NEURAL NETWORK SYSTEM TO RECOGNIZE PARASITES ON FISH IMAGES

A NEURAL NETWORK BASED SYSTEM TO RECOGNIZE, DETECT AND LOCATE SEALWORM PARASITIC INFESTATIONS ON COD FISH FILLET IMAGES

BY

EMMANUEL B. ARYEE, B.Sc.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

April 1991

MASTER OF SCIENCE (1991)
(Computation)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: **A Neural Network Based System to Recognize, Detect and Locate Sealworm Parasitic Infestations on Cod Fish Fillet images**

AUTHOR: **Emmanuel B. Aryee,**
B.Sc. (Univ. of Science & Tech., Kumasi, Ghana)

SUPERVISOR: **Dr. S. Poehlman**

NUMBER OF PAGES: x, 137

ABSTRACT

In this project, an investigation of a neural network based system is used to examine the following:

a) the possibility and practicability of analysing and recognising parasites/sealworms on a parasite/sealworm infested cod fish images,

b) the most efficient but robust way of presenting data to the neural network for efficient training and generalisation.

The basic problem is to automate the sorting of sealworm infested cod fish from good normal cod fish using a neural network based system.

The generalised back propagation supervised learning algorithm is used and both steepest descent and conjugate gradient methods are investigated. Various data representation schemes in unprocessed and processed formats before presentation for training of the neural network, are also examined. Finally the level of recognition achieved by the neural network when presented with the cod fish images is computed.

Thus in this project an attempt is made to analyse and find the best components for solving the basic problem and then use this information to develop a neural network based system to recognise, detect and locate parasite/sealworms on cod fish images.

ACKNOWLEDGEMENTS

I wish to express my thanks to Dr. W. F. S. Poehlman, my supervisor, for his direction and assistance with this project work. I would also like to thank the other members of Dr. Poehlman's research group, particularly Peter Lind, Darel Mesher and Sam Demooy for their assistance and help throughout this project. I would also like to acknowledge CANPOLAR INC, and CANPOLAR EAST INC, especially Dr. James R. Rossiter of CANPOLAR INC. for making available the data used in this work and some of their confidential technical reports. Finally I would like to thank the office of the Department of Computer Science and Systems for their help and support during my studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This project investigates and analyses the recognition of image characteristics using a neural network based system. The supervised back propagation model [6] is the most commonly implemented neural network model, perhaps because of its general purpose capabilities. It belongs to a class of artificial neural network that is comprised of few to many non-linear computational elements operating in parallel and arranged in patterns similar to biological neural nets [5]. This massive parallelism is thought by researchers to be essential for high performance speech and image recognition. Thus the approach is to develop an adaptive network model which, when presented with a set of training patterns, learns an internal representation of the input distribution and retains this knowledge via connecting strengths. This learning is referred to as supervised since the input patterns are accompanied by their desired outputs applied by a supervisor(teacher).

Pattern Recognition is the science which attempts to simulate the processes that permit humans to achieve visual and auditive perception by using optical and vision methods to analyse a scene in an image or signals in a sound environment. Image recognition invariably falls under this science and because of the massive parallelism in neural networks, neural networks have been used in the development of systems for assistance in medical diagnosis, in pattern recognition to classify undersea sonar returns, speech and handwritings, etc., and new applications under research include simple vision systems [9]. It is therefore appropiate to consider using the neural network model to analyse cod fish images, and attempt to recognise various attributes or characteristics occuring in these kinds of images.

Processing of image data presents various complexities and difficulties which must be either reduced or eliminated before presentation to the neural network for efficient manipulation:

-the data must be in a form or format that can be numerically processed,

-the data size must be considerably reduced in order to make the analysis with a neural network model practical,

-partitioning of the images down to basic component level such that their characteristics can be used to differentiate a parasitic/sealworm background from a non-parasitic/sealworm background.

The theoretical basis for selecting or choosing of neural network architectures and parameters for learning or training procedures are non-existent and although various researchers are investigating algorithms that improve the training of neural network and remove dependencies upon these learning parameters [11], empirical methods still tend to be the norm presently.

In this project we thus investigate the possibility, practicability and efficiency of using the neural network back propagation technique to analyse images of such complex nature as the cod fish and to recognise and locate parasites/sealworms on these images. Such work should aid development of an Automatic fish inspection system. By virtue of this investigation, the project also looks at various empirical methods used for selecting network parameters, various forms of data representations, and network search methods for training.

The outline of this project is as follows. Chapter 2 describes the inspection system involved and the evolution of research performed for this project. Basic theories of computer vision and neural networks which are utilised in this project are given in Chapter 3. Chapter 4 deals with parameter selection and data representation while Chapter 5 elaborates on the analysis, experimentation and the results obtained. Chapter 6 gives the conclusion to this analysis and suggestions for the future work to extend this research.

On the opposite page, Fig. 1.1 is a picture of the worst case of an infested cod fish fillet showing a cross-section of the types of parasites/sealworms circled and numbered, normally encountered by the fishing industry. The picture is divided into four sections as by the rectangles and labelled COD1, COD2, COD3 and COD4 in this project.

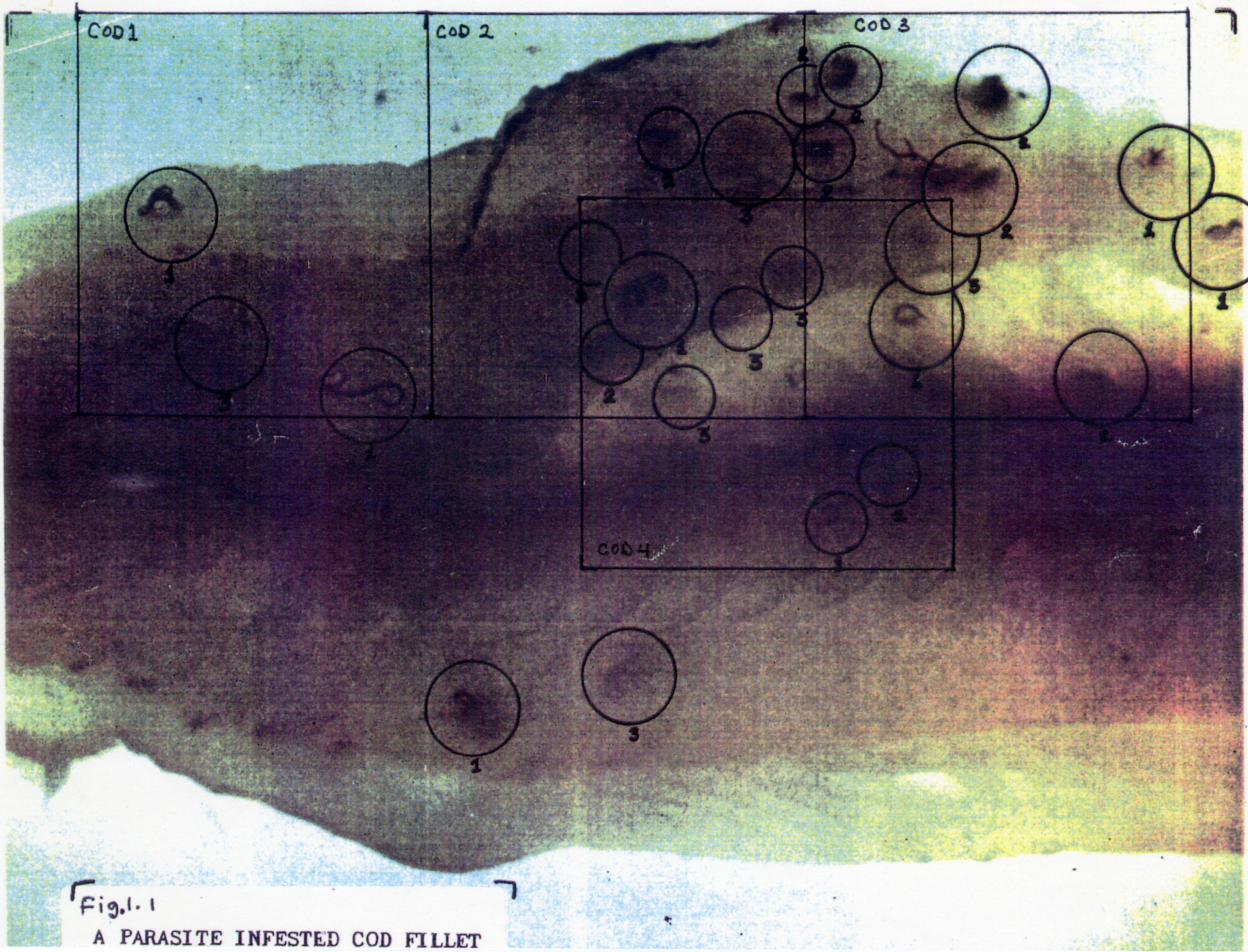


Fig. 1.1

A PARASITE INFESTED COD FILLET

CHAPTER 2

AUTOMATED FISH INSPECTION SYSTEMS

2.1 Introduction

This chapter gives background to Fish Inspection Systems and their objectives, included is a description of an inspection system under development, the motivation for automated systems and a description of an envisaged automated system.

2.2 Background

Parasite/sealworm infestation of fish in general has concerned the fish industry for sometime now and apart from its unacceptability aesthetically, the health concerns to consumers cannot be ignored. Atlantic Canadian Fish Processors incur estimated costs of $50 million annually related to parasites/sealworms in cod fish [21]. As the market for these delicacies increases, pressure for finding efficient and faster methods of identifying and removing these parasite/sealworm becomes more important, more necessary and economically justifiable or viable for even small percentages such as 5-10% of parasite/sealworm infestation.

These parasite/sealworm are known to occur as two varieties, the <u>phocanema</u> and <u>anisakis</u> [20]. Information on these kinds of parasites are still sketchy but are known to be curvy, spiral or wriggly in shape and tend to keep these shapes on attachment to the skin of a cod fish. Thus they are slightly different and more difficult to track than the consistent round shaped parasite/sealworm that are found on flat fish. The parasite/sealworm found on cod fish normally become lodged on the surface skin, in which case, their shape and contrast is easily discernible, as circled and numbered 1 in the cod fish picture given in Fig. 1.1, but in other cases they become deeply lodged in the skin of the cod fish such that they appear as blobs in which case their shape and contrast is not easily discernible. This is shown in the numbered circles 2 & 3 of Fig. 1.1. Their detection and location is therefore complicated by the fact that

-they appear in so many different shapes,

-their contrast levels in the image easily match many of other sections of the fish muscle background,

-more deeply lodged ones apppear in the same contrast and shape as portions of the fish muscle,

-some parasites have hairline shapes that appear similar to the hairline bony structure of the cod fish.

Their detection and complete removal or sorting is thus difficult to achieve without the help of the human perception system.

The traditional ways of detecting, and removing or sorting out these parasite/sealworm infested cod fish from the good fillets has included the very old system of fishermen directly sorting the fish during offloading, to the manual inspection systems being researched and developed by CANPOLAR INC. [20].

2.3 Inspection Systems

CANPOLAR INC. has researched and is developing a prototype inspection system which incorporates the Parasensor(TM), an optical imaging technology, that will greatly provide for reliable inspection of parasite/sealworm in cod fish fillets up to one inch thick, improved inspection reliability, reduced handling costs and improved inspection of over 80% of all fillets produced in Atlantic Canada [21].

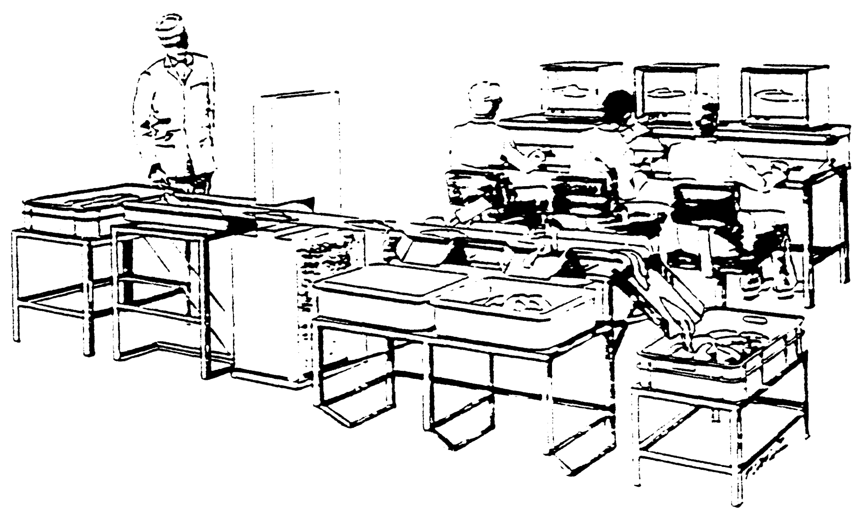


Fig. 2.1 Artist's conception of the prototype [21] (with the permission of CANPOLAR INC.).

An artist's conception of the system under development by CANPOLAR INC. is illustrated in Fig. 2.1. The cod fish is washed, thoroughly cleaned, unwanted parts removed and then processed into fillets. These are then manually loaded unto a conveyor belt which is rolled past an optical imaging system (camera/laser/radar) which captures the image or scan of the fillet which is displayed on three monitors, that have three inspectors who scan the fish images for parasites/sealworms. Upon the sighting of a parasite/sealworm, a button is pressed and the fillet is moved into an appropriate bin reserved for the purpose.

CANPOLAR INC. is actively pursuing research into areas aimed at improving the inspection system such as using laser technologies or radar impulse systems that will pick up deeply embedded parasites/sealworms with the same sensivity as surface infestations of the cod fish fillet. Processors are also seeking technologies for automated detection and removal of these parasites/sealworms [21] and improvements of these methods will be a step towards automation.

2.4 Automated Inspection System

The objective is to reduce the human error and boredom factor (due to the performance of a constant monotonous task) which exist with the manual inspection systems, thereby improving the performance, efficiency, reliability and speed of inspection and thus reducing the cost of operation. Such an automated system requires automatic detection and recognition techniques to separate the infested cod fish fillets from the healthy ones which, in the manual system, is performed by human inspectors. Several computer vision techniques exist in research fields that could be investigated and they tend to be ideal for image enhancement procedures but are very time consuming (shape analysis and edge detection) and very sensitive to noisy backgrounds such that they are unable to perform within practical time constraints.

Humans in their perceptions are able to tolerate ambiguities in input patterns and yet make very good and correct decisions where the input is close or approximately close to the original input. Since neural networks attempt to simulate the workings of the human brain, although not perfectly in any measure, they are able to correctly classify objects when presented with a close approximation of an input pattern on which they have been trained.

Since the time consuming job of training is done off line, recognition during production use is extremely rapid meeting most time critical delivery deadlines in real time. It is therefore prudent to consider using the neural network system in the automatic detection and recognition system of an automated inspection system.

2.4.1 Flowchart of envisaged system

Fig. 2.2 presents the flowchart of the envisaged automated inspection sysytem.

2.4.2 Description of the flowchart

1. After the fish has been washed, thoroughly cleaned, all unwanted parts removed and processed into fillets, it is loaded unto a conveyor belt system, which is configured to move at a maximum rate of one fish per second.

2. A vision system - CAMERA/LASER/RADAR - mounted by the conveyor system captures and digitizes the fish image to grey scale levels as it rolls on the conveyor belt.

3. The digitized image is sent to a hardware/software system that uses the neural network based technique to analyse and process the image and make a determination as to whether a parasite/sealworm is present or not.

4. The determination instruction is sent to an acceptor/implementor as a signal for implementation.

5. A robot arm assembly then implements the determination and directs the fish on the conveyor belt into one of a group of bins.

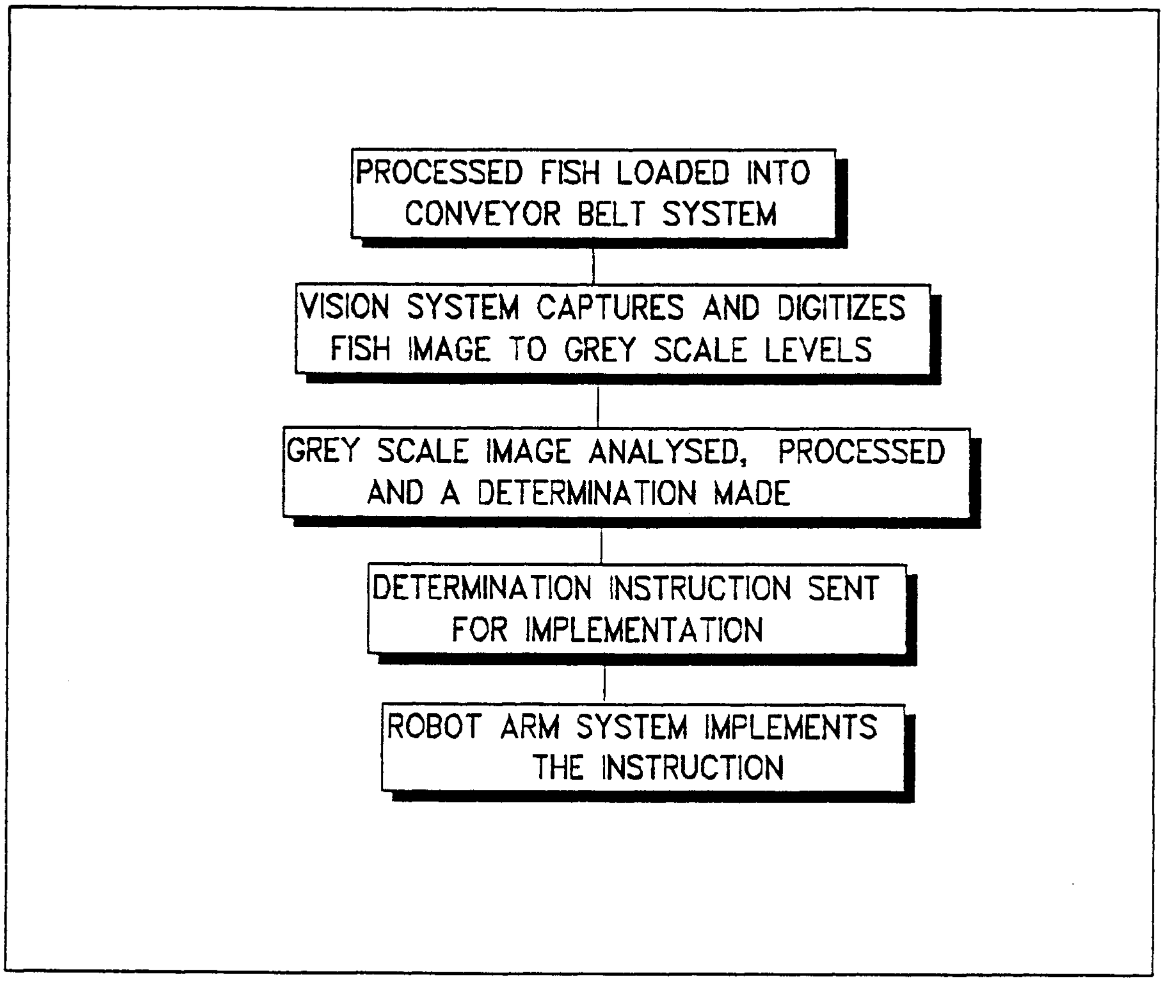


Fig.2.2 Illustration of the envisaged system.

CHAPTER 3

COMPUTER VISION AND NEURAL NETWORK

3.1 Introduction

In this chapter, the computer vision methods that are used in this investigation and analysis are introduced and briefly described. Neural Networks are also introduced where the backpropagation algorithm is described. Two of the optimization methods used for the backpropagation algorithm are described along with a brief summary of the neural network backpropagation algorithm currently used in research.

3.2 Computer Vision

Computer vision is concerned with extracting information about a scene by analysing images of that scene [1]. It combines graphics for outputting images from non-pictorial information and image processing techniques. To process an image by the computer, the image is first converted into digitized image form, a discrete array of elements representing brightness or color values, numerical-valued elements called pixels of grey scale levels. The general goal of computer vision is to recognise objects of various types that may be present in the scene. An object is an arrangement of parts whose properties (eg. grey scale levels, textures, sizes, shapes) and relations at the same time satisfy certain constraints [1]. Thus to recognise an image, there are requirements for identifying the parts that refer to the object parts and satisfying the appropriate constraints. Techniques used to detect parts in an image include segmentation, thresholding, edge detection, feature extraction, processing of grey scale levels etc.

Segmentation is employed to identify distinctive sub-populations of pixels or used to partition an image into connected regions each of which is "homogenous" or "uniform" in some sense either by color or by a statistical analysis of one or groups of pixels in the image.

Thresholding in its simplest form provides a representation for the image which requires less storage than the original [3]. Its purpose is to segment the image into regions which can subsequently be analysed based on their characteristics. It involves choosing a grey scale level T such that all grey levels greater than T are mapped unto the "object" label and all other grey levels are mapped unto the "background" label.

Edge detection is a technique used to search for edges between regions. A gradient operator is used followed by a threshold operation on the gradient in order to decide whether an edge has been found [2].

The processing of grey scale levels of an image can take the form of a grey level histogram, sometimes referred to as the cumulative relative frequency distribution of the grey scale levels in an image, which is a function showing for each grey scale level, the number of pixels in the image which has that grey level. All spatial information is discarded, therefore the histogram for any image is unique although the reverse is not the case [4].

3.2.1 Feature extraction

This technique is used to identify special types of local patterns in the image eg. step edges, lines, curves, spots, corners etc. [1], which are then extracted and analysed. A feature is usually defined by its properties and can thus be detected by comparisons, or by computations of various levels of curvature, or by extracting a string of symbols which uniquely represents the features.

3.2.2 Neighbourhood averaging

This is an image preprocessing technique for doing various smoothing operations on an image. A section or a neighbourhood of a selected pixel is averaged to a smoothed image point to represent that section of the image according to a mathematical criteria as follows [4],

$$g(x,y) = 1/M\Sigma_{(n,m)\nu S}f(n,m).$$

where

g(x,y) -> the smoothed image point

M -> the number of points in the neighbourhood.

S -> represents the co-ordinates of points in the neighbourhood.

f(n,m) -> represents the original image section.

3.3 Neural Network

Neural network studies are known by different names, neuro computing, parallel distributed processing or connectionist modelling, etc. Networks are basically formed by simulated neurons connected together much the same way that human brain neurons present in the nervous system are connected. The neurons behave as the processing or computational elements operating in parallel, and these communicate to each other via links or connections

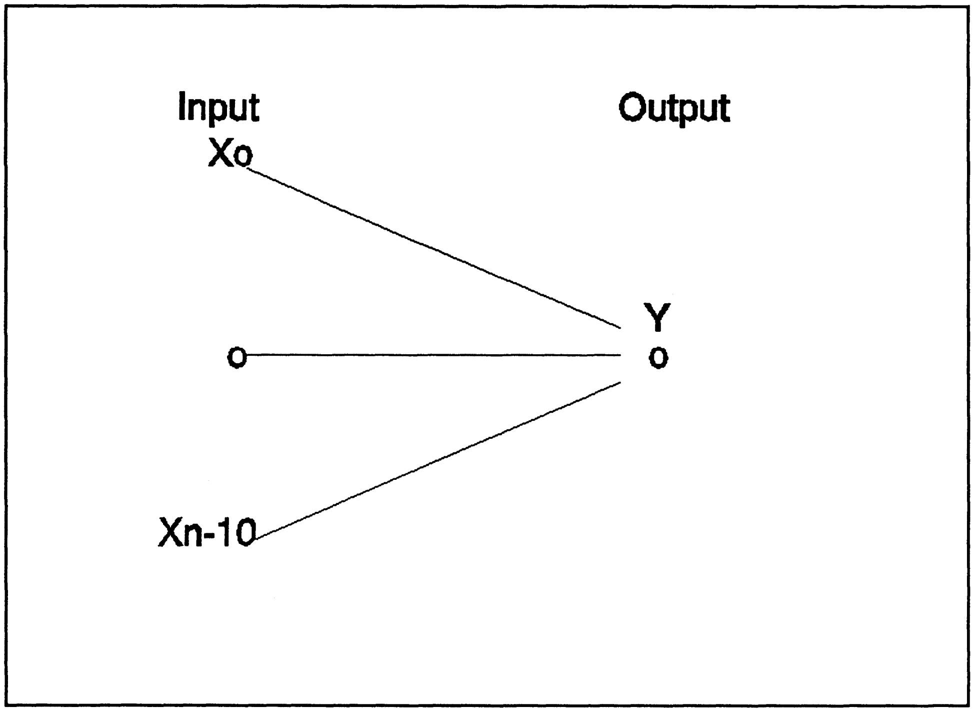


Fig. 3.1 A simple network.

with variable weights which are adjusted in the learning/teaching/training phase according to a learning rule. An activation function sums the weighted inputs which is passed through a non-linear or transfer function (usually hard limiters, threshold logic elements, and sigmoidal non-linearity functions) [5]. Input signals can be either excitatory or inhibitory, i.e., it can fire or not fire. By adapting to changes in the input, a neural network learns or is trained by accumulating experience as the connecting strengths of the elements are adjusted. Thus the computational functions are embedded in the network.

Neural networks solve problems that humans do well at, such as problems of association, evaluation and pattern recognition which are difficult to compute and do not require perfect answers. Neural networks are poor at precise calculations, predicting or recognising things that do not inherently contain some sort of pattern [9]. These networks belong to a class of classifiers that do various tasks such as identifying which class best represents a noisy input pattern.

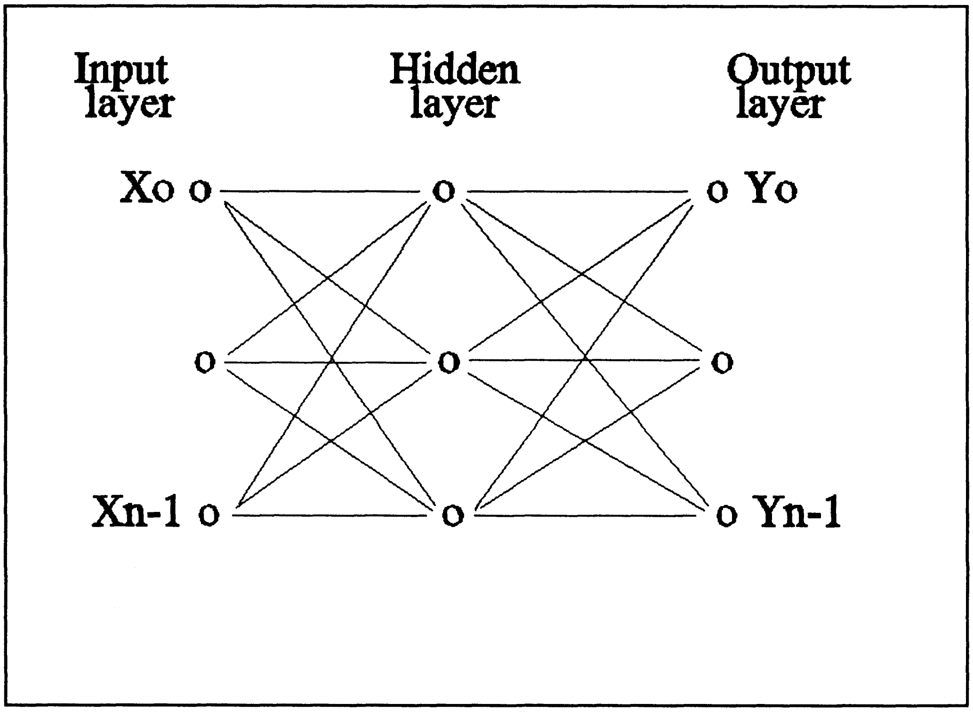


Fig. 3.2 A multi-layer network.

3.3.1 Perceptron

A single layer perceptron [6] is a feed forward linear associator which is implemented using the perceptron convergence theorem. It has the capacity of generating two decision regions but has several limitations [7]. A two layer perceptron can form any possibly unbounded convex region in the input space. A three-layer perceptron can form arbitrarily complex decision regions [5], as complex as those formed using nearest-neighbour classifiers A multi-layer perceptron overcomes many of the limitations of the single, two-layer perceptrons and with the recent advancement in the training algorithms for a multi-layer perceptron [6], it has become one of the most important tools for the neural network technique.

3.3.2 Supervised back propagation model

Most learning rules have evolved from the Hebbian rule which in its basic form states that when two neurons fire at the same time, then their connections strengths should increase,

$\Delta w_{ij} = \epsilon e_i a_j$
where
w_{ij} - *connection between two neurons*
ϵ - *the gain term*
e_i - *the error measure*
a_j - *the activation*

The delta rule [6], (also known as the Widrow-Hoff learning rule or the least mean square rule), is to adjust the strengths of the connections to reduce the difference or the error measure between the actual output and the desired output pattern during training [6]. In essence the attempt is to minimise the energy or error function.

There are two main types of training algorithms, surpervised and unsupervised, supervised since the training is by a "teacher", in that it has the knowledge of the desired response and any error between the desired response and the actual response is therefore used to correct the network. This becomes more difficult in networks of multi-layers because of the connections that have to be made to the hidden layer as depicted in Fig. 3.2. Unsupervised learning, on the other hand is the case where training is done without the use of a "teacher" or knowledge of the desired response, in order to adjust the connection strengths.

The supervised backpropagation network is a multi-layer feed forward network that uses the generalised form of the delta rule. It was popularised by Rummelhart et al [6]. It provides a mathematical explanation for the dynamics of the learning process. Also known as the "backward" error propagation algorithm(backprop), it is a generalisation of the least mean square algorithm. It uses a gradient search technique to minimise a cost function equal to the mean square difference between the desired output and the actual output. The network is trained by initially selecting small random weights and internal thresholds and then presenting all training data to the network repeatedly. Weights are adjusted after every trial using side information specifying the correct class, until weights converge and the cost function is reduced to an acceptable limit.

3.3.3 Back propagation algorithm

The algorithm [5] is described in the following steps:

Step 1: **Preprocess the input data into a format for efficient use.**

A normalisation process.

Step 2: **Initialise the weights and thresholds.**

Initialise all the connnecting weights and thresholds in the network to small random values in a specific range excluding zero.

Step 3: **Present input vector and desired output vector.**

Input vector X(m) and desired vector D(n) are presented until the weights stabilise.

where X(m) - m-dimensional vector

D(n) - n-dimensional vector

Step 4: **Calculate the actual output and test for adapting weights.**

Using the sigmoid logistic non-linearity to compute the output, O(n)

$$f(\alpha) = \frac{1}{(1 + e^{-(\alpha - \theta)})}$$
$$\textit{where } \alpha = \sum_i w_{ij} x_i$$
$$\textit{and } \theta - \textit{threshold}$$

$$O_l = f\left(\sum_k w_{kl} x_k - \theta_l \right)$$
$$\theta_l - \textit{ouput neuron threshold}$$
$$0<l<n-1$$

x_k - activation of node in the layer following the output node l

w_{kl} - connecting weight between the node following the output node l and the output node l

If the weights stabilise, then adaptation is stopped, i.e, when the error function has been reduced below an established limit:

$$E = \frac{1}{2} \sum_n (D_n - O_n)^2$$

Step 5: **Adapt the weights.**

Weights are adjusted by working back recursively from the output nodes to the first hidden layer. Adjust weights by

$w_{ij}(t+1) = w_{ij}(t) + \epsilon\delta_j x_i$
where
$w_{ij}(t)$ - *weight to hidden node i from input node j at time t*
x_i - *activation of node i*
ϵ - *gain term*
δ_j - *the error term for node j*

if node j is an output node, then the error term is

$$\delta_j = y_j(1-y_j)(d_j-y_j)$$
where
y_j - *actual output*

if node j is internal and hidden then the

error term is

$$\delta_j = x_j(1-x_j)\sum_k \delta_k w_{jk}$$

where k sums over all nodes in the layer following node j.

Step 6: **Repeat by going back to step 3.**

3.3.4 Optimization methods

Network learning is mainly accomplished by the optimisation of a criterion function E, which is a measure of the error between the target function and the approximating function. The object is to determine a set of weights which minimises this function. The methods used are based mainly on the same strategy. The minimisation is a local iterative process in which the approximation to the function in a neighbourhood of the current point in weight space, w, is minimised [12]. Among the optimisation algorithms that can be used to solve these types of problems, gradient based ones have proven to be very effective in a variety of applications [17]. Basically the weight vector, w_k at the kth iteration is updated by taking a step s_k, in the search direction d_k, as below:

```
for (k=0; evaluate(w_k) != converge; ++k)
{       d_k   = determine_search_dir();
        s_k   = determine_step();
        w_k+1 = w_k + s_k.d_k; -- update the weight vector
} [18]
```

A set of random weights, w_0, in the weight space w, are selected for the initial iteration k=0. The criterion function E(the error function) which controls the convergence of the algorithm(discussed in section 3.3.7) is evaluated using this initial set of weights in order to determine whether this set of weights satisfy the convergence criterion. If the convergence criterion is satisfied then the iteration procedure is stopped, otherwise the search direction d_k, the gradient of the criterion or the error function with respect to each weight is

$$d_k = \frac{\delta E}{\delta w_{kj}}$$

(For the detailed mathematical proof and explanation refer to [22])

computed and the step size s_k, the learning rate or the gain term is also determined. These are then used to update the weights, thus moving the weight vector in the weight space. The iteration is increased, k+1, and the criterion function evaluated using the new set of weights to find if convergence has been achieved and the procedure continued if not, till convergence is achieved.

Thus a sufficiently small step in the direction of descent d_k will therefore reduce the error function E [18].

3.3.5 Steepest descent

This is the most classical gradient based optimization algorithm [18]. From section 3.3.4 when the search direction d_k, is set to the negative of the gradient g(w) and the step size s_k is made equal to a constant ν, then the algorithm becomes the steepest descent algorithm. In the context of neural networks this is the back propagation algorithm without the momentum term [6] presented in section 3.3.3. The error function is computed and each weight is gradually moved "down" the error surface towards the local minima [6]. Through an iterative procedure, the steepest descent method minimises the error function $E(w_k)$. Unfortunately the convergence rate of the steepest descent method is only linear [18] and therefore requires a large number of iterations to converge to a reasonable limit. Inclusion of the momentum term [6] is not able to speed up the algorithm considerably, yet it introduces another user dependent parameter, which makes the algorithm less robust [12].

3.3.6 Conjugate gradient methods

Conjugate gradient optimisation systems improve the convergence rates by making use of second-order derivatives information without the expensive estimation and storage of these derivatives. From section 3.3.4, the search direction d_{k+1} in this case is set to a combination of the negative of the weight gradient, $-g(w_{k+1})$ as in section 3.3.5 and a factor of the previous search direction d_k,

$$d_{k+1} = -g_{k+1} + \beta_k . d_k$$

the factor β_k is determined by various rules [12] depending on the particular application, one of which is the Polak-Ribiere rule [18] in which β_k is determined from g_{k+1} and g_k according to

$$b_k = (g_{k+1} - g_k)^T.g_{k+1}/g_k^T.g_k.$$

If matrix operations are involved then the gradient vector g, is a column vector, g^T its transpose, becomes a row vector. The step size is determined by the normal linesearch method [18] which is iterative and involves the evaluation of the error function for some values of the step size to determine which value minimises the function best. In practise, a typical linesearch in a network problem terminates in two or three iterations [18]. By keeping a copy of the next best step size, a reinitialisation procedure can be restarted [23] if convergence is not achieved after a number of iterations. Detailed proof and explanations can be found in [11, 17, 18, 23].

3.3.7 Convergence criterion

Choice of the convergence criterion is rather important since it controls the limit to which the iteration should proceed in order to establish a "good" to "perfect" training, i.e., training must be terminated when the error function has been sufficiently minimised. Several criteria used include an error threshold, an iteration limit or even using both.

An error threshold is set by making the error function attain a threshold limit μ,

$$E() = \mu$$

after which convergence is declared. In the case where the error surface contains some "bad" local minimum, it is possible the error threshold will be unattainable [18]. To guarantee termination of the algorithm, despite an unattainable error threshold, an iteration limit can be used, by which a limit or bound on the number of iterations is set at which the algorithm is curtailed. For practical problems, this works best where the limit is known from previous experiments but works poorly if the limit is not known. Using both criteria is another option.

A necessary condition for the weights to be at their minima, either local or global is that the weight gradient, explained in section 3.3.6, be equal to zero.

$$g(w) = \frac{\delta E}{\delta w} = 0$$

Thus the convergence criterion for the optimisation algorithm can be set as

$$\|g(w_k)\| \leq \epsilon$$
where ϵ is a sufficiently small gradient threshold

The downside to using this as a convergence test is that, for successfull trials, learning will be longer than they would be in the case of an error threshold [18].

Pao [22] shows that under a situation where the algorithm gets trapped in some local minimum or at some stationary point or perhaps oscillates between such points, the error remains large regardless of the number of iterations carried out. The criterion thus selected to a large extent depends on its practicability in relation to the specific problem being analysed or solved.

3.3.8 Other models

Back propagation implementation algorithms are very popular and several modifications are showing up in research literature recently, all aimed at reducing some of the problems associated with the original back propagation model. A few are considered here briefly, such as the cascade correlation archictecture [13], the quick prop [14], and the conjugate gradient models.

The original back propagation learning algorithm that uses the steepest descent method popularised by Rummelhart [6] is rather time consuming due to problems involving the step size and the moving target problems [13]. The introduction of a momentum term to reduce this problem increases the number of the user-dependent parameters on which the convergence of the network depends.

Cascade correlation [13] is well suited to choosing or selecting the basic network topology to use as a starting point model for an investigation but is lacking especially when large networks are involved. The algorithm works best with problems involving small to non-complex networks. It performs unpredictably, even complete failure when used for large and complex practical networks. Since the algorithm considers a node at a time while freezing the other nodes, their influences are also frozen. In a small network, the sum of these influences may have a negligible effect on the convergence of the network, but in a bigger network this sum is very prominent in all aspects of the network scheme and cannot therefore be neglected.

The quick prop algorithm [14] is among some of the fastest algorithms for network training purposes since it uses the second order derivatives to plot a parabola and descends directly to the minimum on that curve. The estimation and storage of these tend to be very expensive, and when the network is large and complex, this is compounded further.

Models that use the conjugate gradient systems tend to be robust and fast at the same time. Like the quick prop algorithm, they make use of the second order derivative information without the estimation and storage of the second order derivatives [18] and seem to be the best models available at present.

CHAPTER 4

PARAMETER SELECTION AND DATA REPRESENTATION

4.1 Introduction

In this chapter the techniques for selecting the neural network parameters are described and an in depth look at the format in which the data is presented to the network including the processed format and the unprocessed format. It also describes the cumulative frequency distribution and the two algorithms developed to generate these distributions in a fixed format for efficient use by the neural network.

4.2 Learning parameters

In the back propagation algorithm, two parameters are made use of, the gain term and the momentum term [6] which control the rate of learning and the approach to convergence in the training phase respectively. These terms are user specified and because their selection have no formal theoretical basis, empirical methods are used for their determination. A correlation method due to Apey [10] is employed in a slightly modified form to select and adjust these parameters in the back propagation with the steepest descent optimization technique.

Essentially the parameters are set and adjusted according to a measure of the changes in the connecting weights of the neurons:

$$c_j(t) = \sum_{i=0}^{N} f\,(\,\Delta w_{ij}(t),\ \Delta w_{ij}(t-1)\,)$$

where $c_j(t)$ - correlation factor which measures the changes in the connecting weights

f - function of the connecting weights

The algorithm in its simplest form [10] is as follows:

if $c_j(t) < 0$ *then*
$\epsilon \rightarrow 0.01$
$\alpha \rightarrow 0.00$
else
if($c_j(t) < (c_j(t-1) + 0.05 * c_j(t-1)\,)$ *or*
$(\epsilon < \max \epsilon)$ *then*
$\epsilon \rightarrow \epsilon + \epsilon$ *rate*
$\alpha \rightarrow \alpha + 0.01$
endif
endif
where ϵ - *gain term*
α - *the momentum term*
maxϵ - max *gain term*
ϵ *rate* - *gain term rate*

Thus if there is no learning then the parameters are drastically reduced, but if there is learning then the parameters are increased for more learning to continue.

4.3 Initial weights

The second step in the back propagation algorithm(section 3.3.3) is the choice of the initial weights. It is often good practise to use initial weights that are of very small orders to minimise the mathematical problems encountered during computation, and also to place the initial decision boundaries within reasonable limits to reduce the number of iterations to achieve convergence. This therefore makes the selection of the initial weights very important.

An empirical method was used to compare two different methods. Random weights were selected and

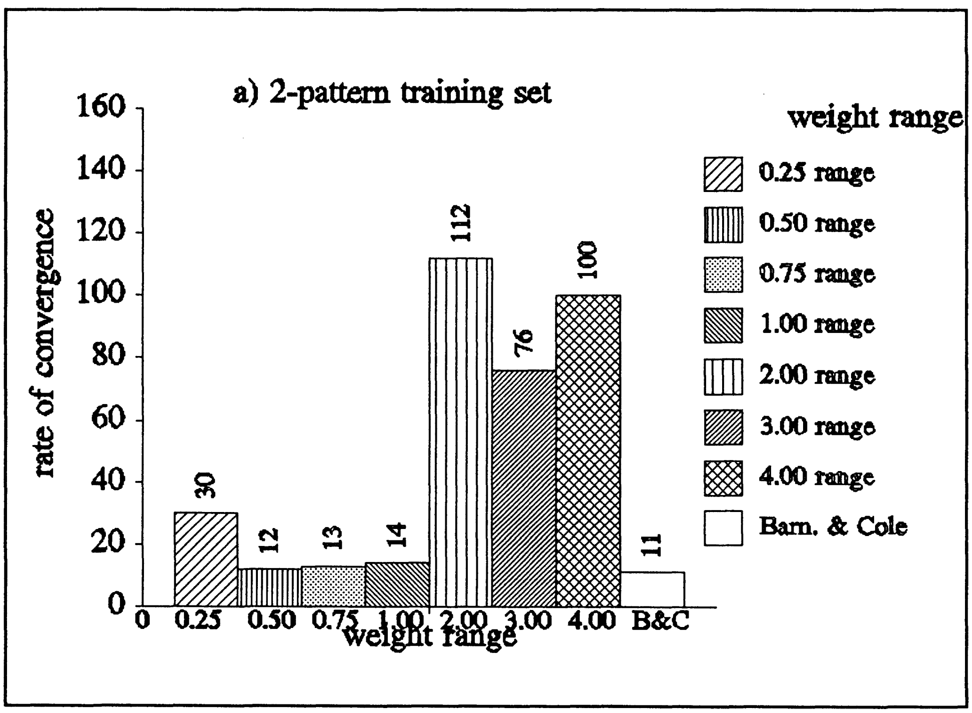
1. Constrained to a selected range and,

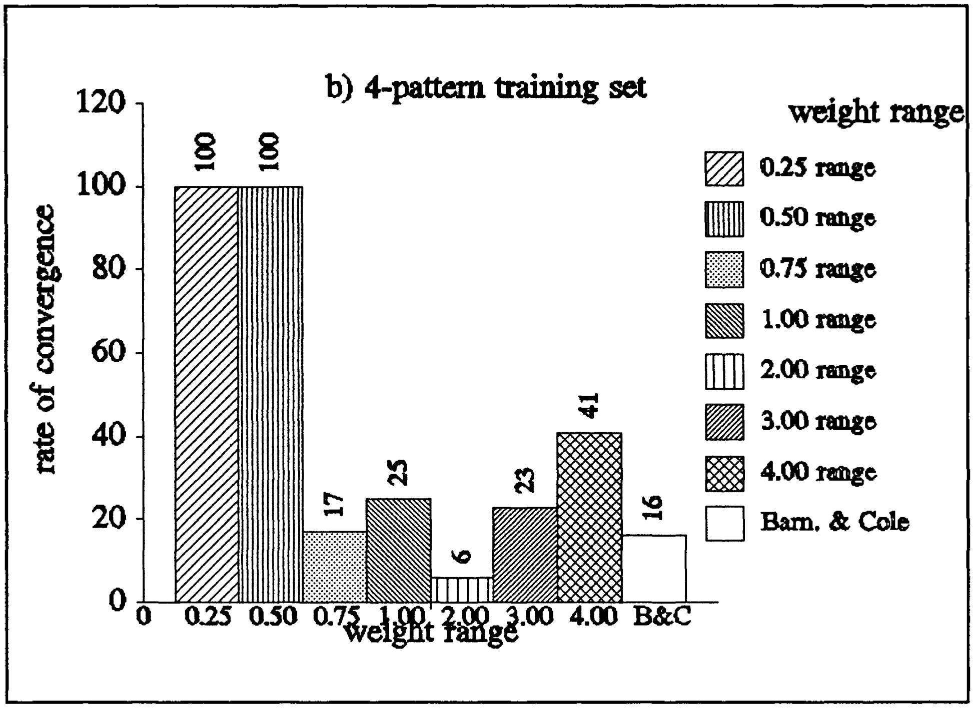
2. A method employed by Barnard and Cole [11] - in which the weights to a node are scaled to be uniformly distributed between

$$\pm\ 3\ \sqrt{n}$$

where n - the number of connections to that node

thus ensuring that the sum of the weights leading to a node is a random variable with approximately zero mean and variance of 3. The training set consisted of 10 patterns(5 parasitic and 5 non-parasitic) from the feature extraction input data (samples presented in the appendix) with a topology of 32 input neurons, 65 hidden neurons and 2 output neurons.





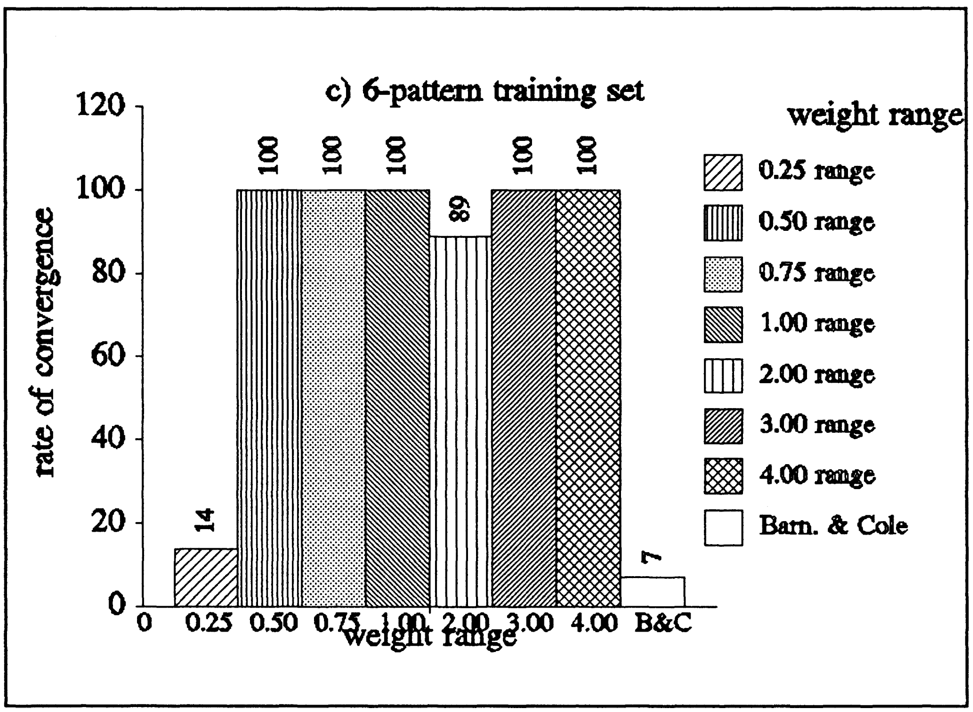
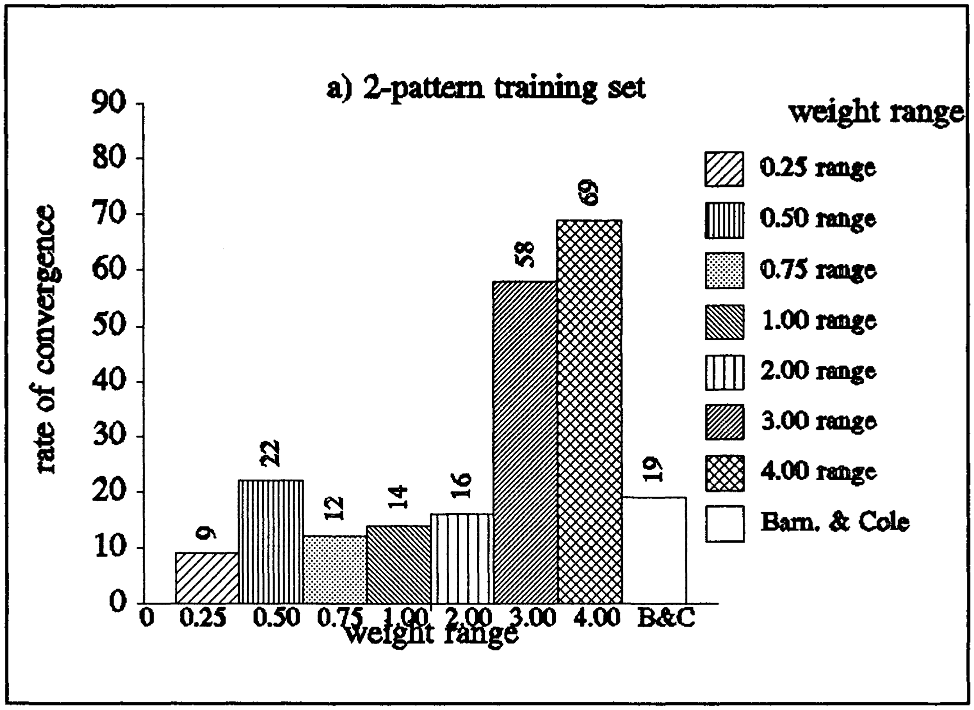
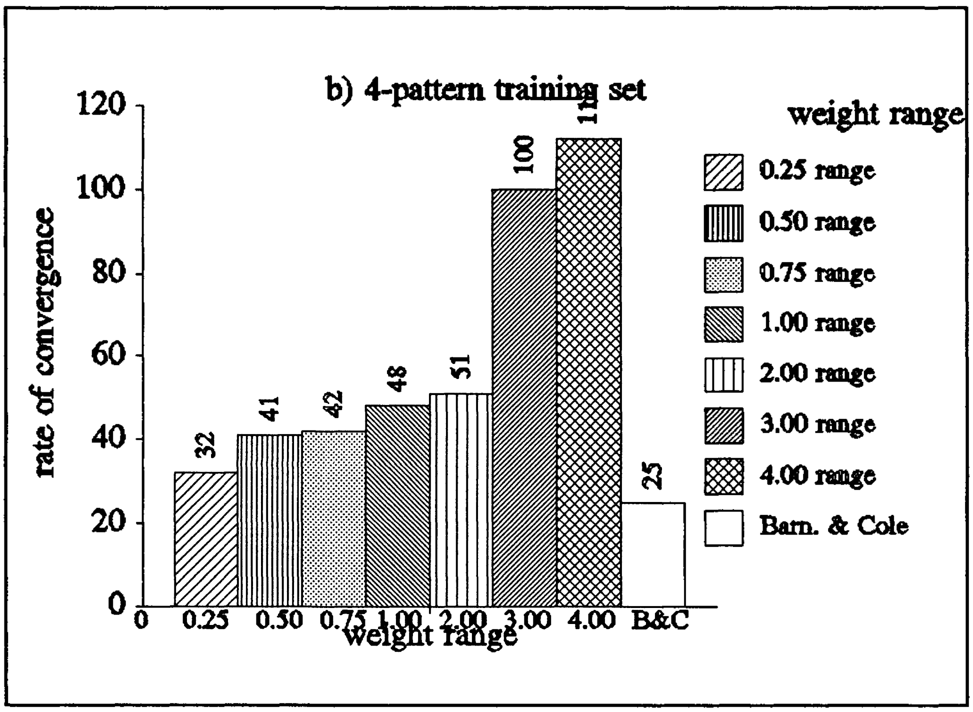
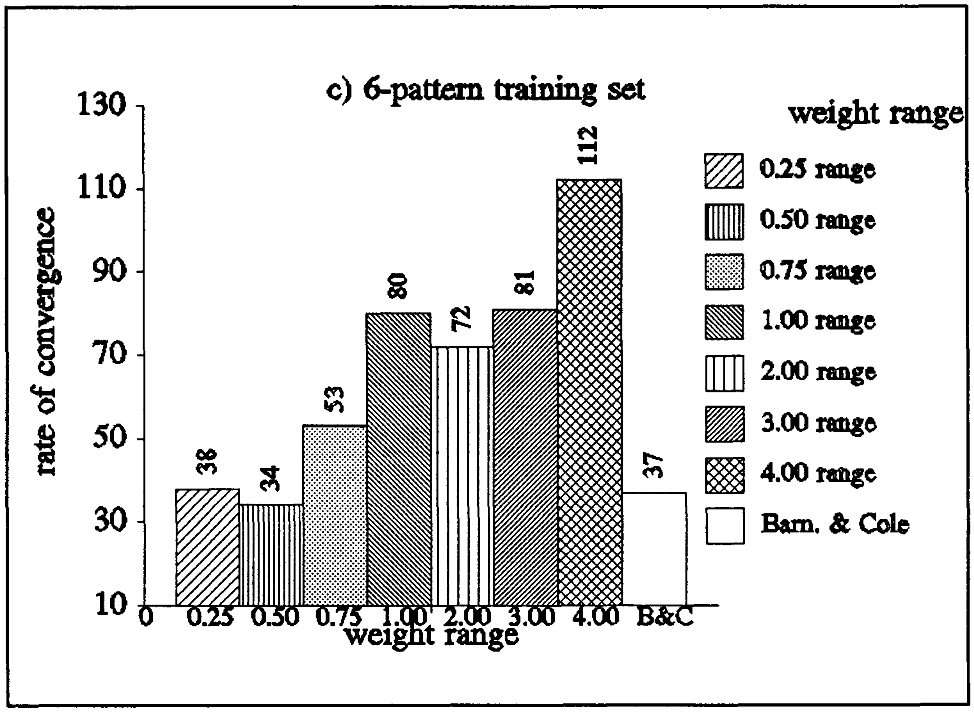
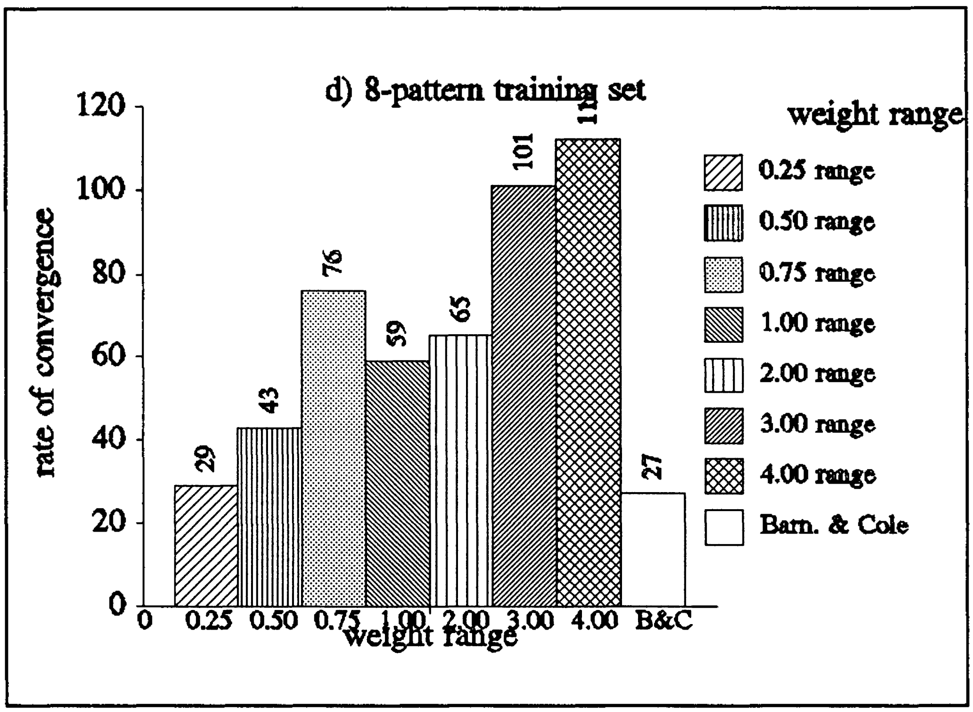


Fig. 4.1a,b,c Plots of rate of convergence against ranges of initial weights for the steepest descent method.


a) 2-pattern training set
rate of convergence
weight range
0 0.25 0.50 0.75 1.00 2.00 3.00 4.00 B&C
9
22
12
14
16
58
69
19
weight range
0.25 range
0.50 range
0.75 range
1.00 range
2.00 range
3.00 range
4.00 range
Barn. & Cole


b) 4-pattern training set
rate of convergence
weight range
0 0.25 0.50 0.75 1.00 2.00 3.00 4.00 B&C
32
41
42
48
51
100
25
weight range
0.25 range
0.50 range
0.75 range
1.00 range
2.00 range
3.00 range
4.00 range
Barn. & Cole


c) 6-pattern training set
rate of convergence
130
110
90
70
50
30
10
0
0.25
0.50
0.75
1.00
2.00
3.00
4.00
B&C
weight range
38
34
53
80
72
81
112
37
weight range
0.25 range
0.50 range
0.75 range
1.00 range
2.00 range
3.00 range
4.00 range
Barn. & Cole


d) 8-pattern training set
rate of convergence
120
100
80
60
40
20
0
0
0.25
0.50
0.75
1.00
2.00
3.00
4.00
B&C
weight range
29
43
76
59
65
101
27
weight range
0.25 range
0.50 range
0.75 range
1.00 range
2.00 range
3.00 range
4.00 range
Barn. & Cole

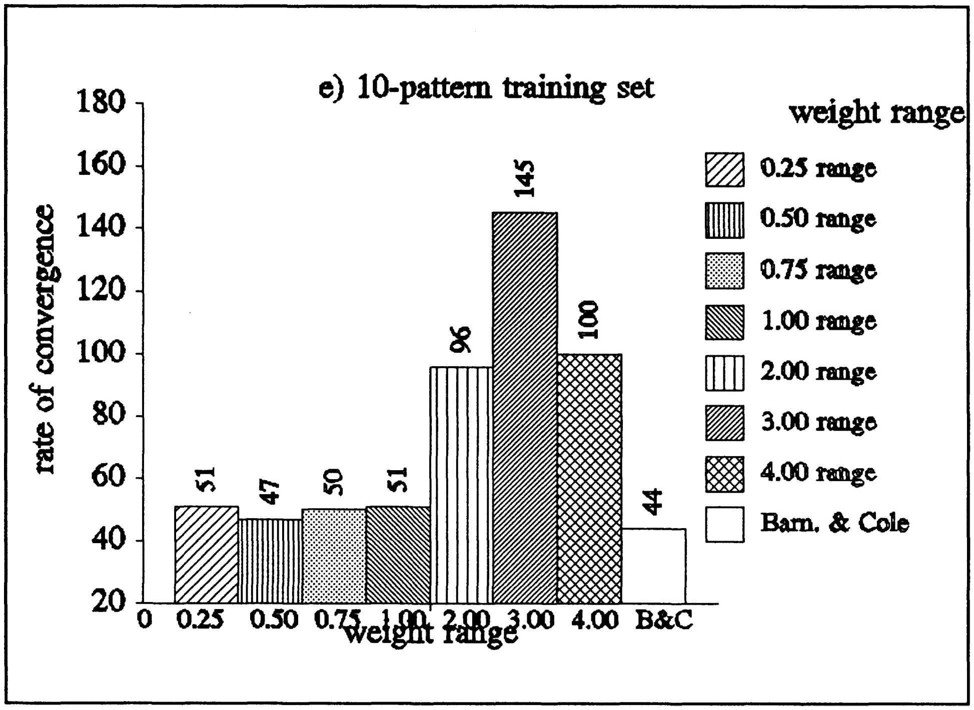


Fig. 4.2a,b,c,d,e,f,g Plots of rate of convergence against ranges of initial weights for the conjugate gradient method.

The rates of convergence were measured by the number of iterations for convergence to be achieved for an error threshold of 0.001 for the steepest descent algorithm and 0.00000001 for the conjugate gradient algorithm.

From the plots in Fig. 4.1 and Fig. 4.2, the method employed by Barnard and Cole [11] seems to offer a fairly consistent, and better rate of convergence although oscillations do occur in a few cases and therefore its use was carefully monitored.

4.4 Architecture

The aim is to select a neural network architecture or topology such that

-the total number of training patterns to solve the problem eventually is low

-a minimum amount of training time is needed to solve the problem

-the mathematical computations and storage requirements are minimal

-a minimum number of iterations is needed for convergence

-convergence is eventually achieved

-the level of classification after convergence is what is desired.

The architecture or the topology is comprised of the output, the hidden and the input layers. In this project a three-layer network with one hidden layer was used.

4.4.1 Output layer

This is the layer from which the outcome of the processing undertaken by the network is passed to the outside. Since the object is to recognise, detect and locate parasites/sealworms on the cod fish images, this layer is made up of two units which indicate:

1) Fish background with parasite and

2) Fish background without parasite.

4.4.2 Hidden layer

This is the layer which neither receives inputs directly nor are given direct feedback but is the layer within which new features and internal representations [6] can be created in the network in order to approach the decision region accurately. The selection of the number of units for this layer is critical to the desired performance of the network. Again no formal methods exist for this selection and thus empirical methods are relied upon, wherein experimentation and judgement are used.

The training set consisted of 7 patterns (4 parasitic and 3 non-parasitic) and a testing set of 31 patterns selected from the 4 COD sections Fig. 1.1. The conjugate gradient algorithm was used with an error threshold of 0.00000001 for the convergence criterion. The % of incorrect classification was measured on the testing set and this was used in combination with the rate of convergence to select the number of hidden nodes that gave the best results.

For the feature extraction scheme(section 3.2.4 and 4.5.3) with an input vector of 32x1, the Kolmogorov's number [15] was used as a starting point around which to experiment, with a slightly modified form of the Kolmogorov's rule [15], as below:

1. HN = IN + 1 --> 33 units

2. HN = 2IN + 1 --> 65 units --> Kolmogorov's number

3. HN = 3IN + 1 --> 97 units

where HN - units in the hidden layer and

IN - units in the input layer

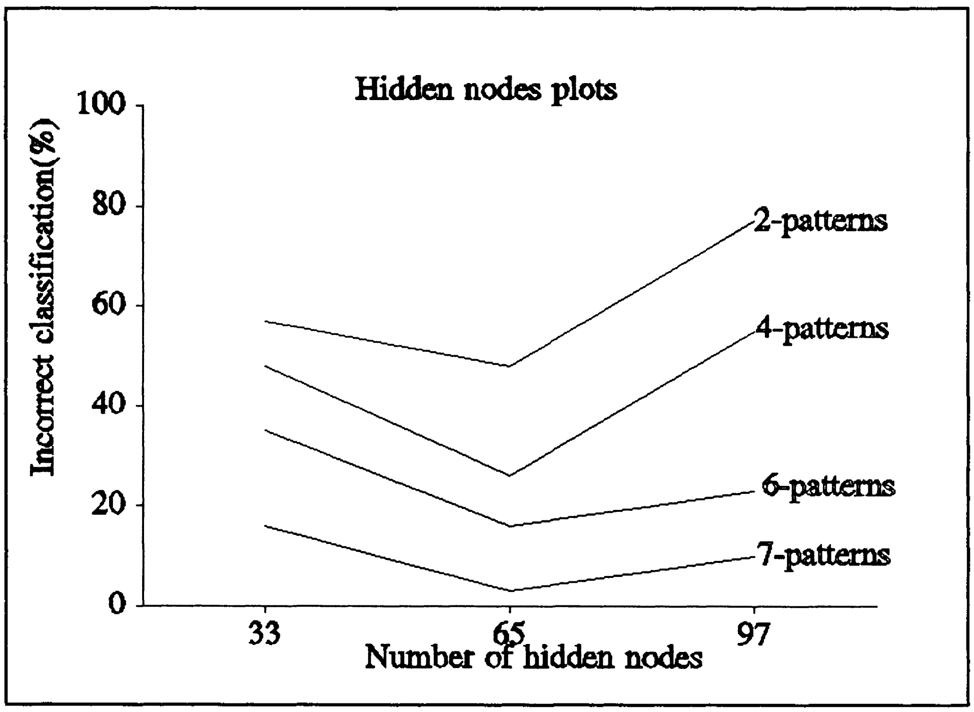


Fig.4.3 Plots of incorrect classification(%) against the number of hidden nodes.

From the graph plots in Fig. 4.3 representing incorrect classification versus the number of patterns, the Kolmogorov's number seems to give the best results of the three, in terms of the number of hidden units, i.e, the lowest percentage of incorrect classifications.

For the pixel representation(section 4.5.2), and the curve map/binarised curve map(section 4.5.2) with input vectors of 32x32 and 32x11 respectively, both of which is in excess of 60 input nodes, the average value number formula, as suggested in [16] was used as the starting point for experimentation,

HN = (IN + OU)/2 --> 513 units

OU - units in the output layer

The best results were achieved with a minimum average value of

HN = (IN + OU)/32 --> 32 units

and for the curve map/binarised curve map with an input vector of 32x11, the best results were achieved at a minimum average value of

HN = (IN + OU)/11 --> 64.

4.4.3 Input layer

This is one of the most important layers of the network topology since it is through this layer that the training pattern is presented directly to the network.

From step 5(section 3.3.3) of the back propagation algorithm, the adaptation rule is

$$\Delta w_{ij} = \varepsilon \delta_i x_j$$

which means that the weight update is directly proportional to the input signal or feature. To maximise the effectiveness of the network, it is therefore prudent to scale the inputs for the following reasons:

-the input features must be in a numerical format for mathematical processing.

-to reduce local minimum problems which the gradient search methods are used to locate.

-to constrain the input data to be "centred about the origin".

-to reduce big weight updates which cause weights to move to positions where they cannot be changed or updated further.

Two scaling methods were considered in this project, the bipolar(-1,+1) and the binary(0,+1) scaling of the data. Bipolar data scaling is popular for the simple reason that no

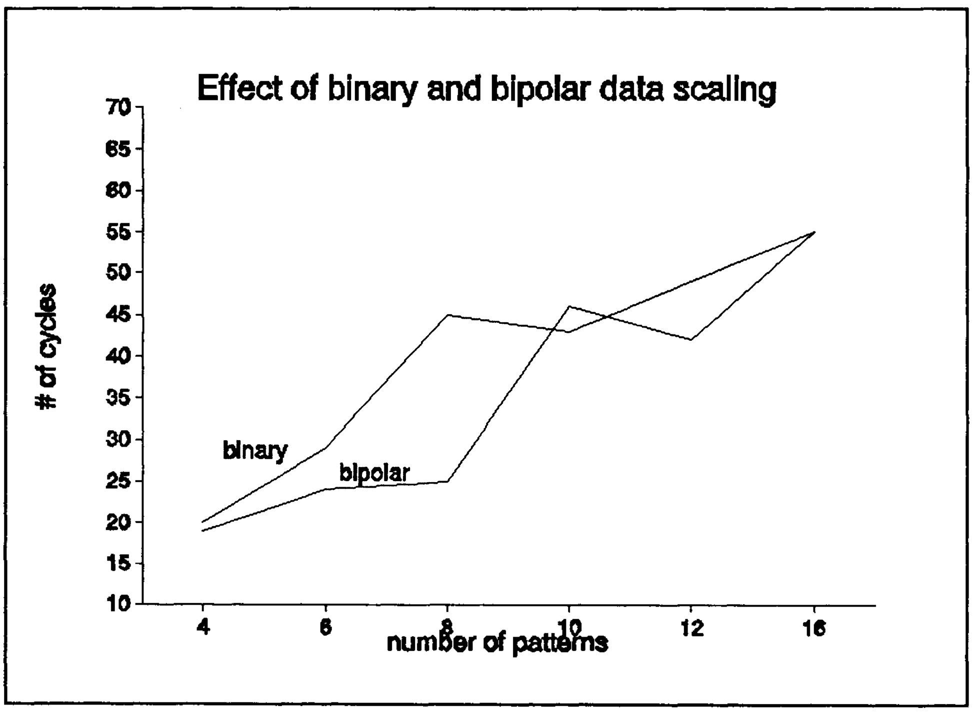


Fig.4.4a Plots of # of cycles versus the number of patterns.

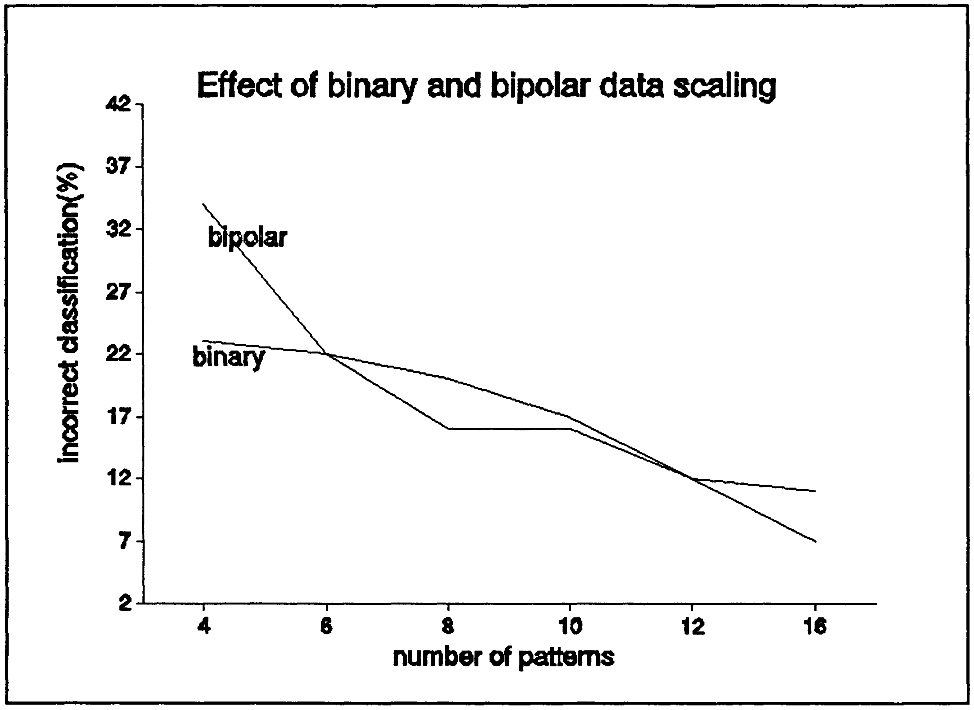


Fig.4.4b Plots of incorrect classification(%) versus the number of patterns.

learning is achieved when the input signal or feature is zero as in the binary case since the weight change is directly proportional to the input signal as explained above.

An empirical experiment to consider which of these two performs better, was undertaken. Working with a training set of 10 patterns and starting with 4 patterns and an artificially created testing set of 113 patterns and using the conjugate gradient algorithm, a plot of the percentage of incorrect classifications and the number of cycles i.e., the number of iterations when convergence of the network is achieved, against the number of training patterns gave the plots in Fig. 4.4a and Fig. 4.4b, which rather indicate very slight improvements using bipolar over binary scaling.

4.5 Data Representation

This is the format in which the data is presented to the network through the input layer either directly or indirectly, directly implies an unprocessed format and indirectly, a processed format.

4.5.1 Diagram of the data representation

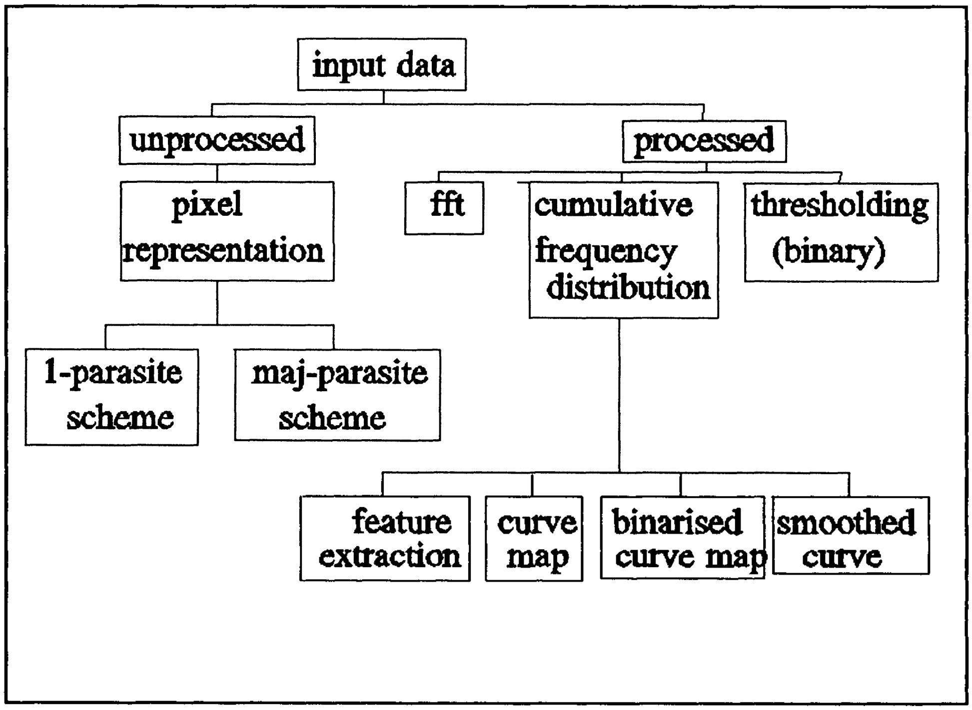


Fig.4.5 Flowchart of the data representation.

4.5.2 Unprocessed format

In this format the data is presented directly to the input layer of the network, in the raw form, i.e., grey scale levels. Pixel representation is by a pixel numerical value (0-255) which represents the colour as defined by the grey scale level of a point in the image matrix.

An image consisting of 512x512 matrix of pixel points is therefore presented as a 512x512 input vector to the input layer of the network, and similarly for 256x256 and so on.

Two schemes are used under the pixel representation data format; the one-parasite scheme and the majority-parasite scheme.

The one-parasite scheme is the scheme in which one network is trained to recognise one particular parasite on a fish image. A database of these networks is then set up in a queue and during the on-line implementation or the production phase of the neural network,

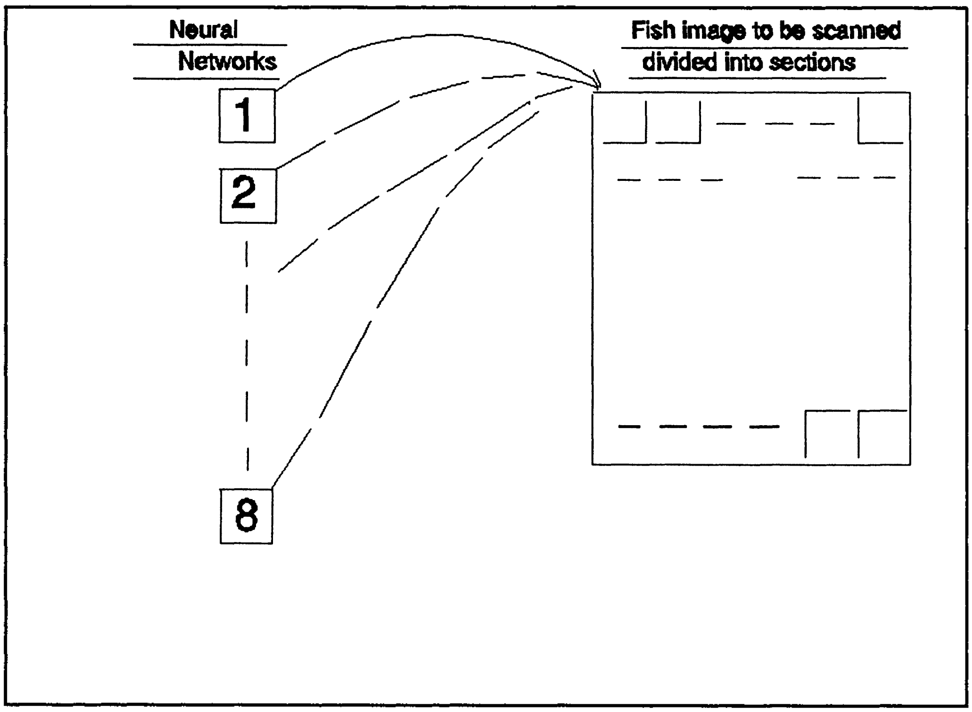


Fig.4.6 Illustration of the one-parasite scheme.

a controller releases these networks one at a time, one after the other, to scan the fish image respectively for parasite recognition, as illustrated in Fig. 4.6. Since the objective is to detect and locate parasites in the fish image, the controller is informed if one is detected, otherwise the release of the networks is continued till the queue becomes empty.

In the majority parasite scheme, one network is trained to recognise a number of parasites at the same time. The training starts with one parasite pattern. Next the number of training patterns is increased with the unrecognised patterns, after each training in order to reduce the percentage of unsuccessful recognition. The aim is to reach a level where the neural network can generalise to an extent that it will recognise most parasite patterns that it comes across, using only one network.

4.5.3 Processed format

In this format, the data is presented indirectly to the network. Before presentation, the data is pre-processed either in a one step process or a two step process. The three kinds of preprocessing undertaken included, thresholding(binarisation), fast fourier transforms and cumulative frequency distribution using the histogram translation and scaling algorithm.

Four schemes are used under the cumulative frequency distribution data format, described in section 4.7; the feature extraction, curve map, binarised curve map, and the smoothed curve formats.

In the feature extraction scheme, the coordinates of the distribution curves are extracted as features and the network is trained to differentiate between these as applied to fish backgrounds with parasites and without parasites. Fig. 4.7a shows a typical distribution curve and the features extracted. The norm. cum. freq. axis represents the normalised cumulative relative frequencies of the grey scales in the image and the linear spaces axis represents the scaled grey scale levels of 0-255 of the pixels to 0-31. This is done with the histogram translation and scaling algorithm(section 4.7.1).

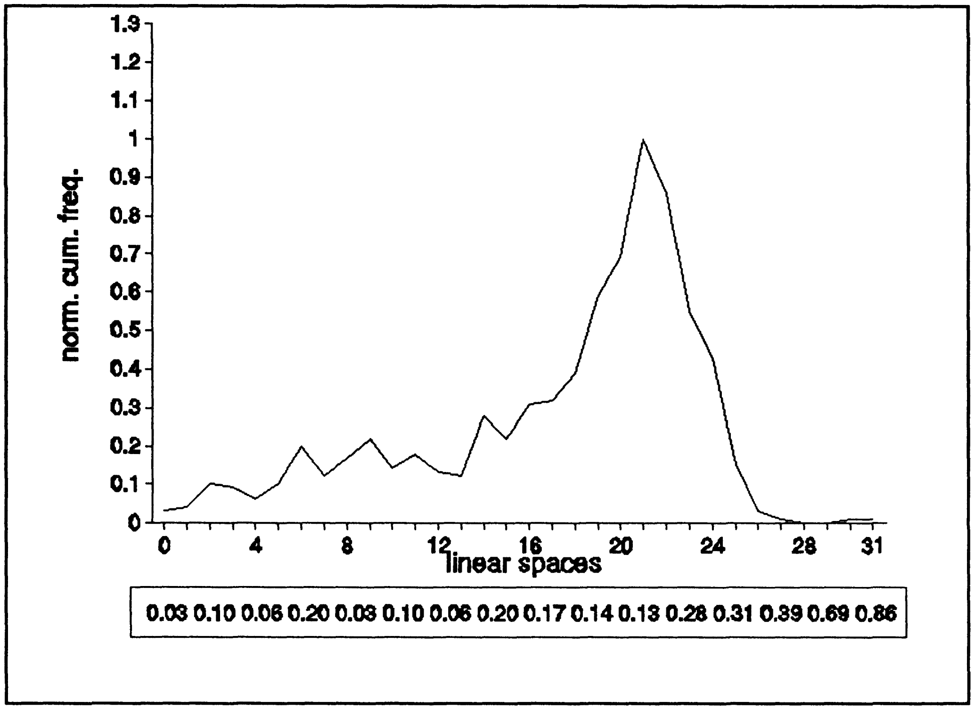


Fig.4.7a A typical distribution curve and the features extracted.

In the curve map, the distribution curve is presented to the network in a matrix map form, a typical curve map is illustrated in Fig. 4.7b, and again the network is trained to

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.59	0.00	0.55	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.22	0.18	0.00	0.22	0.00	0.00	0.00	0.00	0.15	0.00	0.00	0.00	0.00
0.00	0.09	0.16	0.12	0.00	0.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01

Fig.4.7b A typical curve map.

differentiate between these typical patterns as described above.

The binarised curve map is a curve map in which the curve points in the matrix are replaced with ones(1's) and the neural network is trained with that map as in the curve map.

In the smoothed curve, a second preprocessing is done using the smoothing algorithm(section 4.7.2) to smooth the distribution curve before the features are extracted as in the feature extraction scheme and then presented to the network.

4.6 Choice of patterns

Experimenting with numerals and alphabets, generated artificially, it was observed that the neural network tends to learn in relation to position. For instance, when the neural network is trained to recognise the alphabet A as in Fig. 4.8a, recognition is achieved in Fig. 4.8a through to Fig. 4.8e, but recognition is partially or not achieved at all when the alphabet is translated in both the lateral and longitudinal directions from its central position, as in Fig. 4.8f through to Fig. 4.8i. This particular observation occurred with the numerals also.

This knowledge, coupled with scientific judgement is employed and made use of in the selection of the parasitic patterns for the training of the network, i.e., for a typical parasitic pattern in a window section, the training set would include 5 different patterns of the parasite, although not in all cases;

1 - the parasite in its actual position.

2 - the parasite translated laterally to the left side of the window,

3 - the parasite translated laterally to the right side of the window,

4 - the parasite translated longitudinally to the upper side of the window,

5 - the parasite translated longitudinally to the lower side of the window,

Two advantages emerge from this action: first, recognition is achieved irrespective of the position by which the section falls inside the window being scanned during checking by the network and secondly, this means that the scanning of the whole image can be done 32 pixels

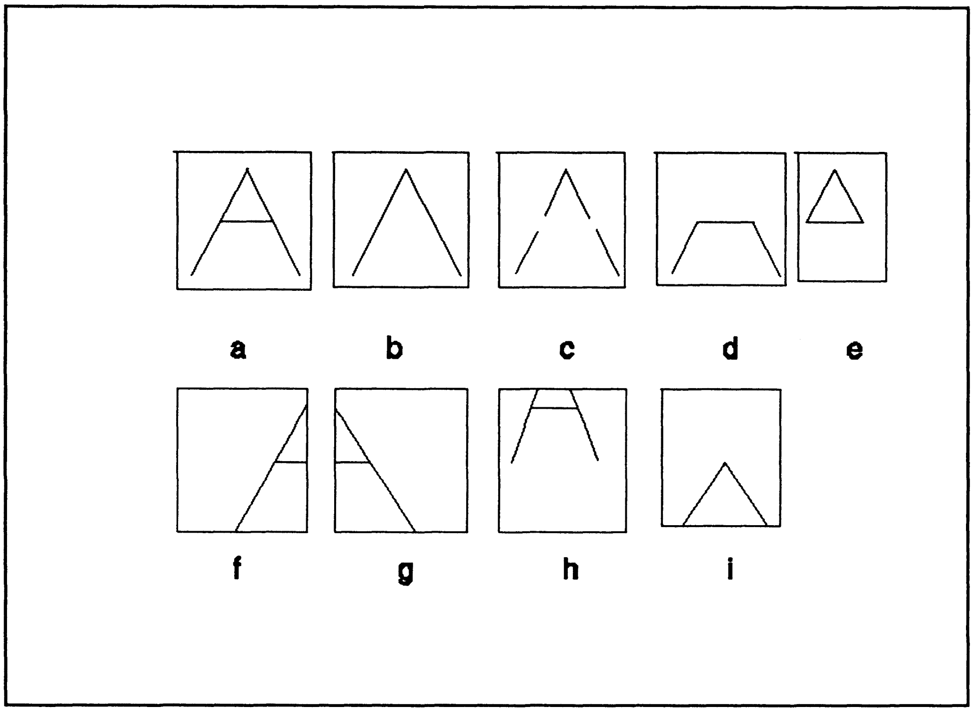


Fig.4.8 Illustrations of the alphabet A.

at a time instead of moving a pixel at a time in order to ensure all positions have been fully analyzed during on-line production use. The effect is that there is a tremendous saving in time. The expensive time consuming factor is actually transferred indirectly to the off-line training phase.

4.7 Cumulative frequency distributions

Through visual inspection it was observed that most of the parasites/sealworms that were encountered in the available fish images that were used in this project seem to fit in both shape and size into 32x32 pixel window sections of the reduced 256x256 cod fish images. Plots of the cumulative frequency distribution of the grey scale levels in these 32x32 window sections of the fish image, revealed two approximating, basic distribution formats.

1. A fish muscle background with a parasite in a section shows a bimodal curve.

2. A fish muscle background without a parasite shows a unimodal distribution curve.

Illustrations of cumulative frequency distribution plots for sections taken out of the four COD FISH images used are presented in Fig. 4.9(more are given in the appendix I) which clearly show the formats mentioned above. Some slight differences do occur, such as when the parasite is deeply embedded, or when a muscle tissue follows the shape and size of a parasite.

From section 4.6, it has been observed that a neural network learns in relation to position. This means that in order to make use of these unimodal and bimodal distribution curves to differentiate between fish background with parasites and fish background without parasite, the neural network is trained to recognise these unimodal and bimodal curves. Next the relative positions of these curves must be fixed. Although the curves in Fig. 4.9 show the unimodal and bimodal shapes, their positions differ to a very large extent both laterally and longitudinally, although the two basic approximating formats are still prominent. For example, although Fig. 4.9d and Fig. 4.9f both show a unimodal shape, their positions in the y and x axis are on different sections of both axes. In order to fix the positions to be relatively the same without changing the shapes of the curves, a histogram translation and scaling algorithm was developed and used for this purpose and is described in the next section.

The algorithm basically scales the spread of the grey scale levels of the curve and then translates it into 32 linear spaces starting from the origin, and also scales the cumulative frequency of occurrence of the grey scales to between 0 and 1. Thus if the distribution spread is greater than 32, then it is REDUCED to fit into 32 linear spaces. If it is less than

32, then it is ENLARGED and mapped into the 32 linear spaces. If it is exactly 32, then it is maintained and mapped into the 32 linear spaces.

4.7.1 Histogram translation and scaling algorithm

Step 1: Compute the cumulative relative frequency of occurrence of each grey scale level in the 32x32 window section of the image.

Step 2: Find the spread of the cumulative relative frequency distribution by calculating the difference between the beginning and the end of the distribution as below,

a) Get the first non-zero grey scale pixel with a non-zero cumulative relative frequency, Sbeg

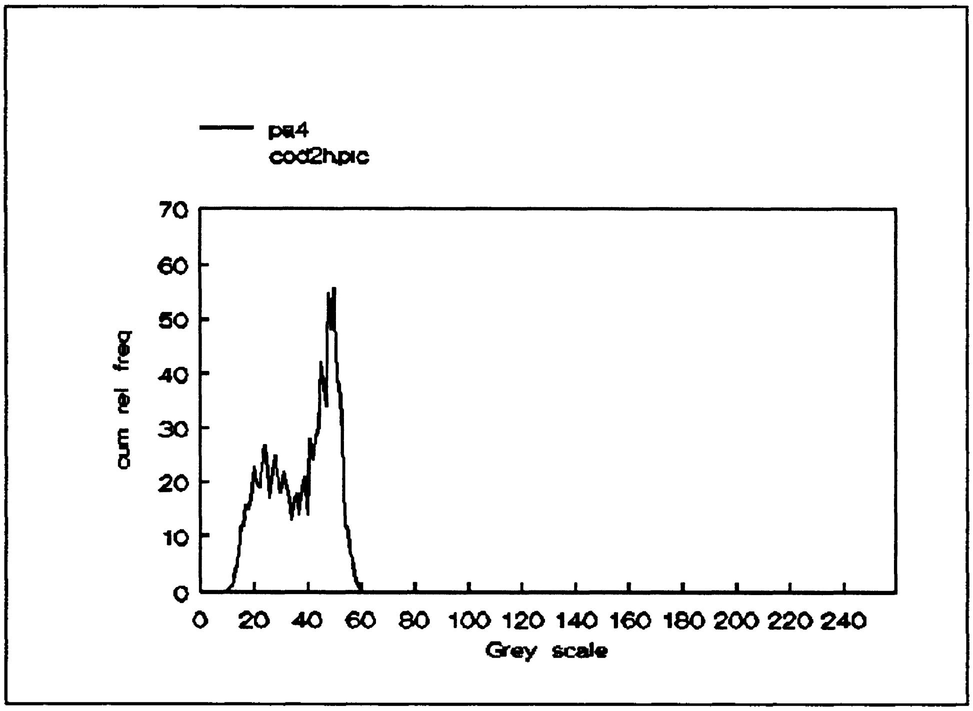
b) Get the last non-zero grey scale pixel with a non-zero cumulative relative frequency, Slast

c) Compute the spread, S = Slast - Sbeg.

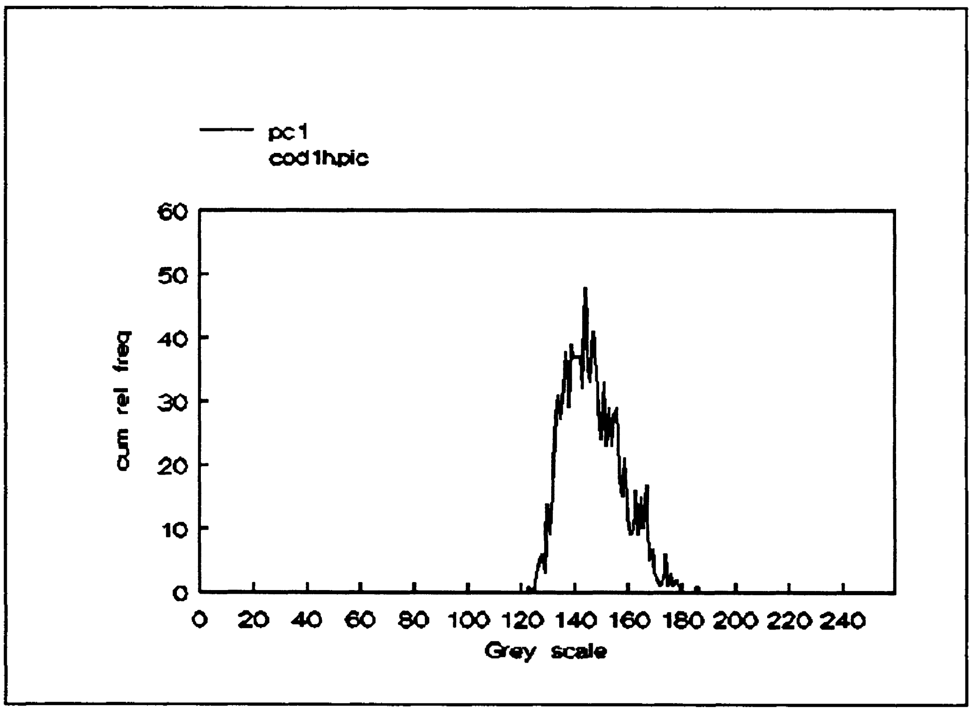
Step 3: Map the spread into a 32 linear space or cells using the area under the curve for each particular space or cell as below.

a) **REDUCTION** - If the spread of the cumulative distribution is greater than 32 grey scales, then a reduction procedure is undertaken as follows:

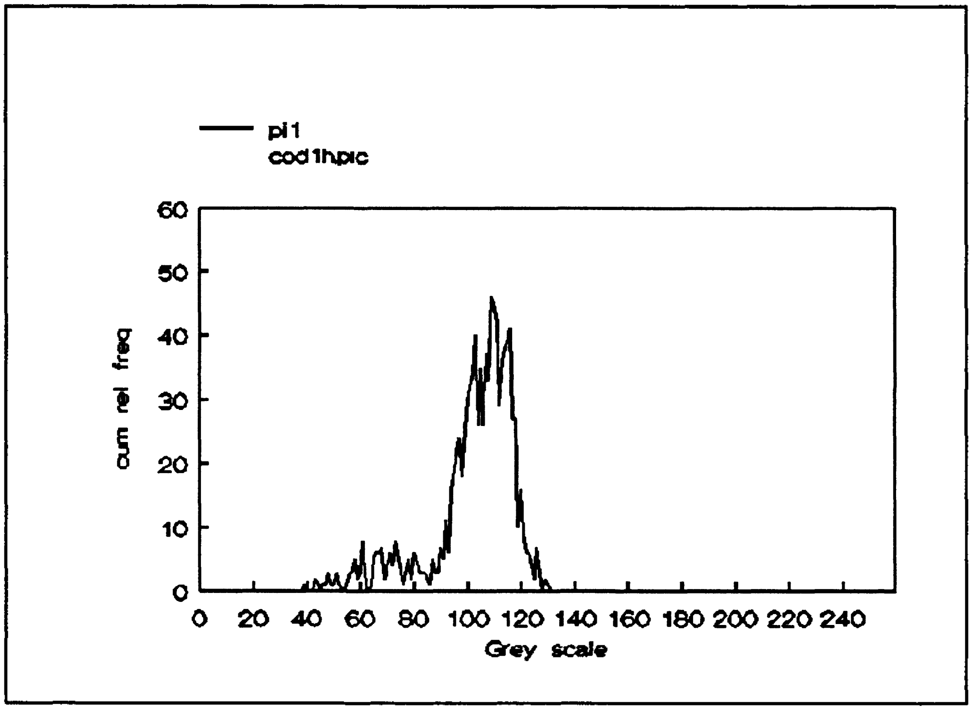
First divide the spread into 32 linear spaces or cells. For each space or cell, starting with the first cell, using a statistical measure, such as the mean, mode etc., (the mean was used in this project), then assign the mean of the cumulative frequencies of the grey scales that fall inside this cell as the new cumulative frequency of the space/cell. For example, for a spread of 62 grey scales, divided into 32 cells will yield 2 grey scales per cell. If therefore the first two



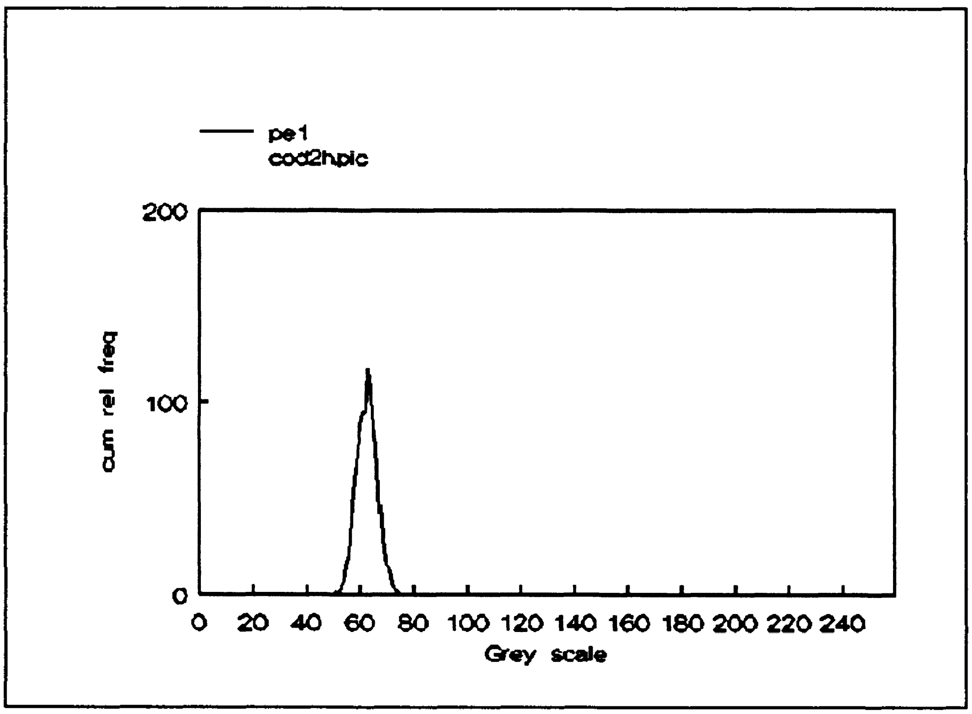
a) Section with a curvy worm.



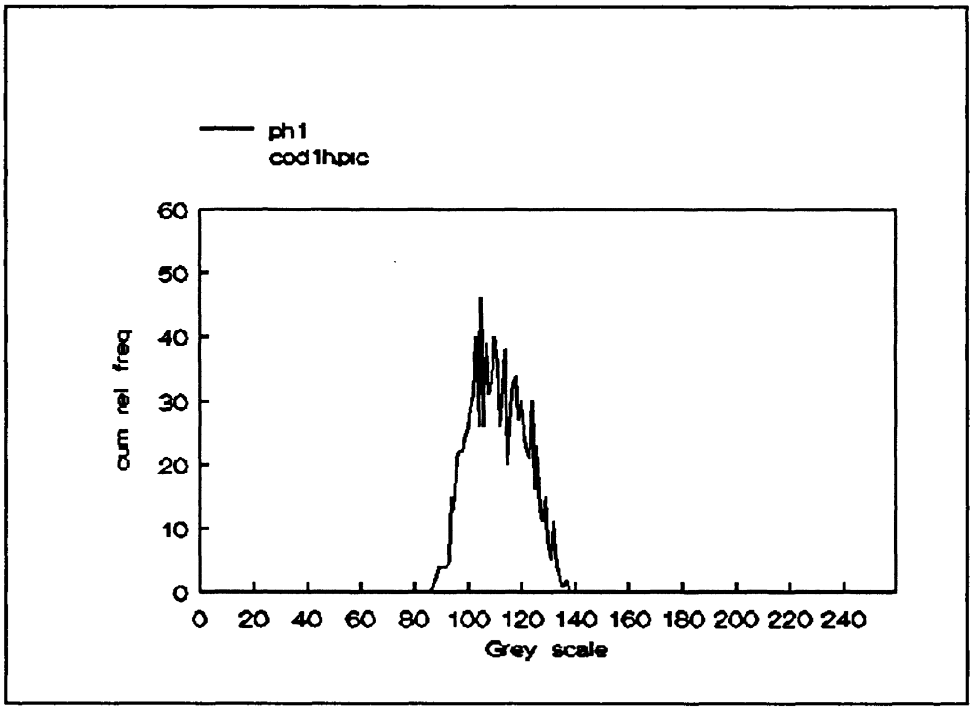
b) Section with image background.



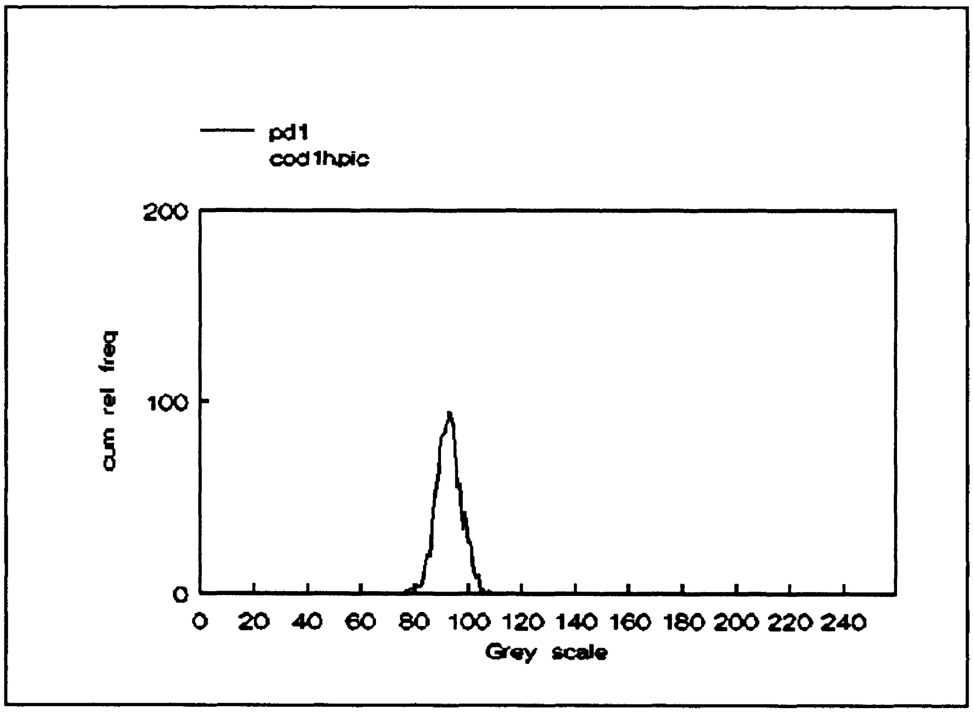
c) Section with 1/2 cut of curvy worm.



d) Section with dark muscle tissue.



e) Section with image background/fish muscle.



f) Section with light muscle tissue.

Fig.4.9a, b, c, d, e, and f. are illustrations of cumulative frequency distribution curves of 32x32 for sections with different patterns.

grey scales have cumulative frequencies of 23, 24 then their mean will be 23.5. This value will then be assigned to the cell containing these two grey scales which happens to be the first cell. This is then repeated to the end of the spread.

b) **ENLARGEMENT** - If the spread of the cumulative distribution is less than 32 grey scales, then an enlargement procedure is undertaken as follows:

First divide 32 linear spaces\cell into the number of grey scales in the spread. Then starting with the first linear cell, assign the cumulative frequency of the first grey scale to the first linear space of the group of linear spaces that fall within that grey scale. The cumulative frequencies for the remaining linear spaces is then found by interpolation of the cumulative frequency of the first grey scale to the cumulative frequency of the next grey scale. For example, for a spread of 16 grey scale levels, after the division will produce 2 linear spaces/cells per grey scale. Therefore if the first two grey scales have cumulative frequencies of 23 and 24, then the first linear space is assigned a cumulative frequency of 23 and the next linear space is assigned 23.5, by interpolation to the cumulative frequency of the next grey scale. Then move to the next grey scale and repeat the whole procedure again. Repeat to the end of the spread.

c) **NO CHANGE** - If the spread of the cumulative frequency distribution is exactly equal to 32 grey scales, then map the cumulative frequencies directly into the 32 linear spaces.

Step 4: Scale the cumulative frequencies to a workable range, e.g using the largest cumulative frequency occurring in the section to scale it to between 0 to 1.

From the curves displayed in Fig. 4.10, there is little to no change in both shape and size of the curves after the application of the algorithm and thus most of the curves features are still preserved.

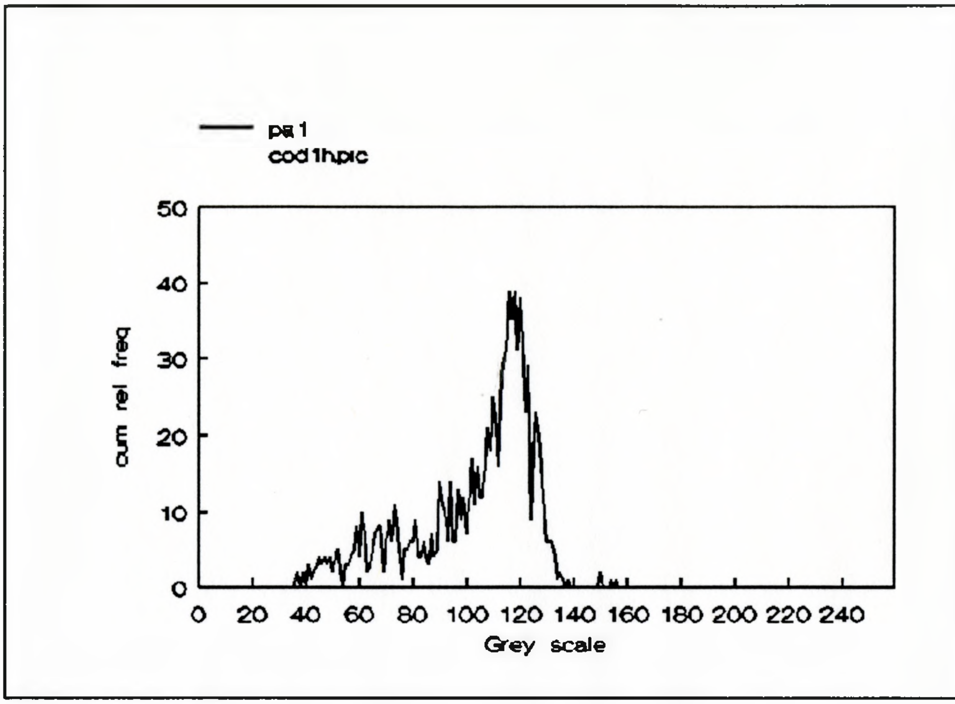


Fig.4.10a Typical curve for a parasite background.

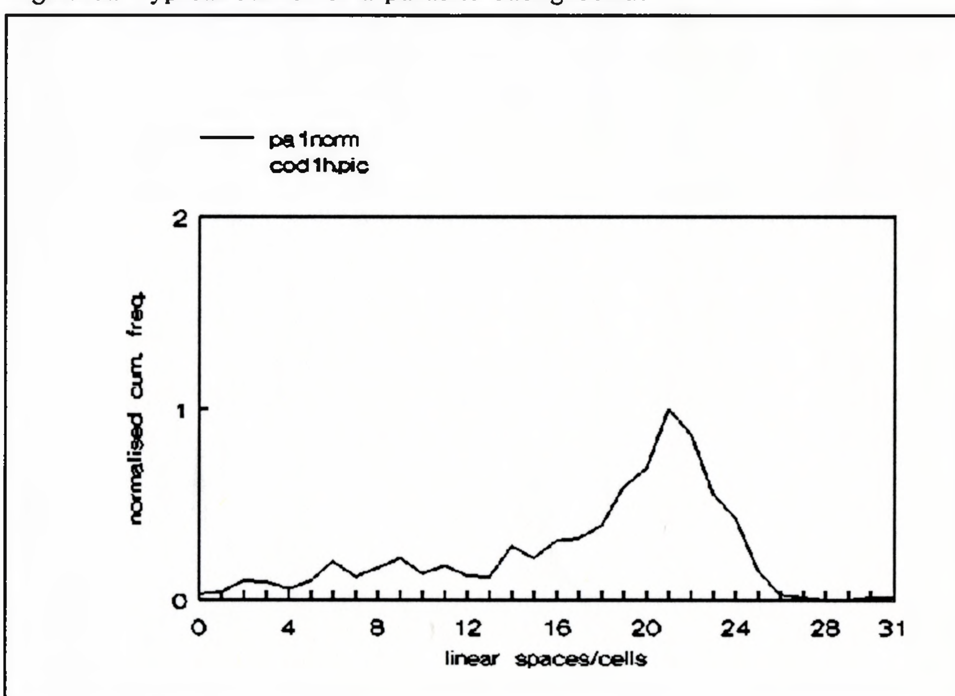


Fig.4.10b Curve in (a) after algorithm applied.

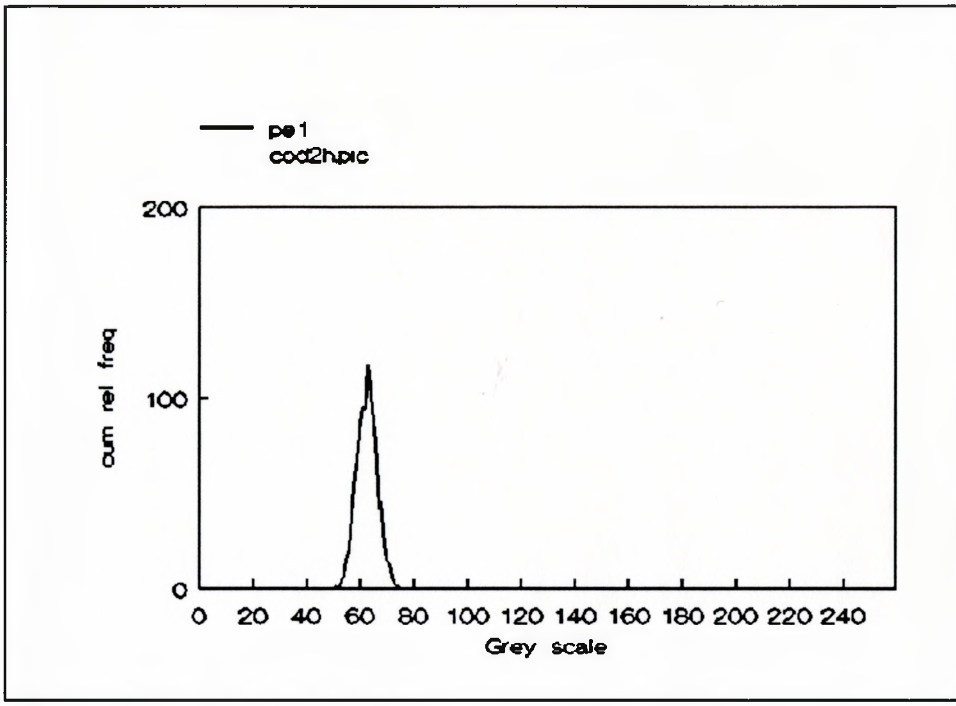


Fig.4.10c Typical curve for a normal background.

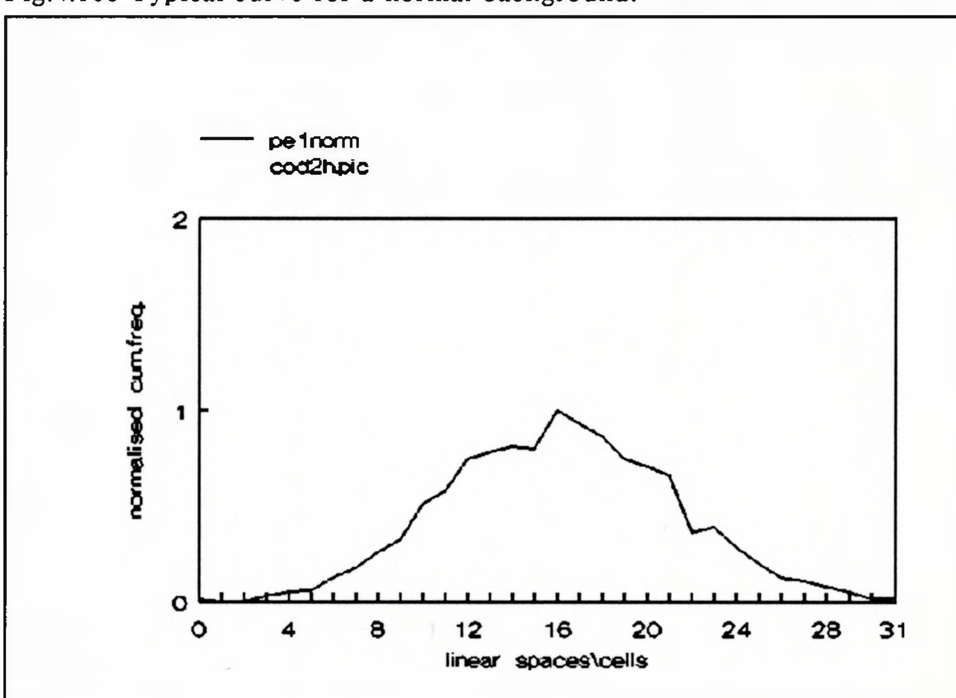


Fig.4.10d Curve in (c) after algorithm applied.

4.7.2 Smoothing algorithm

This is a simple tracing algorithm which follows the shape of a curve and smooths the spikes that occur at various positions of the curve. This way the network learns the shape of a smoothed curve rather than that of a spiked curve.

Step 1: Start at the beginning grey scale of the curve and assign it a cumulative frequency of 0.

Step 2: Move to the next grey scale of the spread and check the cumulative frequency of that grey scale against the previous grey scale's cumulative frequency.
If the present grey scale's cumulative frequency is greater than that of the previous grey scale by 0.5, then assign the previous grey scale's new cumulative frequency + 0.1 to the present grey scale.
elseif the present grey scale's cumulative frequency is less than that of the previous grey scale's cumulative frequency by 0.5, then assign the previous grey scale's new cumulative frequency - 0.1 to the present grey scale.
else assign the previous grey scale's new cumulative frequency to the present grey scale.

Step 3: Repeat step 2 to the end of the curve.

CHAPTER 5

EXPERIMENTAL ANALYSIS

5.1 Introduction

This chapter describes the experimental analysis which includes the possibility, practicability and efficiency of using the neural network technique to detect, locate and recognise parasites on a cod fish image. It also describes the efficiencies of the various schemes used in this analysis described in earlier chapters and the results and concludes with the processing and production times of these schemes.

5.2 Analysis

As described in the data representation section, the unprocessed and processed formats of the input data were used in the experimentation carried out in this investigation. The cod fish images provided and used for the analysis contained four sections of 512x512 matrix sizes, COD1, COD2, COD3, and COD4. To reduce

-the difficulties encountered with visual/display observations of these images,

-the impracticability of analysis with image sizes of that order,

-the size of image files generated,

-the slow processing during the on-line production use of the network,

a reduction of the images was undertaken. This included using the neighbourhood averaging method described in section 3.2.2 for image reduction to a workable size. Now a quarter of its original size, window sections of 32x32 pixel of the images were used throughout this investigation.

5.2.1 Unprocessed format

This is the format in which the input data is presented to the neural network in its raw form. The pixel representation falls under this format where the actual grey scale levels of the image patterns are used directly by the neural network. As explained earlier, patterns of 32x32 window sections were chosen selectively including:

fish muscle background with parasite sections,

fish muscle background without parasite sections and

an image background,

from the fish images. These were then used to train the neural network using the two schemes under the pixel representation described in earlier sections.

a) **The majority parasite scheme** - Using the criteria elaborated in the section 4.6 coupled with scientific judgement, the patterns for training were selected starting with

-two parasitic patterns

-one fish muscle pattern

-one image background pattern

which meant that where the parasitic pattern was approximately equal to the 32x32 section being considered then the five different positions outlined in section 4.6 was fully duplicated in the training set, otherwise scientific judgement was used to select the number of positions that safely satisfied the criteria in section 4.6. Therefore the first set used was made up of

10 patterns selected from COD1 for the training set:

1a. Fish background with parasite - No.1 parasite comprised of 5 different patterns

- -parasite in the centre position
- -parasite translated to the left position
- -parasite translated to the right position
- -parasite translated to the top position
- -parasite translated to the bottom position

1b. Fish background with parasite - No.2 parasite comprised of 3 different patterns

- -parasite in the centre position
- -parasite translated to the top position
- -parasite translated to the bottom position

2. Fish background without parasite - 2 different patterns

- -fish muscle background without parasite
- -image background.

These are patterns which best represent the pattern being selected, i.e., the fish muscle with parasite, the fish muscle background without parasite and image background. Thus in some of the selections two fish muscle background patterns were chosen since one pattern was not fully representative of the entire fish muscle background in the particular image. Also there were cases when patterns consisting of fish muscle "blob" tissues were included to differentiate the muscle "blobs" from that of the tissue muscle containing parasites.

Four different sets of training patterns were generated for use from the four cod images, set1 contained patterns extracted from COD1 and similarly set2, set3, set4 from COD2, COD3 and COD4 respectively.

set1 --- 10 training patterns

set2 --- 16 training patterns

set3 --- 20 training patterns

set4 --- 15 training patterns

with a network topology using an input vector of 32x32

1024 units for input layer

32 units for hidden layer

2 units for output layer

the selections of which have been described in earlier sections. Training of the neural network was done with these four training sets respectively and then tested on the four cod fish images. The results are presented in the form of a confusion matrix showing the level of recognition achieved listed in Table 1 in section 5.2.1c below.

A second procedure undertaken was to combine the training sets, i.e., combining the number of patterns from one set of patterns with the other sets to make four different sets. Com1 is made up of the patterns in the set1 training set, com2 is made up of the combined patterns in the set1 and set2 training sets, com3 is made up of set1, set2 and set3, and com4 is made up of set1, set2, set3, and set4.

com1(set1) --- 10 training patterns

com2(set1+set2) --- 26 training patterns

com3(set1+set2+set3) --- 46 training patterns

com4(set1+set2+set3+set4) --- 61 training patterns

Again using the topology described above, the neural network was trained and then tested. The testing results are shown in the form of a confusion matrix in Table 2 in section 5.2.1c below.

b) **The one parasite scheme** - This followed the same procedure described above with 8 networks. Each network was trained to recognise one particular parasite pattern. The testing results were similar to that of the majority parasite scheme but it had two serious drawbacks. The first problem is that, it has large time constraints. For example with the 8 networks it takes 8 times as much time as that taken under the majority parasite scheme, with a training set that includes the 8 parasitic patterns recognised in the one parasite scheme. The second problem is that since the 8 networks must be stored and loaded at the same time during the on-line production phase, the storage requirements are very enormous. Thus investigation in this area was not pursued further.

c) **Confusion matrices** - The results of the testing for the level of recognition are shown in the confusion matrices below for the two sets described above with the first

	cod1	cod2	cod3	cod4
set1	100	81	76.5	69
set2	97	86	80	72
set3	69	44	86	36
set4	55	56	72	65

Table 1

procedure in which the training patterns were extracted from CODl, COD2, COD3, and COD4 in Table 1. in percentages. From the confusion matrix in Table 1, it can be observed that except for set4, the recognition level achieved with an image from which the patterns for training were selected, was above 85%, as shown along the diagonal of the matrix.

Setl and set2 also achieve recognition levels of 70% and above, not only for images that the training patterns were selected from, but for images from which no patterns for training had been selected, whilst set3 and set4 did poorly on images that their training patterns were not selected from.

It is clearly observed from these results that the neural network performs rather well on images that the training patterns has been selected from, and poorly or may even be unpredictable for images on whose patterns it has not been trained.

	cod1	cod2	cod3	cod4
com1	100	81	76.5	69
com2	100	87.5	80	67
com3	89	80	89	66
com4	77	75	86	66

Table 2

In Table 2, the results from the second procedure in which the patterns from the training sets in the first procedure has been combined are presented and these observations are almost similar. Recognition levels except for com4, along the diagonal reach 85% and above for all the training sets and the recognition level improves as the set of patterns from another image is added to the set of training patterns. Apart from the poor recognition levels achieved for unknown images, there is a loss of information as the training patterns increases by the addition of patterns from the other training sets. This is depicted by the column information in the Table 2 representing the recognition levels for each particular image. This means that increment in the number of training patterns must be done and monitored carefully.

The conclusion from these observations can be summarized simply by saying that the network does not generalise easily and might therefore perform poorly or even unpredictably when presented with an unknown image especially with the pixel representation format.

5.2.2 Processed format

This is the format in which the input is not presented to the neural network in the raw form but is preprocessed in one of several ways: thresholding, performing a fast fourier transform on the input data or extracting the cumulative frequency distribution of the grey scale levels in the input data. After this processing, features are extracted from the processed data and presented to the neural network. In one of the methods under the cumulative frequency distribution procedure, a second preprocessing referred to as smoothing of the curve is performed to the data before presentation to the neural network.

a) **Thresholding** - Global, local and dynamic thresholding all aim at trying to scientifically segment the image into regions that can easily be subsequently analyzed as described in section 3.2. One of these is binarisation, but in general, visual or automatic selection of the threshold factor, T, is very difficult in practise, especially in images in which the "object" areas do not possess distinct characteristics that differentiate it from the "background" areas. Both global and local thresholding failed to process the cod images into "object" and "background" segments that could easily separate parasitic backgrounds from non-parasitic backgrounds for good utilisation by the neural network. Dynamic thresholding on the other hand provided a better tool especially when applied to the 32x32 window sections of the images which tend to reduce the image complexity considerably. The drawback is the requirement for large time constraints. More time is taken by the dynamic thresholding processing than is utilised by the neural network processing, thus making its use rather impractical. It could be improved if an efficient algorithm could be developed in conjunction with a high speed hardware implementation [4], which would make its investigation worthwhile.

b) **Fast fourier transforms** - Nothing particularly worthy of investigation was observed when the fast fourier transforms were applied to the different segments of the images as investigated in other areas in this project and thus its investigation was not further pursued.

c) **Cumulative frequency distribution** - As described earlier in section 4.7.1, input patterns of 32x32 window sections from the cod fish images were processed using the histogram translation and scaling algorithm to generate distribution curves or shapes that were then presented to the neural network in the form of extracted features, a curve map, a binary curve map, and extracted features from a smoothed curve. The objective here is to train the neural network to recognise the shapes or curves of the cumulative frequency distributions and thus be able to differentiate between the bi-modal curve structure with a smaller hump adjacent to a bigger hump weighed to the origin of the curve, which is

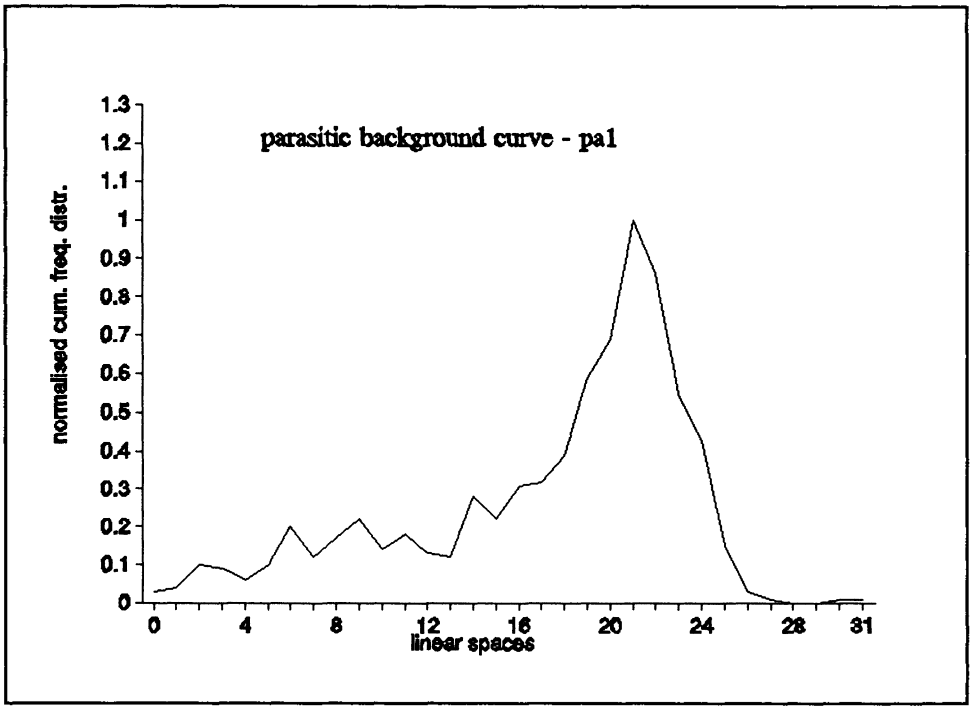


Fig.5.1a Parasitic background curve.

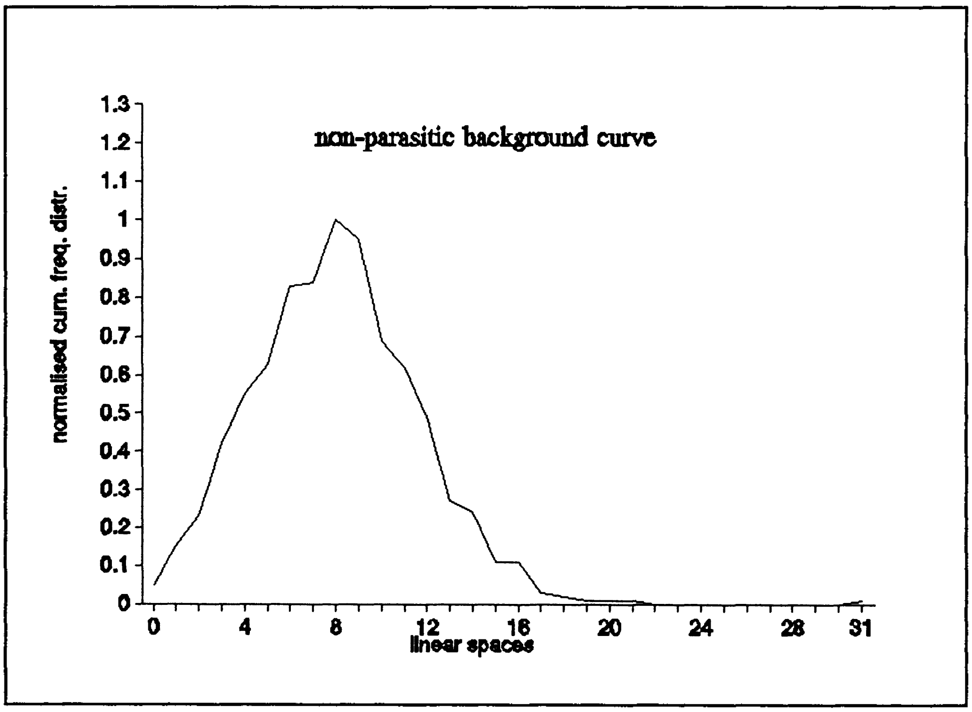


Fig.5.1b. Non-parasitic background.

representative of fish background with parasite, as against a uni-modal curve structure with slightly different characteristics that is representative of fish background without a parasite as described in section 4.7.

d) **Feature extraction** - In this format the coordinates of the distribution curves were extracted as features and presented to the neural network as a 32x1 input vector for training. Starting with two patterns, as displayed in Fig. 5.1, one pattern representing a parasite background and the other representing a non-parasitic background and using an architecture of

32 units for the input layer

65 units for the hidden layer

2 units for the output layer

the selection of which has been described earlier, the neural network was trained and then tested on the four cod images. The number of patterns was increased in the training set whilst training continued to 16 different patterns. Fig. 5.2 illustrates these curves. Testing was done and the levels of recognition achieved are presented in Table 3 below. From the results, it is rather interesting to observe that as the number of patterns are increased the level of recognition increases significantly and that from the training set of four patterns, the level of recognition rises to 55% and higher. Also for the training set of 4, 6, 8, 10, 12, and 16 patterns(which reflect an equal weighting between the number of patterns that contain parasitic background and those that contain non-parasitic background), the level of recognition is above 65%.

Since the distribution curves that the neural network is trained to recognise, are not exactly representative of a section from a particular image, but rather a representation of a general background, it means that the generalisation achieved with this format is better than

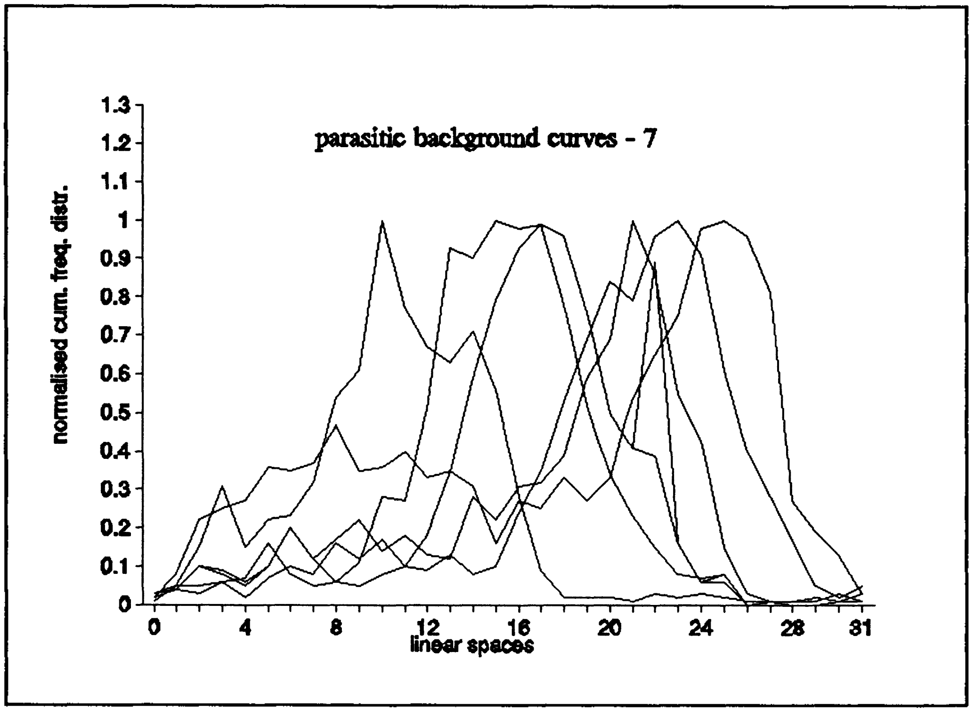


Fig.5.2a. Parasitic background curves.

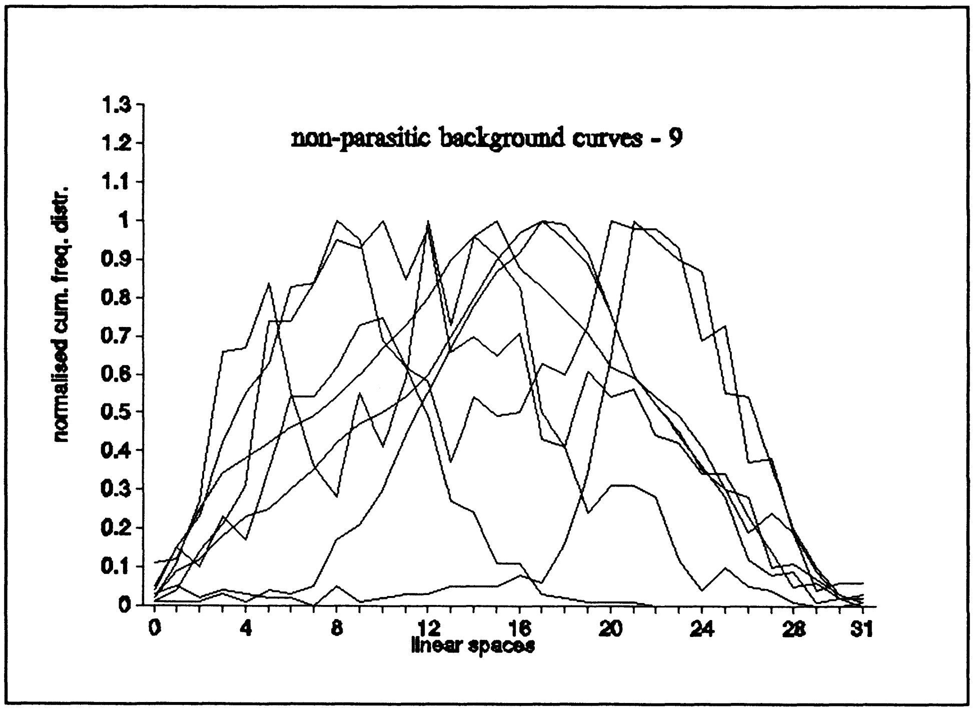


Fig.5.2b. Non-parasitic background curves.

tr. pats.	2	3	4	5	6	7	8	10	12	15	16
cod1	34	27	82	81	81	83	81	84	79	85	87.5
cod2	30	25	67	58	80	80	80	69	84	70	70
cod3	45	45	67	64	70	69	70	69	75	73	76
cod4	33	25	76	73	80	78	78	80	84	81	81

Table 3

that of the unprocessed format of the pixel representation. The best level of recognition seems to occur with a training set of 12 patterns, in which case the level of recognition reaches 80%.

e) **Curve map** - In this format the distribution were presented to the neural network in a shape form in which the exact strengths of the coordinates representing the shape of the distribution curve is embedded in a background of zeros to form a sort of curve map as shown in Fig. 4.7b.

An input vector of 32x11 was used for the presentation with a topology of

352 units for the input layer

64 units for the hidden layer

2 units for the output layer

training and testing was done and the results are presented in the Table 4.

From the results, the level of recognition achieved with this format is very low and to some extent not very consistent. For example, the level of recognition for a training set of 6 and 8 patterns is around 50%, then drops for the training set of 10 and 12 patterns and then rises again for the training set of 16 patterns. This means that this particular format is not very suitable and does not improve the efficiency of the network training.

tr. pats.	2	3	4	5	6	7	8	10	12	15	16
cod1	22	8	27	23	53	36	50	38	31	48	69
cod2	23	22	34	33	56	42	53	42	33	41	61
cod3	34	27	39	44	58	42	61	48	45	55	64
cod4	38	22	47	45	61	47	58	50	44	55	70

Table 4

f) **Binary curve map** - This is similar to the curve map format where the actual coordinate strengths that define the shape of the curve are replaced by binary signals; that is, ones(1's) representing the coordinates or shape of the curve while the rest of the map is represented by zeros as described in section 4.5.3. Using the same architecture as in the curve map procedure, training and testing was done and the results are presented in Table 5.

Although there is a slight improvement in the level of recognition over that of the curve map scheme, the results are generally weak.

tr. pats.	2	3	4	5	6	7	8	9	10	12	15	16
cod1	33	17	56	56	61	55	62	52	59	59	58	63
cod2	25	22	50	47	59	56	59	55	59	56	47	56
cod3	38	30	58	55	56	55	61	50	61	55	55	63
cod4	41	17	61	58	72	67	75	66	72	72	75	73

Table 5

g) **Smoothed curve** - In this format, a second preprocessing is done on the distribution curve generated using the smoothing algorithm described in section 4.7.2. This smooths the distribution curve by removing some of the spikes that occur in the curves which tend to skew the exact modal structure of the curve and thereby improve the learning by the neural network. After this processing, features were extracted from the distribution curve, i.e., the coordinates of the processed curve, and were presented to the neural network in an input vector of 32x1. Using the same topology as in the feature extraction procedure, training and testing was carried out and the results are presented in the Table 6.

Again as observed in the feature extraction section 5.2.2c, as the training set is increased there is a considerable increase in the level of recognition. From a training set of 6 patterns and above, the level of recognition rises to 50% and higher, and the best results occur for a training set of 15 patterns.

tr. pats.	2	3	4	5	6	7	8	9	10	12	15	16
cod1	31	25	69	80	75	77	75	69	70	67	77	77
cod2	25	22	45	48	64	61	61	61	66	58	63	56
cod3	44	34	50	69	63	64	63	52	53	50	66	56
cod4	28	27	61	67	70	75	75	64	63	64	72	64

Table 6

5.2.3 Processing times

Two tables are presented below showing the processing times for training the neural network under three data representation formats used; pixel representation for the unprocessed form, curve map and feature extraction formats for the processed forms. The individual times represent the mean of ten trials. Table 7 was generated on the SUN 4.1/280 miniframe system with variable system load at any time, and the variations of these times were within 0.1 to 1.0 second. Table 8 was generated on a SUN SPARC work station with no system load except for the root programs, and the variations were within 0.1 second.

The variations on the processing times are pronounced for the small number of patterns, for example, for 2 patterns in Table 7 the feature extraction method is 20 times faster than the pixel representation method and 12 times faster than the curve map method, Processing times on SUN in seconds

# of pats	2	4	6	8	10	12	14	16
pix.rep	7.8	57.8	65.5	92.4	121.9	202.5	292.0	228.8
cur.map	4.8	14.1	18.5	27.8	37.1	43.3	52.3	50.1
fea. ext.	0.4	0.7	1.0	6.7	17.3	11.5	16.2	33.0

Table 7

Processing times on SUN SPARC in seconds

# of pats	2	4	6	8	10	12	14	16
pix.rep	4.8	34.5	37.7	66.4	72.8	160.7	177.0	134.2
cur. map	3.0	8.4	11.2	16.7	21.6	25.4	30.6	29.4
fea. ext.	0.3	0.4	0.6	4.2	11.2	7.3	10.7	21.2

Table 8

whereas when the number of patterns is 12, these become 17 and 4. The same trend shows in Table 8. Also from 14 patterns in the training set, the processing times start to reduce. For the unprocessed format, in particular, the pixel representation, the drop is sharp which may indicate overlearning or overtraining or a phenomenon in which the network has reached its maximum knowledge acquisition capacity and to some extent begins to lose information as more is fed to it as was observed in section 5.2.1c. Those training sets with the processed format exhibit a more robust knowledge acquisition capacity, as the curve for its processing times shows a positive gradient. Which is an indication of its capacity to continue to absorb more knowledge without losing information, although that of the curve map format starts to flatten. A more consistent comparison was computed using the area under the curves in plots of processing times versus the number of patterns shown below in Fig. 5.3a.

The area ratios are the same for the two plots, pixel rep.: curve map : feature extr.

14 : 3 : 1

which indicates that the feature extraction method is 14 times faster than the pixel

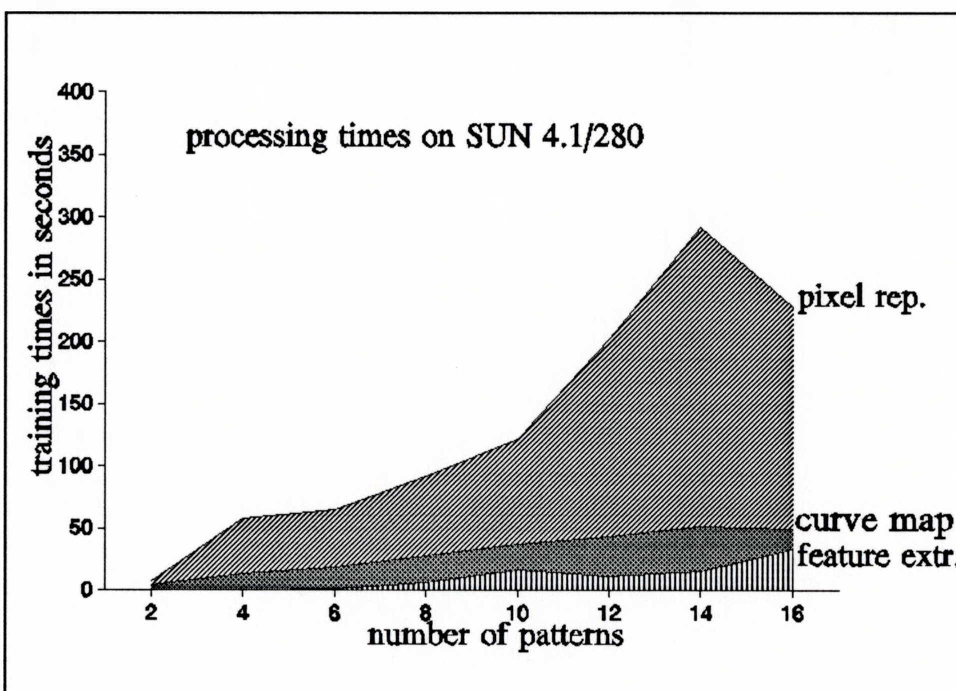


Fig.5.3a Processing times versus # of patterns on the SUN.

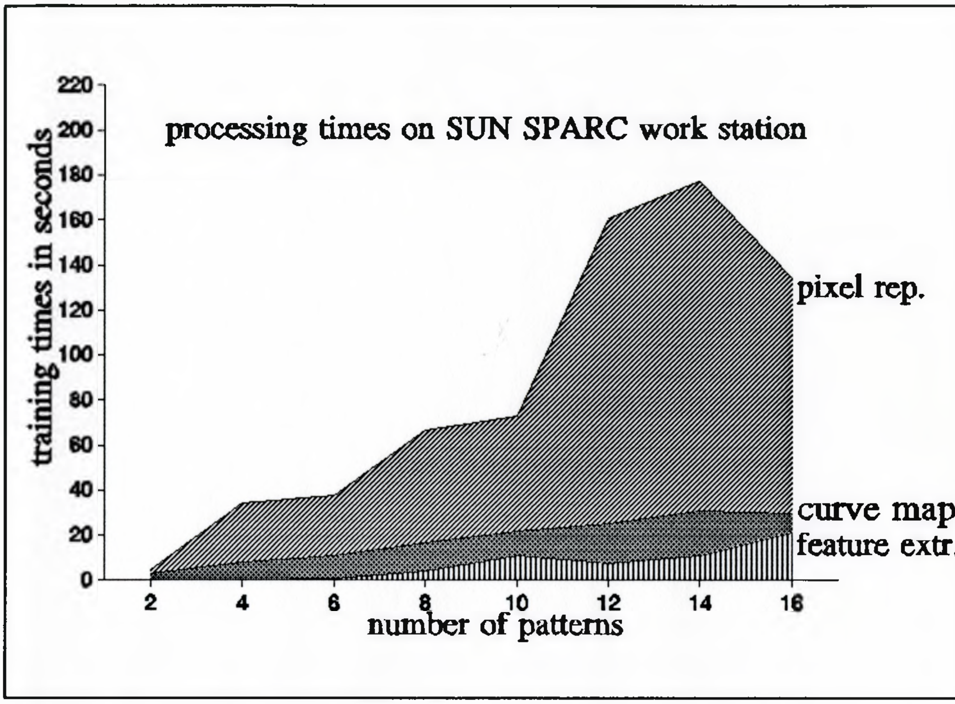


Fig.5.3b Processing times versus # of patterns on the SPARC station.

representation method, 5 times faster than the curve map method, and the curve map is 3 times faster than the pixel representation method. This clearly shows that it is faster using the processed form of the data representation than the unprocessed format. This is particularly noteworthy since the preprocessing of the raw data to the features or the curve map, i.e., processing the 32x32 grey scale pixels, on the two machines takes fractions of a second.

5.2.4 Production times

This section gives an indication of the on-line performance of the network with the various data representation after the off-line training phase. The four COD images in Fig.

1.1 were used as the testing data. Since input data from 32x32 pixels sections were used, during the production the network scans the 64 sections contained in the 256x256 pixels COD fillets images section by section. The production time therefore represents the time taken by the network to do a complete scan of one of these images. Sample data of 32x32 sections are presented in the appendix.

Table 9 shows the times in seconds for the three data representation formats used in section 5.2.3, with the individual times representing the mean of ten trials on the two machines as discussed in section 5.2.3. Again the feature extraction gives the fastest time compared to the curve map and the pixel representation. Thus the processed format in the form of the feature representation is about ten times faster than the unprocessed format in the form of the pixel representation.

Production times in sec.

	Sun	Sparc
pixel rep.	14.2	8.9
curve map	10.1	6.2
feature extr.	1.6	0.9

Table 9

CHAPTER 6

CONCLUSIONS AND EXTENSIONS OF THE WORK

6.1 Introduction

This investigation was done with a small number of cod fish images than was originally expected to be available due to funding problems with the cod fish image production project. The conclusions to the analysis and experimentations in this investigation are therefore based largely on these images which were used. It should be noted that images with substantially less parasites is more normal so that worst case conditions have been under study here. This chapter presents these observations and which directions that further investigation should be undertaken.

6.2 Neural network technique

From the results of the experimentation on the recognition levels of the parasite/sealworm from the cod fish images, this clearly shows that the neural network technique is very useful for solving such a complex problem. The steepest descent optimisation method [6] has very large time constraints whilst the conjugate gradient optimisation method [18] performed better within practical real time constraints. It therefore allows for faster implementation of the neural network technique to solve this problem with speed and efficiency. Further investigation at applying improved versions of the conjugate gradient and other faster optimisation techniques [12] can be pursued.

6.3 Data representation

Better results were achieved in terms of recognition levels, processing and production time and level of generalisation, using the processed format of data than the unprocessed format which shows that this system works better when the input data is processed to some format using vision techniques, either in a one step or several steps processing before presentation to the neural network. The processing methods can thus be further investigated for improvement such as using edge or area detection schemes which are aimed at trying to differentiate parasites/sealworms background from the cod fish image by some form of area or edge measures.

6.4 Image reduction

The reduction of the cod fish images before use proved to be very useful and improved the time for both the off-line training phase and the on-line production phase of the neural network. This can be further improved by encoding this routine in the hardware for implementation since it is a simple routine using the neighbourhood averaging method or another equally effective vision method. This can easily then become part of the automatic inspection system.

6.5 Summary

This investigation thus establishes clearly that it is very possible to use the neural network technique in combination with some form of vision technique to develope a neural network based system to analyse, recognise and detect parasite/sealworm on a cod fish image. The level of generalisation achieved by the network when trained with a selected number of patterns is very dependent to a large extent on the limits of the format in which the data was presented. The network thus tends to generalise better when presented with a processed data format because any input data presented is processed to fall within the limits of the training data used by the network. On the other side of this, the network does poorly or unpredictable when presented with an unprocessed data format since a large number of the input data presented to the network easily fall outside the limits of the training data used by the network. Therefore the practicability and efficiency of the neural network based system depends on the data representation format and in this investigation it is very clear that the best results occur with the processed format.

Further work should therefore concentrate on investigating other processing methods aimed at producing data representation formats that would improve some of the results achieved in this investigation especially with more cod fish images to work on.

BIBLIOGRAPHY

[1] Azriel Rosenfeld, "Computer Vision: Basic Principles", Proceedings of the IEEE, vol. 76, #8, 1988, pp. 863-868.

[2] Theo. Pavlidis, Algorithms for Graphics and Image Processing, Computer Science Press, 1981.

[3] Joan S. Weszka, "A Survey of Threshold Selection Techniques", Computer Graphics and Image Processing 7, 1978, pp. 259-265.

[4] D. W. Capson, Lecture Notes for E.C.E 763, Computational Vision, McMaster University, 1990.

[5] Richard P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, April, 1987, pp. 4-22.

[6] J. L. McClelland, D. E. Rumelhart, Explorations in Parallel Distributed Processing, MIT Press, 1988.

[7] Minsky and Papert, Perceptron-Expanded Edition, MIT Press, 1988.

[8] R. O. Duda, P. E. Hart, Pattern Classification and Scene Analysis, John Wiley and Sons, New York, 1973.

[9] Jeanette Lawrence, "Untangling Neural Nets", Dr. Dobb's Journal, April, 1990, pp. 38-44.

[10] Emilio A. Apey, "Back Error Propagation Simulator(BPS)", George Mason University, September, 1989.

[11] E. Barnard, R. Cole, "A Neural-net Training Program Based on Conjugate Gradient Optimisation", O.G.C Tech. Report No. C.S.E. 89 - 014, Oregon Graduate College.

[12] Martin F. Moller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", Computer Science Dept., Aarhus University, Aarhus, Denmark, [Private communication from Martin F. Moller, Nov., 1990].

[13] Scot E. Fahlman, Christian Lebierre, "The Cascade - Correlation Learning Architecture", CMU-CS-90-100, Carnegie-Mellon University, School of Computer Science.

[14] Scot E. Fahlman, "Faster Learning Variation on Back-Propagation - An Empirical Study", Proceeding of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1988, pp. 38-51.

[15] R. Hecht-Nielsen, "Kolmogorov's Mapping Neural Network Existence Theorem", 1st IEEE International Conference on Neural Net, San Diego, 1987.

[16] Raymond Ho, "A Neural Network System for Recognition of Evoked Potential", (Master's Thesis), McMaster University, Department of Computer Science and Systems, December, 1989.

[17] Mordecai Avriel, Nonlinear Programming, Analysis and Methods, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[18] A. H. Kramer, A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Network", Advances in Neural Information Processing Systems 1, edited by D.S. Touretzky, Morgan Kaufmann Publishers, 1989, pp. 40-48.

[19] CANPOLAR INC., Toronto, Ontario, "Evaluation of Potential Automated Parasite Recognition Technology", September, 1988.[1]

[20] CANPOLAR EAST INC., St. John's, Newfoundland, "Parasensor(TM) Development Milestone Report", January, 1990.[1]

[21] CANPOLAR INC., Toronto, Ontario, "Operations Report", 1988.[1]

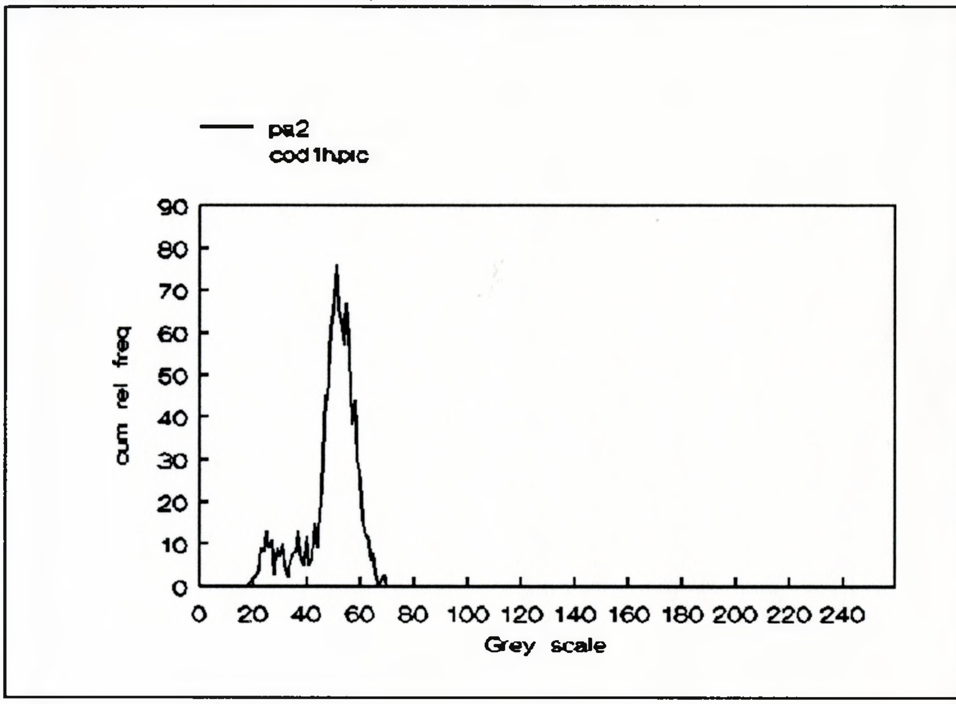
[22] Yoh-Han Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley Publishing Company, Inc., 1989.

[23] M. J. D. Powell, "Restart Procedures for the Conjugate Gradient Method", Mathematical Programming 12, 1977, pp. 241-251.

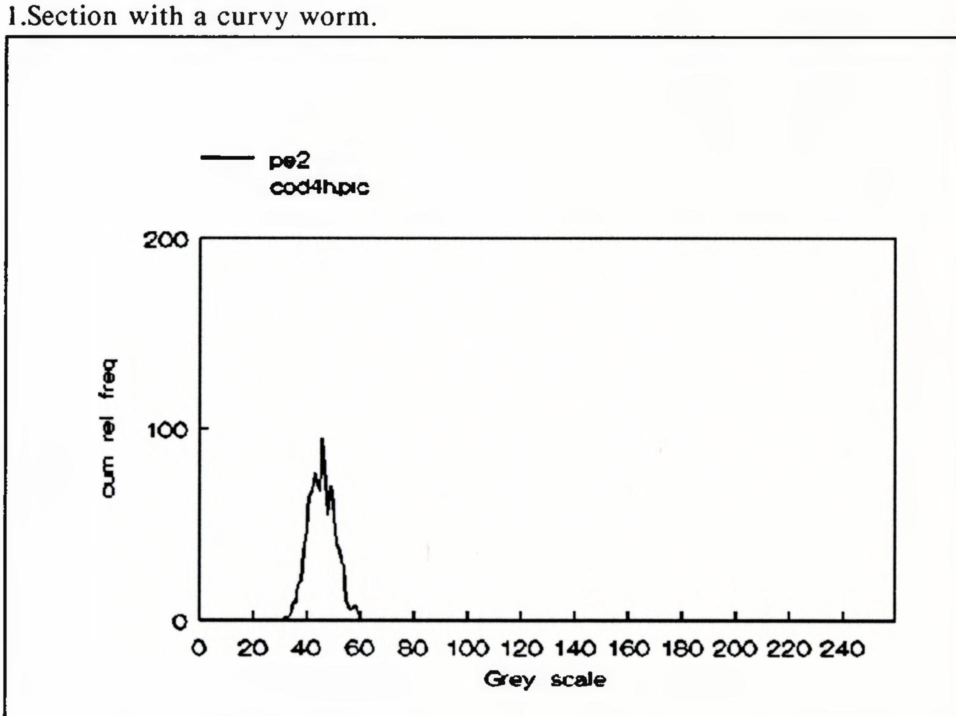
[1]Has been kindly provided by Dr. James R. Rossiter, President, CANPOLAR INC.

APPENDIX I:

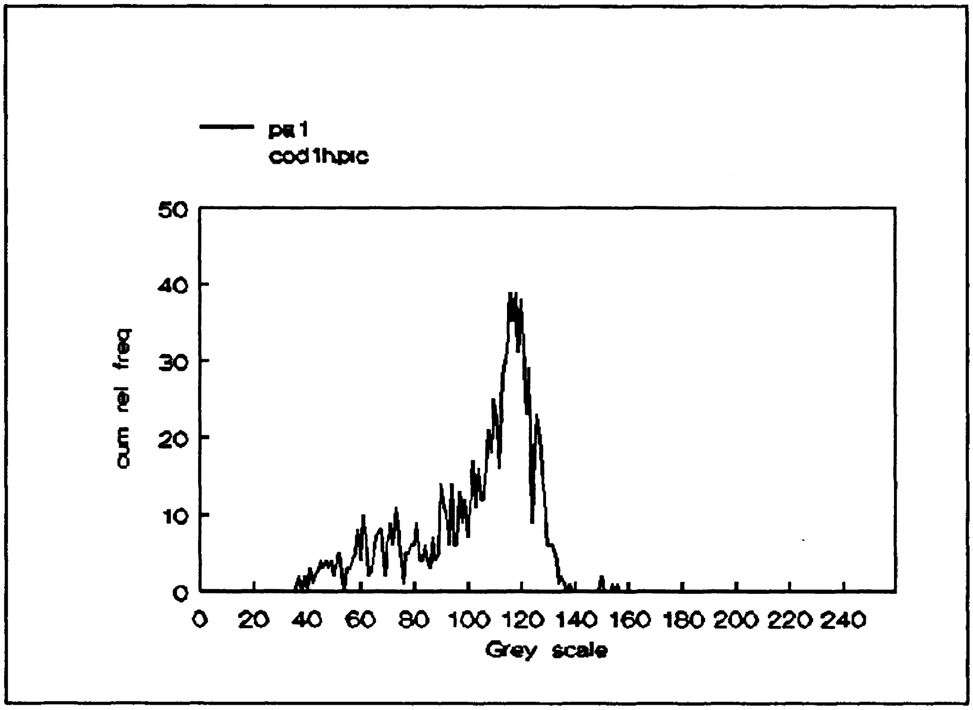
Illustrations of cumulative frequency distribution curves.



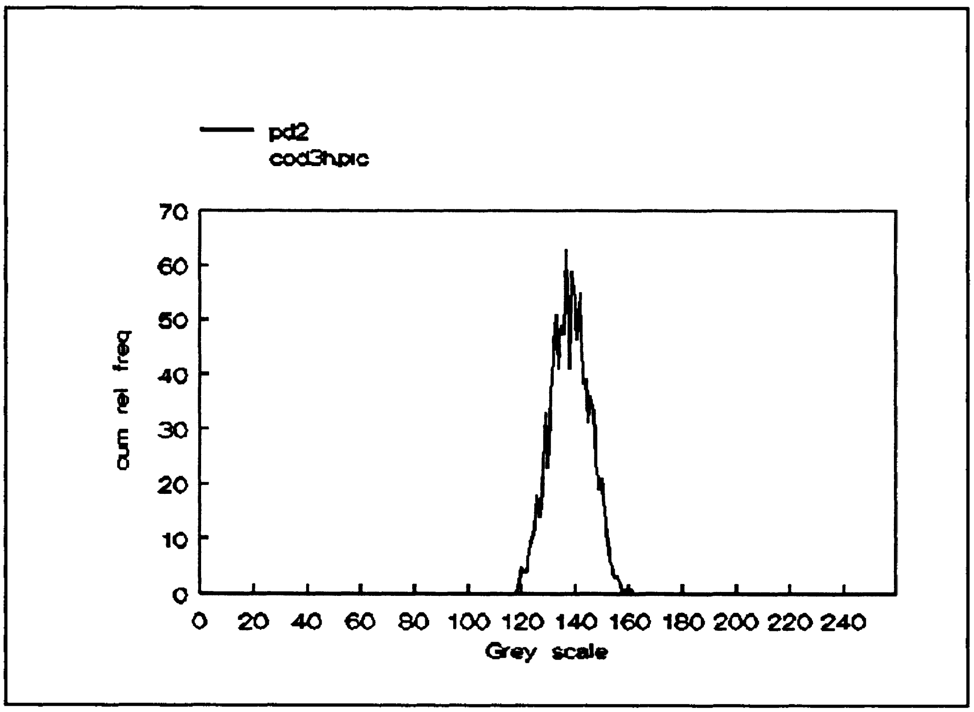
1.Section with a curvy worm.



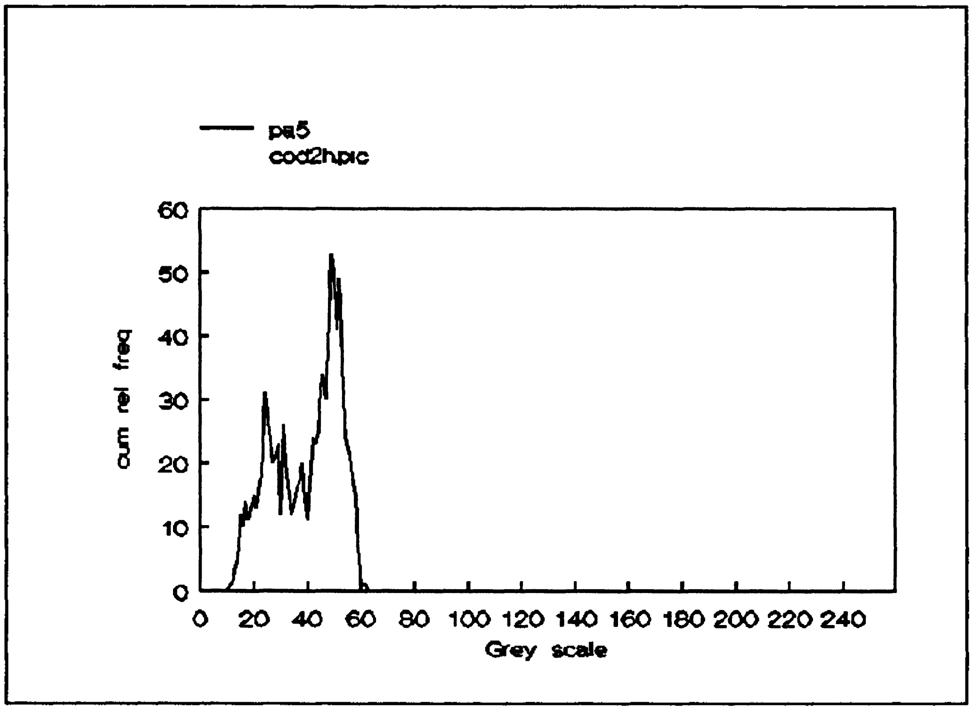
2.Section with dark muscle tissue.



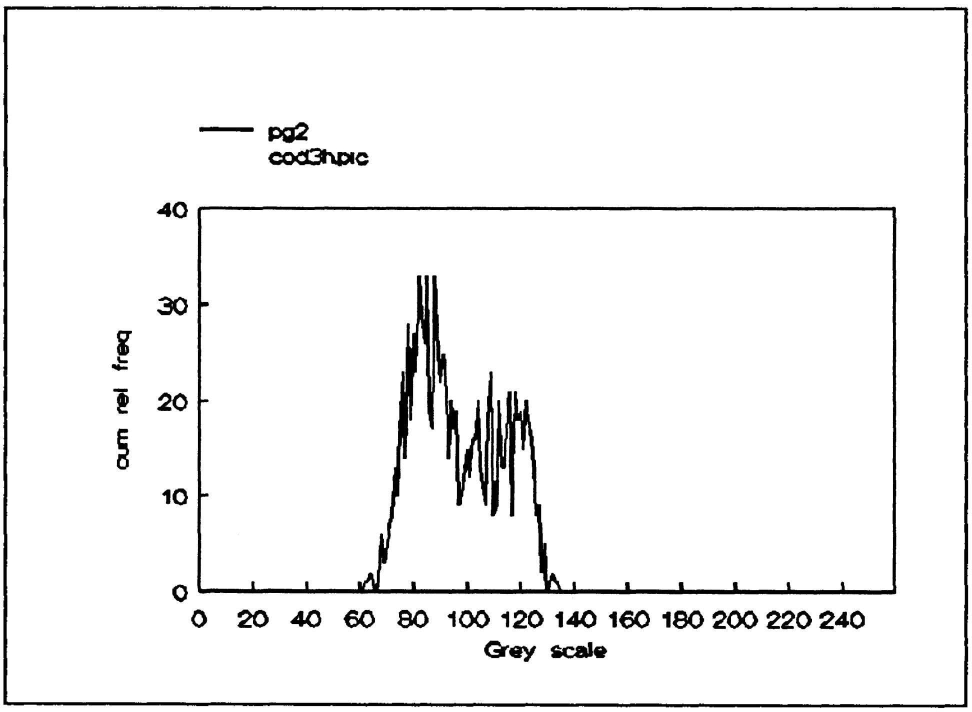
3.Section with a curvy worm.



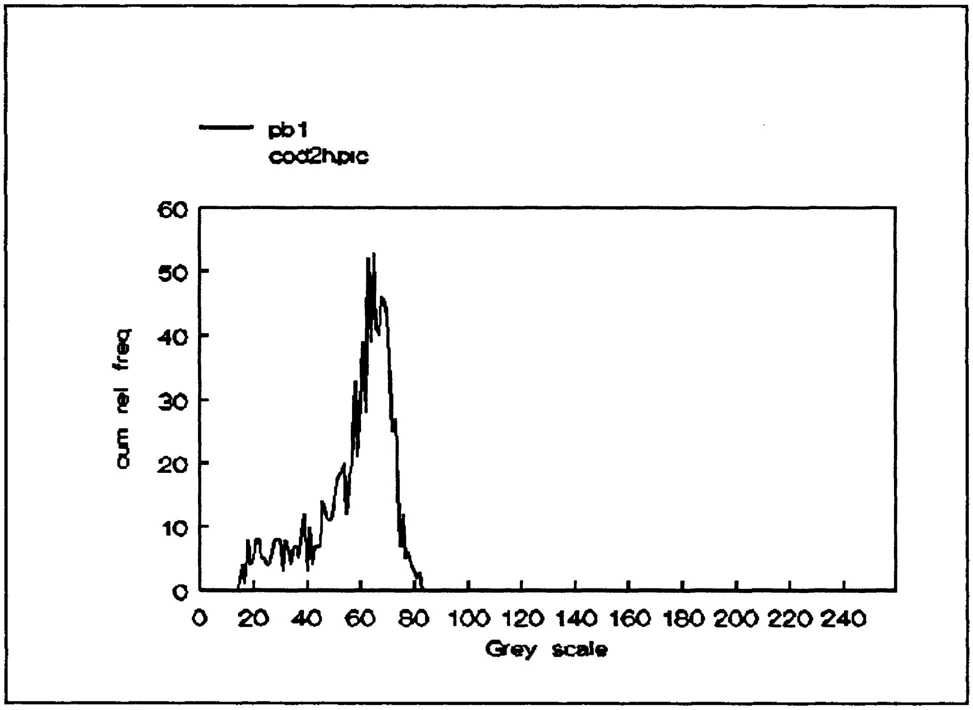
4.Section with light muscle tissue.



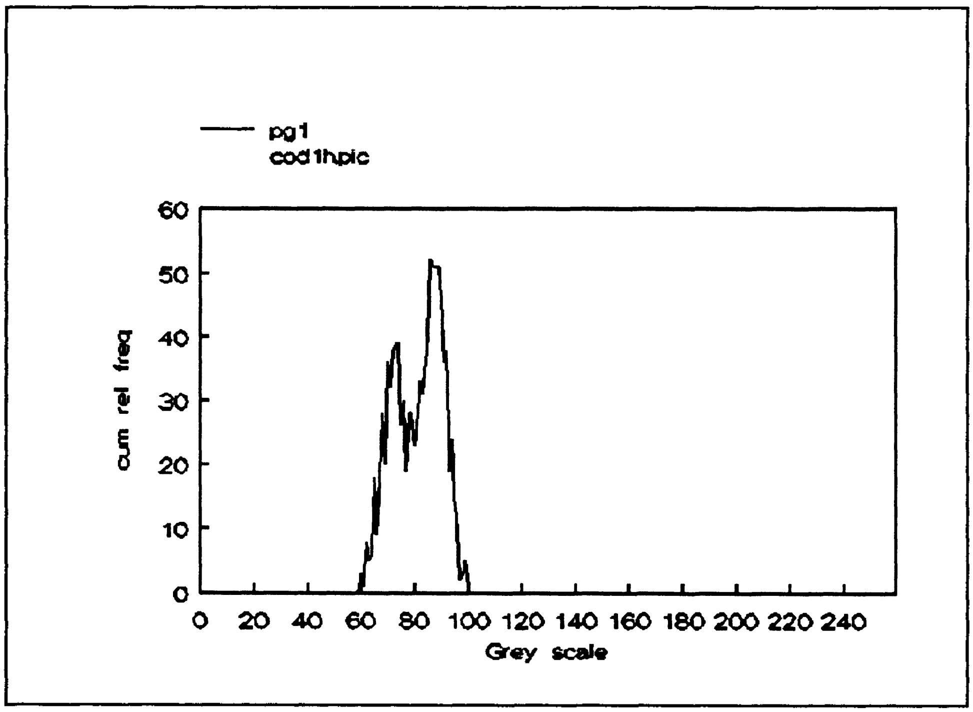
5.Section with a curvy worm.



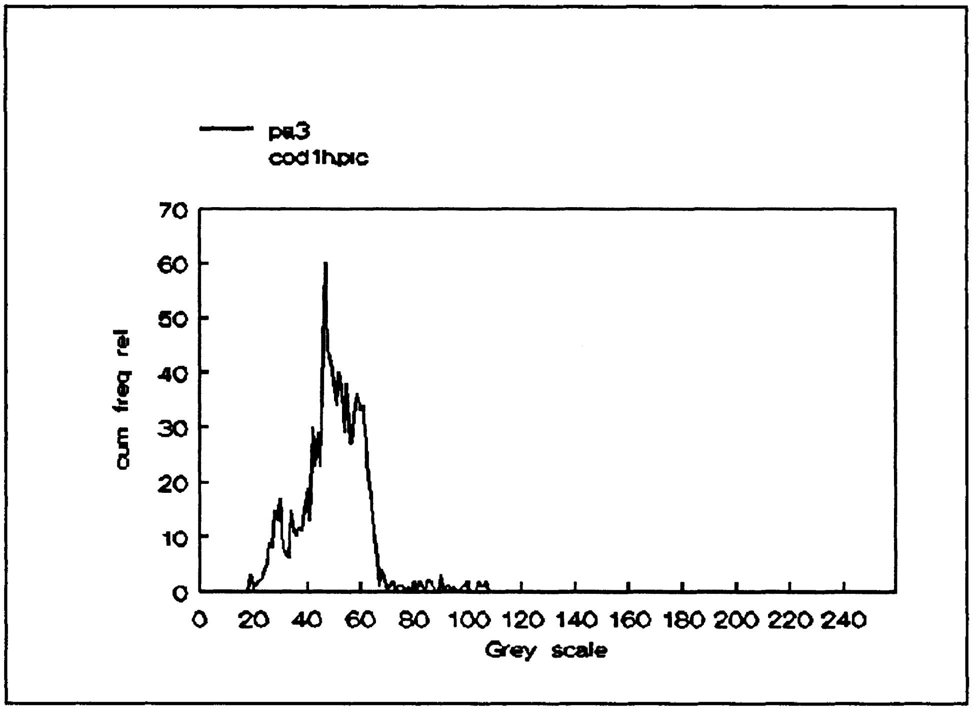
6.Section with light/dark muscle tissue.



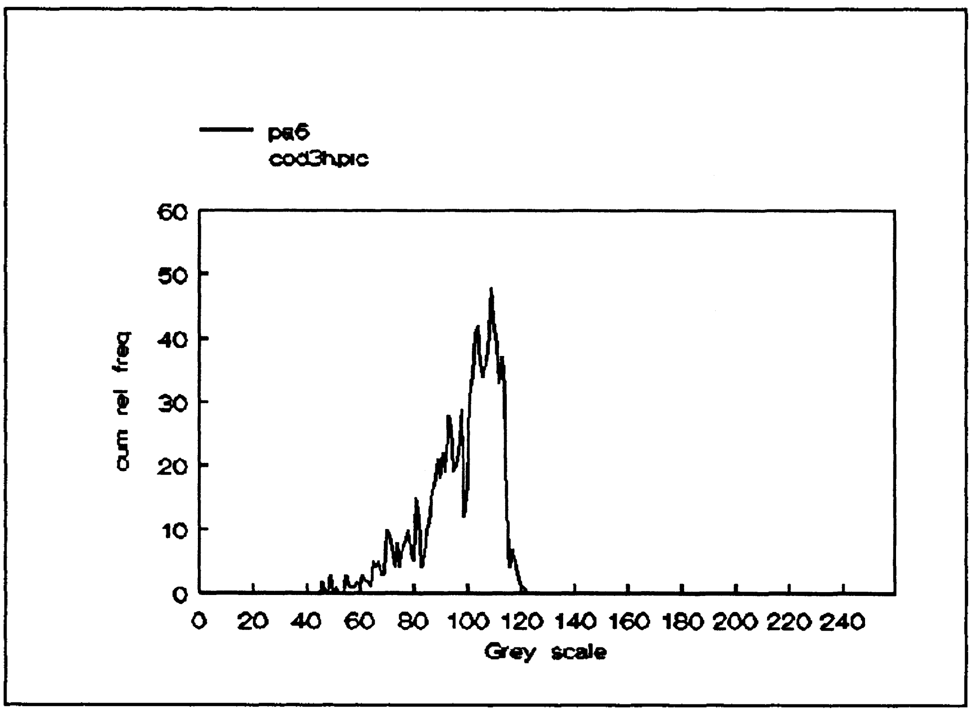
7. Section with an embedded worm.



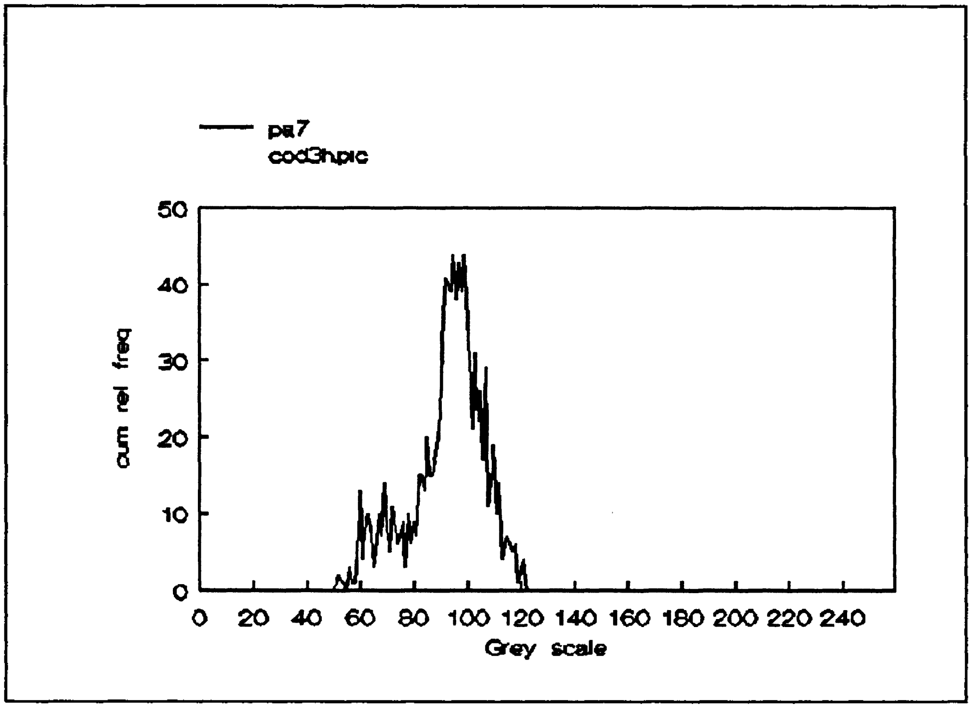
8.Section with light/dark muscle.



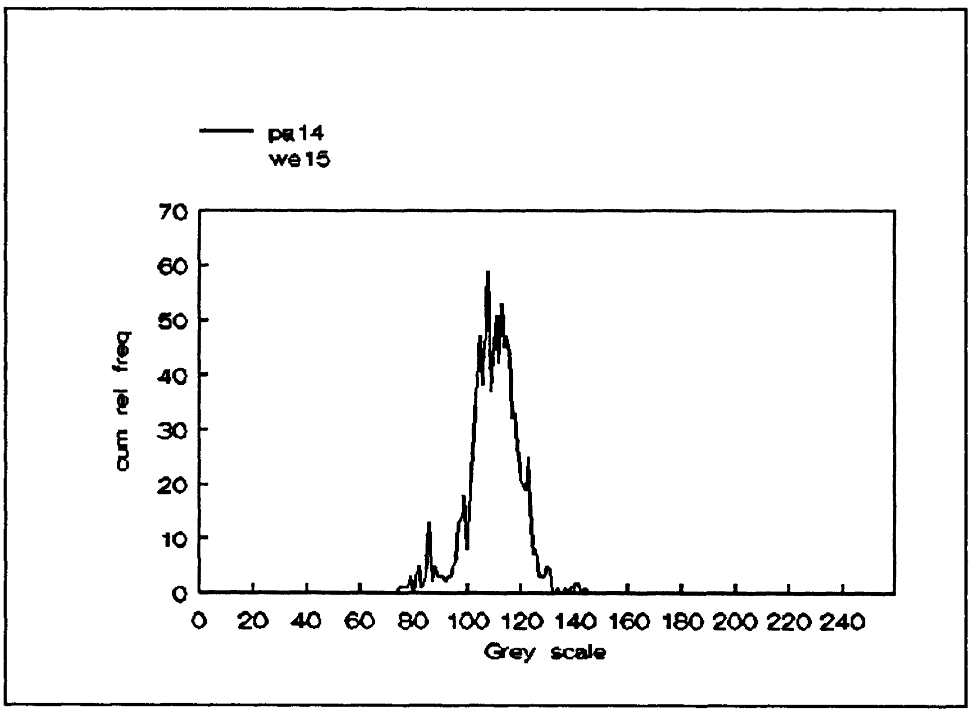
9. Section with a curvy worm.



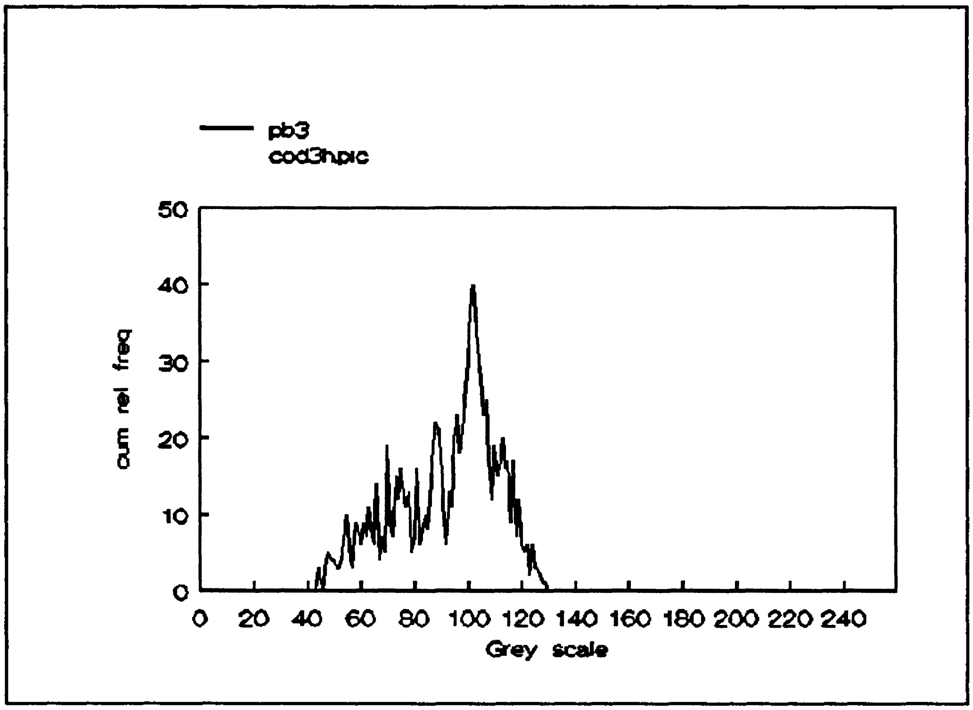
10.Section with a curvy worm.



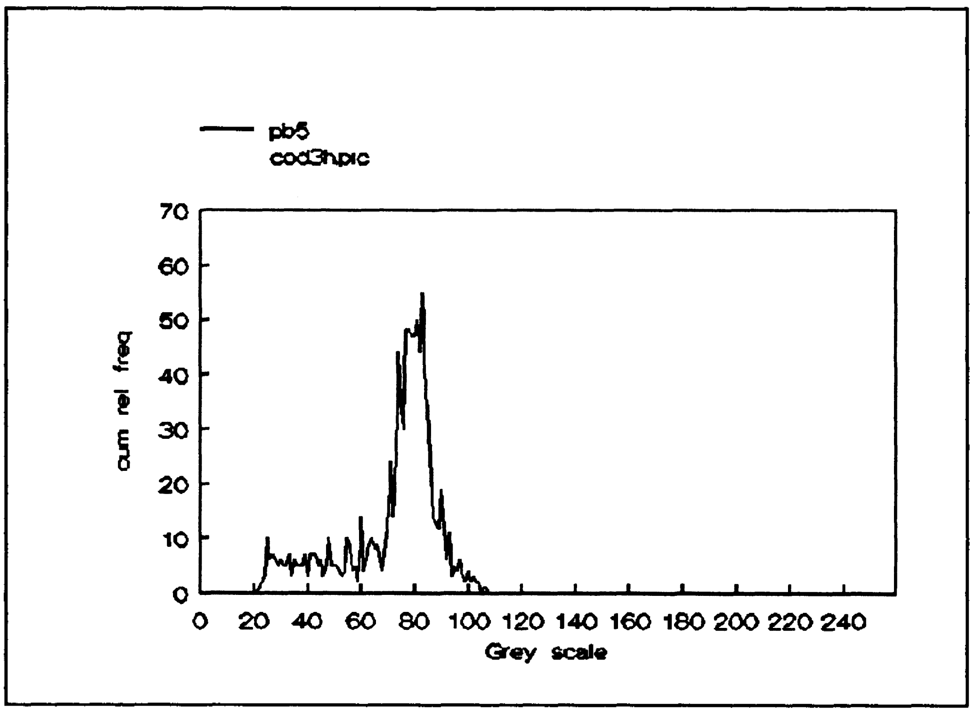
11. Section with a curvy worm.



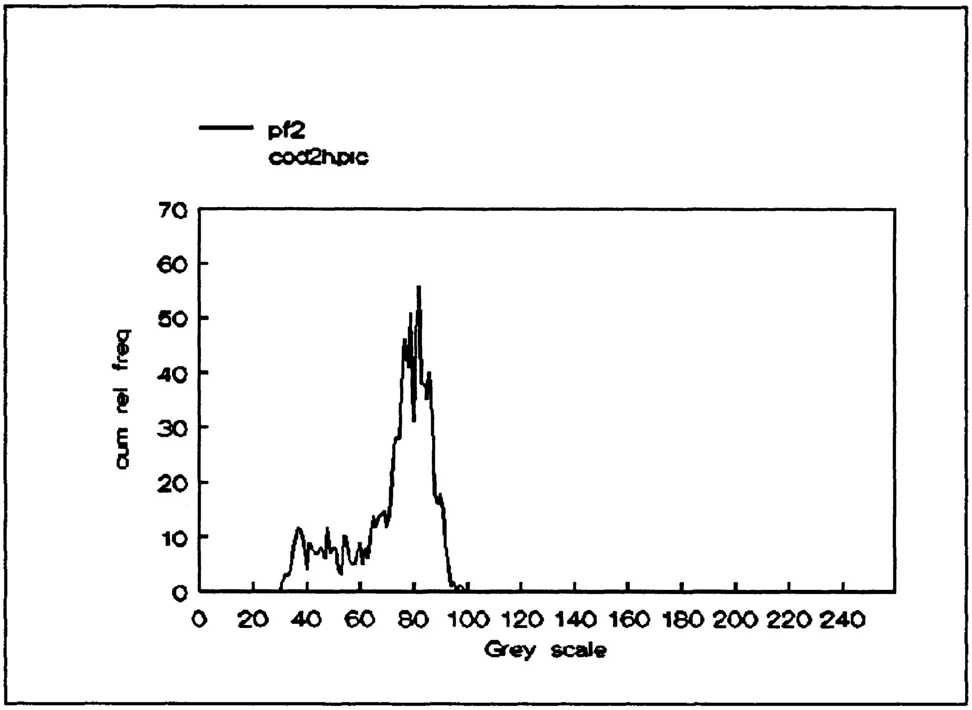
12.Section with a curvy worm.



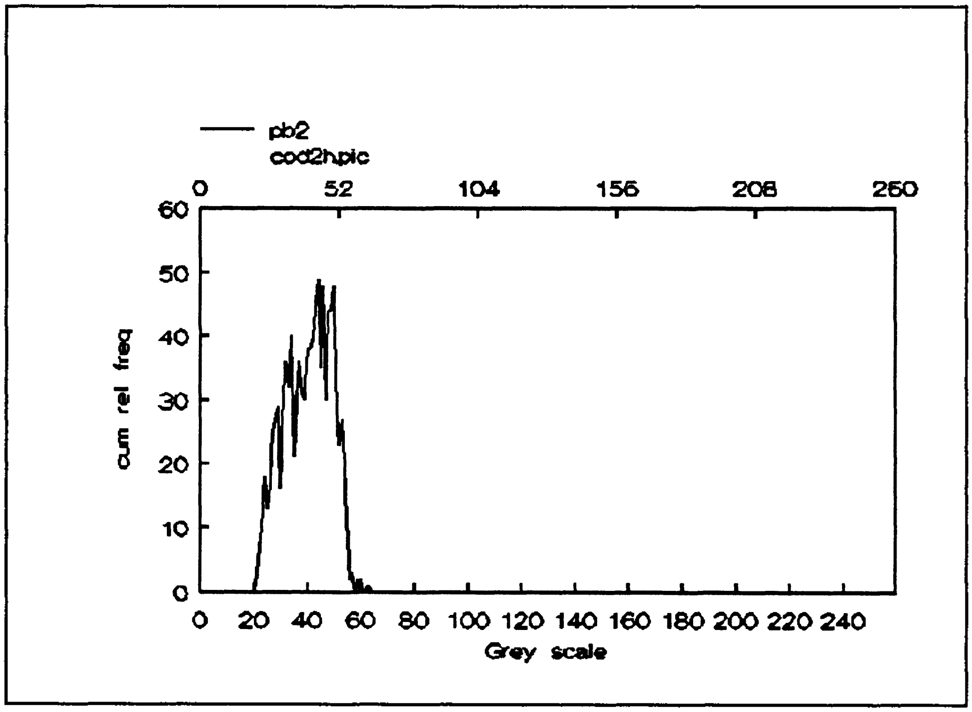
13. Section with an embedded worm.



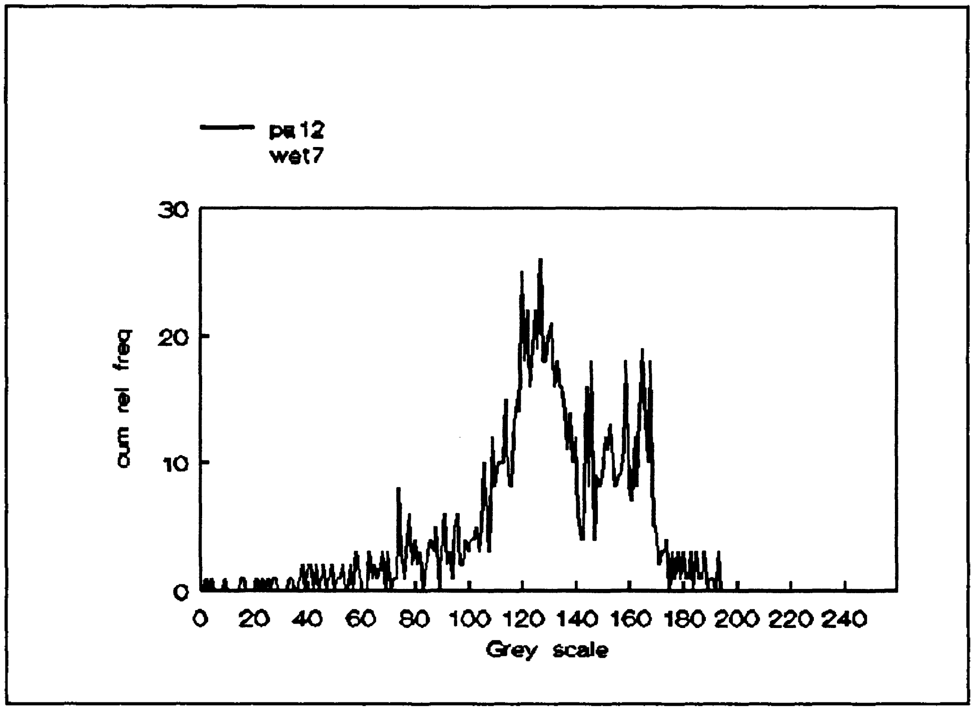
14.Section with an embedded worm.



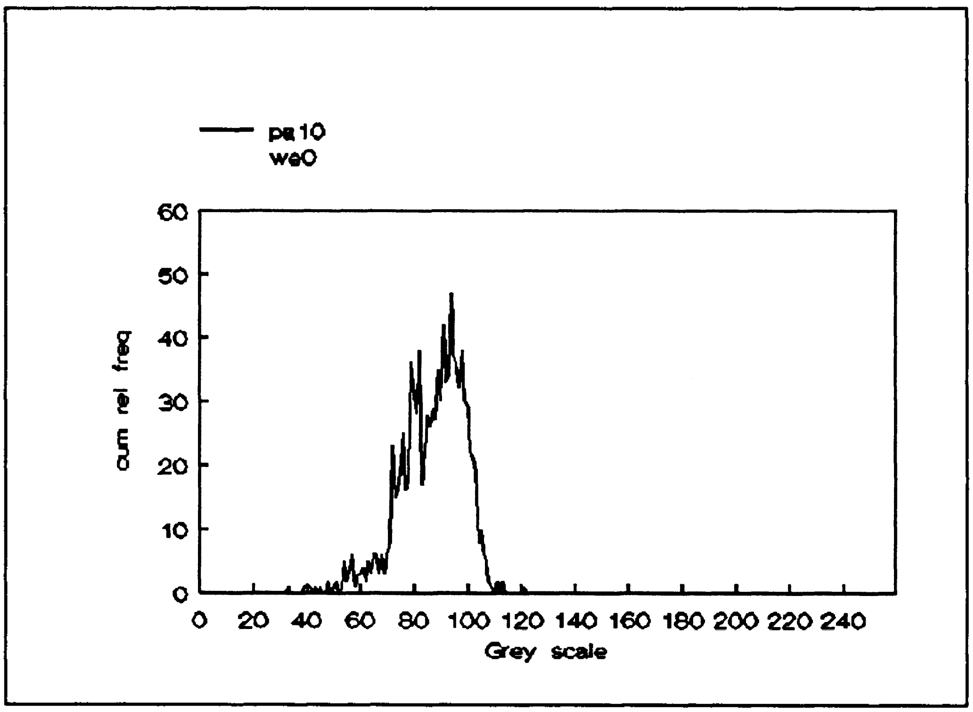
15. Section with muscle blobs.



16.Section with an embedded worm.



17. Section with a curvy worm.



18.Section with a curvy worm.

APPENDIX II

SAMPLE INPUT DATA

1. Input Data of the Pixel Representation Format(32x32 pixel section)

a) Parasitic Background

par.pr

72	99	134	132	129	121	132	128	121	127	130	133	127	119	123	127	128	128	119	112	122	121	126	120	117	114	121	123	121	110	111	126
118	113	125	128	123	127	130	129	116	120	123	135	132	123	120	135	127	125	119	122	113	117	125	125	118	111	120	120	123	117	117	122
118	118	117	132	129	125	113	113	111	119	123	120	121	127	120	128	132	121	119	113	114	119	124	118	119	112	112	119	121	118	120	125
123	115	125	126	128	128	108	108	113	119	109	116	112	120	133	122	126	130	121	126	120	115	118	118	115	111	120	123	120	115	115	123
118	119	117	112	120	117	118	122	122	119	112	119	129	128	130	118	117	122	119	129	118	126	122	114	126	119	114	117	118	113	104	114
115	112	126	118	120	114	119	120	127	127	118	123	128	121	128	123	116	125	123	124	122	117	122	117	116	118	120	123	121	121	110	116
116	118	121	118	117	114	115	115	127	126	127	123	121	123	131	126	118	119	117	106	111	100	106	104	120	121	126	127	129	118	108	115
115	124	119	116	119	124	121	116	128	117	118	122	127	129	123	129	107	98	92	79	82	80	74	80	97	121	127	123	125	116	117	115
117	116	124	117	118	118	119	119	118	123	113	110	120	120	118	108	88	75	71	67	74	72	68	70	78	95	115	126	120	116	116	117
108	116	127	119	116	119	121	120	115	118	114	116	117	110	99	91	72	68	72	75	86	84	88	87	73	81	107	120	118	114	113	116
113	111	120	113	112	116	118	118	117	120	111	121	111	98	71	63	67	73	68	65	73	80	90	87	73	67	95	109	115	109	110	114
111	117	116	107	110	117	122	127	121	123	119	114	94	73	52	53	61	68	66	66	64	66	74	71	65	65	89	116	110	111	110	120
110	112	126	118	115	126	122	125	118	117	117	100	73	49	45	51	56	66	68	61	65	67	60	58	58	71	91	109	123	117	105	118
110	111	118	117	118	133	128	121	117	123	110	81	63	52	46	45	48	51	49	48	61	70	66	56	50	60	81	107	118	113	110	114
108	114	122	118	127	121	124	115	119	120	100	72	52	55	49	44	43	46	39	43	47	51	55	61	57	58	77	92	109	107	108	120
111	106	128	123	124	123	127	125	121	112	85	59	42	36	41	39	44	56	57	57	56	51	48	52	58	59	71	78	101	103	111	110
114	119	126	125	126	119	126	122	128	110	81	52	41	41	49	59	87	94	102	93	90	74	61	61	60	59	61	75	82	92	102	108
108	113	126	122	123	121	121	117	124	100	77	45	37	47	67	101	113	113	113	113	116	102	90	65	61	66	62	75	78	85	104	107
105	113	120	122	124	125	121	114	113	101	70	44	37	62	90	116	116	114	115	113	120	117	108	83	74	68	70	69	67	78	92	106
105	111	117	122	117	106	94	99	91	83	59	46	47	79	105	115	127	112	115	117	105	102	108	95	83	71	67	71	68	76	93	94
97	109	120	115	104	94	84	84	75	61	45	47	57	82	101	102	116	110	106	110	110	106	109	97	92	77	73	69	70	77	79	87
94	103	119	103	90	91	90	79	62	59	50	55	71	86	97	99	111	105	104	113	114	105	106	97	90	80	72	73	73	81	84	91
94	90	116	93	77	86	85	81	74	61	59	70	74	81	93	93	103	100	104	109	112	101	104	107	96	82	80	74	78	73	80	92
95	97	109	90	60	62	68	67	64	62	71	79	87	97	94	88	99	109	111	102	114	99	103	101	94	87	90	83	84	81	89	95
102	103	113	91	66	62	59	62	64	79	87	90	91	90	89	89	91	98	107	95	101	102	97	92	96	96	92	90	81	85	94	102
108	102	116	109	93	73	72	74	84	91	99	99	94	91	90	98	91	94	100	97	103	101	105	99	104	89	96	88	92	102	103	111
110	101	123	120	107	96	100	98	106	105	99	101	104	102	99	90	96	98	107	110	103	97	108	101	102	98	97	94	99	105	103	108
111	109	119	122	116	116	115	113	97	101	104	94	102	102	99	92	97	105	111	110	109	115	110	107	110	106	102	112	106	109	107	103
111	109	116	118	113	115	116	116	106	104	109	104	104	98	105	96	97	116	108	115	113	106	113	109	113	113	116	116	114	114	116	103
102	114	117	110	111	115	110	108	115	115	114	109	108	112	104	108	110	113	109	102	115	109	107	108	111	109	118	115	114	114	112	104
107	110	119	116	120	119	114	107	114	106	122	113	117	114	103	111	114	121	116	104	112	116	107	108	111	114	112	116	120	116	114	112
116	115	116	119	109	112	113	117	113	121	113	117	116	117	116	113	117	111	117	121	124	121	108	105	114	116	114	115	126	119	120	116

b) Data in (a) After Normalisation

par.nm

0.28	0.39	0.53	0.52	0.51	0.47	0.52	0.50	0.47	0.50	0.51	0.52	0.50	0.47	0.48	0.50	0.50	0.50	0.47	0.44	0.	0.47	0.49	0.47	0.46	0.45	0.47	0.48	0.47	0.43	0.44	0.49
0.46	0.44	0.49	0.50	0.48	0.50	0.51	0.51	0.45	0.47	0.48	0.53	0.52	0.48	0.47	0.53	0.50	0.49	0.47	0.48	0.	0.46	0.49	0.49	0.46	0.44	0.47	0.47	0.48	0.46	0.46	0.48
0.46	0.46	0.46	0.52	0.51	0.49	0.44	0.44	0.44	0.47	0.48	0.47	0.47	0.50	0.47	0.50	0.52	0.47	0.47	0.44	0.	0.47	0.49	0.46	0.47	0.44	0.44	0.47	0.47	0.46	0.47	0.49
0.48	0.45	0.49	0.49	0.50	0.50	0.42	0.42	0.44	0.47	0.43	0.45	0.44	0.47	0.52	0.48	0.49	0.51	0.47	0.49	0.	0.45	0.46	0.46	0.45	0.44	0.47	0.48	0.47	0.45	0.45	0.48
0.46	0.47	0.46	0.44	0.47	0.46	0.46	0.48	0.48	0.47	0.44	0.47	0.51	0.50	0.51	0.46	0.46	0.48	0.47	0.51	0.	0.49	0.48	0.45	0.49	0.47	0.45	0.46	0.46	0.44	0.41	0.45
0.45	0.44	0.49	0.46	0.47	0.45	0.47	0.47	0.50	0.50	0.46	0.48	0.50	0.47	0.50	0.48	0.45	0.49	0.48	0.49	0.	0.46	0.48	0.46	0.45	0.46	0.47	0.48	0.47	0.47	0.43	0.45
0.45	0.46	0.47	0.46	0.46	0.45	0.45	0.45	0.50	0.49	0.50	0.48	0.47	0.48	0.51	0.49	0.46	0.47	0.46	0.42	0.	0.39	0.42	0.41	0.47	0.47	0.49	0.50	0.51	0.46	0.42	0.45
0.45	0.49	0.47	0.45	0.47	0.49	0.47	0.45	0.50	0.46	0.46	0.48	0.50	0.51	0.48	0.51	0.42	0.38	0.36	0.31	0.	0.31	0.29	0.31	0.38	0.47	0.50	0.48	0.49	0.45	0.46	0.45
0.46	0.45	0.49	0.46	0.46	0.46	0.47	0.47	0.46	0.48	0.44	0.43	0.47	0.47	0.46	0.42	0.35	0.29	0.28	0.26	0.	0.28	0.27	0.27	0.31	0.37	0.45	0.49	0.47	0.45	0.45	0.46
0.42	0.45	0.50	0.47	0.45	0.47	0.47	0.47	0.45	0.46	0.45	0.45	0.46	0.43	0.39	0.36	0.28	0.27	0.28	0.29	0.	0.33	0.35	0.34	0.29	0.32	0.42	0.47	0.46	0.45	0.44	0.45
0.44	0.44	0.47	0.44	0.44	0.45	0.46	0.46	0.46	0.47	0.44	0.47	0.44	0.38	0.28	0.25	0.26	0.29	0.27	0.25	0.	0.31	0.35	0.34	0.29	0.26	0.37	0.43	0.45	0.43	0.43	0.45
0.44	0.46	0.45	0.42	0.43	0.46	0.48	0.50	0.47	0.48	0.47	0.45	0.37	0.29	0.20	0.21	0.24	0.27	0.26	0.26	0.	0.26	0.29	0.28	0.25	0.25	0.35	0.45	0.43	0.44	0.43	0.47
0.43	0.44	0.49	0.46	0.45	0.49	0.48	0.49	0.46	0.46	0.46	0.39	0.29	0.19	0.18	0.20	0.22	0.26	0.27	0.24	0.	0.26	0.24	0.23	0.23	0.28	0.36	0.43	0.48	0.46	0.41	0.46
0.43	0.44	0.46	0.46	0.46	0.52	0.50	0.47	0.46	0.48	0.43	0.32	0.25	0.20	0.18	0.18	0.19	0.20	0.19	0.19	0.	0.27	0.26	0.22	0.20	0.24	0.32	0.42	0.46	0.44	0.43	0.45
0.42	0.45	0.48	0.46	0.50	0.47	0.49	0.45	0.47	0.47	0.39	0.28	0.20	0.22	0.19	0.17	0.17	0.18	0.15	0.17	0.	0.20	0.22	0.24	0.22	0.23	0.30	0.36	0.43	0.42	0.42	0.47
0.44	0.42	0.50	0.48	0.49	0.48	0.50	0.49	0.47	0.44	0.33	0.23	0.16	0.14	0.16	0.15	0.17	0.23	0.22	0.22	0.	0.20	0.19	0.20	0.23	0.23	0.28	0.31	0.40	0.40	0.44	0.43
0.45	0.47	0.49	0.49	0.49	0.47	0.49	0.48	0.50	0.43	0.32	0.20	0.16	0.16	0.19	0.23	0.34	0.37	0.40	0.36	0.	0.29	0.24	0.24	0.24	0.23	0.24	0.29	0.32	0.36	0.40	0.42
0.42	0.44	0.49	0.48	0.48	0.47	0.47	0.46	0.49	0.39	0.30	0.18	0.15	0.18	0.26	0.40	0.44	0.44	0.44	0.44	0.	0.40	0.35	0.25	0.24	0.26	0.24	0.29	0.31	0.33	0.42	0.42
0.41	0.44	0.47	0.48	0.49	0.49	0.47	0.45	0.44	0.40	0.27	0.17	0.15	0.24	0.35	0.45	0.45	0.45	0.45	0.44	0.	0.46	0.42	0.33	0.29	0.27	0.27	0.27	0.26	0.31	0.36	0.42
0.41	0.44	0.46	0.48	0.46	0.42	0.37	0.39	0.36	0.33	0.23	0.18	0.18	0.31	0.41	0.45	0.50	0.44	0.45	0.46	0.	0.40	0.42	0.37	0.33	0.28	0.26	0.28	0.27	0.30	0.36	0.37
0.38	0.43	0.47	0.45	0.41	0.37	0.33	0.33	0.29	0.24	0.18	0.18	0.22	0.32	0.40	0.40	0.45	0.43	0.42	0.43	0.	0.42	0.43	0.38	0.36	0.30	0.29	0.27	0.27	0.30	0.31	0.34
0.37	0.40	0.47	0.40	0.35	0.36	0.35	0.31	0.24	0.23	0.20	0.22	0.28	0.34	0.38	0.39	0.44	0.41	0.41	0.44	0.	0.41	0.42	0.38	0.35	0.31	0.28	0.29	0.29	0.32	0.33	0.36
0.37	0.35	0.45	0.36	0.30	0.34	0.33	0.32	0.29	0.24	0.23	0.27	0.29	0.32	0.36	0.36	0.40	0.39	0.41	0.43	0.	0.40	0.41	0.42	0.38	0.32	0.31	0.29	0.31	0.29	0.31	0.36
0.37	0.38	0.43	0.35	0.24	0.24	0.27	0.26	0.25	0.24	0.28	0.31	0.34	0.38	0.37	0.35	0.39	0.43	0.44	0.40	0.	0.39	0.40	0.40	0.37	0.34	0.35	0.33	0.33	0.32	0.35	0.37
0.40	0.40	0.44	0.36	0.26	0.24	0.23	0.24	0.25	0.31	0.34	0.35	0.36	0.35	0.35	0.35	0.36	0.38	0.42	0.37	0.	0.40	0.38	0.36	0.38	0.38	0.36	0.35	0.32	0.33	0.37	0.40
0.42	0.40	0.45	0.43	0.36	0.29	0.28	0.29	0.33	0.36	0.39	0.39	0.37	0.36	0.35	0.38	0.36	0.37	0.39	0.38	0.	0.40	0.41	0.39	0.41	0.35	0.38	0.35	0.36	0.40	0.40	0.44
0.43	0.40	0.48	0.47	0.42	0.38	0.39	0.38	0.42	0.41	0.39	0.40	0.41	0.40	0.39	0.35	0.38	0.38	0.42	0.43	0.	0.38	0.42	0.40	0.40	0.38	0.38	0.37	0.39	0.41	0.40	0.42
0.44	0.43	0.47	0.48	0.45	0.45	0.45	0.44	0.38	0.40	0.41	0.37	0.40	0.40	0.39	0.36	0.38	0.41	0.44	0.43	0.	0.45	0.43	0.42	0.43	0.42	0.40	0.44	0.42	0.43	0.42	0.40
0.44	0.43	0.45	0.46	0.44	0.45	0.45	0.45	0.42	0.41	0.43	0.41	0.41	0.38	0.41	0.38	0.38	0.45	0.42	0.45	0.	0.42	0.44	0.43	0.44	0.44	0.45	0.45	0.45	0.45	0.45	0.40
0.40	0.45	0.46	0.43	0.44	0.45	0.43	0.42	0.45	0.45	0.45	0.43	0.42	0.44	0.41	0.42	0.43	0.44	0.43	0.40	0.	0.43	0.42	0.42	0.44	0.43	0.46	0.45	0.45	0.45	0.44	0.41
0.42	0.43	0.47	0.45	0.47	0.47	0.45	0.42	0.45	0.42	0.48	0.44	0.46	0.45	0.40	0.44	0.45	0.47	0.45	0.41	0.	0.45	0.42	0.42	0.44	0.45	0.44	0.45	0.47	0.45	0.45	0.44
0.45	0.45	0.45	0.47	0.43	0.44	0.44	0.46	0.44	0.47	0.44	0.46	0.45	0.46	0.45	0.44	0.46	0.44	0.46	0.47	0.	0.47	0.42	0.41	0.45	0.45	0.45	0.45	0.49	0.47	0.47	0.45

c) Image Background

ima.pr

32	32	150	148	152	144	146	136	135	143	142	139	148	143	141	139	130	129	126	134	136	134	137	129	123	134	134	144	137	134	128	140
139	133	149	144	149	151	142	139	134	131	132	136	146	146	146	134	130	130	142	144	143	128	139	138	144	142	138	141	137	137	137	131
134	135	153	151	142	141	140	134	137	131	140	133	135	130	141	142	142	131	137	148	147	146	142	138	141	147	140	138	142	144	146	138
143	138	155	149	148	140	136	143	144	135	136	131	139	139	140	142	137	139	141	158	142	147	149	143	144	145	141	131	141	143	145	149
143	142	141	132	135	141	141	154	142	137	137	136	139	140	140	148	138	140	149	152	145	149	146	144	139	137	130	133	137	140	143	145
140	138	145	136	137	137	136	138	147	137	134	138	145	145	132	128	130	133	144	142	148	147	145	145	134	135	128	138	129	131	139	140
131	137	147	137	139	148	142	148	145	138	138	136	145	145	134	134	140	134	143	137	142	144	152	144	137	135	143	139	135	133	137	139
128	135	151	146	145	153	150	144	142	147	143	138	135	144	135	130	144	143	140	141	147	148	144	142	136	137	137	142	131	139	135	139
137	127	157	154	158	160	148	144	143	137	136	144	151	143	142	135	150	151	141	146	142	141	146	140	142	141	147	145	136	145	139	142
136	134	148	149	159	162	144	137	136	141	144	143	139	133	137	142	153	143	146	144	149	150	154	145	152	144	146	153	142	143	139	142
139	137	161	142	157	155	149	140	143	145	140	137	139	144	139	150	149	153	144	146	134	142	144	136	150	152	149	150	141	142	134	140
141	141	154	145	151	165	148	135	143	141	133	138	138	133	144	147	141	149	143	144	143	139	145	136	138	144	142	136	133	133	137	142
145	146	153	146	149	165	147	134	141	144	147	139	139	132	132	142	143	156	155	147	139	149	155	147	151	141	147	137	141	142	140	147
143	132	160	158	168	171	150	136	138	140	146	140	134	135	126	134	142	152	144	136	139	138	144	149	150	134	136	144	142	137	134	144
135	127	163	163	147	160	150	143	131	139	145	140	145	137	133	135	141	145	136	139	140	135	137	147	150	132	140	138	146	148	152	139
135	137	156	151	150	163	159	144	147	152	151	142	134	134	145	135	140	144	136	145	137	138	133	136	150	145	148	146	142	144	145	136
143	147	152	139	152	161	141	133	145	139	138	138	128	141	149	138	136	146	146	139	126	134	137	138	134	141	134	136	139	127	138	141
143	142	148	156	153	150	144	139	141	134	148	148	139	151	148	140	147	143	149	138	139	138	137	136	140	146	143	144	137	140	136	145
143	143	144	149	154	152	142	141	144	143	146	141	148	158	151	149	147	146	144	142	144	146	144	147	135	134	137	131	132	141	142	144
139	145	160	155	162	159	154	148	153	152	152	137	147	143	146	149	142	138	150	151	147	147	145	140	133	138	147	133	134	137	138	143
143	149	151	159	149	156	162	152	152	151	142	146	148	146	143	143	140	146	146	144	139	138	134	139	132	139	144	127	141	145	136	150
145	148	152	142	151	161	161	155	156	154	142	139	146	152	145	149	149	145	139	143	145	153	139	139	143	143	145	149	143	146	148	161
159	152	154	146	149	161	146	149	157	148	141	138	147	156	153	151	144	135	133	138	146	155	132	137	139	141	134	147	152	143	151	146
146	140	149	153	156	157	153	151	161	154	152	142	150	161	154	142	141	137	137	141	150	137	133	140	139	138	141	152	148	145	155	150
143	140	151	145	151	155	156	145	162	156	144	152	141	139	147	136	142	130	132	142	139	133	136	132	136	135	146	143	144	145	147	145
144	142	162	146	150	152	155	155	153	143	150	152	135	141	134	140	148	132	144	146	133	133	132	138	128	124	136	149	148	138	142	143
138	142	166	151	159	163	160	160	158	152	145	151	145	141	142	146	139	148	142	140	130	132	137	135	135	133	141	142	148	138	141	145
142	146	162	154	161	167	157	157	159	147	145	149	153	136	140	136	125	151	145	144	133	139	146	137	141	144	144	152	146	148	140	147
149	154	166	157	154	165	160	157	152	142	141	147	145	146	144	133	135	149	147	147	141	137	138	146	144	142	150	148	140	135	139	142
149	146	163	162	160	157	167	163	157	144	147	141	147	141	144	141	139	146	155	150	139	145	146	146	144	144	136	138	140	138	137	138
142	145	167	163	159	154	160	151	140	150	156	163	149	134	137	134	142	152	142	142	136	142	141	146	140	146	138	147	149	141	143	143
149	142	169	157	157	155	159	156	144	155	155	150	147	144	140	135	147	151	132	139	148	148	143	144	137	139	132	137	143	140	140	149

d) Data in (c) After Normalisation

ima.nm

0.13	0.13	0.59	0.58	0.60	0.56	0.57	0.53	0.53	0.56	0.56	0.	0.58	0.56	0.55	0.55	0.52	0.51	0.49	0.53	0.	0.53	0.54	0.51	0.48	0.53	0.53	0.56	0.54	0.53	0.50	0.55
0.55	0.52	0.58	0.56	0.58	0.59	0.56	0.55	0.53	0.51	0.52	0.	0.57	0.57	0.57	0.53	0.51	0.51	0.56	0.56	0.	0.50	0.55	0.54	0.56	0.56	0.54	0.55	0.54	0.54	0.54	0.51
0.53	0.53	0.60	0.59	0.56	0.55	0.55	0.53	0.54	0.51	0.55	0.	0.53	0.51	0.55	0.56	0.56	0.51	0.54	0.58	0.	0.57	0.56	0.54	0.55	0.58	0.55	0.54	0.56	0.56	0.57	0.54
0.56	0.54	0.61	0.58	0.58	0.55	0.53	0.56	0.56	0.53	0.53	0.	0.55	0.55	0.55	0.56	0.54	0.55	0.55	0.62	0.	0.58	0.58	0.56	0.56	0.57	0.55	0.51	0.55	0.56	0.57	0.58
0.56	0.56	0.55	0.52	0.53	0.55	0.55	0.60	0.56	0.54	0.54	0.	0.55	0.55	0.55	0.58	0.54	0.55	0.58	0.60	0.	0.58	0.57	0.56	0.55	0.54	0.51	0.52	0.54	0.55	0.56	0.57
0.55	0.54	0.57	0.53	0.54	0.54	0.53	0.54	0.58	0.54	0.53	0.	0.57	0.57	0.52	0.50	0.51	0.52	0.56	0.56	0.	0.58	0.57	0.57	0.53	0.53	0.50	0.54	0.51	0.51	0.55	0.55
0.51	0.54	0.58	0.54	0.55	0.58	0.56	0.58	0.57	0.54	0.54	0.	0.57	0.57	0.53	0.53	0.55	0.53	0.56	0.54	0.	0.56	0.60	0.56	0.54	0.53	0.56	0.55	0.53	0.52	0.54	0.55
0.50	0.53	0.59	0.57	0.57	0.60	0.59	0.56	0.56	0.58	0.56	0.	0.53	0.56	0.53	0.51	0.56	0.56	0.55	0.55	0.	0.58	0.56	0.56	0.53	0.54	0.54	0.56	0.51	0.55	0.53	0.55
0.54	0.50	0.62	0.60	0.62	0.63	0.58	0.56	0.56	0.54	0.53	0.	0.59	0.56	0.56	0.53	0.59	0.59	0.55	0.57	0.	0.55	0.57	0.55	0.56	0.55	0.58	0.57	0.53	0.57	0.55	0.56
0.53	0.53	0.58	0.58	0.62	0.64	0.56	0.54	0.53	0.55	0.56	0.	0.55	0.52	0.54	0.56	0.60	0.56	0.57	0.56	0.	0.59	0.60	0.57	0.60	0.56	0.57	0.60	0.56	0.56	0.55	0.56
0.55	0.54	0.63	0.56	0.62	0.61	0.58	0.55	0.56	0.57	0.55	0.	0.55	0.56	0.55	0.59	0.58	0.60	0.56	0.57	0.	0.56	0.56	0.53	0.59	0.60	0.58	0.59	0.55	0.56	0.53	0.55
0.55	0.55	0.60	0.57	0.59	0.65	0.58	0.53	0.56	0.55	0.52	0.	0.54	0.52	0.56	0.58	0.55	0.58	0.56	0.56	0.	0.55	0.57	0.53	0.54	0.56	0.56	0.53	0.52	0.52	0.54	0.56
0.57	0.57	0.60	0.57	0.58	0.65	0.58	0.53	0.55	0.56	0.58	0.	0.55	0.52	0.52	0.56	0.56	0.61	0.61	0.58	0.	0.58	0.61	0.58	0.59	0.55	0.58	0.54	0.55	0.56	0.55	0.58
0.56	0.52	0.63	0.62	0.66	0.67	0.59	0.53	0.54	0.55	0.57	0.	0.53	0.53	0.49	0.53	0.56	0.60	0.56	0.53	0.	0.54	0.56	0.58	0.59	0.53	0.53	0.56	0.56	0.54	0.53	0.56
0.53	0.50	0.64	0.64	0.58	0.63	0.59	0.56	0.51	0.55	0.57	0.	0.57	0.54	0.52	0.53	0.55	0.57	0.53	0.55	0.	0.53	0.54	0.58	0.59	0.52	0.55	0.54	0.57	0.58	0.60	0.55
0.53	0.54	0.61	0.59	0.59	0.64	0.62	0.56	0.58	0.60	0.59	0.	0.53	0.53	0.57	0.53	0.55	0.56	0.53	0.57	0.	0.54	0.52	0.53	0.59	0.57	0.58	0.57	0.56	0.56	0.57	0.53
0.56	0.58	0.60	0.55	0.60	0.63	0.55	0.52	0.57	0.55	0.54	0.	0.50	0.55	0.58	0.54	0.53	0.57	0.57	0.55	0.	0.53	0.54	0.54	0.53	0.55	0.53	0.53	0.55	0.50	0.54	0.55
0.56	0.56	0.58	0.61	0.60	0.59	0.56	0.55	0.55	0.53	0.58	0.	0.55	0.59	0.58	0.55	0.58	0.56	0.58	0.54	0.	0.54	0.54	0.53	0.55	0.57	0.56	0.56	0.54	0.55	0.53	0.57
0.56	0.56	0.56	0.58	0.60	0.60	0.56	0.55	0.56	0.56	0.57	0.	0.58	0.62	0.59	0.58	0.58	0.57	0.56	0.56	0.	0.57	0.56	0.58	0.53	0.53	0.54	0.51	0.52	0.55	0.56	0.56
0.55	0.57	0.63	0.61	0.64	0.62	0.60	0.58	0.60	0.60	0.60	0.	0.58	0.56	0.57	0.58	0.56	0.54	0.59	0.59	0.	0.58	0.57	0.55	0.52	0.54	0.58	0.52	0.53	0.54	0.54	0.56
0.56	0.58	0.59	0.62	0.58	0.61	0.64	0.60	0.60	0.59	0.56	0.	0.58	0.57	0.56	0.56	0.55	0.57	0.57	0.56	0.	0.54	0.53	0.55	0.52	0.55	0.56	0.50	0.55	0.57	0.53	0.59
0.57	0.58	0.60	0.56	0.59	0.63	0.63	0.61	0.61	0.60	0.56	0.	0.57	0.60	0.57	0.58	0.58	0.57	0.55	0.56	0.	0.60	0.55	0.55	0.56	0.56	0.57	0.58	0.56	0.57	0.58	0.63
0.62	0.60	0.60	0.57	0.58	0.63	0.57	0.58	0.62	0.58	0.55	0.	0.58	0.61	0.60	0.59	0.56	0.53	0.52	0.54	0.	0.61	0.52	0.54	0.55	0.55	0.53	0.58	0.60	0.56	0.59	0.57
0.57	0.55	0.58	0.60	0.61	0.62	0.60	0.59	0.63	0.60	0.60	0.	0.59	0.63	0.60	0.56	0.55	0.54	0.54	0.55	0.	0.54	0.52	0.55	0.55	0.54	0.55	0.60	0.58	0.57	0.61	0.59
0.56	0.55	0.59	0.57	0.59	0.61	0.61	0.57	0.64	0.62	0.56	0.	0.55	0.55	0.58	0.53	0.56	0.51	0.52	0.56	0.	0.52	0.53	0.52	0.53	0.53	0.57	0.56	0.56	0.57	0.58	0.57
0.56	0.56	0.64	0.57	0.59	0.60	0.61	0.61	0.60	0.56	0.59	0.	0.53	0.55	0.53	0.55	0.58	0.52	0.56	0.57	0.	0.52	0.52	0.54	0.50	0.49	0.53	0.58	0.58	0.54	0.56	0.56
0.54	0.56	0.65	0.59	0.62	0.64	0.63	0.63	0.62	0.60	0.57	0.	0.57	0.55	0.56	0.57	0.55	0.58	0.56	0.55	0.	0.52	0.54	0.53	0.53	0.52	0.55	0.56	0.58	0.54	0.55	0.57
0.56	0.57	0.64	0.60	0.63	0.65	0.62	0.62	0.62	0.58	0.57	0.	0.60	0.53	0.55	0.53	0.49	0.59	0.57	0.56	0.	0.55	0.57	0.54	0.55	0.56	0.56	0.60	0.57	0.58	0.55	0.58
0.58	0.60	0.65	0.62	0.60	0.65	0.63	0.62	0.60	0.56	0.55	0.	0.57	0.57	0.56	0.52	0.53	0.58	0.58	0.58	0.	0.54	0.54	0.57	0.56	0.56	0.59	0.58	0.55	0.53	0.55	0.56
0.58	0.57	0.64	0.64	0.63	0.62	0.65	0.64	0.62	0.56	0.58	0.	0.58	0.55	0.56	0.55	0.55	0.57	0.61	0.59	0.	0.57	0.57	0.57	0.56	0.56	0.53	0.54	0.55	0.54	0.54	0.54
0.56	0.57	0.65	0.64	0.62	0.60	0.63	0.59	0.55	0.59	0.61	0.	0.58	0.53	0.54	0.53	0.56	0.60	0.56	0.56	0.	0.56	0.55	0.57	0.55	0.57	0.54	0.58	0.58	0.55	0.56	0.56
0.58	0.56	0.66	0.62	0.62	0.61	0.62	0.61	0.56	0.61	0.61	0.	0.58	0.56	0.55	0.53	0.58	0.59	0.52	0.55	0.	0.58	0.56	0.56	0.54	0.55	0.52	0.54	0.56	0.55	0.55	0.58

e) Fish Muscle Background

mus.pr

104	163	89	88	87	81	87	86	83	84	81	80	78	81	78	77	81	77	80	78	73	74	77	78	74	71	72	78	76	79	80	74
76	74	85	85	80	82	89	90	82	83	78	78	78	81	81	73	75	80	83	84	76	75	79	73	72	75	75	73	77	80	80	80
78	74	84	82	83	83	83	87	75	79	77	75	77	81	77	81	79	76	80	83	86	76	69	71	76	74	80	77	81	80	75	74
80	82	87	80	77	84	91	89	77	78	78	72	77	83	78	78	78	76	81	86	84	79	78	81	83	78	75	79	82	74	73	77
77	79	91	89	80	82	88	83	85	76	86	72	73	76	76	77	74	76	77	80	81	84	83	86	86	79	75	75	72	71	78	71
71	77	91	90	78	77	77	85	77	76	76	70	67	77	82	91	80	78	83	82	84	90	86	80	83	77	75	76	71	73	79	72
72	80	90	94	81	74	78	81	81	87	77	77	75	78	82	84	82	78	87	84	82	85	85	79	80	80	77	80	74	71	68	69
76	75	87	90	92	77	79	80	80	82	77	78	79	81	75	74	78	80	79	77	79	81	78	78	68	65	71	78	75	72	73	76
65	72	85	85	84	77	86	84	79	81	79	73	73	77	76	73	77	78	76	77	83	79	80	77	79	71	70	75	78	75	82	79
74	71	74	80	80	75	75	77	78	77	77	76	71	75	68	77	76	82	81	79	85	76	78	76	75	78	75	74	78	77	76	73
68	72	77	76	76	74	73	82	83	73	78	73	72	76	75	77	78	78	86	85	83	84	80	75	69	73	79	77	75	80	77	70
76	72	91	80	72	72	82	84	86	80	78	76	79	76	77	75	82	85	83	82	86	78	80	74	69	73	79	83	78	81	75	75
71	73	86	86	73	79	83	86	80	77	75	75	74	75	74	75	80	89	79	80	79	78	76	74	76	83	86	78	78	76	74	78
71	72	80	85	80	81	77	89	81	81	76	74	72	77	79	79	78	80	74	79	79	79	78	78	84	78	80	76	73	73	80	75
72	70	79	83	81	80	80	84	78	75	72	76	79	78	82	79	80	73	78	72	78	75	73	71	80	83	83	82	80	79	78	77
72	75	77	79	76	77	80	81	76	72	72	76	72	82	90	79	80	77	77	75	78	75	77	72	81	84	78	77	76	73	70	76
73	74	82	89	80	73	79	74	84	74	74	72	79	80	84	78	75	79	76	85	82	76	73	78	77	82	74	72	72	73	68	72
72	74	86	79	76	79	78	73	83	81	73	70	73	77	74	78	78	79	75	85	83	77	74	71	78	74	69	70	67	74	66	67
69	72	86	75	78	75	83	85	79	77	78	72	72	74	75	79	83	74	77	81	80	74	81	79	75	77	78	69	68	72	69	73
71	71	92	80	71	74	82	84	82	76	75	74	74	83	78	78	84	79	86	79	82	73	84	80	79	79	73	73	68	69	70	70
74	75	85	75	72	72	76	79	77	75	68	75	80	75	72	78	80	80	78	79	74	69	81	82	74	84	77	69	74	69	68	68
70	73	88	87	79	76	78	83	78	80	70	73	75	79	77	86	85	75	72	76	75	75	84	85	81	85	77	76	72	67	74	71
78	70	83	81	74	78	79	86	80	74	76	73	77	76	75	77	74	75	74	74	75	73	76	84	80	81	77	74	72	66	66	64
75	71	88	82	70	76	84	83	75	79	75	74	75	69	73	70	71	73	69	76	67	71	80	84	83	78	72	77	73	63	68	73
71	72	83	80	71	75	69	72	77	78	69	70	76	70	66	72	69	74	74	74	76	71	75	72	75	74	66	71	73	66	72	67
69	74	89	83	78	76	73	72	79	76	79	71	81	73	71	74	70	71	69	68	73	73	75	76	70	75	75	75	70	69	71	70
69	69	76	78	76	78	82	72	86	75	78	74	75	78	75	79	75	81	76	71	71	67	67	72	66	75	78	77	73	70	67	67
68	65	72	80	81	82	76	80	77	77	74	75	77	74	73	79	74	78	75	77	75	75	74	75	71	77	73	71	71	66	69	61
63	69	72	80	80	75	81	81	77	74	80	77	75	74	72	76	71	76	78	81	75	71	72	74	72	71	75	73	78	74	71	69
66	64	73	72	79	69	73	73	72	75	78	72	63	70	74	74	73	84	69	73	78	72	66	68	69	74	73	71	69	72	70	69
68	68	75	72	74	78	71	71	68	68	72	71	71	72	70	73	75	77	76	76	80	80	77	76	71	75	70	70	67	74	77	72
71	72	71	80	78	75	66	66	64	62	70	66	68	67	74	76	73	75	71	71	76	72	74	73	77	73	67	72	64	73	72	72

f) Data in (e) After Normalisation

mus.nm

0.41	0.64	0.35	0.	0.34	0.32	0.34	0.34	0.33	0.33	0.32	0.31	0.31	0.32	0.31	0.30	0.32	0.30	0.31	0.31	0.29	0.29	0.30	0.31	0.29	0.28	0.28	0.31	0.30	0.31	0.31	0.29
0.30	0.29	0.33	0.	0.31	0.32	0.35	0.35	0.32	0.33	0.31	0.31	0.31	0.32	0.32	0.29	0.29	0.31	0.33	0.33	0.30	0.29	0.31	0.29	0.28	0.29	0.29	0.29	0.30	0.31	0.31	0.31
0.31	0.29	0.33	0.	0.33	0.33	0.33	0.34	0.29	0.31	0.30	0.29	0.30	0.32	0.30	0.32	0.31	0.30	0.31	0.33	0.34	0.30	0.27	0.28	0.30	0.29	0.31	0.30	0.32	0.31	0.29	0.29
0.31	0.32	0.34	0.	0.30	0.33	0.36	0.35	0.30	0.31	0.31	0.28	0.30	0.33	0.31	0.31	0.31	0.30	0.32	0.34	0.33	0.31	0.31	0.32	0.33	0.31	0.29	0.31	0.32	0.29	0.29	0.30
0.30	0.31	0.36	0.	0.31	0.32	0.35	0.33	0.33	0.30	0.34	0.28	0.29	0.30	0.30	0.30	0.29	0.30	0.30	0.31	0.32	0.33	0.33	0.34	0.34	0.31	0.29	0.29	0.28	0.28	0.31	0.28
0.28	0.30	0.36	0.	0.31	0.30	0.30	0.33	0.30	0.30	0.30	0.27	0.26	0.30	0.32	0.36	0.31	0.31	0.33	0.32	0.33	0.35	0.34	0.31	0.33	0.30	0.29	0.30	0.28	0.29	0.31	0.28
0.28	0.31	0.35	0.	0.32	0.29	0.31	0.32	0.32	0.34	0.30	0.30	0.29	0.31	0.32	0.33	0.32	0.31	0.34	0.33	0.32	0.33	0.33	0.31	0.31	0.31	0.30	0.31	0.29	0.28	0.27	0.27
0.30	0.29	0.34	0.	0.36	0.30	0.31	0.31	0.31	0.32	0.30	0.31	0.31	0.32	0.29	0.29	0.31	0.31	0.31	0.30	0.31	0.32	0.31	0.31	0.27	0.25	0.28	0.31	0.29	0.28	0.29	0.30
0.25	0.28	0.33	0.	0.33	0.30	0.34	0.33	0.31	0.32	0.31	0.29	0.29	0.30	0.30	0.29	0.30	0.31	0.30	0.30	0.33	0.31	0.31	0.30	0.31	0.28	0.27	0.29	0.31	0.29	0.32	0.31
0.29	0.28	0.29	0.	0.31	0.29	0.29	0.30	0.31	0.30	0.30	0.30	0.28	0.29	0.27	0.30	0.30	0.32	0.32	0.31	0.33	0.30	0.31	0.30	0.29	0.31	0.29	0.29	0.31	0.30	0.30	0.29
0.27	0.28	0.30	0.	0.30	0.29	0.29	0.32	0.33	0.29	0.31	0.29	0.28	0.30	0.29	0.30	0.31	0.31	0.34	0.33	0.33	0.33	0.31	0.29	0.27	0.29	0.31	0.30	0.29	0.31	0.30	0.27
0.30	0.28	0.36	0.	0.28	0.28	0.32	0.33	0.34	0.31	0.31	0.30	0.31	0.30	0.30	0.29	0.32	0.33	0.33	0.32	0.34	0.31	0.31	0.29	0.27	0.29	0.31	0.33	0.31	0.32	0.29	0.29
0.28	0.29	0.34	0.	0.29	0.31	0.33	0.35	0.31	0.30	0.29	0.29	0.29	0.29	0.29	0.29	0.31	0.35	0.31	0.31	0.31	0.31	0.30	0.29	0.30	0.33	0.34	0.31	0.31	0.30	0.29	0.31
0.28	0.28	0.31	0.	0.31	0.32	0.30	0.35	0.32	0.32	0.30	0.29	0.28	0.30	0.31	0.31	0.31	0.31	0.29	0.31	0.31	0.31	0.31	0.31	0.33	0.31	0.31	0.30	0.28	0.28	0.31	0.29
0.28	0.27	0.31	0.	0.32	0.31	0.31	0.33	0.31	0.29	0.28	0.30	0.31	0.31	0.32	0.31	0.31	0.29	0.31	0.28	0.31	0.29	0.29	0.28	0.31	0.33	0.33	0.32	0.31	0.31	0.31	0.30
0.28	0.29	0.30	0.	0.30	0.30	0.31	0.32	0.30	0.28	0.28	0.30	0.29	0.32	0.35	0.31	0.31	0.30	0.30	0.29	0.31	0.29	0.30	0.28	0.32	0.33	0.31	0.30	0.30	0.29	0.27	0.30
0.29	0.29	0.32	0.	0.31	0.29	0.31	0.29	0.33	0.29	0.29	0.28	0.31	0.31	0.33	0.31	0.29	0.31	0.30	0.33	0.32	0.30	0.29	0.31	0.30	0.32	0.29	0.28	0.28	0.29	0.27	0.28
0.28	0.29	0.34	0.	0.30	0.31	0.31	0.29	0.33	0.32	0.29	0.27	0.29	0.30	0.29	0.31	0.31	0.31	0.29	0.33	0.33	0.30	0.29	0.28	0.31	0.29	0.27	0.27	0.26	0.29	0.26	0.26
0.27	0.28	0.34	0.	0.31	0.29	0.33	0.33	0.31	0.30	0.31	0.28	0.28	0.29	0.29	0.31	0.33	0.29	0.30	0.32	0.31	0.29	0.32	0.31	0.29	0.30	0.31	0.27	0.27	0.28	0.27	0.29
0.28	0.28	0.36	0.	0.28	0.29	0.32	0.33	0.32	0.30	0.29	0.29	0.29	0.33	0.31	0.31	0.33	0.31	0.34	0.31	0.32	0.29	0.33	0.31	0.31	0.31	0.29	0.29	0.27	0.27	0.27	0.27
0.29	0.29	0.33	0.	0.28	0.28	0.30	0.31	0.30	0.29	0.27	0.29	0.31	0.29	0.28	0.31	0.31	0.31	0.31	0.31	0.29	0.27	0.32	0.32	0.29	0.33	0.30	0.27	0.28	0.27	0.27	0.27
0.27	0.29	0.35	0.	0.31	0.30	0.31	0.33	0.31	0.31	0.27	0.29	0.29	0.31	0.30	0.34	0.33	0.29	0.28	0.30	0.29	0.29	0.33	0.33	0.32	0.33	0.30	0.30	0.28	0.26	0.29	0.28
0.31	0.27	0.33	0.	0.29	0.31	0.31	0.34	0.31	0.29	0.30	0.29	0.30	0.30	0.29	0.30	0.29	0.29	0.29	0.29	0.29	0.29	0.30	0.33	0.31	0.32	0.30	0.29	0.28	0.26	0.26	0.25
0.29	0.28	0.35	0.	0.27	0.30	0.33	0.33	0.29	0.31	0.29	0.29	0.29	0.27	0.29	0.27	0.28	0.29	0.27	0.30	0.26	0.28	0.31	0.33	0.33	0.31	0.28	0.30	0.29	0.25	0.27	0.29
0.28	0.28	0.33	0.	0.28	0.29	0.27	0.28	0.30	0.31	0.27	0.27	0.30	0.27	0.26	0.28	0.27	0.29	0.29	0.29	0.30	0.28	0.29	0.28	0.29	0.29	0.26	0.28	0.29	0.26	0.28	0.26
0.27	0.29	0.35	0.	0.31	0.30	0.29	0.28	0.31	0.30	0.31	0.28	0.32	0.29	0.28	0.29	0.27	0.28	0.27	0.27	0.29	0.29	0.29	0.30	0.27	0.29	0.29	0.29	0.27	0.27	0.28	0.27
0.27	0.27	0.30	0.	0.30	0.31	0.32	0.28	0.34	0.29	0.31	0.29	0.29	0.31	0.29	0.31	0.29	0.32	0.30	0.28	0.28	0.26	0.26	0.28	0.26	0.29	0.31	0.30	0.29	0.27	0.26	0.26
0.27	0.25	0.28	0.	0.32	0.32	0.30	0.31	0.30	0.30	0.29	0.29	0.30	0.29	0.29	0.31	0.29	0.31	0.29	0.30	0.29	0.29	0.29	0.29	0.28	0.30	0.29	0.28	0.28	0.27	0.27	0.24
0.25	0.27	0.28	0.	0.31	0.29	0.32	0.32	0.30	0.29	0.31	0.30	0.29	0.29	0.28	0.30	0.28	0.30	0.31	0.32	0.29	0.28	0.28	0.29	0.28	0.28	0.29	0.29	0.31	0.29	0.28	0.27
0.26	0.25	0.29	0.	0.31	0.27	0.29	0.29	0.28	0.29	0.31	0.28	0.25	0.27	0.29	0.29	0.29	0.33	0.27	0.29	0.31	0.28	0.26	0.27	0.27	0.29	0.29	0.28	0.27	0.28	0.27	0.27
0.27	0.27	0.29	0.	0.29	0.31	0.28	0.28	0.27	0.27	0.28	0.28	0.28	0.28	0.27	0.29	0.29	0.30	0.30	0.30	0.31	0.31	0.30	0.30	0.28	0.29	0.27	0.27	0.26	0.29	0.30	0.28
0.28	0.28	0.28	0.	0.31	0.29	0.26	0.26	0.25	0.24	0.27	0.26	0.27	0.26	0.29	0.30	0.29	0.29	0.28	0.28	0.30	0.28	0.29	0.29	0.30	0.29	0.26	0.28	0.25	0.29	0.28	0.28

2. Input Data of the Normalised Feature Extraction Format

a) Parasitic Background

```
par.fe
0.03 0.04 0.08 0.09 0.09 0.05 0.09 0.15 0.17 0.14 0.16 0.25 0.14 0.15 0.17 0.13 0.13 0.28 0.27 0.24 0.26 0.40 0.38 0.64 0.71 0.99 1.00 0.76 0.45 0.46 0.12 0.07
```

b) Image Background

```
ima.fe
0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.13 0.49 0.85 1.00 0.74 0.40 0.22 0.14 0.06 0.02
```

c) Fish Muscle Background

```
mus.fe
0.03 0.12 0.34 0.78 1.00 0.96 0.45 0.29 0.09 0.06 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

3. Input Data of the Normalised Curve Map

a) Parasitic Background

```
par.cm
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.99 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.07
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.76 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.71 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.64 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.46 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.40 0.38 0.00 0.00 0.00 0.00 0.00 0.45 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.28 0.27 0.00 0.26 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.15 0.17 0.00 0.16 0.25 0.00 0.00 0.17 0.00 0.00 0.00 0.00 0.24 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.08 0.09 0.09 0.05 0.09 0.00 0.00 0.14 0.00 0.00 0.14 0.15 0.00 0.13 0.13 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.12 0.07
0.03 0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

b) Image Background

```
ima.cm
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.85 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.74 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.49 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.40 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.22 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.13 0.00 0.00 0.00 0.00 0.00 0.00 0.14 0.06 0.00
0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.02
```

c) Fish Muscle Background

```
mus.cm
0.00 0.00 0.00 0.00 1.00 0.96 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.78 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.45 0.00 0.00 0.06 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.34 0.00 0.00 0.00 0.00 0.29 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.12 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

4. Input Data of the Normalised Binarised Curve Map

a) Parasitic Background

```
par.bc
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00 1.00 1.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 1.00 0.00 0.00 1.00 1.00 0.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00
1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

b) Image Background

```
ima.bc
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
```

c) Fish Muscle Background

```
mus.bc
0.00 0.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 1.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
```

5. Input Data of the Normalised Smoothed Curve Format

a) Parasitic Background

```
par.sc
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.10 0.10 0.10 0.10 0.20 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.20 0.20 0.20 0.20 0.30 0.30 0.40 0.50 0.60 0.60 0.50 0.40 0.40 0.30 0.20
```

b) Image Background

```
ima.sc
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.10 0.20 0.30 0.40 0.30 0.20 0.10 0.00 -0.10 -0.10
```

c) Fish Muscle Background

```
mus.sc
0.00 0.10 0.20 0.30 0.40 0.40 0.30 0.20 0.10 0.10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

APPENDIX III

SOURCE CODE LISTINGS

1. **Processing Programs**

   a) Rannet.c - neural net program using the steepest descent optimisation method.

   b) Ranopt.c - neural net program using the conjugate gradient optimisation method.

2. **Production Programs**

   a) Wortest.c - classification program for the pixel representatiom format.

   b) Wavtest.c - classification program for the feature extraction format.

   c) Histest.c - classification program for the curve map/binarised curve map format.

   d) Smthtest.c - classification program for the smoothed curve format.

```
/****************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the neural net program that implements
*  the steepest descent optimization methods. It contains the
*  routines for the off-line processing and the on-line production
*  of the network.
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
****************************************************************/


 /***********************
 * RANNET.C:   This neural net simulator implements the steepest
 *             descent optimization method. It contains all the
 *             necessary routines for creating the network,
 *             training the network with image patterns and
 *             classifying the image patterns. It is implemented
 *             as a classifier.
 *****************************/

#define PC    /* define if using PC */
#define UNIX  /* define if using UNIX system */
              /* include the necessary libraries */
#ifdef PC
 #include <stdlib.h>
 #include <string.h>
 #include <conio.h>
 #include <alloc.h>
#endif

#ifdef UNIX
 #include <curses.h>
#endif

#include <stdio.h>
#include <math.h>

#define ERRORLIMIT 0.001

static float *(*L);         /* for the layers and each layer nodes */
static float *(*input);     /* for the inputs */
static float *(*output);    /* for the outputs */
static float *(*(*W));      /* for the connecting weights between the nodes */
static float *(*(*Wtmp)); /* the temp weight holder */
static float *D;            /* the array of actual or expected outputs */
static int *N;              /* the number of nodes in each layer */
static float gterm = 0.01;        /* the gain term */
static float gtermrate = 0.01;  /* the rate of increase of the gain term */
static float maxgterm = 0.7;      /* the max gain term */
static float alpha = 0.0;         /* the momemtum factor */
static float maxalpha = 0.2;      /* the max momemtum factor */
static float corre;         /* the correlation term */
static float prcorre = 0.0;       /* the previous correlation */
static float tsserror;       /* the sum of square error term */
static int layers = 0;      /* the number of layers */
static int mlayer = 0;      /* the max number of nodes in the network */
static int p = 0;           /* the number of patterns */
static float limit = 0.0;  /* the error limit */
static int sweeps = 0;
static count;               /* number of epochs */
```

```
double random_number();    /* the random number generator */
void displayweights();    /* displays the weights and outputs */
void save();

/**********************
 * create():     this routine creats the network topology with the connecting
 *               edges assigned with information from the network topology file.
 ********************************/

void create(fnet)
char fnet[15];
{
 FILE *fn;
 char line[15];
 int i, j;

 if (( fn = fopen(fnet,"r")) == NULL )
  {
   printf("\nCannot open the file %s.",fnet);
   exit(0);
  }

 fscanf( fn,"%s",line );    /* read off the title of the file */
 fscanf( fn,"%d",&layers );  /* read the number of layers less one */

      /* allocate memory for the layers */
 L = (float *(*))calloc(layers+1,sizeof(float *));
 N = (int *)calloc(layers+1,sizeof(int));

 if ( !L )
  {
   printf("\nNot enough memory for layers.");
   exit(0);
  }

     /* get the number of nodes for each layer and allocate memory */
 for( i=0; i<=layers; i++ )
   {
    fscanf( fn,"%d",&N[i] );

    if ( N[i] > mlayer )
      mlayer = N[i];

    L[i] = (float *)calloc(N[i],sizeof(float ));
    if ( !L[i] )
     {
      printf("\nNot enough memory for layer %d.",i+1);
      exit(0);
     }
    } /* end of the for */

  fclose(fn);

 /* allocate memory for the expected outputs */
 D = (float *)calloc(N[layers], sizeof(float));

 /* allocate memory for the connecting weights between the nodes(units) */
 W = (float *(*(*)))calloc(layers, sizeof(float *(*)));
 Wtmp = (float *(*(*)))calloc(layers, sizeof(float *(*)));
 if ( !W || !Wtmp )
  {
   printf("\nNot enough memory for the weights.");
   exit(0);
  }
```

```
 for( i=0; i<layers; i++ )   /* for the nodes in each layer */
   {
    W[i] = (float *(*))calloc(N[i]+1, sizeof(float *));
    Wtmp[i] = (float *(*))calloc(N[i]+1, sizeof(float *));
    if ( !W[i] || !Wtmp )
     {
      printf("\nNot enough memory for layer %d weights.",i+1);
      exit(0);
     }
    for( j=0; j<=N[i]; j++ )  /* weights per each node */
      {
       W[i][j] = (float *)calloc(N[i+1], sizeof(float));
       Wtmp[i][j] = (float *)calloc(N[i+1], sizeof(float));
       if ( !W[i][j] || !Wtmp[i][j] )
        {
         printf("\nNot enough memory for node %d of layer %d weights.",j,i+1);
         exit(0);
        }
      } /* end of the second for */
   }  /* end of the first for loop */

}  /* end of create */

/************************
 * w_range():    this routine initialises the network by assigning random values
 *               to the connecting weights or reading the weights from a file.
 ************************************/

void w_range(fwdir)
char fwdir[15];
{
 FILE *fw, *fw1;
 char ch, line[20], fname[15];
 int l, i, j;
 float range;

 if ( (fw=fopen(fwdir,"r")) == NULL)
  {
   printf("\nCannot open file %s",fwdir);
   exit(0);
  }

  fscanf(fw,"%s",line);  /* read off the title */
  fscanf(fw,"%c",&ch);    /* read the space off */
  fscanf(fw,"%c",&ch);    /* read the option */
  fscanf(fw,"%s",fname); /* read the range or file name */

  if ( ch == 'r' )
   {

   for( l=0; l<layers; l++ )
    {
     range = sqrt( (double)(N[l]+1) )/3;
     for( i=0; i<=N[l]; i++ )
       for( j=0; j<N[l+1]; j++ )
        {
         W[l][i][j] = (2.0 * random_number() - 1.)/range ;
         Wtmp[l][i][j] = W[l][i][j];
        }
     } /* end of the first for */
  } /* end of the if */
 else if ( ch == 'f' )
    {
     if (( fw1 = fopen(fname,"r") ) == NULL )
      {
```

```
      printf("\nCannot open file %s.",fname);
      exit(0);
     } /* end of the if */
    else
       {
        fscanf( fw,"%s",line);    /* read off the file name */
        for( l=0; l<layers; l++ )
          for( i=0; i<=N[l]; i++ )
            for( j=0; j<N[l+1]; j++ )
             {
              fscanf( fw,"%f",&W[l][i][j] );
              Wtmp[l][i][j] = W[l][i][j];
             }
       }  /* end of the second else */
   fclose(fw);
   fclose(fw1);
   }  /* end of the first else */
 else
   {
    printf("\nOption has to be r or f not %c",ch);
    exit(0);
   }
 fflush(stdin);
} /* end of w_range */

/*************************
 * random_number():      this routine generates the random numbers.
 *************************************/

double random_number()
{
 static double x = 3141592654.0;
        double a = 3141592654.0;
        double c = 2718281828.0;
        double m = 1.0e+10;

 x = fmod(a*x+c, m);
 return(x/m);

}  /* end of random_number */

/**********************
 * loadtrainset():               this routine loads the training set comprising of
 *                       the input data and the desired outputs.
 ****************************/

void loadtrainset(ftrain)
char ftrain[15];
{
 FILE *ft;
 char ch, line[20];
 int err = 0, i, j;

 if (( ft = fopen( ftrain, "r" )) == NULL )
  {
   printf("\nFile %s cannot be opened. ", ftrain);
   exit(0);
  }
 else
    {
     fscanf(ft,"%s",line);   /* read off the title */
     fscanf(ft,"%d",&p);      /* read the number of patterns */
     fscanf(ft,"%c",&ch);     /* read the space off */
     fscanf(ft,"%c",&ch);     /* read the option for convergence */
     if ( ch == 'e' )
```

```
      fscanf(ft,"%f",&limit);  /* read the error limit */
     else if ( ch == 's' )
           fscanf(ft,"%d",&sweeps);  /* read the number of sweeps */
     else
        {
         printf("\nOption has to be e or s not %c",ch);
         exit(0);
        }

     fscanf(ft,"%f",&gterm);  /* read learning the parameters */
     fscanf(ft,"%f",&gtermrate);
     fscanf(ft,"%f",&maxgterm);
     fscanf(ft,"%f",&alpha);
     fscanf(ft,"%f",&maxalpha);
                                  /* allocate memory */
         input = (float *(*))calloc(p,sizeof(float *));
         output = (float *(*))calloc(p,sizeof(float *));

         for( j=0; j<p; j++ )    /* loop through the patterns */
           {   /* load the input for a pattern from the file */
            input[j] = (float *)calloc(N[0],sizeof(float));
            for( i=0; i<N[0]; i++ )
              if ( fscanf( ft, "%f", &input[j][i] ) < 1 )
               err = 1;

             /* load the output from the file */
            output[j] = (float *)calloc(N[layers],sizeof(float));
            for( i=0; i<N[layers]; i++ )
              if ( fscanf( ft, "%f", &output[j][i] ) < 1 )
               err = 1;

            if ( err )
             {
              printf("\nError in input file!\n");
              break;
             }  /* end of if */
           } /* end of the for */
        fclose(ft);
        }  /* end of the second else */

    if ( err )  /* if any error is present then stop the program */
     {
      printf("\nError in training/input file!\n");
      exit(0);
     }

}  /* end of loadtrainset */

/*******************************
 * loadclass():  this routine loads the classification set.
 **********************************************/

void loadclass(fclass)
char fclass[15];
{
 FILE *fc;
 char line[20];
 int err = 0, i;

 if (( fc = fopen( fclass, "r" )) == NULL )
      {
       printf("\nFile %s cannot be opened. ", fclass);
       exit(0);
      }
     else
```

```
        {    /* load the classification data from the file */
         fscanf(fc, "%s", line);  /* read off the title */
         for( i=0; i<N[0]; i++ )
           if ( fscanf( fc, "%f", &L[0][i] ) < 1 )
           err = 1;

         fclose(fc);

        }  /* end of the second else */

    if ( err )  /* if any error is present then stop the program */
     {
      printf("\nError in production file!\n");
      exit(0);
     }

}  /* end of loadclass */

/******************************
 * changeparameters():   this routine allows a user to change the learning
 *                       parameters.
 ********************************************/

void changeparameters()
{
 char num[15];

 printf("\nPresent values of the learning parameters are as follows:");
 printf("\n\tGain term = %5.3f.",gterm);
 printf("\n\tGain term rate = %5.3f.",gtermrate);
 printf("\n\tMaximum gain term = %5.3f.",maxgterm);
 printf("\n\tMomemtum factor = %5.3f.", alpha);
 printf("\n\tMaximum momemtum factor = %5.3f.",maxalpha);
 printf("\n\nEnter the new gain term: ");
 gets(num);
 gterm = atof(num);

 do
  {
   printf("\nEnter the new maximum gain term, it has to be >= %5.3f:",gterm);
   gets(num);
   maxgterm = atof(num);
  } while ( maxgterm < gterm );

 do
  {
   printf("\nEnter the new gain term rate, it has to lie between 0.00 ");
   printf("\nand %5.3f.",maxgterm);
   gets(num);
   gtermrate = atof(num);
  } while ( gtermrate > maxgterm );

 printf("\nEnter the new momemtum factor:");
 gets(num);
 alpha = atof(num);

 do
  {
   printf("\nEnter the new maximum momemtum factor, it has to be >= %5.3f:",alpha);
   gets(num);
   maxalpha = atof(num);
  } while ( maxalpha < alpha );

} /* end of changeparameters */
```

```
/******************************
 * adjustparameters():   this routine uses the weight correlation to internally
 *                       adjusts the learning parameters.
 *******************************************/

void adjustparameters()
{
 int inc = 0, red = 0;  /* flags for adjusting the paramters */

   /* if weight change is small or bigger and error limit has been reached  */
 if ( ((corre <= 0.0000) || (corre > 0.0000)) && ( tsserror <= (ERRORLIMIT+(ERRORLIMIT*0.05))) )
  red = 1;  /* reduce the terms else if wt change is nil limit not reached */
 else if ( (corre <= 0.0000) && (tsserror > (ERRORLIMIT+(ERRORLIMIT*0.05))) )
       inc = 2; /* double the increment rate else if the wt change is bigger */
 else if ( (corre > 0.0000) && (tsserror > (ERRORLIMIT+(ERRORLIMIT*0.05))) )
       {  /* check if the wt change is decreasing */
        if ( corre < (prcorre + (prcorre*0.05)) )
         inc = 1; /* increase the terms moderately */
       } /* end of the else if */
 else    /* otherwise maintain the terms */
     inc = 1;
 if ( inc == 2 )
  {
   gterm +=  2*gtermrate;
   alpha +=  0.01;
  }
 else if ( inc == 1 )
       {
        gterm += gtermrate;
        alpha +=  0.005;
       }
 else if ( red == 1 )
       {
        gterm = 0.01;
        alpha = 0.0;
       }
 else
     ;

 if ( gterm > maxgterm )  /* reset the gterm if greater */
   gterm = maxgterm;

 if ( alpha > maxalpha )  /* reset the alpha term */
   alpha = maxalpha;
 prcorre = corre; /* save the correlation computed */

} /* end of adjustparameters */

/******************************
 * compute_out():                this routine computes the sigmoid logistic function.
 *******************************************/

void compute_out()
{
 int i, j, l;
 float sum = 0.0;
 float ulimit = -log(1E39);
 float llimit = -log(1E-38);

/* computation of outputs for the layer l+1 */
for( l=0; l<layers; l++ )
  for( j=0; j<N[l+1]; j++ )  /* scan the nodes of the next layer */
    {    /* let the first weight be the assumed threshold */
     sum = -W[l][0][j];
```

```
    for( i=0; i<N[l]; i++ )  /* compute the weighted inputs into a */
      sum += W[l][i+1][j] * L[l][i];    /* node in the next layer */

    if ( sum < ulimit )   /* check the limits of the sum */
     L[l+1][j] = 1.0;
    else if ( sum > llimit )
       L[l+1][j] = 0.0;
    else
       L[l+1][j] = 1.0/(1.0 + exp(-sum));  /* compute the new output */
   }                                    /* using a non-linearity function */
}  /* end of compute_out */

/******************************
 * adaptweights():               this routine adapts the weights if they have not stabilised.
 *********************************************/

void adaptweights( )
{
 int i, j, k, l;
 float y=0.0, tempwt=0.0;  /* temporary holders */
 float a, wtchg, prwtchg;
 float *delta, *delta1; /* pointers to the delta rasters */
 float *temp;
 float sum;       /* for computing the deltas */

 /* allocate memory for the deltas */
 delta = (float *)calloc(mlayer+1, sizeof(float));
 delta1 = (float *)calloc(mlayer+1, sizeof(float));

 /* adapt the weights starting from the output layer */
 l = layers - 1;

 for( j=0; j<N[layers]; j++ ) /* go through the nodes in the output layer */
 {      /* act x (1 - act) x (target - act) */
  y = 0.0;
  delta[j] = L[layers][j] * (1 - L[layers][j]) * (D[j] - L[layers][j]);
  y = delta[j] * gterm;
  for( i=0; i<N[layers-1]; i++ )  /* adjust the connecting weights from the */
  {        /* j node of the output layer to the nodes in the nearest hidden layer */
   if ( i==0 )   /* adjust the 1st weight by the gterm to start */
     a =0.0;
   else
      a = alpha;
   prwtchg = fabs(W[l][i+1][j] - Wtmp[l][i+1][j]); /* compute the old weight change */
   tempwt = W[l][i+1][j];  /* save the old weight before changing it */
   W[l][i+1][j] = W[l][i+1][j] + ( y*L[l][i] ) + ( a*prwtchg ); /* change the weight */
   wtchg = fabs(W[l][i+1][j] - tempwt); /* compute the new weight change */
   Wtmp[l][i+1][j] = tempwt;  /* switch the old wt to the temp holder */

   corre = corre + ( (wtchg+prwtchg)/2 ); /* calculate the correlation */

   } /* end of the 2nd for */
   W[l][0][j] = W[l][0][j] - y;  /* adjust the thresholds */
   Wtmp[l][0][j] = Wtmp[l][0][j] - y;
  } /* end of the 1st for */

 /* adapt the weights in the hidden layers */

 for( l=layers-1; l>0; l-- )  /* loop through the hidden layers */
 {   /* go through the nodes of the l hidden layer */
  for( j=0; j<N[l]; j++ )
  {             /* get the delta weights of the l+1 layers */
                 /* connections into the j node of the l hidden layer */
   sum = 0.0;          /* initialise the sum */
   for( k=0; k<N[l+1]; k++ )
```

```
     sum = sum + delta[k]*W[l][j+1][k];  /* sum the delta weights */
    delta1[j] = L[l][j]*(1-L[l][j])*sum; /* use that to compute the new delta */
    y = gterm*delta1[j];

    /* adjust the connecting weights from the j node of the hidden layer to */
    for( i=0; i<N[l-1]; i++ )  /* the nodes in the next layer */
    {
     if ( i==0 )   /* adjust the 1st weight by the gterm to start */
      a =0.0;
     else
        a = alpha;
     prwtchg = fabs(W[l-1][i+1][j] - Wtmp[l-1][i+1][j]); /* compute the old weight change */
     tempwt = W[l-1][i+1][j];  /* save the old weight before changing it */
     W[l-1][i+1][j] = W[l-1][i+1][j] + ( y*L[l-1][i] ) + ( a*prwtchg ); /* change the weight */
     wtchg = fabs(W[l-1][i+1][j] - tempwt); /* compute the new weight change */
     Wtmp[l-1][i+1][j] = tempwt;  /* switch the old wt to the temp holder */

     corre = corre + ( (wtchg+prwtchg)/2 ); /* calculate the correlation */

     } /* end of the 3rd of */
     W[l-1][0][j] = W[l-1][0][j] - y;
     Wtmp[l-1][0][j] = Wtmp[l-1][0][j] - y;
    } /* end of the 2nd for */
    temp = delta;
    delta = delta1;  /* swap the deltas */
    delta1 = temp;
   } /* end of the 1st for */


   free(delta);
   free(delta1);
 }   /* end of adaptweights */

/**********************************
 * train():      this routine trains the network during the processing of
 *               an image pattern.
 *******************************************/

int train()
{
 char ch;
 int  b, i, j, k, t;
 int *index, temp, stop = 0;
 float perror, serror;

      tsserror = 0.0;      /* initialise the error term */
      serror = 0.0;
      corre = 0.0;         /* initialise the correlation term */
      count++;             /* increase the number of sweeps */
      srand(count%100);    /* re initialise the random number generator */
                            /* allocate memory and initialise the index */
      index = (int *)calloc(p, sizeof(int));
      for( i=0; i<p; i++ )
        index[i] = i;

      t = p-1;
      while ( t>=0 )  /* go through the number of patterns randomly */
        {
         b = rand()%(t+1);
         k = index[b];

         L[0] = input[k];   /* get the input array */
         D = output[k];     /* get the output array */

         compute_out();   /* compute the activations up to the output layer */
```

```
        perror = 0.0;    /* initialise the error per pattern */
        for( j=0; j<N[layers]; j++ )
          perror += fabs(D[j] - L[layers][j]); /* calculate the error per pattern */

        serror += perror * perror;    /* compute the sum of square error */

        adaptweights();  /* adjust the weights to reduce the error */
                          /* re arrange the index to reflect the remnants */
        temp = index[b]; index[b] = index[t]; index[t] = temp;
        t--;   /* decrement the number of patterns */

       } /* end of second for */

      tsserror = serror;
      corre /= p;
      adjustparameters();

       if ( tsserror < ERRORLIMIT )   /* stop if the error is less than the */
          {                          /* error limit set, stabilisation has occured */
           printf("\nThe tss error is less than %f and the ",ERRORLIMIT);
           printf("number of sweeps is %d.",count);
           stop = 1;
          } /* end of if */
 free(index);
 return(stop);

} /* end of train */

/**********************************
 * learn():      this routine trains the network for the number of epochs or
 *               iterations needed to achieve convergence, i.e., if the weights
 *               have stabilised. A log file of the variables at various positions
 *               is kept.
 ****************************************/

#define INTERVAL 1
void learn(fsav)
char fsav[15];
{
 FILE *fs1;
 char line[15];
 int r, k = 0;

 strcpy(line,"log.");
 strncat(line,fsav,3);
 fs1 = fopen(line,"w");

 if ( sweeps != 0 )     /* convergence is by the number of sweeps */
  {                   /* so get that number */
         /* do till the number of sweeps */
   count = 0;
   for( r=0; r<sweeps; r++ )
     {
      k = train();
      if ( (count%INTERVAL) == 0 )
       {
        fprintf(fs1,"\n\t#%5d.          error:    %7.5f.       eta:    %4.3f.        alpha:
%4.3f.",count,tsserror,gterm,alpha);
        save(fsav);
       }

      if ( k == 1 )
       break;
     }  /* end of the for */
```

```
    printf("\n\nThe tss error is %7.5f and number of sweeps is %d.",tsserror,count);
  } /* end of the first if */
 else if ( limit != 0 )
   {         /* convergence is by an error limit from the keyboard */
    count = 0;

    do
     {
      k = train();
      if ( (count%INTERVAL) == 0 )
       {
        fprintf(fs1,"\n\t#%5d.          error:    %7.5f.       eta:    %4.3f.       alpha:
%4.3f.",count,tsserror,gterm,alpha);
        save(fsav);
       }

      if ( k == 1 )
       break;
     } while ( tsserror > limit );

    printf("\n\nThe tss error is %7.5f and number of sweeps is %d.",tsserror,count);
   } /* end of the else */
 else
   {
    printf("\n#sweeps or errorlimit has to be greater than 0.");
    exit(0);
   }

} /* end of the learn */

/**************************
 * checkparasite():      this routine is used to scan an image file to classify
 *                       the patterns on it.
 ***************************************************/

#define S 32
#define B 256
#define GSCALE 255

void checkparasite()
{
 FILE *im;
 char image[15], ch;
 int i;
 int stop, counter=0;
 float val;
          /* get the file name and open the file */
 printf("\nEnter the name of the image file to be checked:");
 gets(image);
 if ((im = fopen(image,"r")) == NULL)
  {
   printf("\nCannot open file %s.",image);
   exit(0);
  }

 stop = 64;  /* get the total number of sections on the image*/
 do
  {
   for( i=0; i<N[0]; i+=S ) /* load the data into the input nodes */
     {
        val = fgetc(im);    /* normalise the data before loading it */
        L[0][i] = val/GSCALE;
     } /* end of first for */

   compute_out();       /* display the classifications */
```

```
  printf("\nL[0]=%4.3f.L[1]=%4.3f.section %d.",L[layers][0],L[layers][1],counter+1);

  counter++;   /* increment the counter */
 } while ( counter < stop );

}  /* end of checkparasite */

/*************************
 * displayweights():     this routine displays the weights and the output
 *                       activations at any particular time during a run.
 ***************************************/

void displayweights()
{
 int l, j, i;
 char show;
 printf("\nEnter 'w' to look at the weights and the outputs or 'o' for ");
 printf("outputs only(w/o)!");
 while ((( show = getch() ) != 'w') && ( show != 'o'));
 if ( show == 'w' )
  {
   printf("\n\n\tLayer    Node    Weight   Node   Layer  ");
   printf("\n\t-----------------------------------------");

   for( l=0; l<layers; l++ )
     for( i=0; i<N[l]; i++ )
       for( j=0; j<N[l+1]; j++ )
         {
          printf("\n\t  %d       %d     %9.5f    %d       %d ",l+1,i+1,W[l][i+1][j],j+1,l+2);
         }  /* end of for */
   printf("\n\t-----------------------------------------");
   printf("\n\nEnter a char to view the output values so far.");
   getch();

   printf("\n\n\tValues for the output layer. ");
   printf("\n\t   Node         Values    ");
   printf("\n\t-------------------------");

   for( i=0; i<N[layers]; i++ )
     printf("\n\t    %d         %9.5f  ",i+1,L[layers][i] );
   printf("\n\t---------------------------\n");
   printf("\n\nEnter a char to end viewing:\n ");
   getch();
  }  /* end of if */
 else
   {
    printf("\n\n\tValues for the output layer. ");
    printf("\n\t   Node         Values    ");
    printf("\n\t-------------------------");

    for( i=0; i<N[layers]; i++ )
      printf("\n\t    %d         %9.5f  ",i+1,L[layers][i] );
    printf("\n\t---------------------------\n");
    printf("\n\nEnter a char to end viewing:\n ");
    getch();
   }  /* end of else */
 fflush(stdin);
}  /* end of displayweights */

/***************************
 * save():      this saves the connecting weights when required.
 **************************************/

void save(fsave)
char fsave[15];
```

```
{
 FILE *fs;
 int l, i, j;

 fs = fopen(fsave,"w");   /* open the file for writing */

 fprintf(fs,"%s",fsave);  /* write the name of the file */
 fprintf(fs,"\n");

 for( l=0; l<layers; l++ )
   {
    for( i=0; i<=N[l]; i++ )
      {
       for( j=0; j<N[l+1]; j++ )
         fprintf( fs,"%9.5f",W[l][i][j] );
       fprintf( fs,"\n" );
      }  /* end of the second for */
    fprintf(fs,"\n");
    }  /* end of the first for */

   fclose(fs);

} /* end of save */

/*****************************
 * main():       this is the main driver routine of this neural net program
 *               where the necessary files and options for use by the program
 *               are checked and loaded. It is a command line run program and
 *               calls the necessary routines depending on the options defined
 *               in the files used.
 *************************************/

void main(number, names)
int number;
char **names;
{
 char wtname[15];

 if ( number < 5 )   /* check if correct number of fnames */
  {
   puts("\nUse of this program is as follows:");
   puts("\nnetline <rannet> <wtdirfile> [option t-c-p] <train or class or parasite file> ");
   exit(0);
  }

 initscr();

 srand(37);
 create(names[1]);
 w_range(names[2]);
 if ( names[3][0] == 't')
  {
   strncpy(wtname,names[1],3);
   strcat(wtname,".wt");

   loadtrainset(names[4]);
   learn(wtname);
   save(wtname);
  }
 else if ( names[3][0] == 'c')
    {
     loadclass(names[4]);
     compute_out();
     displayweights();
    }
```

```
 else if ( names[3][0] == 'p' )
     checkparasite(names[4]);
 else
    {
     printf("\nOption has to be t-c-p not %c",names[3]);
     exit(0);
    }

 fflush(stdin);
 endwin();
}   /* end of main */
/*  end of the Rannet.c  */
```

```
/***************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the neural net program that implements
*  the conjugate gradient optimization methods. It contains the
*  routines for the off-line processing of the network.
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
***************************************************************/

 /***********************
 * RANOPT.C:     This neural net simulator implements the conjugate
 *               gradient optimization method. It contains all the
 *               necessary routines for creating the network and
 *               training the network with image patterns. It is
 *               implemented as a classifier.
 ******************************/

#include "ranopt.h"
#include "conjugate.c"

#define MAXLOOP 10        /* max number of weight initialisation */
#define MAXPT 0.7         /* upper cut-off point for the output */
#define MINPT 0.3         /* lower cut-off point for the output */
#define CLAS "train.set" /* training set file */
#define CONST "netwk.arc"       /* network arch. file */
#define OUTP "weight.fil"        /* weight file */
                          /* announce these function in conjugate.c */
extern FTYPE     critfc();
extern FTYPE     conjugate();
                          /* declare the global variables */
FTYPE    *w, **output, **delta, **input;
int      layers, pats, *units, *target, *conwts;
int      prt_flag;

void main (argc, argv)
int    argc;
char   **argv;
{
 FTYPE  tsserror, num, tol, *w;
        int     n, j, maxunit, inum;
        int     totconn, nit,i, counter=0;
        FILE    *inp, *outp, *wp;
        char    name[100];
        int     errflag = 0, cont_flag = 0, data_set = 0;

 if (argc == 4)
 {
        data_set = atoi (argv[1]);
        cont_flag = atoi (argv[2]);
        prt_flag = atoi (argv[3]);
                          /* check the number or tag for the file */
        if (data_set < 0 || data_set > 99)
          errmsg ("input file # must be between 0 and 99");
                          /* check if weights are to be loaded from a file
                          /* or from initial randum values */
        if (cont_flag < 0 || cont_flag > 1)
          errmsg ("continue flag must be either 0 or 1");
                          /* check if error values to be displayed */
        if (prt_flag < 0 || prt_flag > 1)
          errmsg ("print flag must be either 0 or 1");
 }
```

```
      else           /* display use of the program message */
         errmsg("\nUsage: ranopt <# for data file> <#cont-flag> <#prtflag>");

strcpy(name, CONST);     /* get the network arc. file for opening */
strncat(name, argv[1], 2);
if ((inp = fopen(name,"r")) == NULL)     /* if cannot open */
 errmsg("Network arch. file cannot be opened.");  /* display error message */
fscanf(inp,"%d %d %f", &layers, &nit, &thres);  /* get parameters that */
                                /* the network archictecture   */
if (layers != 3)               /* check if a simple network or multi-layer network */
 errmsg("Number of layers is not 3.");  /* message if wrong */
                              /* allot memory for the units in the layers */
if ((units = (int *)malloc(layers * sizeof(int))) == NULL)
 errmsg("Not enough memory for the units.");   /* display error if memory */
                                   /* allocation fails  */
mxunit = 0;              /* find the layer with the max number of units */
for(n=0; n<layers; n++)
{                        /* get the # of units from the network file */
 fscanf(inp," %d", &units[n]);
 if (n=0)                /* allow for the variable threshold in the */
  units[n]++;                     /* first layer */
 if (units[n] > mxunit)   /* find the layer with the max units */
  mxunit = units[n];
} /* end of the for */
fclose(inp);
                           /* allot memory for the connections between the layers */
if ((conwts = (int *)malloc(layers*sizeof(int))) == NULL)  /* display error */
 errmsg("Not enough memory for the connecting weights.");  /* message */

totconn = 0;         /* compute the number of connections from */
for(n=0; n < (layers-1); n++)  /* layer to layer and the total number of */
{                              /* connections in the network */
 conwts[n] = totconn;
 totconn += units[n] * units[n+1];
}  /* end of the for */
          /* allocate memory for the output and their delta units */
output = (FTYPE **)calloc(layers+1, sizeof(FTYPE));
delta  = (FTYPE **)calloc(layers+1, sizeof(FTYPE));
for(i=0; i<layers; i++)
{
 output[i] = (FTYPE *)calloc(units[i]+1, sizeof(FTYPE));
 delta[i] = (FTYPE *)calloc(units[i]+1, sizeof(FTYPE));
 if ((output[i] == 0) || (delta[i] == 0))    /* display error message if */
  errmsg("Not enough memory for the activations."); /* memory allocation fails */
}  /* end of the for */
                               /* allocate memory for the weights */
if ((w = (FTYPE *)malloc(totconn * sizeof(FTYPE))) == NULL)
 errmsg("Not enough memory for the weights.");

strcpy(name, CLAS);     /* get and open the training-set file */
strncat(name, argv[1], 2)

if ((inp = fopen(name,"r")) == NULL)
 errmsg("Training file cannot be opened.");
fscanf(inp, "%d", &pats);
if (pats < 2)   /* check if less than two patterns */
 errmsg("Number of patterns have to be greater than 2 for classification");
                               /* allocate memory for the input units */
input = (FTYPE **)calloc(pats+1, sizeof(FTYPE));
for(i=0; i<pats; i++)
{
 input[i] = (FTYPE *)calloc(units[0]+1, sizeof(FTYPE));
 if (input == 0)
  errmsg("Not enough memory for input units.");
} /* end of the for */
```

```
                          /* allocate memory for the target units */
if ((target = (int *)malloc(pats*sizeof(int))) == NULL)
 errmsg("Not enough memory for the target units.");

for(n=0; n<pats; n++)     /* get the training data pattern by pattern */
{
 fscanf(inp, "%d", &target[n]);   /* get the desired targets first */
 if (target[n] > units[layers-1])   /* and check it */
  errmsg("There are more targets than output units.");
 for(i=0; i< (units[0]-1); i++)    /* read the data for each pattern */
 {                                 /* and check the data */
  if (fscanf(inp, "%lf", &input[n][i]) == EOF)
   errmsg("Not enough data in training file.");
 }  /* end of 2nd for */
 input[n][units[0]-1] = 1.;    /* assign the threshold unit in the 1st layer */
}   /* end of 1st for */
fclose(inp);
                /* load in the weights */
if (cont_flag)  /* if reading the weights from a file */
{
 strcpy(name, OUTP);
 strncat(name, argv[1], 2);

 if (!(wp = fopen(name, "r")))
    errmsg("Error, weight file cannot be opened");

 fread(&inum, sizeof(int), 1, wp);
 fread(&layers, sizeof(int), 1, wp);
 fread(units, sizeof(int), layers, wp);
 fread(w, sizeof(FTYPE), totconn, wp);
 fread(tsserror, sizeof(FTYPE), 1, wp);
 fclose(wp);
}
else
{
do         /* select initial random weights using the criteria outlined in */
 {         /* the project report and repeat for MAXLOOP times if convergence */
  for(n=0; n < (layers-1); n++)  /* is not achieved */
    for(i=0; i < units[n]; i++)
    {
     num = sqrt((double)units[n])/3.;
     for(j=0; j < units[n+1]; j++)
       w[conwts[n] + j*units[n] + i] = (2.0*randnum() - 1.)/num;
    }  /* end of for */
                    /* optimise using the conjugate gradient method */
  tsserror = conjugate(&w, totconn, nit, thres);  /* adjust the weights */
  if (tsserror > thres)    /* if convergence is not achieved, repeat procedure */
   counter++;              /* with a new set of weights */
  else
     counter = MAXLOOP;    /* else stop the optimisation */
 } while (counter < MAXLOOP);
 } /* end of the else */

                        /* save the weights in the weight file */
  strcpy(name, OUTP);
  outp = fopen(name, "wb");
  fwrite(&nit, sizeof(int), 1, outp);
  fwrite(&layers, sizeof(int), 1, outp);
  fwrite(units, sizeof(int), layers, outp);
  fwrite(w, sizeof(FTYPE), totconn, outp);
  fwrite(&tsserror, sizeof(FTYPE), 1, outp);
  fclose(outp);

}  /* end of ranopt.c */
```

```
/****************************************
 * ranopt.h:     this is the header file where the libraries are included,
 *               simple functions used are defined, and the variable types
 *               are defined and declared.
 ****************************************/

#define PC    /* define if using PC */
#define UNIX  /* define if using UNIX system */
              /* include the necessary libraries */
#ifdef PC
 #include <stdlib.h>
 #include <string.h>
 #include <conio.h>
 #include <alloc.h>
#endif

#ifdef UNIX
 #include <curses.h>
#endif

#include <stdio.h>
#include <math.h>

        /* define the functions if not defined in the libraries being used */
#define max(a,b)          ((a)<(b)?(b):(a))
#define min(a,b)          ((a)<(b)?(a):(b)) */
#define bound(a,b,c)      (max((a),min((b),(c))))
#define round(a)          ((a)>0.? (int)((a)+.5):(int)((a)-.5))
#define hi(a)             (unsigned char)(a/256)
#define lo(a)             (unsigned char)(a%256)
#define sqr(a)            ((a) * (a))

#define ULIMIT 70         /* upper limit of exp-function */
#define LLIMIT -70        /* lower limit of exp-function */
#define RED 0.2           /* the allowed reduction in the gradient to stop */
                          /* the line search */
#define ITER 5            /* Allowed number of iterations for a line search */
#define FTYPE   float

typedef unsigned char     byte;

/*****************************************
 * randnum():    this routine generates the randum numbers.
 ***********************************/

double randnum()
{
 static double x = 3141592654.0;
        double a = 3141592654.0;
        double c = 2718281828.0;
        double m = 1.0e+10;

        x = fmod(a*x+c, m);
        return(x/m);
} /* end of randnum */

/************************
 * f(): this routine computes the sigmoid logistic function.
 *********************/

FTYPE f(sig)
FTYPE sig;

{
 FTYPE x;
```

```
        x = sig;
        if (x > ULIMIT)
                x = ULIMIT;
        else if (x < LLIMIT)
                x = LLIMIT;
        return (FTYPE) 1. / (1. + exp (-(FTYPE) x));
}/* end of f */

/*************************
 *errmsg():      this routine displays an error message.
 **********************/

void errmsg(msg)
char  *msg;

{
 perror (msg);
        exit (0);
}/* end of errmsg */
/* end of the header file ranopt.h */
```

```
/*******************************************
 * conjugate.c:          this program implements the conjugate gradient optimisation
 *                       method using the two routines, conjugate() and critfc().
 ***********************************************/

#include <ranopt.h>

FTYPE critfc();

/************************************************************
 * conjugate():          this routine implements the conjugate gradient method by
 *                       optimising the computation of the error/criterion function
 *                       with reinitialisation and using that to adjust the weights
 *                       in the network.
 ************************************************************/
FTYPE conjugate(x, totconn, nit, thres)
      FTYPE **x, thres;
      int totconn, nit;
{
 register FTYPE *ptr1, *ptr2, *ptr3, *ptr4, temp1, temp2;
 static FTYPE *cps, *gcps, terror, gamma, tgamma, beta;
 static FTYPE *wtptr, sserror, sserror1, gdif, gsq, gsq1;
 static FTYPE *gbeg, *gmin, *sdir, *rsdir, *grsdir, sumgr;
 static FTYPE sumgr1, maxstsize, minstsize, stepsize;
 static FTYPE stsizechg, grs, gr2s, temp;

 static int it, rflag, flag, flag1, flag2, lsflag;
                                      /* allocate memory for the */
 cps    = (FTYPE *)malloc(totconn*sizeof(FTYPE));  /* current position */
 gcps   = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* gradient of position */
 gbeg   = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* gradient at beggining */
 gmin   = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* min gradient so far */
 sdir   = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* search dir for line search */
 rsdir  = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* restart search direction */
 grsdir = (FTYPE *)malloc(totconn*sizeof(FTYPE)); /* restart search dir gradient */
                        /* check if memory allocated else display error message */
 if (!cps || !gcps || !gbeg || !gmin || !sdir || !rsdir || !grsdir)
  errmsg("Memory allocation failure for the search parameters.");

 wtptr = *x;       /* get the weight array */
                    /* do the first feed forward iteration and compute the error */
 sserror1 = critfc(wtptr, gmin, totconn);  /* between the target and actual */
 terror = sserror1;
                    /* initialise the parameters */
 it = 1;           /* number of iterartions */
 rflag = 2;        /* restart flag, 2 for line search in dir of steepest descent */
 flag = 0;         /* 1 if no decrease in the error and 2 if no decrease in */
                    /* gradient during line search */
 flag1 = 0;        /* for decrease in search dir */
 gdif = 0.;        /* diff betw the current position gradient and the beg. gradient */
 gsq1 = 0.;        /* gradient squared */

 do                /* do till the number of iterations allowed */
 {
  if (flag & 1)
   {
    if (flag1)     /* if no decrease dir has been found */
     {             /* display error message */
      printf("No decrease dir found, so stop");
      break;
     }
    else
        flag1 = 1;  /* set the flag for the next round */
   }
  else
```

```
   flag1 = 0;

if (flag & 2)
 rflag = 2;      /* line search in steepest descent dir */
else if (fabs((double) gdif) > 0.2*gsq1 )  /* if gradient diff is greater than */
   rflag = 1;    /* the limit then choose a new restart */

if (rflag == 0)  /* begin initial line search freshly */
 {
  ptr1 = gcps;   /* get gradient of current position */
  ptr3 = ptr1 + totconn;  /* get the last position */
  ptr2 = grsdir;  /* gradient of restart dir */
  ptr4 = rsdir;   /* restart dir */
  temp1 = temp2 = 0.;
  for ( ;ptr1 < ptr3; )
  {
   temp1 += (*ptr1) * (*ptr2++);  /* sum(gcps*grsdir[i])  */
   temp2 += (*ptr1++) * (*ptr1++); /* sum(gcps[i]*rsdir[i]) */
  }
  gamma = temp1/tgamma;   /* compute the gamma factor */

  if ( fabs( beta*sumgr - gamma*temp2 ) > 0.2*gsq1 )
   rflag = 1;
  else
    {
     ptr1 = sdir ;        /* search dir */
     ptr2 = rsdir;        /* restart dir */
     ptr3 = ptr1 + totconn; /* end of the list */
     ptr4 = gcps;         /* gradient of the current position */
     for ( ;ptr1 < ptr3; ptr1++)  /* sum(beta*sdir[i] + gamma*rsdir[i] + */
       (*ptr1) = beta*(*ptr1) + gamma*(*ptr2++) - (*ptr4++); /* gcps[i] */
    }
 }
 if (rflag == 2)    /* line search in steepest descent dir */
 {
  ptr1 = sdir;
  ptr2 = gmin;
  ptr3 = ptr1 + totconn;
  for ( ;ptr1 < ptr3; )
    (*ptr1++) = - (*ptr2++);  /* negate the minimum gradient for the new search dir */
  rflag = 1;      /* set the restart flag for a new one */
 }
 else if (rflag == 1)    /* if set then do a new restart */
   {
    tgamma = sumgr - sumgr1;
    ptr1 = sdir;
    ptr2 = rsdir;
    ptr3 = ptr1 + totconn;
    ptr4 = gcps;
    for ( ; ptr1 < ptr3; ptr1++)
      {
       (*ptr2++) = (*ptr1);   /* save the old search dir */
       (*ptr1) = beta*(*ptr1) - (*ptr4++);  /* compute the new search dir */
      }                          /* beta*sdir[i] - gcps[i] */
    ptr1 = grsdir;
    ptr2 = gbeg;
    ptr3 = ptr1 + totconn;
    ptr4 = gcps;
    for ( ;ptr1 < ptr3; )
      (*ptr1++) = (*ptr4++) - (*ptr2++);  /* get the gradient of the new restart */
                                        /* search dir by the diff. */
    rflag = 0;        /* set the flag for another restart procedure */
   }
            /* line search is done here */
 flag = 0;   /* initialise the flag */
```

```
lsflag = 0;  /* line search flag and counter for line search iterations */
ptr1 = gmin;
ptr3 = ptr1 + totconn;
ptr2 = gbeg;
ptr4 = sdir;
temp1 = 0.;
for ( ;ptr1 < ptr3; )
{      /* sum(gmin*sdir[i] */
 temp1 += (*ptr1) * (*ptr4++);
 (*ptr2++) = (*ptr1++);    /* save the old values */
}

sumgr = temp1;    /* sum the gradients */
sumgr1 = sumgr1;
if (it == 1)    /* if this is the first iteration */
 minstsize = fabs(terror/sumgr);   /* set the min. step size */

if (sumgr1 > 0)
{
 flag = 1;
 rflag = 2;
 continue;
}

sserror = sserror1;   /* save the old error sum */
maxstsize = -1;
flag2 = -1;
stsizechg = (minstsize < (temp1 = fabs(terror/sumgr))) ? minstsize: temp1;
minstsize = 0.;    /* initialise the step size before the next loop */

do
{
 do
 {
  if ( lsflag )
   {
    if ( fch != 0)
     gold += 2*temp/stsizechg;
    if ( maxstsize < -0.5 )
     stsizechg = 9. * minstsize;
    else
       stsizechg = 0.5 * (maxstsize - minstsize);

    gnew = sumgr + stsizechg*gold;   /* compute the new gradient */
    if (sumgr*gnew < 0)
     stsizechg *= sumgr/(sumgr-gnew);
   } /* end of the lsflag if */

   stepsize = minstsize + stsizechg;  /* compute the new step size */
   ptr1 = cps;
   ptr2 = wtptr;
   ptr3 = sdir;
   ptr4 = ptr1 + totconn;
   for ( ;ptr1 < ptr4; )     /* adapt the weights */
     (*ptr1++) = (*ptr2++) + stsizechg*(ptr3++);
                            /* compute the sum of square error between the */
   sserror2 = critfc(cps, gcps, totconn); /* target and the new actual */
   it++; lsflag++;        /* increase the number of iterations */

   ptr1 = gcps;
   ptr2 = sdir;
   ptr3 = ptr1 + totconn;
   temp1 = temp2 = 0.;
   for ( ;ptr1 < ptr3; )
   {
```

```
   temp1 += (*ptr1) * (*ptr1);  /* sum(gcps*gcps) */
   temp2 += (*ptr1++) * (*ptr2++); /* sum(gcps[i]*sdir[i]) */
  }

  sumgr2 = temp1;
  gsq = temp2;
  fch = sserror2 - sserror1;
  gold = (sumgr2 - sumgr)/stsizechg;
  temp = 2*fch/stsizechg - sumgr2 - sumgr;

  if ((sserror2 < sserror1) || ((sserror2 == sserror1) && (sumgr2/sumgr > -1)))
  {
   gsq1 = gsq;
   sserror1 = sserror2;
   ptr1 = wtptr;      /* switch and move the new weight to the current position */
   wtptr = cps;
   cps = ptr1;
   ptr1 = gmin;     /* switch and move the gradient also */
   gmin = gcps;
   gcps = ptr1;
   if (sumgr2*sumgr <= 0)
    maxstsize = minstsize;

   maxstsize = stepsize;
   sumgr = sumgr2;
   stsizechg = -stsizechg;

   if (gsq < thres)  /* check if sum(gcps[i]*sdir[i]) is less than the */
   {                 /* error thres set */
    *x = wtptr;      /* get the new weight */
    return sserror1;
   }
   if (fabs(sumgr/sumgr1) < RED)  /* if the gradient reduction is enough */
    break;                        /* then stop */
  }
  else
     maxstsize = stepsize;  /* else use the max step size */

  if (lsflag > MAXIT)   /* if more than the number of iterations */
   break;               /* then stop */
 } while (1);           /* do till a break */

sserror2 = sserror1;
ptr1 = cps;
ptr2 = wtptr;
ptr3 = ptr1 + totconn;
for ( ;ptr1 < ptr3; )
  (*ptr1++) = (*ptr2++);   /* get the new weights */

ptr1 = gcps;
ptr2 = gbeg;
ptr3 = ptr1 + totconn;
ptr4 = gmin;
temp1 = 0.;
for ( ;ptr1 < ptr3; )
{
 (*ptr1) = (*ptr4++);
 temp += (*ptr1++) * (*ptr2++);
}

gdif = temp;
if (fabs(sumgr - sumgr1) > thres)
{
 beta = (gsq1 - gdif)/(sumgr - sumgr1)
 if (fabs(beta*sumgr) < 0.2*gsq1)
```

```
      break;
    }
    flag2++;  /* update the flag */
    if (flag2 > 0)
     flag += 2;

   } while (flag < 1);

   if (sserror <= sserror1)
    flag += 1;
   terror = sumgr1 * minstsize;
  } while (it < nit);

  *x = wtptr;
  return sserror1;        /* return the optimised error */
} /* end of conjugate */

/***********************************************************
 * critfc()      this routine computes the criterion function or error function
 *               from the adjusted weights and computes the weight gradient for
 *               storage. It returns the least mean square error.
 ***********************************************************/
FTYPE critfc(w, wtchg, totconn)
      int totconn;   /* the total connections in the network */
      FTYPE *w, *wtchg;
{
 register FTYPE *ptr1, *ptr2, *ptr3, errorptr, sumwts;
 FTYPE sserror, delta1;
 int i, n, j;

 errorptr = 0.;   /* initialise the registers and the parameters */
 ptr1 = wtchg;   /* get the first position of the delta values */
 ptr3 = ptr1 + totconn;  /* get the last value */
 for ( ;ptr1 < ptr3; )
   (*ptr1++) = errorptr;  /* initialise the variables */

 sserror = 0.;   /* start with a new error variable */
 for (n=0; n < pats; n++)   /* go through all the patterns */
 {
  ptr3 = w;   /* get the first weight array */
  for (i=0; i < units[1]; i++)  /* go through the first hidden layer units */
  {
   sumwts = 0.;
   ptr1 = ptr3;  /* get the first connecting weight in the layer */
   ptr3 = ptr1 + units[0];  /* get the last one */
   ptr2 = input[n];  /* get the first position in the input data for the pattern */
   for ( ;ptr1 < ptr3; )  /* compute the sum of the activations to the end of */
     sumwts += (*ptr2++) * (*ptr1++);  /* by sum(input[i]*w[i]) */

   output[1][i] = f(sumwts);  /* pass this sum through the sigmoid function */
  }  /* end of the second for */

  for (j=0; j < (layers-1); j++)  /* go through the remaining hidden layers */
    for (i=0; i < units[j+1]; i++)  /* go through the first of these */
    {
     sumwts = 0.;
     ptr1 = ptr3;   /* start from the last position in the weight array */
     ptr3 = ptr1 + units[j];
     ptr2 = output[j];
     for ( ;ptr1 , ptr3; )
       sumwts += (*ptr1++) * (*ptr2++);

     output[j+1][i] = f(sumwts);
    }
                 /* compute the error between the desired output and the actual */
```

```
for (i=0; i < units[layers-1]; i++)  /* which is used to compute the deltas for */
{                /* for adjusting the weights. This is done backwards through */
 if (i == target[n]-1)   /* the network layers */
  errorptr = min(output[layer-1][i] - MAXPT, 0);  /* compute the error using */
 else                                          /* the cut off points */
    errorptr = max(output[layer-1][i] - MINPT, 0);
 sserror += errorptr * errorptr;  /* compute the sum of the square error */
                     /* calculate the delta value for use later */
 delta1 = errorptr * output[layers-1][i] * (1.-output[layers-1][i]);
 delta[layers-1][i] = delta1;
 ptr1 = output[layers-2];  /* get the first output unit of that layer */
 ptr2 = &wtchg[conwts[layers-2] + i*units[layers-2]];  /* first position */
 ptr3 = ptr2 + units[layers-2];  /* and last position of the delta array */
 for ( ;ptr2 < ptr3; )
   (*ptr2++) += errorptr * (*ptr1++);  /* compute the new deltaw values */
}                              /* for this layer */
for ( j=(layers-2); j > 1; j--)    /* go through the next layer to compute */
{                 /* its delta values using that of the layer before this */
 errorptr = 0.;
 ptr1 = delta[j];
 ptr3 = ptr1 + units[j];
 for ( ;ptr1 < ptr3; )    /* initialise the delta array for this layer */
   (*ptr1++) = errorptr;

 for ( i=0; i < units[j+1]; i++)  /* compute the delta values for */
 {                             /* this layer */
  ptr2 = &w[conwts[j] + i*units[j]];
  ptr3 = ptr2 + units[j];
  ptr1 = delta[j];
  for ( ;ptr2 < ptr3; )    /* using the sum(w[i]*delta[i+1]) */
    (*ptr1++) += (*ptr2++) * delta[j+1][i];
 }
 for ( i=0; i < units[j]; i++)  /* move to the next hidden layer */
 {
  delta[j][i] *= output[j][i] * (1.-output[j][i]);
  ptr1 = output[j-1];
  ptr2 = &wtchg[conwts[j-1] + i*units[j-1]];
  ptr3 = ptr2 + units[j-1];
  for ( ;ptr2 < ptr3; )
    (*ptr2++) += (*ptr1++) * delta[j][i];
 }
} /* end of the j for */
                         /* compute the delta values for the first layer */
errorptr = 0.;          /* initialise and get parameters */
ptr1 = delta[1];
ptr3 = ptr1 + units[1];
for ( ;ptr1 < ptr3; )
  (*ptr1++) = errorptr;

for ( i=0; i < units[2]; i++)
{
 ptr2 = &w[conwts[1] + i*units[1]];
 ptr3 = ptr2 + units[1];
 ptr1 = delta[1];
 for ( ;ptr2 < ptr3; )
   (*ptr1+) += (*ptr2++) * delta[2][i];
}
for ( i=0; i < units[1]; i++)
{
 delta[1][i] *= output[1][i] * (1.-output[1][i]);
 ptr1 = input[n];
 ptr2 = &wtchg[units[0]*i];
 ptr3 = ptr2 + units[0];
 for ( ;ptr2 < ptr3; )
   (*ptr2++) += (*ptr1++) * delta[1][i];
```

```
  }
 }  /* end of the starting for */

 sserror *= 0.5;     /* compute the least mean square error */
 if (prt_flag)        /* display the error if the flag is on */
   printf("%10.6lf\n",sserror);

 return sserror;
}  /* end of the critfc */
/* end of conjugate.c */
```

```
/****************************************
 * prod.h:      this is the header file where the libraries are included,
 *              simple functions used are defined, and the variable types
 *              are defined and declared for use in the production programs.
 ****************************************/

#define PC    /* define if using PC */
#define UNIX  /* define if using UNIX system */
              /* include the necessary libraries */
#ifdef PC
 #include <stdlib.h>
 #include <string.h>
 #include <conio.h>
 #include <alloc.h>
#endif

#ifdef UNIX
 #include <curses.h>
#endif

#include <stdio.h>
#include <math.h>

        /* define the functions if not defined in the libraries being used */
#define max(a,b)        ((a)<(b)?(b):(a))
#define min(a,b)        ((a)<(b)?(a):(b)) */
#define bound(a,b,c)    (max((a),min((b),(c))))
#define round(a)        ((a)>0.? (int)((a)+.5):(int)((a)-.5))
#define hi(a)           (unsigned char)(a/256)
#define lo(a)           (unsigned char)(a%256)
#define sqr(a)          ((a) * (a))

#define ULIMIT 50       /* upper limit of exp-function */
#define LLIMIT -50      /* lower limit of exp-function */
#define FTYPE   float

typedef unsigned char   byte;

/**************************
 * f(): this routine computes the sigmoid logistic function.
 **********************/

FTYPE f(sig)
FTYPE sig;

{
 FTYPE x;
        x = sig;
        if (x > ULIMIT)
                x = ULIMIT;
        else if (x < LLIMIT)
                x = LLIMIT;
        return (FTYPE) 1. / (1. + exp (-(FTYPE) x));
}/* end of f */

/**************************
 *errmsg():     this routine displays an error message.
 **********************/

void errmsg(msg)
char  *msg;

{
 perror (msg);
        exit (0);
```

```
}/* end of errmsg */

/****************************************************************************
 * compute_out():        this routine computes the activations using the sigmoid
 *                       logistic function
 ****************************************************************************/

void compute_out(one)
int one;
{
 int    a, b, c;
 FTYPE  sigm;

 for (b = 0; b < units[1]; b++)
 {
  sigm = 0.;
  for (a = 0; a < units[0]; a++)
     sigm += w[0][b][a] * input[one][a];
  output[1][b] = f (sigm);
 }

 for (c = 1; c < layers - 1; c++)
   for (b = 0; b < units[c + 1]; b++)
   {
    sigm = 0.;
    for (a = 0; a < units[c]; a++)
      sigm += w[c][b][a] * output[c][a];
    output[c + 1][b] = f (sigm);
   }
} /* end of compute_out */
```

```
/***************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the classification program for the pixel
*  representation format
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
***************************************************************/

/**********************
 * wortest.c      this is a production program that classifies the sections
 *                of a cod image into parasitic sections and non parasitic
 *                sections using the pixel representation format
 ***************************/

#include "prod.h"

#define GSCALE 255.0
#define CODFIL "worms.da"
#define WT "weight.fil"

FTYPE        ***w, **output, **delta, **input;
int             layers, sec, one=1, *units;
FILE           *inp, *outp, *datp;

void main (argc, argv)
int    argc;
char   **argv;
{
 FTYPE  amax, tsserror, num;
 int    nit, n, i, j, maxunit;
 int    cmax, val, par, nopar;
 char   name[100];

 int    wtfil = 0, wormfil = 0, sec = 0; /* flags and number of sections */

        if (argc == 4)
        {
         wtfil   = atoi (argv[1]);
         wormfil = atoi (argv[2]);
         sec     = atoi (argv[3]);

                if (wtfil < 0 || wtfil > 99)
                        errmsg ("weight file # must be between 0 and 99");
                if (wormfil < 0 || wormfil > 99)
                        errmsg ("worm file # must be between 0 and 99");
                if (sec <= 0)
                        errmsg ("number of sections must be greater than 0");
        }
        else
            errmsg ("Usage: wortest  <#for weight file> <#for cod image file> <# sections in image>");

        strcpy (name, WT);
        strncat (name, argv[1], 2);

        if ((inp = fopen (name, "rb")) == 0)
                errmsg ("Error opening weight file for weights");

        strcpy (name, CODFIL);
        strncat (name, argv[2], 2);

        if ((datp = fopen (name, "r")) == 0)
```

```
        errmsg ("Error opening cod image file for image patterns");

 fread (&nit, sizeof (int), 1, inp);  /* read off the number of iteration */

 fread (&layers, sizeof (int), 1, inp);
 if ((units = (int *) malloc (layers * sizeof (int))) == 0)
        errmsg ("units malloc error");

 fread (units, sizeof (int), layers, inp);

 maxunit = 0;
 for (n = 0;n < layers;n++)
        if (units[n] > maxunit)
          maxunit = units[n];


/* allocate memory  */

 input = (FTYPE **)calloc(2,sizeof(FTYPE));

 for (i=0; i<2; i++)
 {  /* get the memory for input nodes */
  input[i] = (FTYPE *)calloc(units[0]+1,sizeof(FTYPE));
  if (input[i] == 0)
   {
    printf("\nNot enough memory for inputs");
    exit(0);
   }  /* end of if */
 }  /* end of for */

 output = (FTYPE **)calloc(layers+1,sizeof(FTYPE));

 for (i=0; i<layers; i++)
 {  /* get memory for the activations for layers after the first */
  output[i] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
  if (output[i] == 0)
   {
    printf("\nNot enough memory for activations");
    exit(0);
   } /* end of if */
  } /* end of for */

 w = (FTYPE ***)calloc(layers, sizeof(FTYPE));

 for (i=0; i<layers - 1; i++)  /* loop thru the # of layers */
 {   /* get the wts from a node to nodes in the next layer */
  w[i] = (FTYPE **)calloc(units[i+1] + 1,sizeof(FTYPE));
  for (j=0; j<units[i+1]; j++)  /* loop thru nodes of next layer */
  {
   w[i][j] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
   if (w[i][j] == 0)
    {
     printf("\nNot enough memory for weights");
     exit(0);
    } /* end of if */
   } /* end of 2nd for */
  } /* end of 1st for */

 for (n = 0;n < layers - 1;n++)
        for (i = 0; i < units[n + 1]; i++)
                fread (w[n][i], sizeof (FTYPE), units[n], inp);

 fread (&tsserror, sizeof (FTYPE), 1, inp);

 for (n = 0; n < sec; n++)    /* go through the windows sections */
```

```
    {
      for (i = 0; i < units[0] - 1; i++)  /* read the section patterns */
        {
         val = fgetc(datp);
         num = (FTYPE)(val)/GSCALE;
         input[one][i] = num;
        }

      input[one][units[0] - 1] = 1.;  /* assign act for the threshold */

      compute_out(one);  /* calculate the activations */

          /* find most active output neuron */
      amax = output[layers - 1][0];
      cmax = 0;

      for (i = 1; i < units[layers - 1]; i++)
          if (output[layers - 1][i] > amax)
            {
              amax = output[layers - 1][i];
              cmax = i;
             }

       if ( (cmax +1) == 1)
        par++;     /* if a par is found then increase par counter */
       else
          nopar++;  /* otherwise increase the no parasite counter */
                 /* display the section # and its classification */
     /*printf("\npattern %d. target %d. amax %lf.\n",n+1,cmax+1,amax);*/
   }
                 /* display the number of parasite and non parasite sections */
   printf("\n Parasites = %d.  No parasites = %d.\n",par,nopar);

   fclose(inp);
   fclose(outp);
   fclose(datp);

}   /* end of wortest.c */
```

```
/*****************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the classification program for the feature
*  extraction format
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
*****************************************************************/

/**********************
 * wavtest.c     this is a production program that classifies the sections
 *               of a cod image into parasitic sections and non parasitic
 *               sections using the feature extraction format
 ***************************/

#include "prod.h"

#define ARRSIZ 255
#define VECSIZ 32
#define CODFIL "worms.da"
#define WT "weight.fil"

FTYPE        ***w, **output, **delta, **input, vector[VECSIZ];
int             layers, sec, one=1, *units;
int           hist[ARRSIZ];
FILE            *inp, *outp, *datp;

/******************************************
 * wave():       this is used to compute the cumulative freq of an image
 *               and extract the normalised features into a 1-dim vector
 *********************************************/

void wave( )
{
 int h=0,l1=0,l2=0,l=0,j=0,m=0,y1=0, val1;
 FTYPE scale=0., y=0., k=0., max1=0.;

 h=0;
 while (hist[h++] <= 0)   /* find the start of the curve */
 l1 = h;

 for(h=255; h>=0; h--)   /* find the end of the curve */
   if (hist[h] != 0)
    {
     l2 = h;
     break;
    }

 l = l2 - l1;   /* compute the spread of the curve */

 if ( l > 32)            /* case of the spread greater than 32 */
  {          /* get the scale for enlargement and the first position */
   scale =(FTYPE)l/32;
   y = l1; m = l1; j = 0;  /* of the curve */
   while (y <= (FTYPE)l2)  /* process till the end of the curve */
   {  /* get the number of points in that section */
    k=0; val1 = 0;
    y += scale;    /* get the border of the next section */
    for (h=m; h<= y; h++,k++)
      val1 += hist[h];     /* compute the total y values */
    if (j >= VECSIZ)    /* stop if overflow */
        break;
```

```
    vector[j++] = (FTYPE)val1/k;  /* find the mean value */
     m = h;    /* get the x-cordinates for the section */
   }  /* end of the while */
  }   /* end of the if */
 else if ( l < 32 )       /* case of the spread less than 32 */
      {      /* get the scale for reduction and the first position */
       scale = 32/(FTYPE)(l+1); y1 = 0; y = 0; j = l1;  /* of the curve */
       while (j <= l2)  /* process till the end of the curve */
       {
        y += scale;   /* move to the next section of the divided 32 array */
        if (y1 >= VECSIZ)  /* stop if overflow */
         break;
        vector[y1] =(FTYPE)hist[j]; y1++; /* get the value for the first cor */
        if (y1 >= VECSIZ)
         break;
        if ((FTYPE)y1 <= y)        /* if not end of the section */
         {
          vector[y1] =(FTYPE)(hist[j] + hist[j+1])/2; /* interpolate for the next value */
          y1++;
         }
        j++;    /* move to the next section in the original array*/
       } /* end of the while */
      }  /* end of the second if */
 else
     { /* transfer the points to the vector array */
      for (j=0, y1=l1; j<VECSIZ; j++,y1++)
        vector[j] = hist[y1];
     }

 max1 = 0.;
 for (h=0; h<VECSIZ; h++)
   {
    if (vector[h] > max1)
     max1 = vector[h];
 /*   printf("\n%3.2f",vector[h]); */
   }

 for (h=0; h<VECSIZ; h++)
   {
    if (vector[h] != 0.)
      vector[h] = vector[h]/max1;
   }

}  /* end of wave */
/****************************************
 *  the driver of the program
 ***************************************/

void main (argc, argv)
int    argc;
char   **argv;
{
 FTYPE   amax, tsserror, num;
 int     nit, n, i, j, maxunit;
 int     cmax, val, par, nopar;
 char    name[100];

 int     wtfil = 0, wormfil = 0, sec = 0; /* flags and number of sections */

         if (argc == 4)
         {
          wtfil   = atoi (argv[1]);
          wormfil = atoi (argv[2]);
          sec     = atoi (argv[3]);
```

```
        if (wtfil < 0 || wtfil > 99)
                errmsg ("weight file # must be between 0 and 99");
        if (wormfil < 0 || wormfil > 99)
                errmsg ("worm file # must be between 0 and 99");
        if (sec <= 0)
                errmsg ("number of sections must be greater than 0");
}
else
    errmsg ("Usage: wavtest  <#for weight file> <#for cod image file> <# sections in image>");

strcpy (name, WT);
strncat (name, argv[1], 2);

if ((inp = fopen (name, "rb")) == 0)
        errmsg ("Error opening weight file for weights");

strcpy (name, CODFIL);
strncat (name, argv[2], 2);

if ((datp = fopen (name, "r")) == 0)
        'errmsg ("Error opening cod image file for image patterns");

fread (&nit, sizeof (int), 1, inp);

fread (&layers, sizeof (int), 1, inp);
if ((units = (int *) malloc (layers * sizeof (int))) == 0)
        errmsg ("units malloc error");

fread (units, sizeof (int), layers, inp);

maxunit = 0;
for (n = 0;n < layers;n++)
        if (units[n] > maxunit)
          maxunit = units[n];

/* allocate memory  */

input = (FTYPE **)calloc(2,sizeof(FTYPE));

for (i=0; i<2; i++)
{  /* get the memory for input nodes */
 input[i] = (FTYPE *)calloc(units[0]+1,sizeof(FTYPE));
 if (input[i] == 0)
  {
   printf("\nNot enough memory for inputs");
   exit(0);
  }  /* end of if */
}  /* end of for */

output = (FTYPE **)calloc(layers+1,sizeof(FTYPE));

for (i=0; i<layers; i++)
{  /* get memory for the activations for layers after the first */
 output[i] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
 if (output[i] == 0)
  {
   printf("\nNot enough memory for activations");
   exit(0);
  } /* end of if */
 } /* end of for */

w = (FTYPE ***)calloc(layers, sizeof(FTYPE));

for (i=0; i<layers - 1; i++)  /* loop thru the # of layers */
{   /* get the wts from a node to nodes in the next layer */
```

```
 w[i] = (FTYPE **)calloc(units[i+1] + 1,sizeof(FTYPE));
 for (j=0; j<units[i+1]; j++)  /* loop thru nodes of next layer */
 {
  w[i][j] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
  if (w[i][j] == 0)
   {
     printf("\nNot enough memory for weights");
     exit(0);
   } /* end of if */
  } /* end of 2nd for */
 } /* end of 1st for */

for (n = 0;n < layers - 1;n++)
        for (i = 0; i < units[n + 1]; i++)
                fread (w[n][i], sizeof (FTYPE), units[n], inp);

fread (&tsserror, sizeof (FTYPE), 1, inp);


for (n = 0; n < sec; n++)    /* go through the windows sections */
  {
    for (i=0; i<ARRSIZ; i++)     /* initialise the hist array */
        hist[i]=0;
    for (i=0; i<VECSIZ; i++)     /* initialise the vector array */
        vector[i]=0.;
    for (i=0; i<1024; i++)   /* compute the cum freq of the 32x32 */
     {
      val = fgetc(datp);    /* input data of the image */
      hist[val]++;
     }             /* if at end of file then stop */

    /* printf("\n # of times = %d",i);
    for (i=0; i<ARRSIZ; i++)
       printf("\n%d ",hist[i]);  */

    wave();  /* compute the features of the curve */
    for (i = 0; i < units[0] - 1; i++)  /* read the input */
      {
       input[one][i] = vector[i];
  /*     printf("%1.2f ",vector[i]); */
      }
    input[one][units[0] - 1] = 1.;  /* assign act for the threshold */

    compute_out(one);  /* calculate the activations */

        /* find most active output neuron */
    amax = output[layers - 1][0];
    cmax = 0;

    for (i = 1; i < units[layers - 1]; i++)
        if (output[layers - 1][i] > amax)
          {
            amax = output[layers - 1][i];
            cmax = i;
           }

     if ( (cmax +1) == 1)
      par++;     /* if a par is found then increase par counter */
     else
        nopar++;  /* otherwise increase the no parasite counter */
                /* display the section # and its classification */
   /*printf("\npattern %d. target %d. amax %lf.\n",n+1,cmax+1,amax);*/
}
                /* display the number of parasite and non parasite sections */
printf("\n Parasites = %d.  No parasites = %d.\n",par,nopar);
```

```
        fclose(inp);
        fclose(outp);
        fclose(datp);
}   /* end of wavtest.c */
```

```
/****************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the classification program for the curve
*  map/binarised curve map format.
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
****************************************************************/

/**********************
 * histest.c     this is a production program that classifies the sections
 *               of a cod image into parasitic sections and non parasitic
 *               sections using the curve map/binarised curve map format
 ***************************/

#include "prod.h"

#define ARRSIZ 255
#define VECSIZ 32
#define CORD 11
#define CODFIL "worms.da"
#define WT "weight.fil"

FTYPE        ***w, **output, **delta, **input, matrix[CORD][VECSIZ];
int             layers, sec, one=1, *units;
int          hist[ARRSIZ];
FILE           *inp, *outp, *datp;


/****************************************
 *cmap():        this is used to compute the cumulative freq of 32x32 section.
 *               and extracts the normalised features into 32x11 dim curve map
 *               binarised curve map
 **********************************************/

void cmap( )
{
 int h=0,l1=0,l2=0,l=0,j=0,m=0,y1=0, val1;
 FTYPE scale=0., y=0., k=0., max1=0.;
 int vector1[VECSIZ];
 FTYPE vector[VECSIZ], temp=0.;

 for(j=0; j<CORD; j++)
   for(h=0; h<VECSIZ; h++)
     matrix[j][h] = 0.;    /* initialise the matrix array */

 for(j=0; j<VECSIZ; j++)
   {
    vector[j] = 0.;        /* initialise the two vector arrays */
    vector1[j] = 0;
   }

 h=0;
 while (hist[h++] <= 0)   /* find the start of the curve */
 l1 = h;

 for(h=255; h>=0; h--)   /* find the end of the curve */
   if (hist[h] != 0)
    {
     l2 = h;
     break;
    }
```

```
l = l2 - l1;   /* compute the spread of the curve */

if ( l > 32)            /* case of the spread greater than 32 */
 {          /* get the scale for enlargement and the first position */
  scale =(FTYPE)l/32;
  y = l1; m = l1; j = 0;  /* of the curve */
  while (y <= (FTYPE)l2)  /* process till the end of the curve */
  {  /* get the number of points in that section */
   k=0; val1 = 0;
   y += scale;    /* get the border of the next section */
   for (h=m; h<= y; h++,k++)
     val1 += hist[h];     /* compute the total y values */
   if (j >= VECSIZ)    /* stop if overflow */
       break;
   vector[j++] = (FTYPE)val1/k;  /* find the mean value */
    m = h;    /* get the x-cordinates for the section */
  }  /* end of the while */
 }   /* end of the if */
else if ( l < 32 )      /* case of the spread less than 32 */
      {      /* get the scale for reduction and the first position */
       scale = 32/(FTYPE)(l+1); y1 = 0; y = 0; j = l1;  /* of the curve */
       while (j <= l2)  /* process till the end of the curve */
       {
        y += scale;   /* move to the next section of the divided 32 array */
        if (y1 >= VECSIZ)  /* stop if overflow */
         break;
        vector[y1] =(FTYPE)hist[j]; y1++; /* get the value for the first cor */
        if (y1 >= VECSIZ)
         break;
        if ((FTYPE)y1 <= y)       /* if not end of the section */
         {
          vector[y1] =(FTYPE)(hist[j] + hist[j+1])/2; /* interpolate for the next value */
          y1++;
         }
        j++;    /* move to the next section in the original array*/
       } /* end of the while */
      }  /* end of the second if */
else
    { /* transfer the points to the vector array */
     for (j=0, y1=l1; j<VECSIZ; j++,y1++)
       vector[j] = hist[y1];
    }

max1 = 0.;
for (h=0; h<VECSIZ; h++)
  {
   if (vector[h] > max1)
    max1 = vector[h];
/*   printf("\n%3.2f",vector[h]); */
  }

for (h=0; h<VECSIZ; h++)
  {
   if (vector[h] != 0.)
     vector[h] = vector[h]/max1;

   temp = (vector[h]*10) + 0.5;  /* this is to ensure proper round off */
   vector1[h] = floor(temp);

   matrix[vector1[h]][h] = 1.0;  /* use 1.0 to define the point on curve */
   /* matrix[vector1[h]][h] = vector[h];  */
  }

}  /* end of cmap */
```

```
/****************************************/
 * this is the main driver of the program
 ****************************************/

void main (argc, argv)
int    argc;
char   **argv;
{
 FTYPE  amax, tsserror, num;
 int    nit, n, i, j, maxunit;
 int    cmax, val, par, nopar;
 char   name[100];

 int    wtfil = 0, wormfil = 0, sec = 0; /* flags and number of sections */

        if (argc == 4)
        {
         wtfil   = atoi (argv[1]);
         wormfil = atoi (argv[2]);
         sec     = atoi (argv[3]);

                if (wtfil < 0 || wtfil > 99)
                        errmsg ("weight file # must be between 0 and 99");
                if (wormfil < 0 || wormfil > 99)
                        errmsg ("worm file # must be between 0 and 99");
                if (sec <= 0)
                        errmsg ("number of sections must be greater than 0");
        }
        else
            errmsg ("Usage: histest  <#for weight file> <#for cod image file> <# sections in image>");

        strcpy (name, WT);
        strncat (name, argv[1], 2);

        if ((inp = fopen (name, "rb")) == 0)
                errmsg ("Error opening weight file for weights");

        strcpy (name, CODFIL);
        strncat (name, argv[2], 2);

        if ((datp = fopen (name, "r")) == 0)
                errmsg ("Error opening cod image file for image patterns");

        fread (&nit, sizeof (int), 1, inp);

        fread (&layers, sizeof (int), 1, inp);
        if ((units = (int *) malloc (layers * sizeof (int))) == 0)
                errmsg ("units malloc error");

        fread (units, sizeof (int), layers, inp);

        maxunit = 0;
        for (n = 0;n < layers;n++)
                if (units[n] > maxunit)
                  maxunit = units[n];

       /* allocate memory */
        input = (FTYPE **)calloc(2,sizeof(FTYPE));

        for (i=0; i<2; i++)
        { /* get the memory for input nodes */
         input[i] = (FTYPE *)calloc(units[0]+1,sizeof(FTYPE));
         if (input[i] == 0)
          {
           printf("\nNot enough memory for inputs");
```

```
   exit(0);
  } /* end of if */
} /* end of for */

output = (FTYPE **)calloc(layers+1,sizeof(FTYPE));

for (i=0; i<layers; i++)
{ /* get memory for the activations for layers after the first */
 output[i] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
 if (output[i] == 0)
  {
   printf("\nNot enough memory for activations");
   exit(0);
  } /* end of if */
 } /* end of for */

w = (FTYPE ***)calloc(layers, sizeof(FTYPE));

for (i=0; i<layers - 1; i++)  /* loop thru the # of layers */
{   /* get the wts from a node to nodes in the next layer */
 w[i] = (FTYPE **)calloc(units[i+1] + 1,sizeof(FTYPE));
 for (j=0; j<units[i+1]; j++)  /* loop thru nodes of next layer */
 {
  w[i][j] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
  if (w[i][j] == 0)
   {
     printf("\nNot enough memory for weights");
     exit(0);
   } /* end of if */
  } /* end of 2nd for */
 } /* end of 1st for */

 for (n = 0;n < layers - 1;n++)
        for (i = 0; i < units[n + 1]; i++)
                fread (w[n][i], sizeof (FTYPE), units[n], inp);

 fread (&tsserror, sizeof (FTYPE), 1, inp);

 for (n = 0; n < sec; n++)     /* go through the windows sections */
   {
     for (i=0; i<ARRSIZ; i++)      /* initialise the hist array */
         hist[i]=0;
     for (i=0; i<1024; i++)    /* compute the cum freq of the 32x32 */
      {
       val = fgetc(datp);    /* input data of the image */
       hist[val]++;
      }            /* if at end of file then stop */

    for (i=0; i<ARRSIZ; i++)
       printf("\n%d ",hist[i]);  */

     cmap();  /* compute the features of the curve */
     i = 0;
     for (i = 0; i < CORD; i++)  /* read the input */
       for (j =0; j < VECSIZ; j++)
         input[one][i++] = matrix[i][j];

     input[one][units[0] - 1] = 1.;  /* assign act for the threshold */

     compute_out(one);  /* calculate the activations */

         /* find most active output neuron */
     amax = output[layers - 1][0];
     cmax = 0;
```

```
        for (i = 1; i < units[layers - 1]; i++)
            if (output[layers - 1][i] > amax)
              {
                amax = output[layers - 1][i];
                cmax = i;
               }

         if ( (cmax +1) == 1)
          par++;      /* if a par is found then increase par counter */
         else
            nopar++;  /* otherwise increase the no parasite counter */
                     /* display the section # and its classification */
       /*printf("\npattern %d. target %d. amax %lf.\n",n+1,cmax+1,amax);*/
    }
                     /* display the number of parasite and non parasite sections */
    printf("\n Parasites = %d.  No parasites = %d.\n",par,nopar);


    fclose(inp);
    fclose(outp);
    fclose(datp);

}   /* end of histest.c */
```

```
/****************************************************************
*  Department of Computer Science and Systems
*  McMaster University, Hamilton, Ontario, Canada
*
*  This file contains the classification program for the smoothed
*  curve format.
*
*  By:    Emmanuel B. Aryee
*
*  Date:  September, 1990
****************************************************************/

/**********************
 * smthtest.c    this is a production program that classifies the sections
 *               of a cod image into parasitic sections and non parasitic
 *               sections using the smoothed curve format
 ***************************/

#include "prod.h"

#define ARRSIZ 255
#define VECSIZ 32
#define CODFIL "worms.da"
#define WT "weight.fil"

FTYPE       ***w, **output, **delta, **input, vector[VECSIZ], smooth[VECSIZ];
int            layers, sec, one=1, *units;
int         hist[ARRSIZ];
FILE          *inp, *outp, *datp;

/*****************************************
 * smoothed():  this is used to compute the cumulative freq of an image
 *              and extract the normalised features of the curve which
 *              is then smoothed to remove the spikes.
 *********************************************/

void smoothed( )
{
 int h=0,l1=0,l2=0,l=0,j=0,m=0,y1=0, val1;
 FTYPE scale=0., y=0., k=0., max1=0.,temp=0.;

 h=0;
 while (hist[h++] <= 0)   /* find the start of the curve */
 l1 = h;

 for(h=255; h>=0; h--)   /* find the end of the curve */
   if (hist[h] != 0)
    {
     l2 = h;
     break;
    }

 l = l2 - l1;   /* compute the spread of the curve */

 if ( l > 32)            /* case of the spread greater than 32 */
  {          /* get the scale for enlargement and the first position */
   scale =(FTYPE)l/32;
   y = l1; m = l1; j = 0;  /* of the curve */
   while (y <= (FTYPE)l2)  /* process till the end of the curve */
   {  /* get the number of points in that section */
    k=0; val1 = 0;
    y += scale;    /* get the border of the next section */
    for (h=m; h<= y; h++,k++)
      val1 += hist[h];    /* compute the total y values */
    if (j >= VECSIZ)    /* stop if overflow */
```

```
        break;
    vector[j++] = (FTYPE)val1/k;  /* find the mean value */
     m = h;    /* get the x-cordinates for the section */
   }  /* end of the while */
  }   /* end of the if */
 else if ( l < 32 )      /* case of the spread less than 32 */
       {      /* get the scale for reduction and the first position */
        scale = 32/(FTYPE)(l+1); y1 = 0; y = 0; j = l1;  /* of the curve */
        while (j <= l2)  /* process till the end of the curve */
        {
         y += scale;   /* move to the next section of the divided 32 array */
         if (y1 >= VECSIZ)  /* stop if overflow */
          break;
         vector[y1] =(FTYPE)hist[j]; y1++; /* get the value for the first cor */
        if (y1 >= VECSIZ)
          break;
         if ((FTYPE)y1 <= y)       /* if not end of the section */
          {
           vector[y1] =(FTYPE)(hist[j] + hist[j+1])/2; /* interpolate for the next value */
           y1++;
          }
         j++;    /* move to the next section in the original array*/
        } /* end of the while */
       }  /* end of the second if */
 else
     { /* transfer the points to the vector array */
      for (j=0, y1=l1; j<VECSIZ; j++,y1++)
        vector[j] = hist[y1];
     }

 max1 = 0.;
 for (h=0; h<VECSIZ; h++)
   {
    if (vector[h] > max1)
     max1 = vector[h];
 /*   printf("\n%3.2f",vector[h]); */
   }

 for (h=0; h<VECSIZ; h++)
   {
    if (vector[h] != 0.)
      vector[h] = vector[h]/max1;
   }
          /* smoothing of the curve begins */
 for (h=0; h<(VECSIZ - 1); h++)
   {
    temp = vector[h+1] - vector[h];     /* get the difference */
    if (temp >= 0.045)      /* if next point is greater */
      smooth[h+1] = smooth[h] + 0.1;  /* increase that point by 0.1 */
    else if (temp <= -0.045) /* if less */
        smooth[h+1] = smooth[h] - 0.1; /* decrease that point by 0.1 */
    else                             /* if same */
        smooth[h+1] = smooth[h];       /* assign the same value */
   }
} /* end of smoothed */

/****************************************/
 * this is the main driver of the program
 ****************************************/

void main (argc, argv)
int    argc;
char   **argv;
{
 FTYPE  amax, tsserror, num;
```

```
int     nit, n, i, j, maxunit;
int     cmax, val, par, nopar;
char    name[100];

int     wtfil = 0, wormfil = 0, sec = 0; /* flags and number of sections */

        if (argc == 4)
        {
         wtfil   = atoi (argv[1]);
         wormfil = atoi (argv[2]);
         sec     = atoi (argv[3]);

                if (wtfil < 0 || wtfil > 99)
                        errmsg ("weight file # must be between 0 and 99");
                if (wormfil < 0 || wormfil > 99)
                        errmsg ("worm file # must be between 0 and 99");
                if (sec <= 0)
                        errmsg ("number of sections must be greater than 0");
        }
        else
            errmsg ("Usage: smthtest  <#for weight file> <#for cod image file> <# sections in image>");

        strcpy (name, WT);
        strncat (name, argv[1], 2);

        if ((inp = fopen (name, "rb")) == 0)
                errmsg ("Error opening weight file for weights");

        strcpy (name, CODFIL);
        strncat (name, argv[2], 2);

        if ((datp = fopen (name, "r")) == 0)
                errmsg ("Error opening cod image file for image patterns");

        fread (&nit, sizeof (int), 1, inp);

        fread (&layers, sizeof (int), 1, inp);
        if ((units = (int *) malloc (layers * sizeof (int))) == 0)
                errmsg ("units malloc error");

        fread (units, sizeof (int), layers, inp);

        maxunit = 0;
        for (n = 0;n < layers;n++)
                if (units[n] > maxunit)
                  maxunit = units[n];

      /* allocate memory  */

        input = (FTYPE **)calloc(2,sizeof(FTYPE));

        for (i=0; i<2; i++)
        {  /* get the memory for input nodes */
         input[i] = (FTYPE *)calloc(units[0]+1,sizeof(FTYPE));
         if (input[i] == 0)
          {
           printf("\nNot enough memory for inputs");
           exit(0);
          }  /* end of if */
        }  /* end of for */

        output = (FTYPE **)calloc(layers+1,sizeof(FTYPE));

        for (i=0; i<layers; i++)
        {  /* get memory for the activations for layers after the first */
```

```
 output[i] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
 if (output[i] == 0)
  {
   printf("\nNot enough memory for activations");
   exit(0);
  } /* end of if */
 } /* end of for */

w = (FTYPE ***)calloc(layers, sizeof(FTYPE));

for (i=0; i<layers - 1; i++)  /* loop thru the # of layers */
{   /* get the wts from a node to nodes in the next layer */
 w[i] = (FTYPE **)calloc(units[i+1] + 1,sizeof(FTYPE));
 for (j=0; j<units[i+1]; j++)  /* loop thru nodes of next layer */
 {
  w[i][j] = (FTYPE *)calloc(units[i]+1,sizeof(FTYPE));
  if (w[i][j] == 0)
   {
     printf("\nNot enough memory for weights");
     exit(0);
   } /* end of if */
  } /* end of 2nd for */
 } /* end of 1st for */

 for (n = 0;n < layers - 1;n++)
        for (i = 0; i < units[n + 1]; i++)
                fread (w[n][i], sizeof (FTYPE), units[n], inp);

 fread (&tsserror, sizeof (FTYPE), 1, inp);

 for (n = 0; n < sec; n++)     /* go through the windows sections */
   {
     for (i=0; i<ARRSIZ; i++)      /* initialise the hist array */
         hist[i]=0;
     for (i=0; i<VECSIZ; i++)      /* initialise the vector array */
       {
        smooth[i] = 0.;            /* and the smooth array */
         vector[i]=0.;
       }
     for (i=0; i<1024; i++)    /* compute the cum freq of the 32x32 */
      {
       val = fgetc(datp);     /* input data of the image */
       hist[val]++;
      }              /* if at end of file then stop */


     smoothed();  /* compute the features of the curve */
     for (i = 0; i < units[0] - 1; i++)  /* read the input */
        input[one][i] = smooth[i];

     input[one][units[0] - 1] = 1.;  /* assign act for the threshold */

     compute_out(one);  /* calculate the activations */

         /* find most active output neuron */
     amax = output[layers - 1][0];
     cmax = 0;

     for (i = 1; i < units[layers - 1]; i++)
         if (output[layers - 1][i] > amax)
           {
            amax = output[layers - 1][i];
            cmax = i;
           }
```

```
        if ( (cmax +1) == 1)
         par++;      /* if a par is found then increase par counter */
        else
           nopar++;  /* otherwise increase the no parasite counter */
                     /* display the section # and its classification */
      /*printf("\npattern %d. target %d. amax %lf.\n",n+1,cmax+1,amax);*/
    }
                     /* display the number of parasite and non parasite sections */
    printf("\n Parasites = %d.  No parasites = %d.\n",par,nopar);

    fclose(inp);
    fclose(outp);
    fclose(datp);

}   /* end of smthtest.c */
```