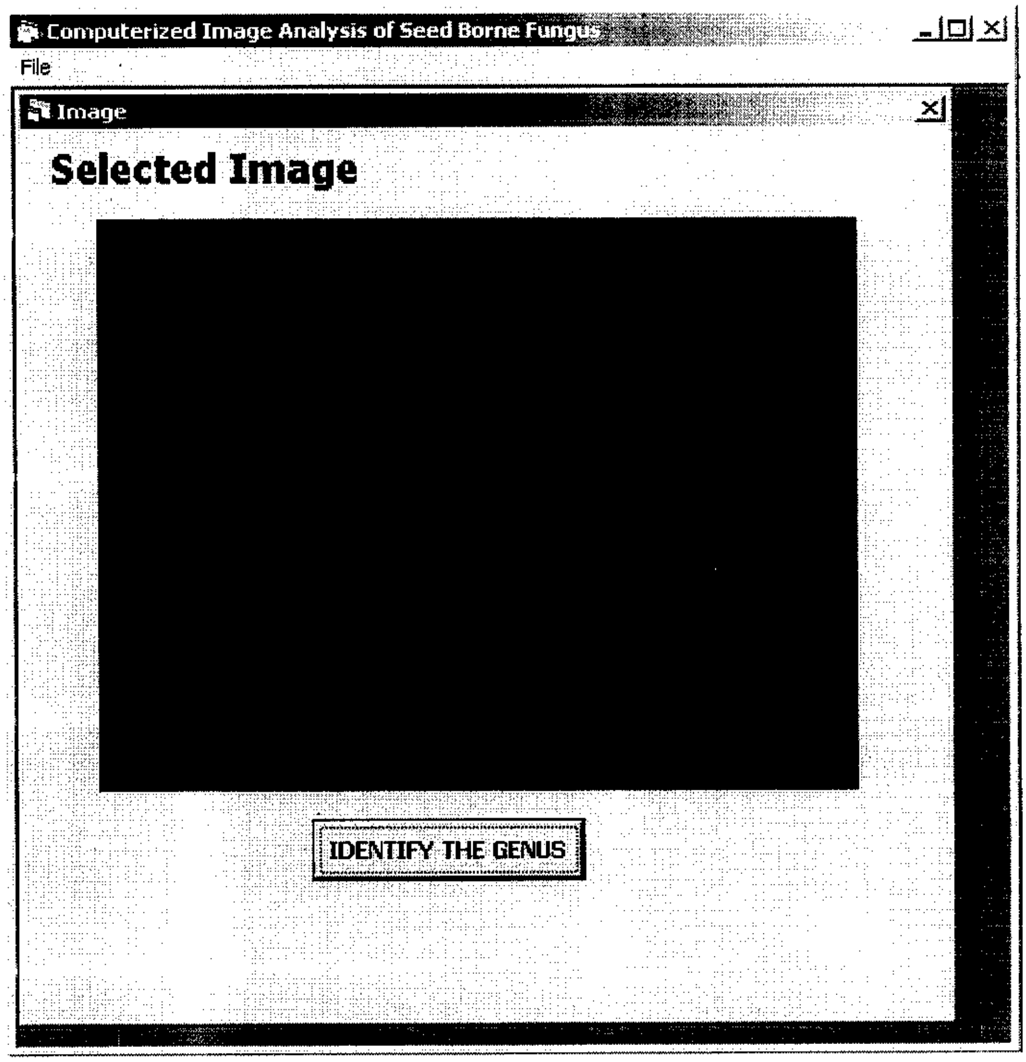


Figure 9.2.1.4. Screen shot of Image Analysis System (iii)

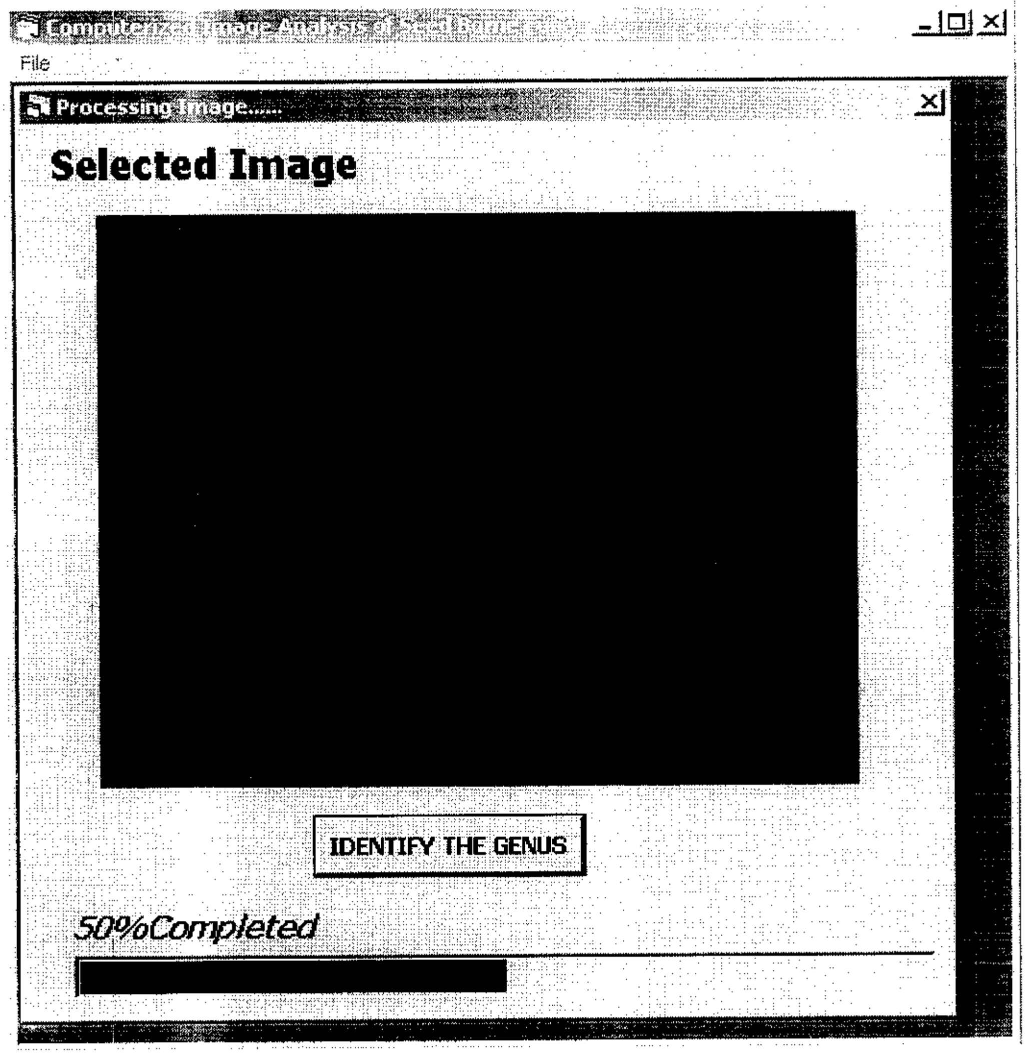
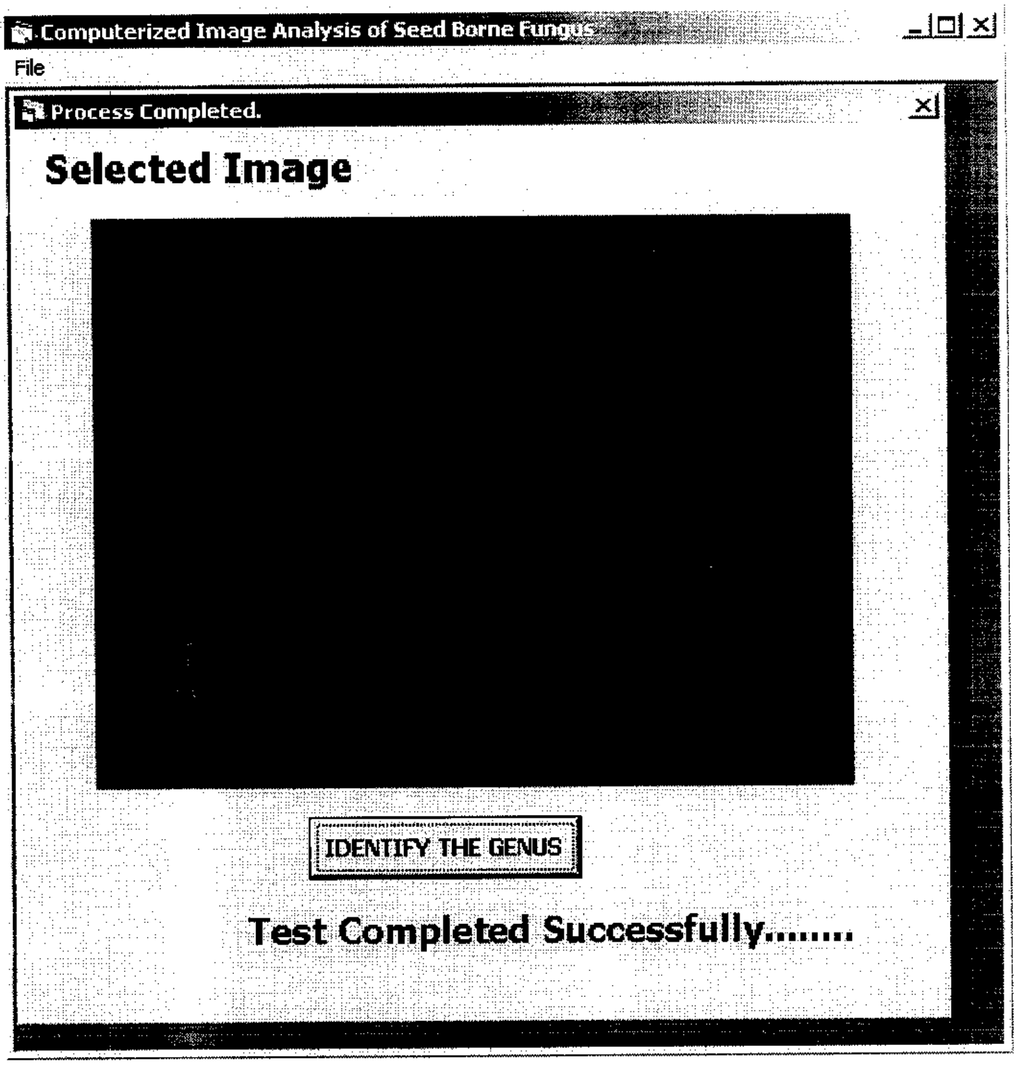


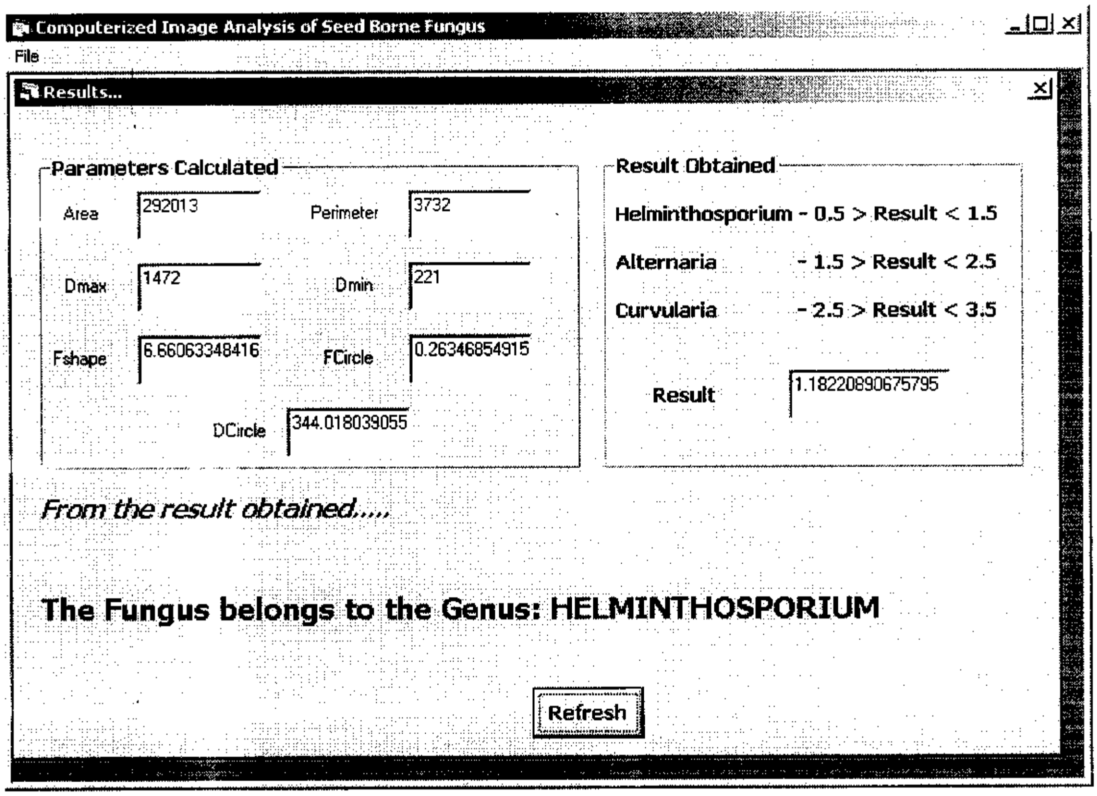
Figure 9.2.1.5. Screen shot of Image Analysis System (iv)



Selected Image

Test Completed Successfully........

Figure 9.2.1.6. Screen shot of Image Analysis System (v)



Computerized Image Analysis of Seed Borne Fungus

File

Results...

Parameters Calculated

Area 292013 Perimeter 3732

Dmax 1472 Dmin 221

Fshape 6.66063348416 FCircle 0.26346854915

DCircle 344.018039055

Result Obtained

Helminthosporium - 0.5 > Result < 1.5

Alternaria - 1.5 > Result < 2.5

Curvularia - 2.5 > Result < 3.5

Result 1.18220890675795

From the result obtained.....

The Fungus belongs to the Genus: HELMINTHOSPORIUM

Figure 9.2.1.7. Screen shot of Image Analysis System (vi)

8.2.2. MATLAB Screen Shots

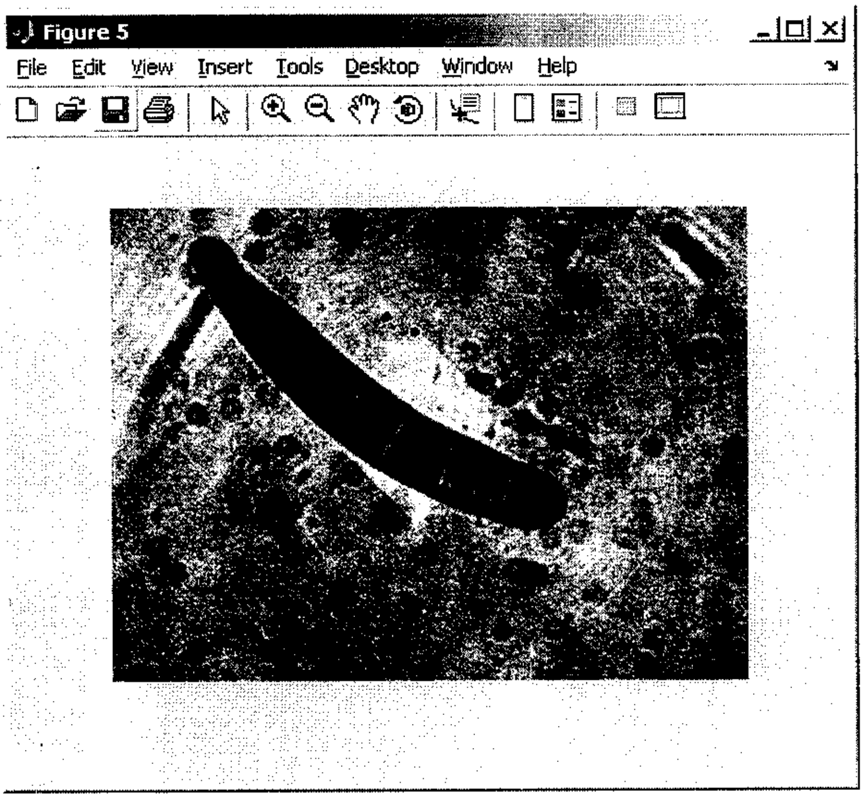


Figure 9.2.2.1. Screen shot After Image Enhancement Phase

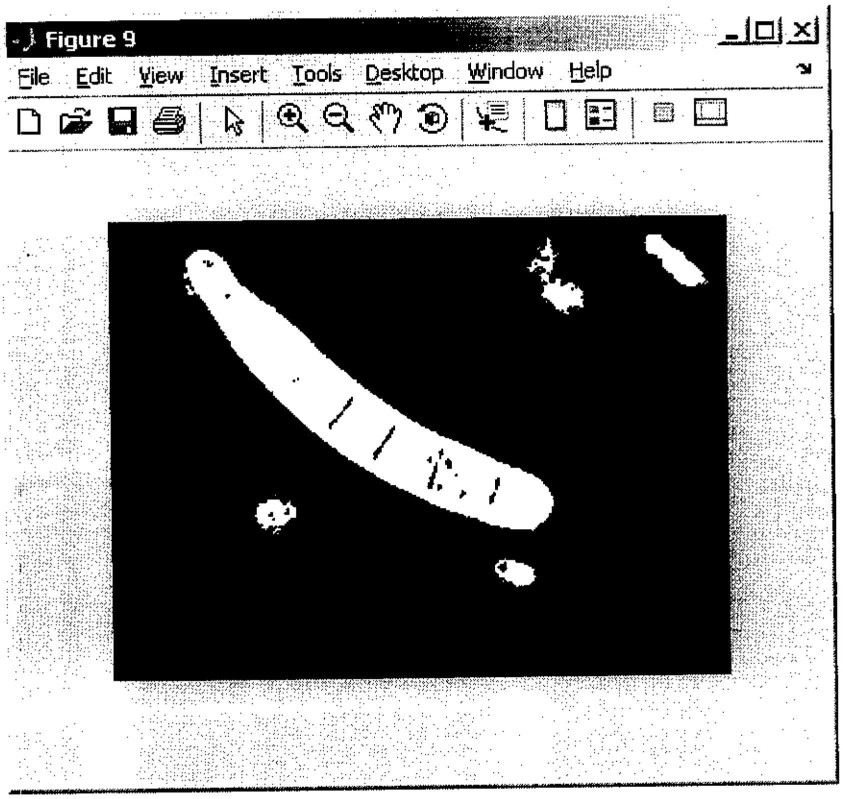


Figure 9.2.2.2. Screen shot After Image Segmentation Phase

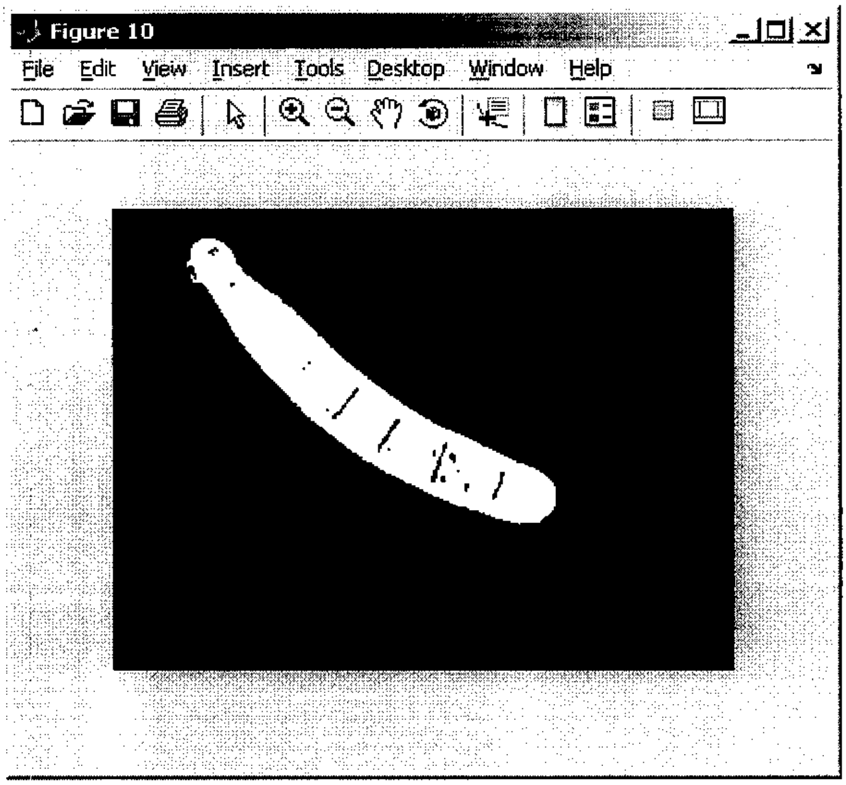


Figure 9.2.2.3. Screen shot after Binary Selection and Correction Procedures Phase

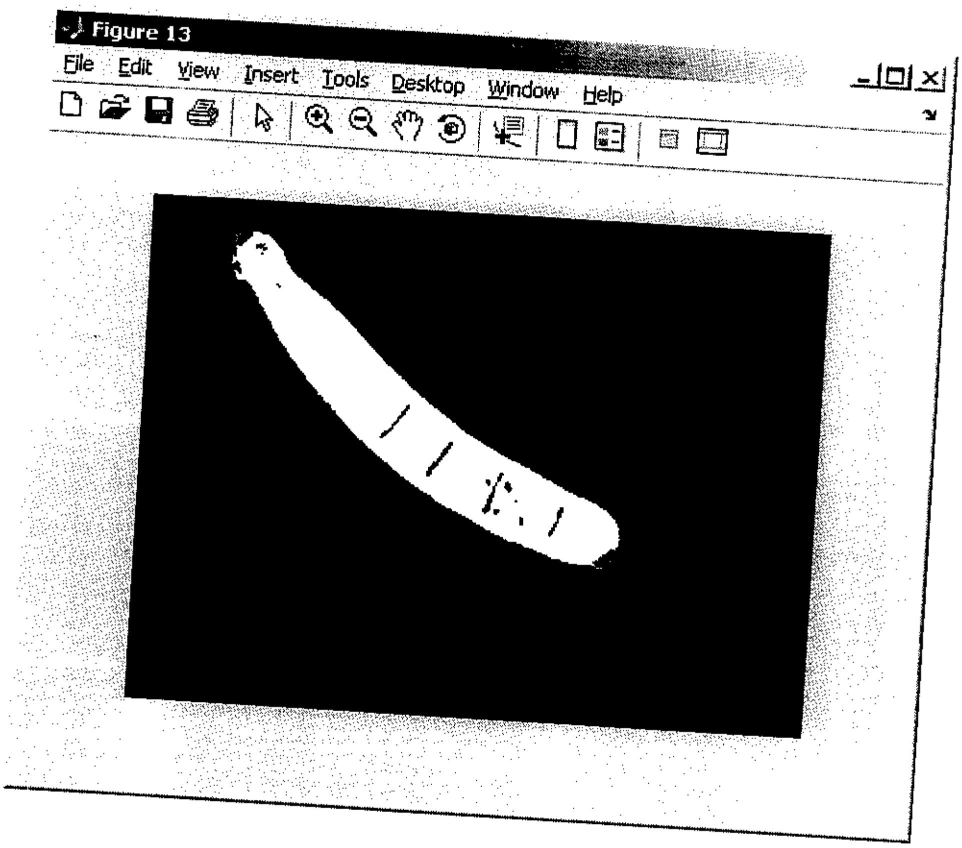


Figure 9.2.2.4. Screen shot after Measurement of Objects Phase

8.2.3. NEURAL NETWORK Screen Shots

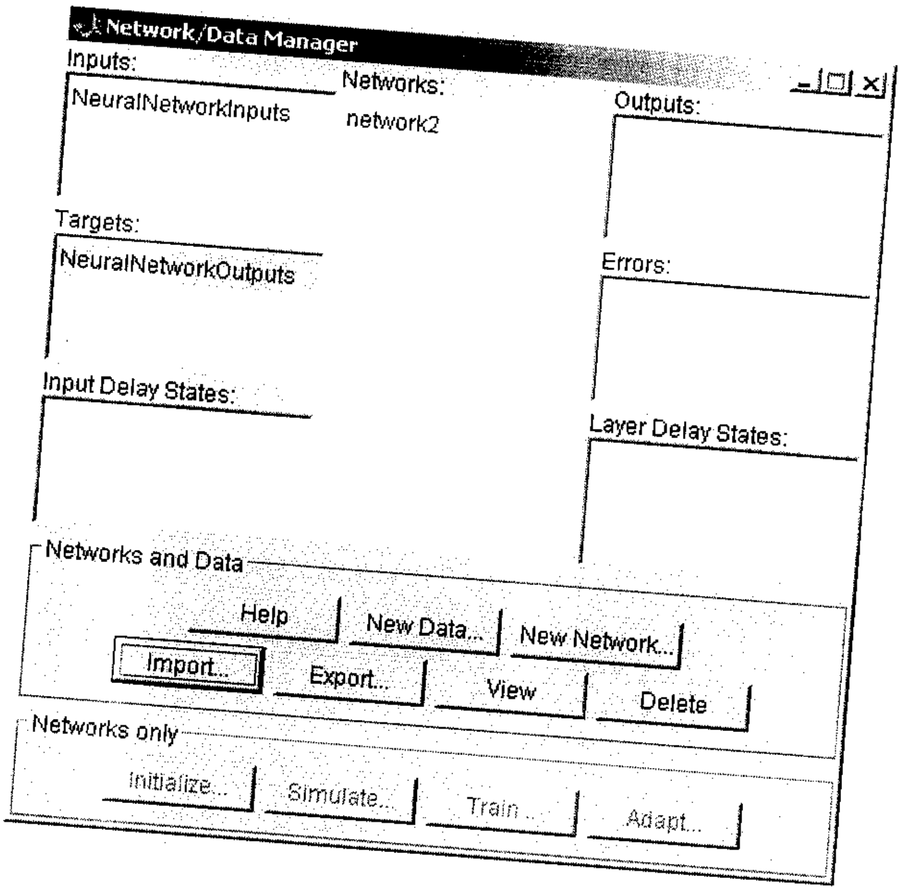
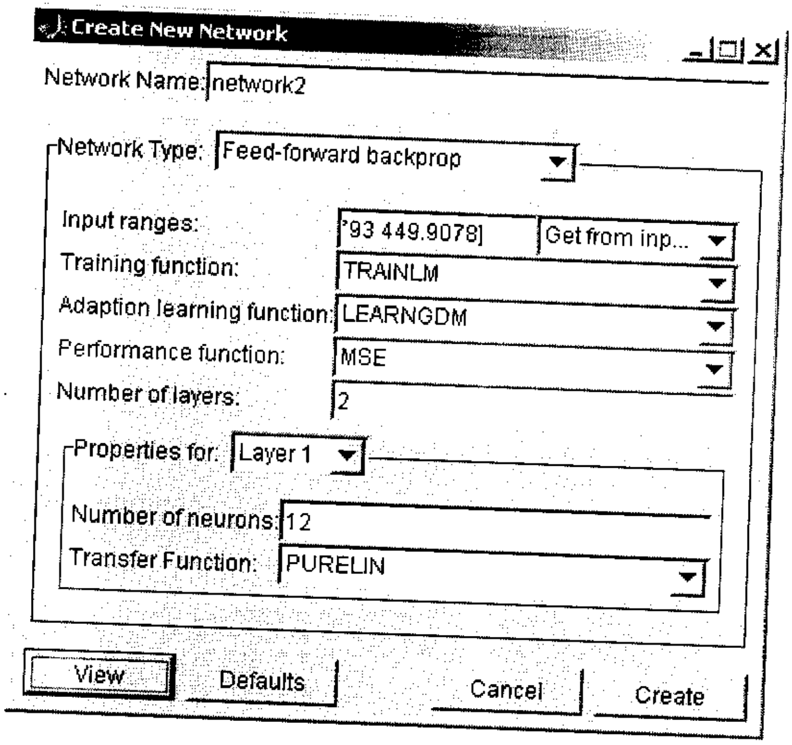


Figure 9.2.3.1. Screen shot of Network Manager



Network Name: network2

Network Type: Feed-forward backprop

Input ranges: '93 449.9078] Get from inp...

Training function: TRAINLM

Adaption learning function: LEARNGDM

Performance function: MSE

Number of layers: 2

Properties for: Layer 1

Number of neurons: 12

Transfer Function: PURELIN

Figure 9.2.3.2. Screen shot of Network Creation

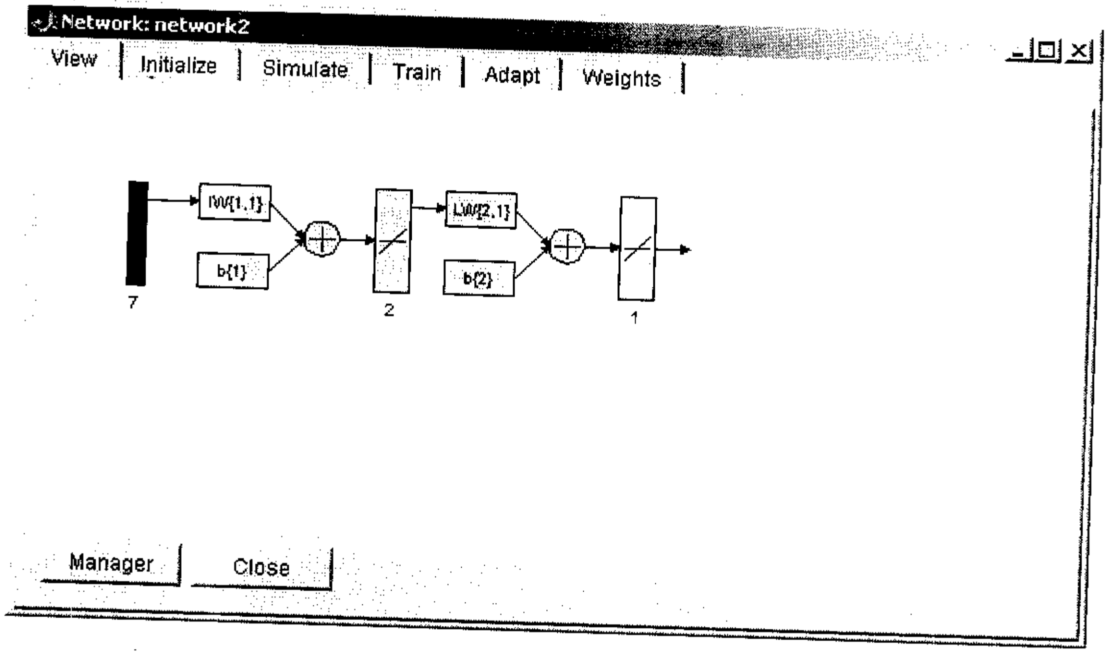
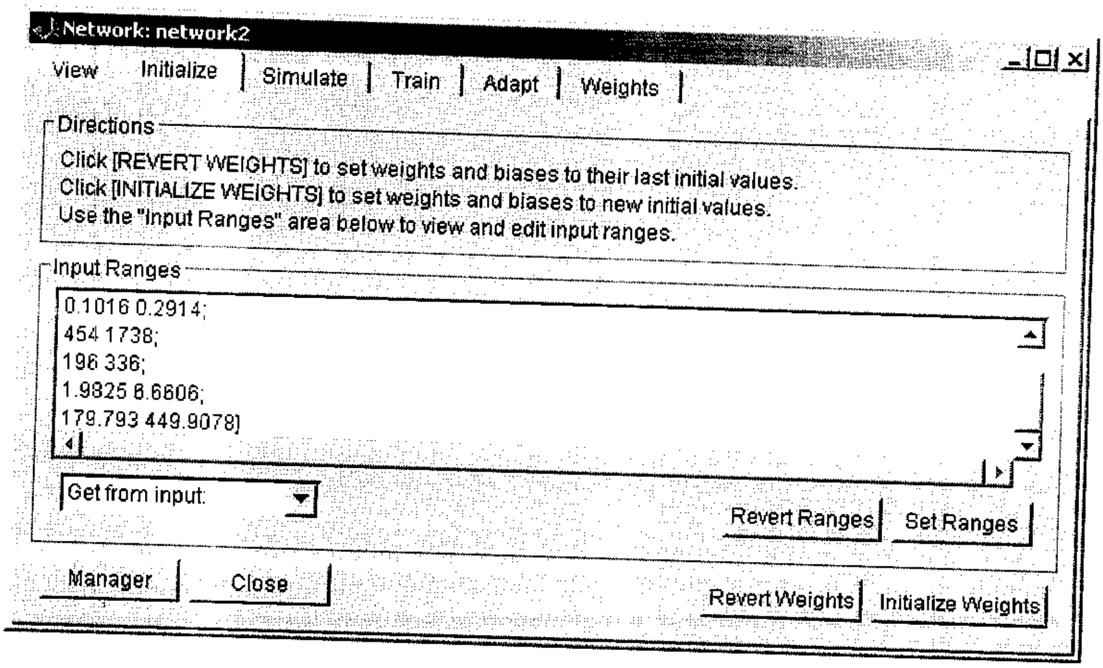


Figure 9.2.3.3. Screen shot of Training the Network (i)



Directions

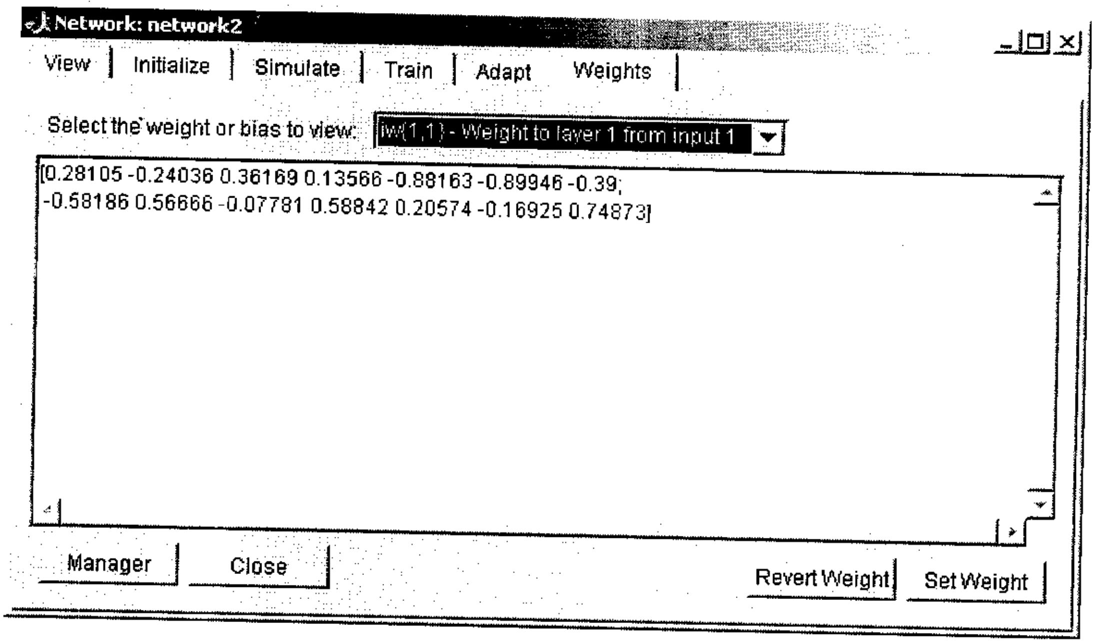
Click [REVERT WEIGHTS] to set weights and biases to their last initial values.
Click [INITIALIZE WEIGHTS] to set weights and biases to new initial values.
Use the "Input Ranges" area below to view and edit input ranges.

Input Ranges

```
0.1016 0.2914;
454 1738;
196 336;
1.9825 6.6606;
179.793 449.9078]
```

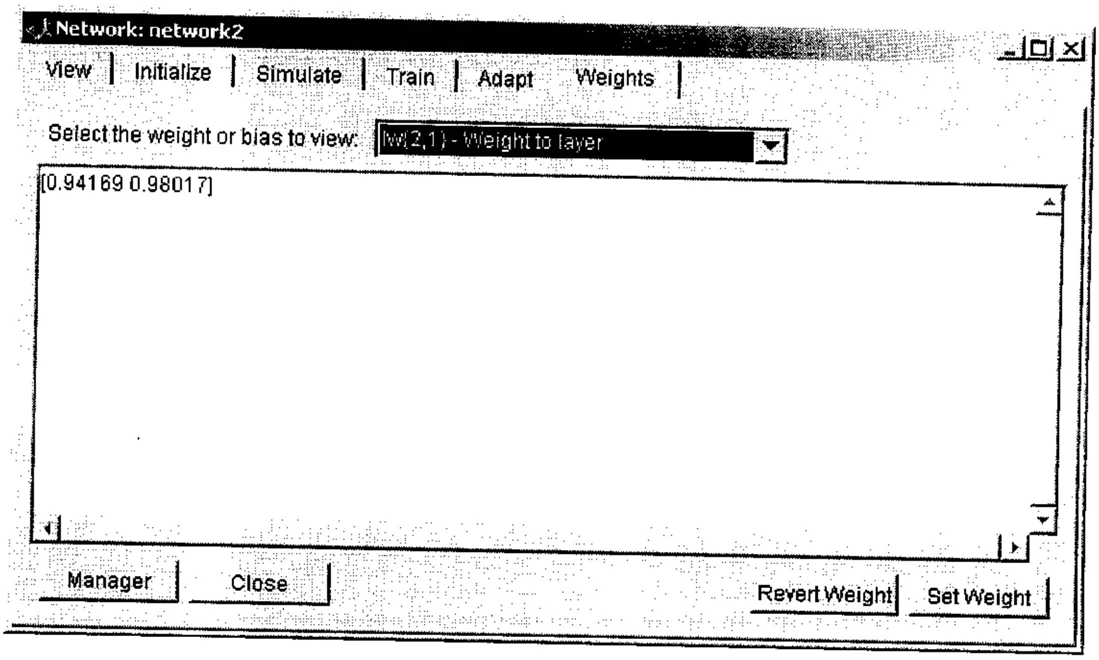
Get from input:

Figure 9.2.3.4. Screen shot of Training the Network (ii)



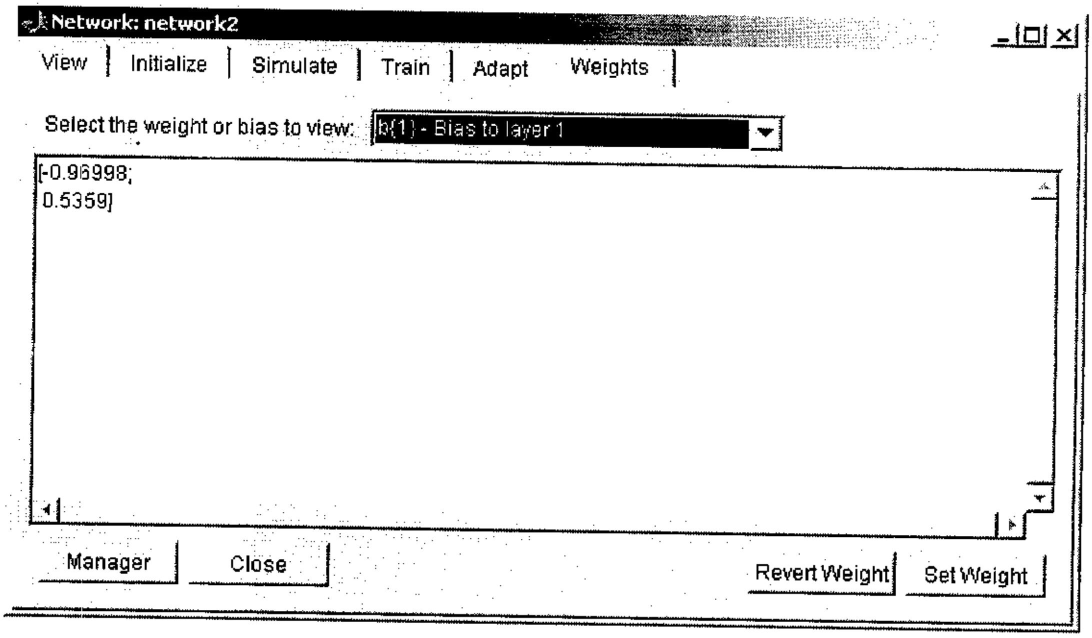
```
[0.28105 -0.24036 0.36169 0.13566 -0.88163 -0.89946 -0.39;
-0.58186 0.56666 -0.07781 0.58842 0.20574 -0.16925 0.74873]
```

Figure 9.2.3.5. Screen shot of Training the Network (iii)



```
[0.94169 0.98017]
```

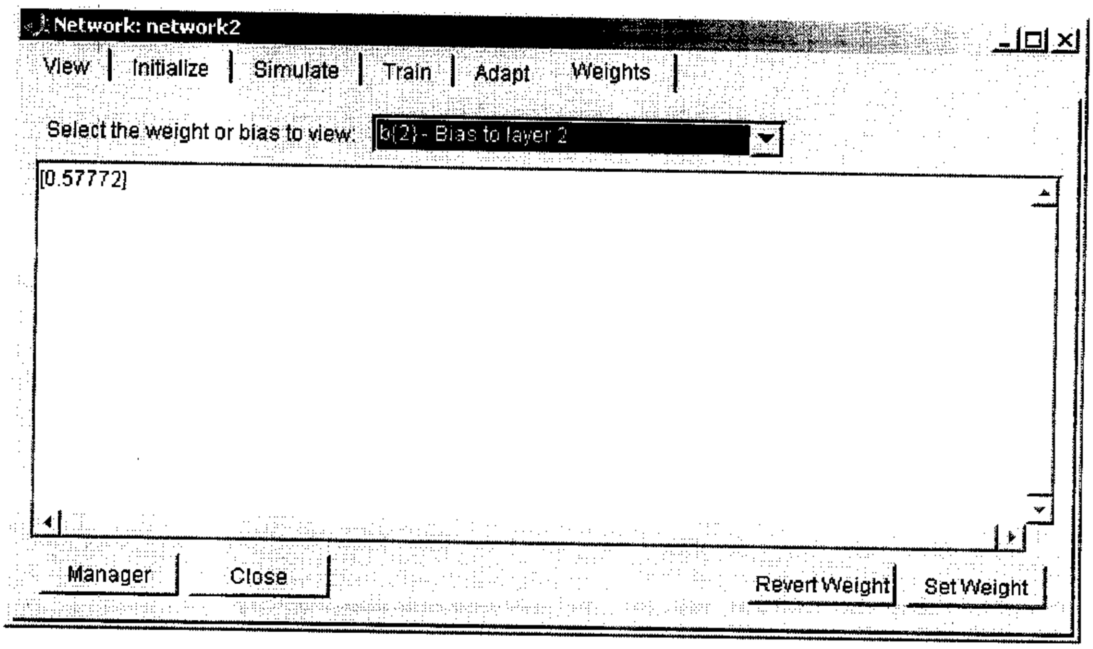
Figure 9.2.3.6. Screen shot of Training the Network (iv)



Select the weight or bias to view: b{1} - Bias to layer 1

```
[-0.96998;
0.5359]
```

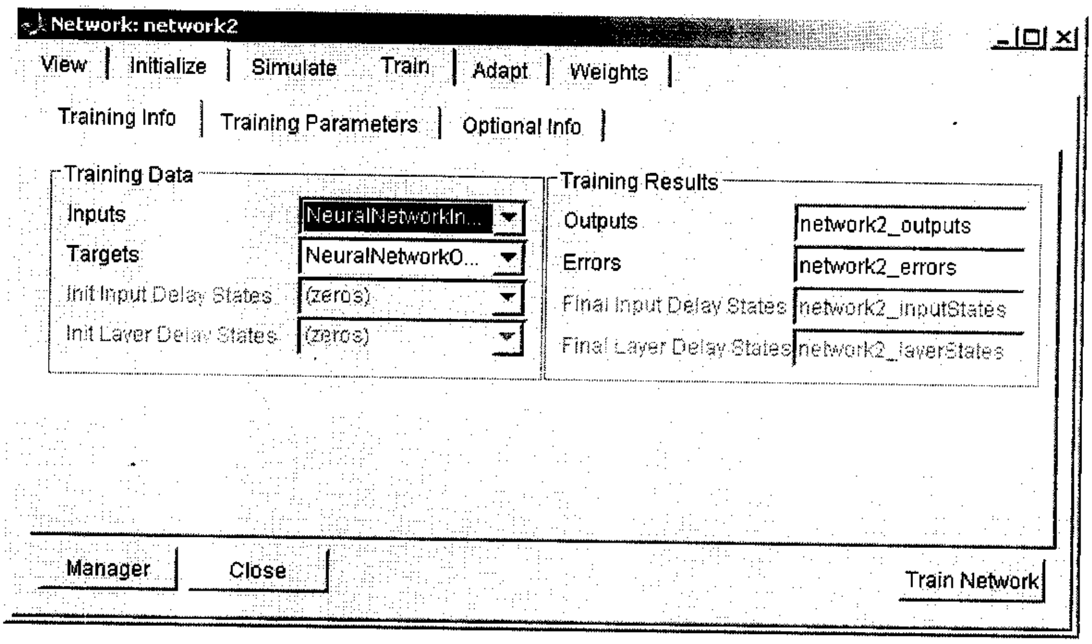
Figure 9.2.3.7. Screen shot of Training the Network (v)



Select the weight or bias to view: b{2} - Bias to layer 2

```
[0.57772]
```

Figure 9.2.3.8. Screen shot of Training the Network (vi)



Training Data

Inputs	NeuralNetworkIn...
Targets	NeuralNetworkO...
Init Input Delay States	(zeros)
Init Layer Delay States	(zeros)

Training Results

Outputs	network2_outputs
Errors	network2_errors
Final Input Delay States	network2_inputStates
Final Layer Delay States	network2_layerStates

Figure 9.2.3.9. Screen shot of Training the Network (vii)

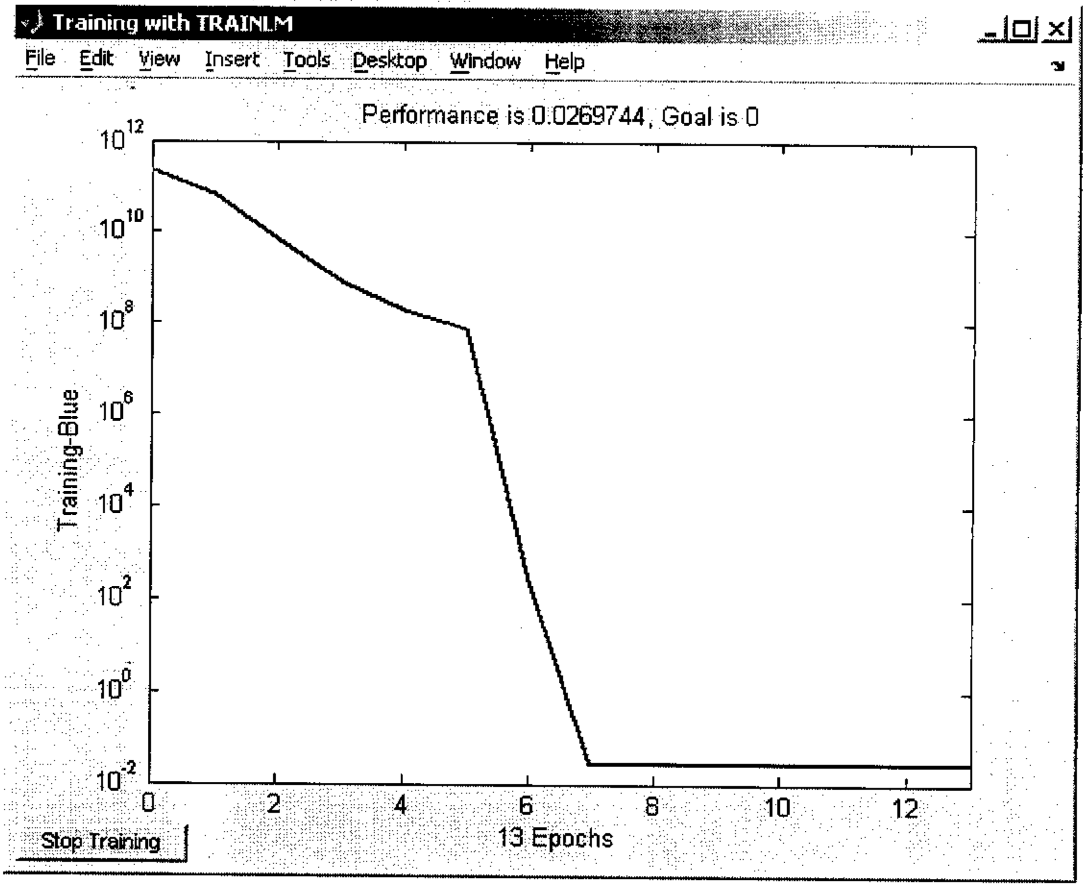


Figure 9.2.3.10. Screen shot of Training Result

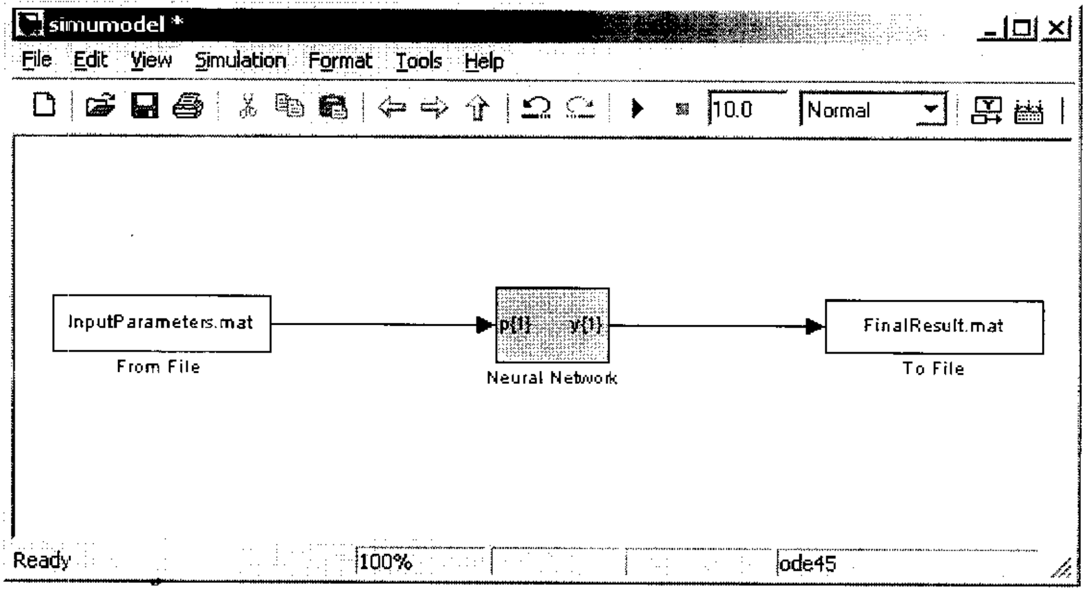


Figure 9.2.3.11. Simulink Model for Trained Network

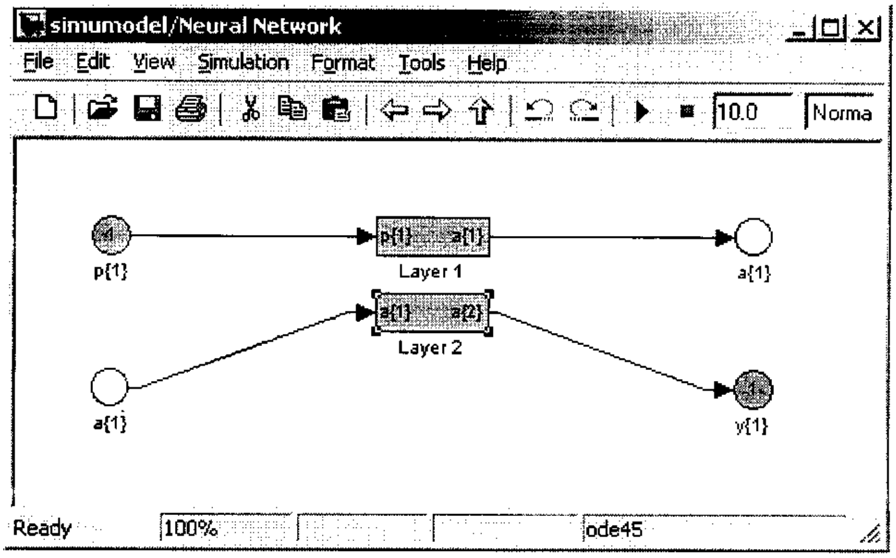


Figure 9.2.3.12. Screen shot of Network Layers

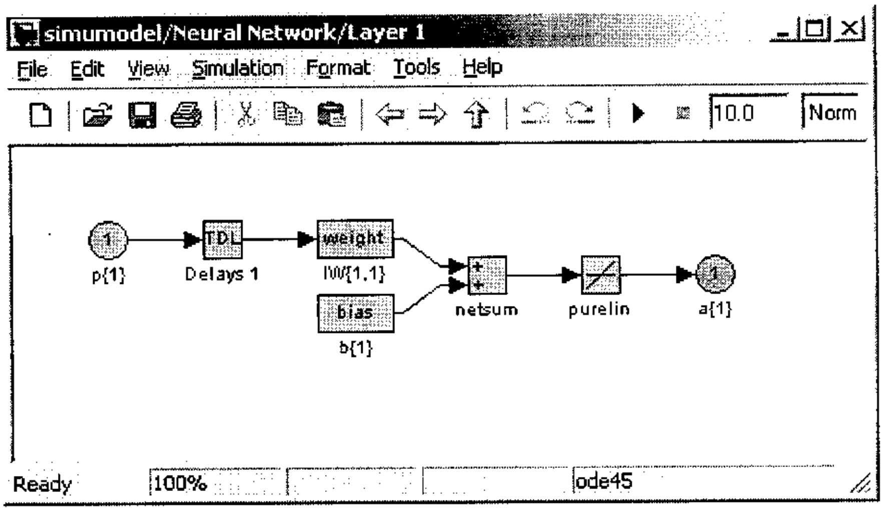


Figure 9.2.3.13. Screen shot of weight assignment for Layer1

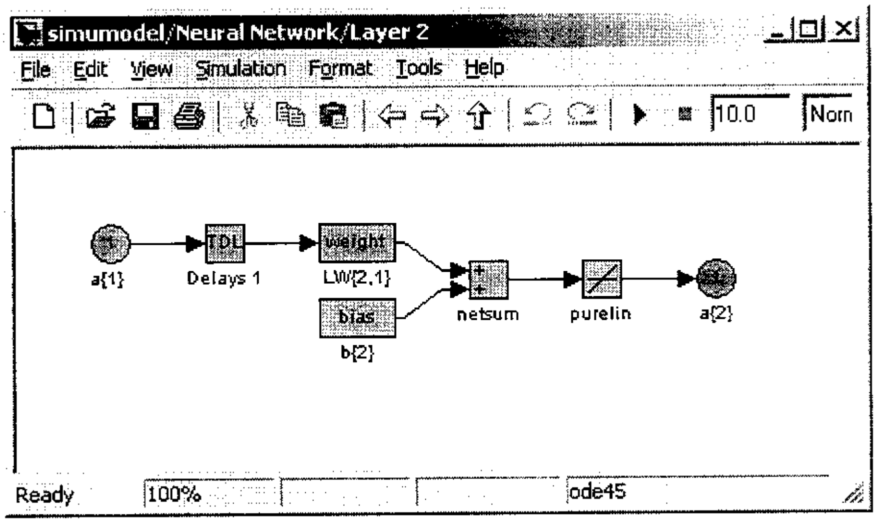


Figure 9.2.3.14. Screen shot of weight assignment for Layer2

REFERENCES

10. REFERENCES

1. Christoph.U.Germeier, Sven Gottwald, and Werner Ruhmann (Feb 2001) – "Computerized Image Analysis in *Fusarium* Taxonomy", Mycol. Res. 105(2):206-214.
2. R.C.Gonzalez and R.E.Woods, 2001 - "Digital Image Processing using MATLAB"
3. R.C.Gonzalez and R.E.Woods, 2001-"Digital Image Processing"
4. MATLAB Image Processing Toolbox, User Guide Version 7.2
5. MATLAB Neural Network Toolbox, User Guide Version 7.2
6. WebLinks - http://www.dsprelated.com
7. WebLinks - www.mathworks.com/**matlab**central
8. WebLinks- www.mathworks.com/ access/helpdesk/ help/techdoc /**matlab**_external/f135590.html
9. Weblinks - http://www.codeproject.com/ KB/DLL/Matlab NN_ BKM.aspx