The Use of Artificial Neural Networks in Classifying Lung Scintigrams Volume 1 (of 2)

Denis Anthony

Thesis Submitted for the Award of PhD in Engineering
Department of Engineering
Warwick University
February 1991
Supervised by Dr. E. L. Hines

SUMMARY

An introduction to nuclear medical imaging and artificial neural networks (ANNs) is first given.

Lung scintigrams are classified using ANNs in this study. Initial experiments using raw data are first reported. These networks did not produce suitable outputs, and a data compression method was next employed to present an orthogonal data input set containing the largest amount of information possible. This gave some encouraging results, but was neither sensitive nor accurate enough for clinical use.

A set of experiments was performed to give local information on small windows of scintigram images. By this method areas of abnormality could be sent into a subsequent classification network to diagnose the cause of the defect. This automatic method of detecting potential defects did not work, though the networks explored were found to act as smoothing filters and edge detectors.

Network design was investigated using genetic algorithms (GAs). The networks evolved had low connectivity but reduced error and faster convergence than fully connected networks. Subsequent simulations showed that randomly partially connected networks performed as well as GA designed ones.

Dynamic parameter tuning was explored in an attempt to produce faster convergence, but the previous good results of other workers could not be replicated.

Classification of scintigrams using manually delineated regions of interest was explored as inputs to ANNs, both in raw state and as principal components (PCs). Neither representation was shown to be effective on test data.

VOLUME 1 : THESIS

VOLUME 2 : APPENDICES

Table of Contents

List of Tables

List of Figures

Acknowledgments

Dr. David Taylor and Mr. John Barham of the Nuclear Medicine Department, Walsgrave Hospital, Coventry for their help and advice in this study, and assistance in experimental procedures.

Mr Philip Neaves for allowing access to impedance images generated by him, advice on the technique of impedance imaging, and assistance with several headaches.

My parents for their support throughout my studies.

Key to Abbreviations

ADALINE	Adaptive Linear Element
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Anterior-posterior
ART	Adaptive Resonance Theory
BAM	Bi-directional Associative Memory
BP	Back Propagation
CVS	Cardiovascular System
COAD	Chronic Obstructive Airways Disease
DVT	Deep Vein Thrombosis
GA	Genetic Algorithm
LL	Left Lateral
LPO	Left Posterior Oblique
MAA	Macroaggregates of Albumin
MADALINE	Multiple Adaptive Linear Element
MLP	Multilayer Perceptron
PA	Posterior-anterior
PC	Principal Component
PCA	Principal Components Analysis
PE	Pulmonary Embolus
PTE	Pulmonary Thromboembolism
PTT	Prothrombin Time
RL	Right Lateral
ROC	Receiver Operating Characteristic
RPO	Right Posterior Oblique
SLE	Systemic Lupus Erythematous
SNR	Signal to Noise Ratio

TB	Tuberculosis
psse	Pattern Sum Square Error
ntsse	Normalised Total Sum Square Error
tsse	Total Sum Square Error

1. OVERVIEW

Previous work by the author[1] , attempted to process scintigraphic images to improve the diagnostic results of human experts. Various colour scales and pre-processing techniques were applied to images to hopefully make abnormalities in images easier to detect. This was not successful. It may be that humans act as sophisticated pre-processors to such an extent that the filters and colour scales used offered no improvement. The failure to improve on human perception by pre-processing led to considering emulating, rather than aiding, the human cognitive skills. To this end an artificial neuron based system was considered.

The aim of this study is to produce a system for classifying nuclear images. An ideal system would take pixel values, which here would relate to the radioactive counts of a scintigram, as data for input, and produce a diagnosis which is both highly sensitive and totally accurate. It would provide the information required by clinicians to enable them to offer the most appropriate treatment to patients. Interrogation of the system should ideally offer rationale and reasoning behind the advice given by the system. The system should work in real time, and not put any extra burden on clinical or other staff.

Many of the above requirements conflict. Accuracy against sensitivity is one such situation, where the more sensitive a system is to a diagnosis (ie. the more likely it is to give a positive result when a condition exists) the less accurate it becomes, as it starts to give false positives. Ie. the ratio of true positives to false positives increases, and true negatives to false negatives decreases as the confidence point above which one declares a diagnosis is raised, and conversely the true positives to false positives decreases, and true negatives to false negatives increases as the confidence point above which one declares a diagnosis is lowered.

Using different methodologies various of the requirements are met with differing levels of success, and a review of the various options showed systems that are optimal in one or more areas. A compromise solution was accepted that did not address the capability to interrogate the system for reasoning, but was fully automatic and did not require user input. Conventional expert systems have been fed pre-processed data e.g[2]. and the proposed system could be embedded in an expert system to provide such information interactively if this was required at a later stage. A similar approach has been implemented by other workers[3], for example, who used an ANN as one component in an expert system.

The following sections will describe the problems and potential solutions in nuclear medical imaging (Chapter 2 and 3), and neural networks (Chapters 4 and 5) and then the current work will be fully discussed (Chapters 6 - 13). Finally a conclusion is presented (Chapters 14).

The thesis will cover the following areas :-

1.1. Nuclear Imaging (Chapter 2)

A general overview of scintigraphic techniques as applied to human studies for diagnostic purposes.

1.2. Lung Scintigrams (Chapter 3)

The particular modality of pulmonary scintigrams will be discussed.

1.3. Introduction to Artificial Neural Networks (Chapters 4 and 5)

Artificial Neural Networks (ANNs) are discussed in general terms, and in some detail the particular model, multilayer perceptrons (MLPs) employing error back propagation, used in this study.

1.4. ANNs Using Raw Data (Chapter 6)

ANNs were utilised to classify lung scintigrams. Initially unprocessed data was used, ie. the pixel values of the image in their raw state. This was unsatisfactory, and the results are presented in this chapter.

1.5. Data Compression Using Principal Components Analysis and Artificial Neural Networks (Chapter 7)

Principal Components Analysis (PCA) gave the most important principal components (PCs) of the data space of scintigram images to allow data compression. This technique was compared with a more novel approach using ANNs.

1.6. ANNs Using Principal Components (Chapter 8)

PCA was shown to be better at compressing images with respect to total sum square error (tsse), and thus PCs were used to provide a reduced data input to an ANN. Some limited ability was discovered in classifying images.

1.7. Dynamic Parameter Tuning of ANNs (Chapter 9)

The dynamic tuning of learning rate and momentum in MLPs using error back propagation was explored. This used data from impedance imaging supplied by an MSc student.

1.8. Local Prediction Techniques (Chapter 10)

Scanning images with a window of an appropriate size to include a typical abnormality was here explored with a view to producing a system which flagged areas of an image with a potential defect.

1.9. Optimising Network Structures With Genetic Algorithms (Chapter 11)

The design of ANNs was automated using Genetic Algorithms (GAs).

1.10. Reducing Connectivity in Compression and Prediction Neural Networks (Chapter 12)

The connectivity of nets is relevant to the speed of execution, and reduced connectivity nets are less likely to fall into uninteresting one to one mappings. Fully connected networks require larger training sets than sparsely connected ones. This chapter explores more sparsely connected networks.

GAs were compared with randomly allocated network structures, which were found to be roughly equivalent to GA designed nets.

1.11. Classification of Defects Using Local Techniques (Chapter 13)

The prediction technique failed to give useful segments for further classification, and areas were subsequently manually identified for the classification network. The classification success rate is presented.

1.12. Conclusion (Chapter 14)

An overall statement of the work is presented.

2. NUCLEAR AND IMPEDANCE IMAGING

2.1. Nuclear Imaging

Nuclear imaging is a particular medical imaging technique, where gamma rays are employed, in place of X-Rays, ultrasonic beams etc. A gamma ray emitting radio-substance with a short half-life is introduced into a patient via intravenous injection inhalation or some other method. Gamma rays are collected by a crystal which is connected to a battery of photo-multipliers. The intensity of the rays and the position of emissions may be calculated by the relative intensities found at the photo-multipliers.

The substance used will distribute itself according to structure and function of the body. Whereas in conventional radiographs (X-Rays) anatomical structure is seen, in scintigraphy function of an organ is more important in determining an image.

Radio-pharmaceuticals used in pulmonary scintigraphy include [4] :

Particles $^{99m}Tc\text{-}HAM$, $^{99m}Tc\text{-}MAA$

Gases ^{133}Xe, ^{15}O, $^{12}CO_2$, ^{11}C *in* CO.

Aerosols ^{99m}Tc.

2.1.1. Display of Images

A problem with nuclear imaging is that the signal to noise ratio (SNR) is low and images are difficult to interpret. Once the image has been collected the display of the image will be affected by the resolution of the screen, the number of grey levels or colours allocated, and the contrast. These parameters may be optimised, but the improvement is limited by the quality of the original data.

2.1.2. Subjectivity of Images

The images produced by nuclear medicine in common with most medical images have the human perceiver as a component in the imaging apparatus. While some quantitative methods are employed (e.g. cardiac ventricular function testing) most images are viewed by a clinical expert to qualitatively reach a diagnosis. In these cases it is therefore not sufficient to evaluate equipment alone, the assessments of observers should be tested. A clinical trial of a system using observers can make use of Receiver Operating Characteristic (ROC) curves[5] (e.g.) to judge the efficacy of an image enhancement technique. Automatic systems may similarly tested, and be compared with observers using ROC analysis.

2.1.3. Physical Limitations of Equipment

The equipment used in scintigraphy is finite in its resolution capabilities, and this will influence the degree to which software can manipulate the image meaningfully. The resolution is determined by the dimensions of the collimators. The longer the collimator, and the smaller its cross sectional area, the more scattered radiation is shielded, but the lower the count. As the radioactive process is a random one, which may be modeled by Poisson noise, very low counts become increasingly affected by background noise. The SNR is thus lowered as resolution increases, and a compromise has to be struck between the two features.

2.1.4. Accuracy of Diagnosis

Nuclear images contain significant levels of noise, and may be difficult to interpret. The use of experts in diagnosing such images is therefore necessary, but even experts disagree with each other on some diagnoses, and one expert may give different opinions on the same image when presented with it on separate occasions.

2.1.5. Automatic Diagnostic System

It would appear advantageous to build an automatic reporting system for nuclear images. Such a system would give more consistent reporting, and may give more accurate reporting. It would also allow analysis of the parameters used to diagnose an image.

Pulmonary radiographs have been classified using texture feature and moment parameter feature analyses[6]. Systems have been built which perform automatic reporting of nuclear images, e.g[7]. and[8].

These systems are tailor made for specific organs, and do not readily transfer to other organs. Specific knowledge is assumed about the organ under consideration, for example[9].

Expert systems have been utilised to give reports[9] , but these are also based on specific information known about the domain under consideration.

Individualised systems are time-consuming to build, and are hence expensive. Rules must be known, stated explicitly, and correctly. One flaw in the program may allow serious errors to result. Consequently the systems are not necessarily robust. The use of fuzzy logic and/or Bayesian statistics can assist in areas where the knowledge is uncertain. Conventional computing techniques may give.optimal solutions to particular problems, but are unlikely to be applicable to other problem areas.

For reasons discussed in later chapters (Chapters 4 and 5) ANNs may offer a more general purpose but robust framework around which to design automatic detection systems.

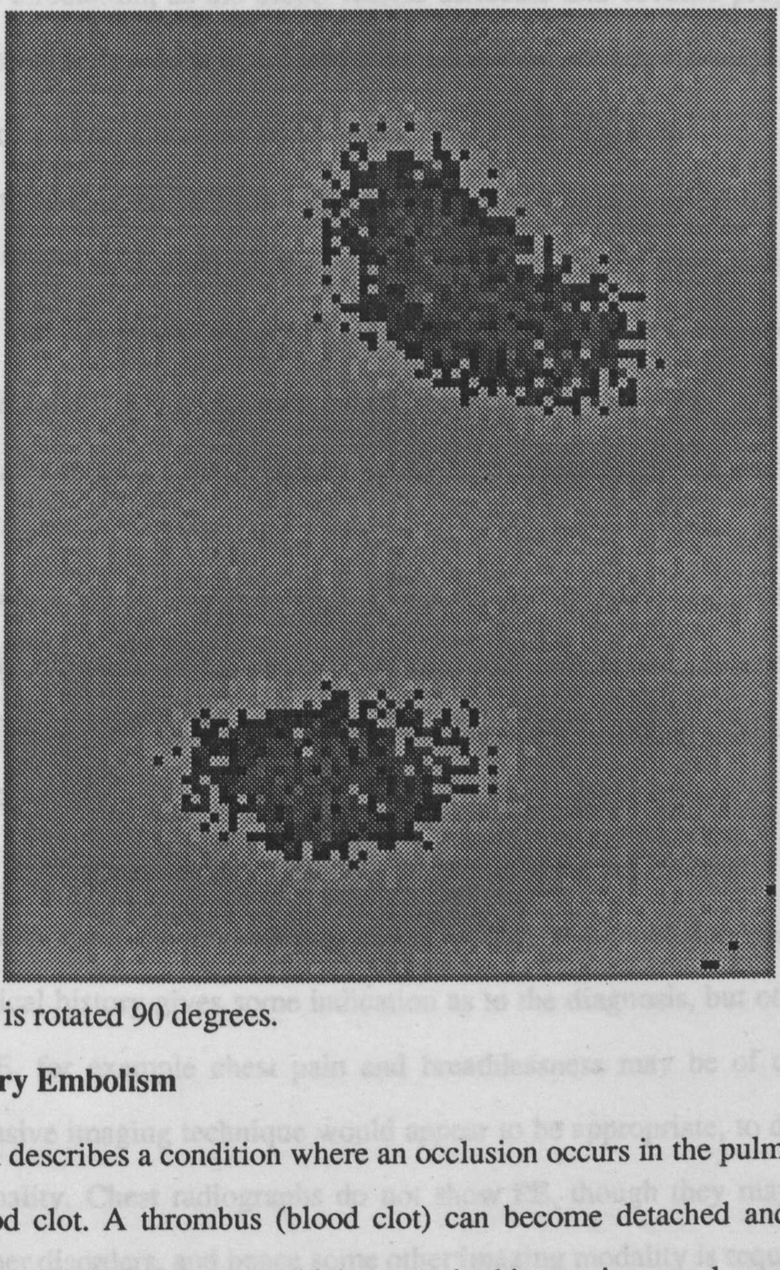
2.2. Impedance Imaging

Images may be constructed from electrical measurements of an organ. The technique consists of surrounding the area to be imaged with a number of electrodes, applying a constant current source across two of the electrodes, and measuring the voltage at all other electrodes[10]. In principle the conductivity distribution may be calculated from these measurements. The mathematics involved in however non-trivial and not fully developed. The relatively new technique of impedance imaging offers several promising advantages. Such systems would be cheap to produce, and could be used for continuous monitoring. Similar problems concerning resolution, subjectivity, physical resolution and accuracy apply to impedance imaging as to nuclear imaging. In a related study by an MSc student this imaging technique has been explored, and the use of ANNs in solving the inverse problem has been attempted[11]. Further analysis of the data from that study has been undertaken, and is explored in Chapter 9.

3. LUNG SCINTIGRAMS

Lung scintigrams, an example of which appears in Figure 3.1, are used for the diagnosis of pulmonary emboli (PE, also known as pulmonary thromboembolism PTE).

Figure 3.1 Pulmonary Scintigram Image



nb. The image is rotated 90 degrees.

3.1. Pulmonary Embolism

This term describes a condition where an occlusion occurs in the pulmonary circulation by a blood clot. A thrombus (blood clot) can become detached and move in the direction of blood flow, a thrombus which moves in this way is termed an embolus. In PE the emboli arise typically from detached thrombi of the deep veins of the leg (deep vein thrombosis or DVT).

DVT is classically a condition associated with immobility, and is common in debilitated, immobile patients. Post-operative patients are at risk from DVT, and the risk increases the more their surgery renders the patient bedridden.

The embolus from a DVT passes through the heart, and is most likely to lodge in the pulmonary circulation, as the blood vessels bifurcate and become progressively narrower. Eventually one is encountered that the embolus cannot pass through.

The effect of an embolus is variable, depending upon the size of the vessel occluded, the number of emboli (multiple emboli are common) and the general condition of the patient. Thus the clinical picture may be anything from asymptomatic, through to chest pain and breathlessness, and to collapse, respiratory arrest and imminent death.

The treatment for PE is anti-coagulant therapy, using enzymes (heparin) or drugs (e.g. warfarin). Failure to treat is dangerous as subsequent PE are not uncommon, and anti-coagulant treatment additionally to removing emboli acts as a prophylaxis for such sequalae. However aggressive treatment of suspected PE from clinical history unsupported from objective tests is inappropriate as post-surgical patients are at risk from internal bleeding if their clotting time (Prothrombin Time, PTT, the time taken for blood to form a clot) increases.

There is an obvious need to get the diagnosis right, to correctly identify those patients who will benefit from anti-coagulant therapy, and those for whom it is inappropriate. Clinical history gives some indication as to the diagnosis, but other conditions may mimic PE, for example chest pain and breathlessness may be of cardiac origin. Some non-invasive imaging technique would appear to be appropriate, to directly visualise the abnormality. Chest radiographs do not show PE, though they may be useful in eliminating other disorders, and hence some other imaging modality is required.

3.2. Scintigraphic Techniques

A pulmonary scintigram of the blood vessels (perfusion scintigram) may be obtained by injecting the patient with a radio-isotope. ^{99m}Tc microspheres are used, but more commonly ^{99m}Tc labelled macroaggregates of albumin (MAA) are employed. These have an average size of 10 - 40 µm, and pass into the pulmonary circulation where the obstruction of a small number of vessels "fixes" them[12] In a normal subject the uptake of the isotope is seen over the whole lung field, with the exception of a cardiac shadow (the heart overlies the lungs and shields part of the radiation output) in the anterior view on the left of centre.

The expert who views scintigraphic images looks for specific patterns which are indicative of known diseases or conditions. For instance the outlines of the lungs should be smooth; in the posterior view the lungs should come down to the same level. A normal image, showing these and other attributes effectively excludes the diagnosis of PE.

In PE areas of reduced uptake are seen where the isotope cannot reach the blood supply of a vessel occluded by an embolus. Unfortunately other conditions may produce such areas, examples of which are given in Table 3.1 :-

Table 3.1 Disorders Seen in Lung Scintigrams
Chronic obstructive airways disease (COAD)
Pleural effusion.
Bronchospasm
Asthma
Pneumonia with Consolidation
Cardiomegaly
Tuberculosis (TB)
Venous Hypertension
Extrapulmonary Abnormalities (e.g. aneurysm)
Tumours

The defects may be segmental, subsegmental or non segmental. Segmental defects are indicative of PE, non segmental defects suggest other pathologies, and sub segmental defects are non specific.

In PE the circulation is affected, but the ventilation system of bronchi, bronchioles and alveoli are either unaffected, or affected to a much lesser degree. Thus a ventilation scintigram will show a mismatch in the areas of reduced uptake, but other conditions do not show this mismatch. Consequently a ventilation study is often performed using a variety of agents with different advantages and restrictions, examples of such agents are

^{133}Xe, ^{127}Xe, ^{81m}Kr, ^{99m}Tc *DTPA* aerosol[12].

Several views are taken as the defect may be easily seen in one view, but obscured in another. Typical views are anterior-posterior (AP), posterior-anterior (PA), right posterior oblique (RPO) and left posterior oblique (LPO).

3.3. Diagnosis of PE

In addition to the factors mentioned above, the clinical history may indicate PE. Chest X Ray may provide useful additional information. The following patterns in Table 3.2 suggest different evaluations for PE. In this table rough probabilities are given, e.g. $p < 0.05$ PE, which means the probability of PE is less than 0.05, or only 5% likely[12].

Table 3.2 Evaluation of PE

Scan	Chest X Ray	Interpretation
Normal	Normal	No PE
Subsegmental perfusion defects or defects with matched ventilation defects	Normal	$p < 0.05$ PE
Segmental, lobar or large non-segmental matched defects	Normal	$p < 0.05$ PE
Bilateral segmental unmatched perfusion defects	Normal	$p > 0.95$ PE
One lung with matched defects, the other with multiple segmental unmatched pulmonary defects	Corresponding abnormality with matched defect	$p > 0.95$ PE Probable pulmonary infarct
Multisegmental perfusion defect in one lung only with normal ventilation	Normal	$p > 0.5$ PE
Matched perfusion/ventilation abnormality	Associated defect	$p > 0.3$ (non diagnostic) Pulmonary angiography may be required

The literature survey has thrown up some rules that may be used in diagnosing PE, and some general comments concerning such diagnosis. For example it is unusual for ventilation images to show defects that are not seen in perfusion images, though this does sometimes happen in pneumonia. Where matched defects are seen typically the perfusion defects are more pronounced. In COAD the primary defect is in the airways obstruction, but this causes a secondary arteriolar obstruction. In PE sometimes minor ventilation defects are seen which correspond to infarction (where the blood supply to an area is totally occluded, and the tissue supplied by that vessel consequently dies). Artificial mismatching may occur in severe parenchymal disease due to clumping of tracer in major bronchi and branches[12].

In addition to a literature survey, medical physics departments were visited in the West Midlands, to interview experts and obtain local information. This information could be used to build an expert system (for an introduction to expert systems in medical systems see [13]), of which the ANN is a constituent part. The details are given in Appendix 5.

4. INTRODUCTION TO ANNs

ANNs are biologically inspired algorithms, where ideas taken from nervous system anatomy and physiology have been used to attempt to simulate a (very simplified) artificial nervous system. The reason for this project is to build systems which are capable of performing actions or analysis that conventional computer programs are poor at, but which living organisms perform well. Introductions to this multi-disciplinary branch of science can be found in[14] , and [15] (e.g).

ANNs are difficult to analyse, and most work has been empirical, based on simulations. These have frequently been "toy" problems such as the XOR problem, found in[16] , or grossly simplified problems, e.g. the synthetic amoeba which learns to find food[17] , and the learning of linguistic concepts (e.g.), , and to represent conceptual structures[18] [19], . ANNs have been implemented in realistic problem domains, to build knowledge processors[20] and for industrial scene analysis (e.g.)[21].

4.1. What is a Network, What are Units

In an analogy with the nervous system of animals, artificial neurons (often called units) are linked by artificial axons (usually called links). The units and links are grossly simplified compared with real neuronal systems. The units are often identical in structure, or consist of only a few different types. Real neurons are in many cases highly specialised although the cerebral cortex contains cells which are not highly differentiated, and so there is some justification for using identical units to model cognitive functions. The links are typically simple summing devices, where the output of several units are weighted by some value, which is potentially different for each unit, and adaptable, and some function is often then applied to the sum. A squashing function such as the logistic function is popular as it ensures outputs that are in a well defined range only are allowed. The logistic function is defined by :

$$f(x)=\frac{1}{1+e^{-x}}$$

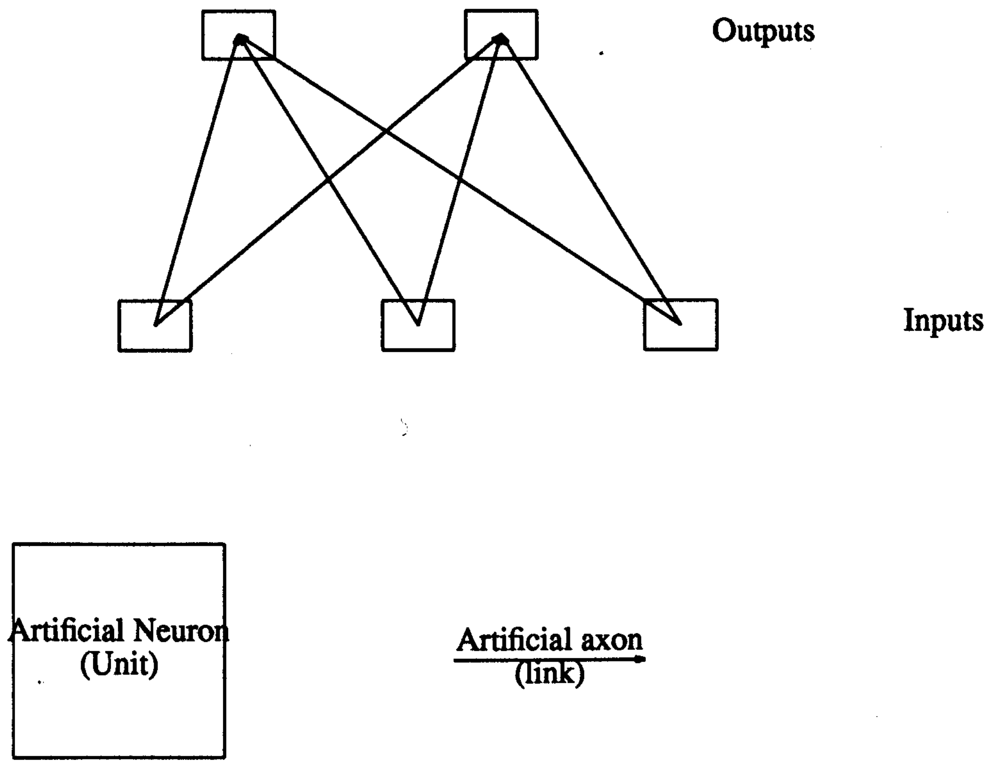
where in this case x is the output value of the unit.

The simplicity of activation functions of artificial networks may be compared with the tens to hundreds of thousands of different neurotransmitters (the number is not known even approximately) found in the human nervous system.

Even these simple artificial neural networks (ANNs) are difficult to analyse mathematically, and before one attempts to employ more complex systems, much work needs to be undertaken both theoretically and experimentally to understand simple systems.

A typical network structure is shown in Figure 4.1.

Figure 4.1 Simple Linear Network



$$o_j = \sum w_{ij} \times i_i$$

where i_i is an input and o_j is an output.

Hebb [22] proposed that if neurons are simultaneously activated, the connections between them become stronger. This rule is the basis of many ANN algorithms, several of which are discussed below. So-called Hebbian learning is epitomised by the update rule for a change in weight Δw :

$$\Delta w_{ij} = \eta a_j t_i \tag{4.1}$$

where a_j = activation of a unit j, t_i = target value of unit i, η= learning rate and w_{ij} = weight between input j and target i[23].

The above would be used to associate an input with a target output, where connections (weights) are made between the units (which are simulations of neurons, grossly simplified). This rule works for linear relations where the input vectors are mutually orthogonal, or at least linearly independent.

If two different input sets (an input set is often referred to as a pattern) are linearly dependent, then a linear network is not able to give a correct answer for both inputs. This is because the linear network is in practice solving a set of linear equations.

For an example take the OR problem. It is the following function :-

(0,0) -> 0

(1,0) -> 1

(0,1) -> 1

(1,1) -> 1

Note the vector (1,1) can be made from (0,1) + (1,0), and hence the inputs are not linearly independent. In a linear net the solution of the following equations will be attempted :-

$$a_1 x_1 + a_2 x_2 = output$$

for each x_i, where a_i is some scalar. This gives :-

$$a_1 0 + a_2 0 = 0 \tag{4.1}$$

$$a_1 0 + a_2 1 = 1 \tag{4.2}$$

$$a_1 1 + a_2 0 = 1 \tag{4.3}$$

$$a_1 1 + a_2 1 = 1 \tag{4.4}$$

equation 4.2 gives a_1=1 and equation 4.3 a_2=1. Thus the following mapping may be given by the net :-

(0,0) -> 0 (Correct)

(0,1) -> 1 (Correct)

(1,0) -> 1 (Correct)

(1,1) -> 2 (Incorrect)

Other learning rules are more appropriate for linearly dependent and for more general (e.g. non linear) cases.

Units may be grouped in levels, and the outputs of one level become the inputs of another. Connections may be made in one direction only (feedforward), in both directions (bi-directional), allowed within a layer, and a unit may feedback to itself. Groups of such units are referred to as networks. Arbitrarily designed networks are notoriously difficult to analyse and predict the behaviour of, and severe restrictions are usually put on a particular network, to allow some form of analysis to be performed. Examples of such restrictions are networks where no feedback is allowed to a unit, or between units of a layer.

4.2. Activation Functions

The output from a set of units, connected to another unit, may be simply summed to provide the input to the receiving unit. Such a simple function ie. $\sum_i a_{ij}$ where j = index of receiving unit, is an example of an activation function. Non-linear functions which "squash" the value into a prescribed range are much more commonly used, eg. the logistic function.

4.3. Rationale for Using Networks

Neural networks are based on the paradigm of information processing, as typified by the cognitive functions of animals. It has been made apparent that the speed of neurons is such that sequential processing is not practicable for real-time usage[24]. This is especially pertinent to vision and speech. The visual system uses concentric rings of retinal cells. These are massively connected to intermediate levels of neurons before termination in the visual cortex. For sufficient speed much of the processing must be completed in parallel, though some sequential steps may be involved as well. It has been pointed out, for example in [25] , that the number of connections in the human visual system is enormous, given roughly 10^8 neurons in the retina, and each cell being connected to between hundreds to tens of thousands of others. Such a system is able, even with the slow neuronal transfer speeds, which are in the order of milliseconds, to allow real-time response to complicated scenes at around 20-30 frames per second. This is far more impressive than the performance of very powerful computers using sequential algorithms. But the number of steps performed by any one neuron can only be in the order of tens to hundreds. It seems inconceivable that the complex visual performance of humans can be handled by so few serial steps, it must involve massive parallelism, whereby many different but small tasks are computed simultaneously.

A better paradigm for image analysis may be to mimic the mammalian visual system by using massively parallel computation, using simple individual processes. A large variety of such artificial neural networks exists, which may be employed dependent upon the application. All the artificial networks have some common characteristics :

1. There are processes which may run independently, and thus in parallel.
2. The networks "learn" or "discover" patterns or results.
3. A network consists of one or more layers of such processes.
4. Individual processes may affect other processes in the same or other layers, or provide feedback into themselves.
5. The inputs from some processes are subject to some function or rule to give an activation value to a process, this value may be used as an output to other units.

4.4. Network Paradigms

4.4.1. Auto-associative :

These networks accept input and produce output using the same units. They attempt to match an input with a previously seen target pattern. Rotational and translational invariance, or pattern filling are typical applications of such nets.

4.4.1.1. Brain State in a Box

This network is similar to a simple linear associator. The units are given a range of possible values though, typically [-1, 1]. An activation rule [16] has the effect of pushing the value into a vertex of the hyper-cube, where in an N unit network, there are 2^N such vertices.

Taking w_{ij} to be the weight between units subscripted i,j. If a_j is in [-1,1].

$$a_j(t+1)=a_j(t)+\sum_i w_{ij}a_i(t) \qquad (4.5)$$

else if $a_j(t+1)>1$ then

$$a_j=1$$

else

$$a_j=-1$$

Learning takes place by adjusting the weights in one of two formulae :

$$\Delta w_{ij}=\eta a_i a_j \qquad (4.6)$$

or

$$\Delta w_{ij}=\eta(t_i-a_i)a_j \qquad (4.7)$$

where t_i is a teaching input.

4.4.1.2. Boltzmann Machines

There is a class of problems that are intractable by exhaustive searching for a solution, which are named NP-Complete. In NP-Complete problems (for a general introduction see [26]) as the number of variables in the problem increases, the time taken to compute the answer increases dramatically. An example quoted in [26] for different time complexity functions is that for size n=60 and linear function $f=n$, *time*=0.00006 secs, for a 5th power function $f=n^5$, *time*=13.0 minutes, (both polynomial functions), but $f=2^n$, *time*=366 centuries, and 3^n, *time*=1.3×10^{13} centuries.

A similar calculation in [27] where each calculation takes one nano-second gives 800 years for an problem with n! solutions where n=20.

The list of known NP-Complete and NP-Hard (a related set of problems, also intractable with exhaustive searching) is large, and contains many real-life problems, e.g. the traveling salesman problem. Optimising with quite small numbers of variables and such high times is clearly impractical for the NP-Complete cases, and approximation methods become pertinent. One such class of approximation techniques are local search methods, where a cost function is minimised by exploring neighbouring solutions. However the upper bound for the time taken to solve a problem is not known for many problems[28]. Consequently some modifications may need to be made to the algorithm. One solution is to restart the algorithm many times with different initial conditions. Another method is to allow a cost function to increase in a probabilistic fashion.

The method named simulated annealing (for a discussion see[29]) uses this latter modification. It is so-named as it resembles the slow cooling of a metal to form a crystal lattice. The probability of the cost function increasing is governed by a function named temperature, where temperature is related to the probability of the cost function being allowed to increase, to avoid local minima. The temperature is slowly decreased, initially the likelihood of the cost function increasing is high, and so if a local minimum is encountered, the system will be unlikely to remain in it, and finally the probability is very low, and a minimum is encountered which will hopefully be close to optimal.

Boltzmann machines perform this type of optimisation, and the algorithm may be performed sequentially or in parallel using a neural net. There are critics of the technique, for example[30] who claim the Boltzmann machines are "not much more powerful then combinatorial circuits built from gates which compute Boolean threshold functions and their negations".

4.4.1.3. Hopfield Net.

The network consists of a fully inter-connected net, each unit may have excitory or inhibitory connections to other units, and the connections are symmetric, ie. the weight from unit i to j is the same as that from j to i. The learning rule is [31] :

$$\Delta w_{ij}=(2x_i-1)(2x_j-1) \tag{4.8}$$

where x_i is output of the current unit, x_j is the input from some other unit.

Hopfield nets have been used in target recognition, and it is possible to implement a large net using optical technology[31].

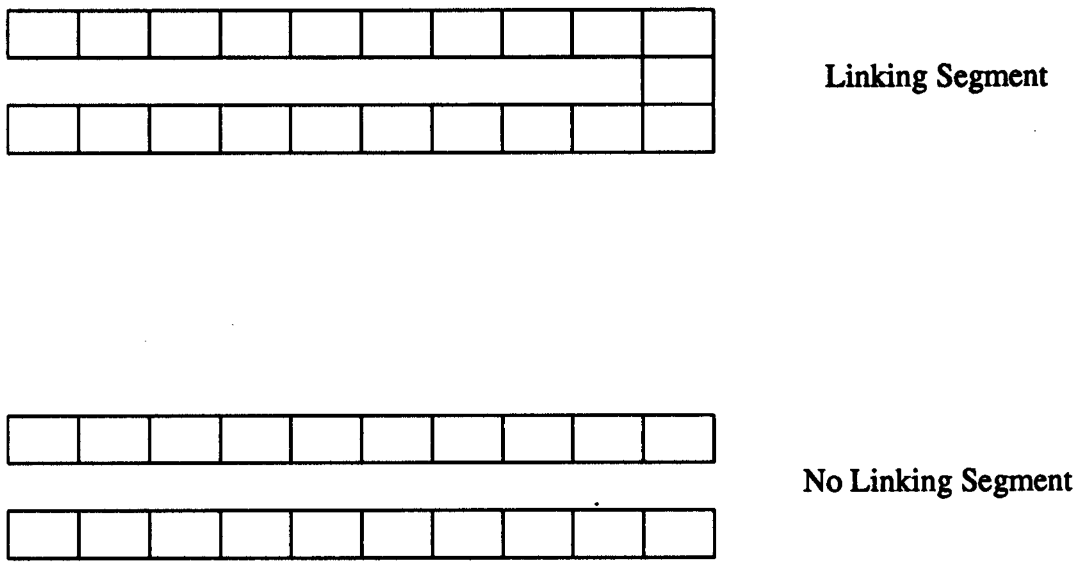
4.4.2. Hetero-associative :

In hetero-associative networks separate units are used for input and output, and the number of inputs units does not in general equal the number of outputs. Classification tasks are typical applications, where frequently a high dimensional input space is mapped to a much smaller classification space, e.g. the inputs of individual dogs could be classified into the breeds of dog.

4.4.2.1. Perceptrons.

These networks are essentially linear associators, with the addition of a threshold for the output of units. It has been shown that these networks are capable of solving any problem for which a linear solution exists[32]. However it has also been shown that problems that are not of this nature may not have a solution [32], and one particular such example is that of connectedness in space. To illustrate this concept imagine a set of pixels which are turned on or off (ie binary). Looking at a small area of an image pixels, which are not joined locally may be thought to be in separate shapes. However as one stands back from the image, on a larger scale the pixels are seen to be in the same shape, or connected together (see Figure 4.2). It can be proved there is no local method of solving this problem. One must look at the whole picture to rule out that two areas are not connected[32].

Figure 4.2 Connected and Unconnected Shapes Not Recognisable as Such by Local Methods



4.4.2.2. Kohonen

Kohonen suggested a network that is unsupervised, and learns to cluster data. An input data space is mapped to a lower dimension output space. The networks are "topologically correct" which means that the topological relations of input and output space are similar. A brief description may be found in [33] and a fuller description by its creator is in[34].

Applications include the exploratory analysis of data[35].

4.4.2.3. Wisard

This classification system is novel in that it is implemented in RAM. A description may be found in[36]

4.4.2.4. Counter Propagation

A marriage between Grossberg learning and Kohonen networks, this hybrid is five layered. Two input layers feed directly to a Kohonen layer, which is connected to two

Grossberg layers. The inputs are presented, and different unit in the Kohonen layer becomes predominant with varying pattern pairs. After learning, incomplete patterns will tend to be filled in. A short review of this network can be found in[37].

4.4.2.5. Bi-Directional Associative Memory (BAM)

Using a four layer structure the BAM network is designed to store associated pairs of vectors. A description of BAM may be found in[38]. The first and fourth layers are input and output buffers, the middle two layers are adaptive. Upon presentation of a noisy pattern pair, oscillations between these inner layers eventually stabilise to give the closest learned association. The BAM inner layers are fully connected, and employ Hebbian learning.

4.4.2.6. Adaline

The delta rule which is described by the relation $\Delta w_{ij}=\eta(t_i-a_i)o_j$, where η is a scalar, t_i a target value and a_i the output for units i. has been used to produce the ADaptive LINear Element (ADALINE)[39]. This rule effectively changes weights to reduce error between input and output of an element. It was the first successful commercial ANN, and is today used in modems. Several ADALINE units may be combined to form a Multiple ADALINE (MADALINE).

5. RATIONALE FOR USING MLPs WITH ERROR BACK PROPAGATION

5.1. Advantages of ANNs

The advantages of the network over traditional AI techniques are :

1. Speed. Each pixel may be computed in parallel.

2. No reasoning is required to be given, or a priori knowledge used, other than the simple constraints used in pre-processing.

3. The system may be applied to other organs, images.

4. Systems are resistant to error and incomplete knowledge. With increasingly incomplete data, or noise, the system exhibits worsening performance, but not a catastrophic failure. (This is usually referred to as "graceful degradation").

5. New rules can be "learned".

5.2. Disadvantages of ANNs

Disadvantages include :

1. The reasoning of the network is difficult to analyse.

2. A solution is not guaranteed, and in particular local minima may be encountered.

3. The networks scale poorly with size.

4. Network architectures are difficult to design optimally.

5.3. Network Paradigm Chosen

The network paradigm chosen was an MLP using error back propagation. The reasons are discussed below.

The images are required to have a particular diagnosis as an output. The inputs will be pixel values, or some representation of pixel values. A hetero-associative network is required rather than an auto-associative, since the inputs and outputs are of different type, and different dimensionality. Auto-associative networks may be used as a pre-processing stage. Such pre-processing might be to obtain invariance for orientation, or to complete partial images, or reduce noise etc. Invariance to size and orientation of images was obtained in other conventional ways where needed, as explained in a later chapters (Chapter 6).

Of the hetero-associative networks, the perceptron may not be appropriate as it would not be able to recognise connected areas, or other non-linear patterns. There are several network paradigms to choose amongst which do permit the use of non-linearities. It would not be possible within the time scale of this study to test the applicability of each type to each problem tackled. Thus the literature was consulted for similar applications to give a lead as to the most appropriate network. The MLP was used to compress data, perform prediction, and to categorise patterns. These particular tasks could be implemented using other network paradigms of course, and such alternatives are referred to in the Chapter 4. However for all of these tasks MLPs have been successfully used in studies for similar ends. Papers showing positive results include[40] in which images were compressed which subjectively appeared close to the original[41], which showed categorisation of images of soda bottles into correct classes was possible, and[42] discuss the use of prediction MLPs, and quote references to studies in which such nets were used successfully. MLPs have been used to make simple expert systems for medical diagnosis of back pain[43] , dyspepsia[44] and headache[45]. An MLP has also been shown to perform as successfully as an expert system in the forecasting of solar flares[46], in which study the authors suggest that if a connectionist system can perform as well as a rule based expert system, then one should, by examining the internal representations of the network, be able to elicit rules. This aspect has been looked at[47] by activating inputs one by one, and in groups, and noting the affect on the outcome. MLPs may be faster to implement, and once trained, faster to run than conventional systems. Quoted in the solar flare study[46] the time taken to build the rule based expert system was one man year, and the processing time for a single prediction was 5 minutes. this compared with one week for the MLP development, and a processing time of a few milliseconds. However in that study the expert system already was designed prior to the ANN being built, and it is possible that the knowledge gained in building the expert system helped speed up the design of the ANN system.

It has been shown that linear MLPs are equivalent to discriminant analysis[48] which is the technique whereby data is organised into well separated clusters. This method would take high dimensional data and project it onto an optimal subspace. The use of nonlinear elements could improve the performance above that of conventional discriminant analysis. Simulations using a hidden layer with more units than the number of expected classes have in fact shown an improvement[48]. Since the scintigrams are to be placed into classes, viz those with particular abnormalities, nonlinear MLPs seem to be a reasonable net design.

An assessment of MLPs versus human observers has been made comparing the recognition of a simulated "nodule". In[49] a noisy background of a 5 by 5 pixel array, had a 3 by 3 pixel signal imposed in 50% of an image set. Receiver operating curves (ROC) showed MLPs at least as good as humans in detecting the signals at low SNR. As the authors admitted the images were much less complicated than real radiological (or scintigraphic) images, but the study shows that in principle the use of MLPs in signal detection in worth exploring.

Conventional techniques such as expert systems and image processing algorithms may also have a complementary place with ANNs.

5.4. Mathematics of Error Back Propagation.

Using linear activation functions a multilayer perceptron(MLP) can be reduced to an equivalent simple perceptron, and so does not solve non-linear problems. For if the network layer is linear, the input to the jth layer from the ith layer may be expressed as

$$J = AI$$

where I is the output from layer i, J is input to layer J, and A is a matrix. But :

$K = BJ$

describes the connection between the kth and jth layer, and

$K = (BA)I$

But BA is just a matrix, and therefore :

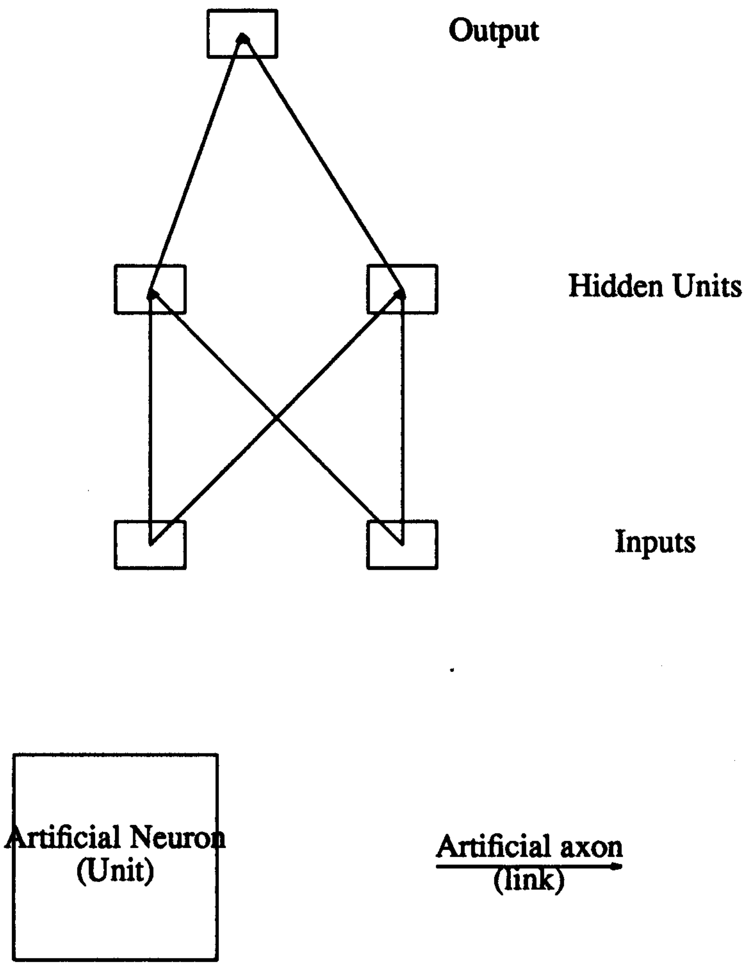
$K = CI$

where $C = BA$ is some matrix, ie. simply equivalent to 2 layers, an input and output.

Using non-linear activation functions, a multi-layer network has been suggested[16]. This is the multilayer perceptron (MLP) using error back propagation as the learning rule.

A variety of activation functions and thresholds may be used, but the basic principle is that the result of sending inputs through a series of layers (with weights originally set to small random values) is compared to some target value, used in "training" the network. Errors are sent back through the network, and used to update the weights in each layer, so as to minimise the error. A typical network layout is shown in Figure 5.1.

Figure 5.1 Simple Multilayer Perceptron



This type of network is known to be able to learn non-linear functions such as the XOR problem that a simple perceptron is unable to "learn". The XOR problem is simply the functional mapping :-

(0,0) -> 0
(0,1) -> 1
(1,0) -> 1
(1,1) -> 0

Since the MLP is the paradigm chosen for this study a slightly more detailed description of the algorithm will be given, which can be found in[16].

Assume initially a linear network. If the rule for changing weights is

$$\Delta_p w_{ji}=\eta(t_{pj}-o_{pj})i_{pi}=\eta\delta_{pj}i_{pi}\ \textit{.LPwhere} \tag{5.6}$$

$\delta_{pj}=(t_{pj}-o_{pj})$

t_{pj} *=jth target output for pattern p*

o_{pj} *=jth output element from pattern p*

i_{pj} *=jth input element from pattern p*

Let an energy term E be defined as

$$E_p=\frac{1}{2}\sum_j(t_{pj}-o_{pj})^2 \tag{5.7}$$

We wish to show that

$$-\frac{\delta E_p}{\delta w_{ji}}=\eta\times\delta_{pj}\,i_{pi} \tag{5.8}$$

Where η is a scalar. Ie. gradient descent occurs because the differential of the error with respect to change in weights is proportional to the current error. The steps are :

$$\frac{\delta E_p}{\delta w_{ji}}=\frac{\delta E_p}{\delta o_{pj}}\frac{\delta o_{pj}}{\delta w_{ji}}$$

$$\frac{\delta E_p}{\delta o_{pj}}=-(t_{pj}-o_{pj})=-\delta_{pj}$$

$$o_{pj}=\sum_i w_{ji}\,i_{pi}$$

$$\frac{\delta o_{pj}}{\delta w_{ji}}=i_{pi}$$

$$-\frac{\delta E_p}{\delta w_{ji}}=\delta_{pj}\,i_{pi} \tag{5.9}$$

QED.

5.5. Adding hidden units

Let

$$net_{pj}=\sum_i w_{ji}\,o_{pj} \tag{5.10}$$

Where $o_i=i_i$ *if unit i is an input unit*

Semilinear functions are functions characterised by a near linear area in one part of their range. A semilinear activation function is given by

$$o_{pj}=f_j(net_{pj}) \tag{5.11}$$

where f is differentiable and non-decreasing. The logistic function which is often used in ANNs is approximately linear in the mid portion of its range, and is increasingly nonlinear as one moves away from the midrange values. Set :

$$\Delta_p w_{ji} = -k \frac{\delta E_p}{\delta w_{ji}} \text{ , } k \text{ is constant} \tag{5.12}$$

$$\frac{\delta E_p}{\delta w_{ji}} = \frac{\delta E_p}{\delta net_{pj}} \frac{\delta net_{pj}}{\delta w_{ji}}$$

$$\frac{\delta net_{pj}}{\delta w_{ji}} = \frac{\delta}{\delta w_{ji}} \sum_k w_{jk} o_{pk}$$

$$= o_{pi}$$

define :

$$\delta_{pj} = -\frac{\delta E_p}{\delta net_{pj}} \tag{5.13}$$

$$-\frac{\delta E_p}{\delta w_{ji}} = \delta_{pj} o_{pi}$$

To make gradient descent make weight changes according to :

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}$$

Applying chain rule

$$\delta_{pi} = \frac{-\delta E_p}{\delta net_{pj}} = -\frac{\delta E_p}{\delta o_{pj}} \frac{\delta o_{pj}}{\delta net_{pj}} \tag{5.15}$$

$$\frac{\delta o_{pj}}{\delta net_{pj}} = f_j'(net_{pj})$$

$$\frac{\delta E_p}{\delta o_{pj}} = -(t_{pj} - o_{pj}) \text{ for output units}$$

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj}) \text{ for output units}$$

$$\sum_k \frac{\delta E_p}{\delta net_{pk}} \frac{\delta net_{pk}}{\delta o_{pj}} = \sum_k \frac{\delta E_p}{\delta net_{pk}} \frac{\delta}{\delta o_{pj}} \sum_i w_{ki} o_{pi} \text{ otherwise}$$

$$= \sum_k \frac{\delta E_p}{\delta net_{pk}} w_{kj}$$

$$= -\sum_k \delta_{pk} w_{kj}$$

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj} \text{ gives a recursive rule}$$

$$\text{if } \Delta_p w_{ji} = \eta \delta_{pj} o_{pi}$$

If $unit_j$ is an output unit

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj})$$

If $unit_j$ is hidden unit

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{5.16}$$

If the activation function is the logistic function then :

$$o_{pj} = \frac{1}{1 + e^{-net_{pj}}} \tag{5.17}$$

$$\frac{\delta o_{pj}}{\delta net_{pj}} = o_{pj}(1-o_{pj})$$

The error for an output unit is :

$$\delta_{pj} = (t_{pj}-o_{pj})o_{pj}(1-o_{pj}) \tag{5.18}$$

The error for an arbitrary hidden unit is :

$$\delta_{pj} = o_{pj}(1-o_{pj})\sum_k \delta_{pk} w_{kj} \tag{5.19}$$

5.6. Software Implementation of Networks

The networks used in this study were implemented initially using a commercially available neural network package (NeuralWare). This system allows rapid prototyping but is rather limited in the number of processing units it will allow under MS DOS. When running under MS DOS a further, more critical, restriction is the memory required to set up the weights for the units. The interface used by NeuralWare was easy to use, but lacked the flexibility required of an experimental system, in particular logging of variables, and dynamic parameter changes were difficult to accomplish. Thus although a good tutorial introduction to ANNs, NeuralWare was not considered a good choice of package for many parts of this study.

Subsequently the networks were reconfigured under Unix, using another package (PDP by McLelland and Rumelhart[50]), to allow bigger networks, and better control. The Rochester system, which is more adaptable still, was considered. This system allows arbitrary connections with a user defined activation function. However as the extra power of this system was not strictly needed for this project, it was not used.

5.7. Parallel Implementation

The above mentioned software runs on sequential processors, and thus does not utilise one of the potential benefits of neural networks in terms of speed, the power of parallel processing. The individual processes are independent, and may be run simultaneously.

Since the processing is essentially parallel in nature, parallel hardware offers greater speed. A system which allowed expansion of processing power would be an advantage in this study, and preferably one which allowed parallel execution. The transputer allows such a system to be built, and initially a single transputer T4 processor has been used on a IBM PC mounted board. The back propagation has been written in parallel C, so that as more processors are added, the system will run faster. This system was not actually used in simulations as it was preferred to use a package with a variety of features such as logging of variables, generation of screen templates etc. built in, rather than create a system from scratch. The routine built on the transputer merely demonstrated that such an approach will work.

5.8. Hardware Used

NeuralWare was run on a variety of IBM compatible computers running under MS DOS. The PDP package also could be run on these machines, but almost all simulations were done on Sun 3 and Sun 4 workstations. Both these simulations use serial algorithms. In principle a network once the weights had been learned using a software simulator could be implemented in RAM, or in direct hardware. A final system would benefit from a network implemented in silicon, as this would be faster, and the algorithm would be a parallel one, to exploit the distributed nature of networks.

6. ANNs USING RAW DATA

In a pilot study [51] coronary scintigram images were classified using an MLP with error back propagation. It was claimed that correct classification was obtained in all cases with 29 scans, 12 which were considered normal and the rest with vessel disease (as judged by a human expert). In that study 15 segments of the images were fed into 45 clusters of input units, a hidden layer of 30 units and a single output unit. It was not stated whether the data was split into training and test sets. Clearly the claim of 100% accuracy of classification is meaningless if the ANN simply classified correctly the data it had been trained on. With such a small data set and a large number of connections the possibility of one to one mappings or local minima is high.

A subsequent paper[52] discussing the same work in more detail gave a test set classification rate of around 80%.

6.1. Type of Data Input

An ANN may be fed input data directly, as in[52] , or after some pre-processing. While it is likely that pre-processing will be advantageous, the simplest possible implementation would be to use raw data. If this failed to produce a reasonable output one would be justified in examining data manipulation, which would typically mean data compression. Accordingly the first experiments were conducted using raw pixel data, with minimal pre-processing, merely a noise reduction median filter, and a standardisation for size.

6.2. Automatic Reporting System

There are a variety of nuclear images which could be analysed with neural networks, e.g. lung scans, dynamic cardiac studies etc. Initially a pulmonary system was implemented. These had certain operational advantages, viz. there were a large supply of them, and they are static images, and without the added complications encountered in cardiac studies (e.g. gating of images). In order to allow the network to function most effectively, the images were pre-processed for reasons given below.

6.3. Acquisition of Images

The images were obtained by "frame grabbing" analogue images taken from an atlas of scintigrams[53].

6.4. Reduction of Noise

Images may be acquired in screen resolutions of various sizes. Further the "screen grabbing" will be affected by how close the image is to the video camera, and subjects have differently sized organs anyway. Also if the uncompressed image is fed directly into a network, various problems are encountered, associated with the size of the data. More input units mean more connections, and a larger usage of computer memory. On micro-computers this may be beyond the resources of the computer. On mainframe or workstation machines this is capable of being dealt with by virtue of virtual memory. But a second problem is less tractable. As the number of connections increases the degrees of freedom of the system increases. This makes it more likely that the network may act as a look-up table, rather than learning general features of the images. Ie. a set of weights may mean a particular image, but a similar image will not necessarily give a similar output, the system has then not generalised well, it is topologically incorrect.

Since one may need to reduce the size of the image, one should employ a data compression technique that also reduces noise. The median filter was employed to reduce noise as the raw images were too large. In a standard median filter approach a mask is moved over the image pixel by pixel, and the median pixel in the mask is saved. For the purpose of file reduction a mask was applied to non-overlapping segments, and one pixel value was saved per mask application. Thus if an n by m mask was applied, an n times m file reduction was achieved. The median filter was employed as features (e.g. edges) are better preserved using median filtering than some other methods, e.g. mean filtering[54].

6.5. Locate Region of Interest

One method of locating the region of interest (which in this case is a lung, or in lateral views where the lungs overlap, both lungs) is to roughly place the area in a rectangle interactively. The maximum row and column counts are obtained, and used to determine the centre of the lung. A segmentation of the lungs from the background is produced by searching for connected pixels from the centre of the lung outwards, given some threshold (discussed later) and requiring the pixel under consideration to be within that threshold, and to be a neighbour of at least one pixel previously considered to be within the lung contour. This method assumes that an operator determines where in the image to start the process, and what the threshold should be.

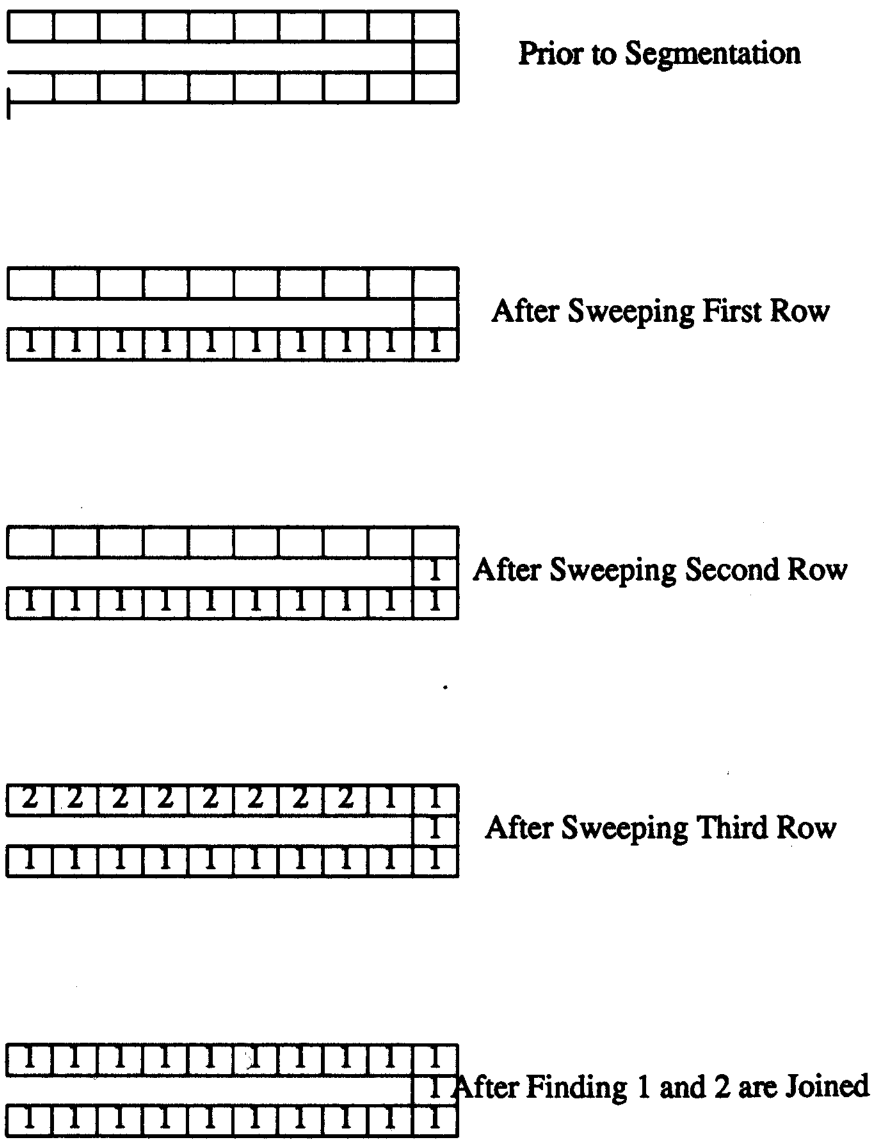
However in a fully automatic system one may not rely on operator interaction. A second method has been developed based on a standard segmentation method[55]. The image is scanned row by row and column by column. A threshold is used which is given as a parameter to the routine. The threshold may be derived as a percentile.

In the images used in this study it was found that a constant value could be used as a threshold, as the images were similar to each other in terms of illumination. This threshold was found experimentally by interactively setting the threshold until the background disappeared. It was kept constant in later images, and in every case gave adequate lung fields. If a pixel value is larger (or smaller, dependent upon whether bright or dark areas are required for segmentation) then the pixel is assumed to have satisfied the threshold requirement. Any pixel not satisfying this requirement is allocated to the background, ie. to no segment. Starting at the bottom left (say) the first pixel which satisfies the threshold requirement in any row is allocated to a new segment, segment 1. If a pixel which satisfies the threshold requirement follows a pixel in the same row which does not, it is allocated to a new segment, segment 2, etc. If the pixel satisfies the threshold requirement, and one of the three neighbouring pixels below is already in a segment, then the current pixel is re-allocated to that segment, it is connected to that pixel.

This procedure has the effect of creating stripes of connected pixels. These stripes are then collected together. See Figure 6.1 for an illustration of the technique. Any segment which is immediately adjacent to another segment is connected to it, and all pixels with that segment number are re-allocated to the segment number of the segment adjacent to the current segment.

This routine creates many potential lung fields, which could be noise, or image borders or other artifacts other than lungs. One could state criteria such that inappropriate regions are "weeded" out, leaving the lung fields as required. This approach contains the flaw of all such custom made rules, it is not possible to apply such a rule in general to other areas, with other organs (for example). Rather than create ad hoc rules it was decided to use neural networks to learn which areas were lungs.

Figure 6.1 Segmentation Procedure in Stages



6.6. Standardisation of Size of Image Segments

The image is reduced or expanded in both x and y scales to produce a standard size image. Compression was achieved using the modified median filter described above, expansion by copying pixel lines in either horizontal or vertical directions. The standardisation of size gives a similar input to the network for each lung field, and avoids the necessity for the network to learn scaling. The images are always oriented in the same direction so orientation likewise does not need to be learned. All values were scaled to lie in the [0,1.0] range.

The standardised pre-processed pixel values are then fed into a neural network. The network was based on the back-propagation method. This network was fully connected between adjacent layers. The design for the network is notoriously difficult to optimise. It has been suggested that two hidden layers are useful, and sufficient[56].

Initially the network was implemented with 300 inputs, two hidden layers of 40 and 20, and 7 outputs. Subsequently a net with 1000 input units, 100 units in a hidden layer, and 7 output units was tried. The output units refer to the presence of a specific abnormality (PE), whether the image is a lung or not (some segments will be artifacts) and the type of view (anterior-posterior (AP), posterior-anterior (PA), left lateral (LL) or right lateral (RL)). Finally an ANN with 400 units, and a hidden layer of 50 was employed.

Segmented images created using the segmentation routine, having been standardised for size, were input to the neural network. Half of the images were used as a training set, and the remainder were used as a test set.

The maximum number of inputs required is determined by the resolution of the gamma camera, which is about 2 cm. Using an image size of pixels which represent 1 cm can be seen to be optimal in this respect. A 40 by 25 pixel image yielding 1000 units as an input layer gives around this resolution. The number of units in the hidden layer is usually determined by trial and error, and the units in the hidden layer were changed in a number of simulations to obtain an optimal number. An exhaustive search is not possible, due to the potentially infinite number of structures possible. This problem of network architecture is addressed later in the study, in chapters 11 and 12.

Given a set of inputs from a training set, and the known diagnostic output expected from the network, a series of training cycles were executed to train the network. Subsequently new images not previously seen by the network were fed in to test the accuracy and sensitivity of the network.

The following experiments were performed using images of lung scans taken from an atlas of lung scintigrams [53] :

1. Various lung images from a variety of projections (anterior-posterior, posterior-anterior, left and right lateral) were trained using the known projection to ascertain whether the network could work out which projection was given.

2. A variety of images with either an abnormality (in this case PE) or with no abnormality were trained to see whether the network could identify one specific abnormality (PE).

3. Segments generated from the segmentation routine were used to train a network to recognise artifacts from lung fields.

The outputs will never be identical to the target values, as the smaller the error, the smaller the adjustment made to the weights in training. A threshold is used to determine whether an output is close enough to be considered correct. A value of above 0.9 was considered to be 1.0, and one below 0.1 to be zero. .

6.7. Results

6.7.1. Initial Net: 300 Inputs, Hidden Layers of 40 and 20 Units, and 7 Outputs

Using a network structure of 300 inputs, hidden layers of 40 and 20, and 7 outputs, and using the automatic segmentation routine, the network converged on the training set, ie. it gave the correct outputs for the data it was trained on. Using an "unseen" test set the network was found to be able to distinguish between lung and non-lung segments. Thus the network could be used to determine which of several candidate segments are lung segments, and therefore could be used for further processing. It did not seem able to distinguish any of the other features (aspect, type, disease state).

In subsequent experiments all the images were lung images, no artifact images were analysed, and therefore the output unit corresponding to the presence of a lung was always clamped to unity.

6.7.2. Large Classification Net: 1000 Inputs, Single Hidden Layer of 100 Units, and 7 Outputs

A larger net was tried to determine whether a higher resolution helps.

The network was trained on a set of 30 image segments. The network converged on the training set. That is to say the training set when put through the network gave results almost identical with the expected outcome which it had used as a target to aim at.

A test set of similar images, which the network had not "seen" gave very poor results. The network seemed unable to distinguish between PE and other types of scan. Furthermore the network was unable to distinguish between various views of the lung.

6.7.3. Small Classification Net: 400 Inputs, Single Hidden Layer of 50, and 7 Outputs

The original network consumed a large amount of processor time, taking in the order of days to converge using a Sun 3 processor, and it was decided to try a smaller network of 400 input units and a single hidden layer of 50 units to speed up processing.

Using the same data as the large network, the net converged again, but did not give better results.

6.8. Observations

The network failure could be attributed to one of several reasons :

1. The network was not given sufficient examples. This is likely as [57] has stated that to avoid one-to-one mappings the number of patterns should be O(N) where N = number of connections, which clearly is not so in this case. Alternatively one may consider that the number of connections were too many allowing one-to-one mappings (and adding computational cost). These aspects will be further discussed in Chapters 11 and 12.

2. The network was not given sufficient iterations. The error curve in training

should asymptote to a value. It is possible that insufficient iterations may result in a network with a large error remaining. This is not the case as the training set would then result in large error on recall, which was not the case.

3. The network was too small, ie. of too low a resolution. This if true would be an impracticable problem to address by increasing the net size as networks scale poorly with size, and again inappropriate computer time would be required. A solution may be to compress the input data. This is explored in chapter 7.

4. The problem is not amenable to network analysis. It is not known a priori whether this is the case, and there exists no known way to test whether this is the case. However biological networks (the human brain) can solve such problems, so there is an existence theorem that more complex nets can categorise scintigrams. The question remains as to whether simple ANNs as used in this study can do likewise.

7. DATA COMPRESSION USING PRINCIPAL COMPONENTS ANALYSIS AND ANNs

7.1. Introduction

Barber and Nijran[58] have shown that the data of scintigrams is highly correlated. The redundancy may be exploited by techniques such as Principal Components Analysis (PCA) to produce a much smaller data set which contains a given level of the information in the image (as measured by the amount of variance accounted for). PCA analysis was used to reduce the number of parameters describing "dixel" (dynamic pixel) curves. PCA analysis has also been used to produce an automatic reporting system[59].

If compression of images can be achieved to a representative subset, that subset may be input to a feature detecting network. This part of the study explores compression of typical lung scintigrams, of a type which would form a set of images to train a feature detecting network.

ANNs have been used to compress image data using error back propagation in an MLP[40]. In this study 8 by 8 segments of an image were used to train an MLP to learn the identity mapping. As Cottrell noted[60] such a network can be considered auto-associative and unsupervised. It is auto-associative since the input is the target, and unsupervised since the error signal is derived from the input. An alternative method of compressing images[61] is to use several subnets to learn portions of an image. An ANN compressed 128 by 128 images split into 256 8 by 8 blocks. These blocks were used to feed 256 subnets, which learned mappings to lower dimensional spaces in parallel. The subnets learned independently the different segments. This would be appropriate if similar images were to be compressed, as the subnets would learn the statistics of different regions of the images. Later workers[62] quoted this study, and stated that a novel adaptation would be to use the one net and pass through all 8 by 8 segments of images. The network architecture was identical to[40] which[62] seemed ignorant of, claiming the technique as their own. Competitive networks have been used to compress data[63] , as have

Hopfield nets[64].

A compression network may be used to reduce the number of inputs to a subsequent classification ANN. As previously stated and in[65] and [66] raw lung scintigram data has had no success in classifying the images into normal and abnormal

The failure may in part be due to a lack of pre-processing of the inputs. One possibility is to employ data reduction using conventional techniques such as principal component analysis (PCA) or the more novel ANN data compression technique.

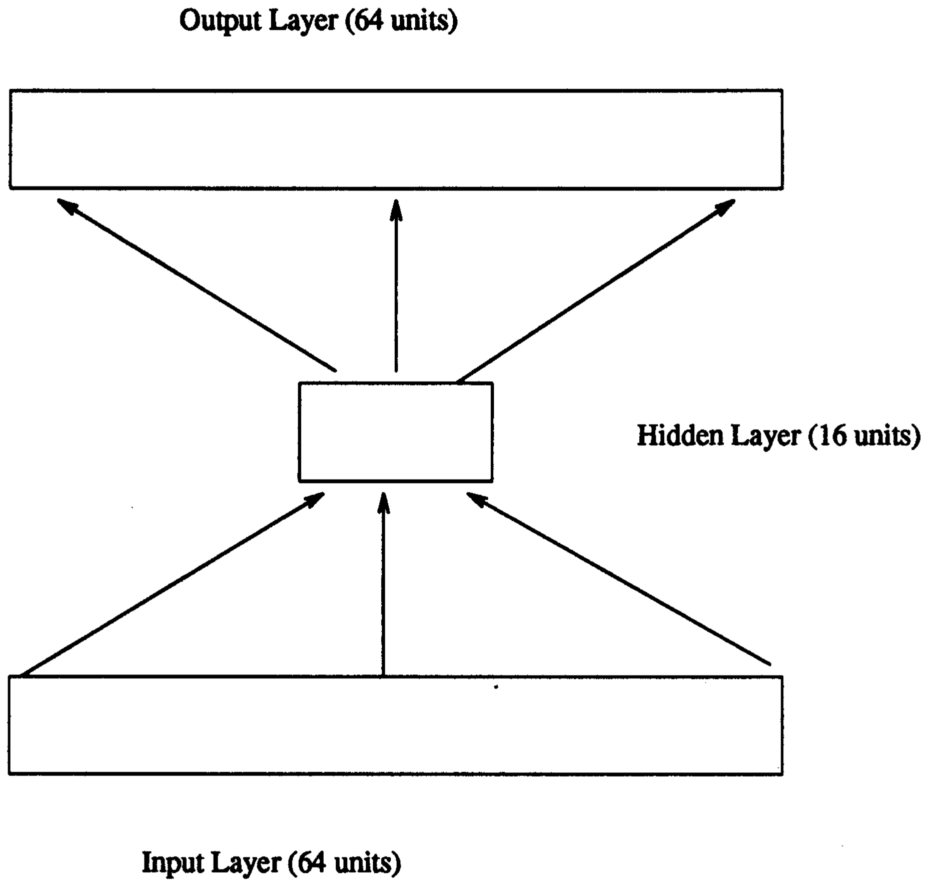
Neural networks and PCA are related[67] but networks have some theoretical advantages since they are capable of learning non-linear mappings[40] unlike PCA which is a linear mapping[68]. It has been shown for linear nets employing Hebbian learning that a model neuron tends to extract the principal component from an input vector sequence[69] and a layer of neuron units yields a principal component subspace[70]. The nonlinear MLP ANN empirically has been shown to produce a hidden layer which spans the subspace generated by PCs[40]. However the experimental work by Cottrell et al[40] has shown that image compression using a linear network performs similarly to non-linear networks such as back propagation. They also found that the variances of hidden layer outputs are roughly equal. This is in contrast to PCA where the successive principal components (PCs) are monotonically decreasing. The equal variance found in ANN hidden units is a potential advantage where damage may occur to units, as corruption of a unit containing a large percentage of the information would cause extensive damage.

In order to ascertain whether the ANN technique was an appropriate pre-processing method for the medical images used in this study, it was compared with PCA[71].

There are theoretical reasons for assuming that ANNs cannot outperform PCA with regard to data compression[72]. However if ANNs did not show a substantial increase in error compared with PCA (say) then they might still be considered preferable, as once trained the ANN would be much faster than PCA, especially if neural hardware was utilised. When computing PCs it is necessary to create a covariance matrix. This has the dimensionality of N^2 where N = number of dimensions. The number of dimensions that one may compute such a matrix for is limited by the memory of the computer, and disk swap space. Using 64 by 32 images (2048 dimensions), the PCs were not capable of being computed because one would require 2048^2 floating point numbers, and there was not sufficient swap space even on a Sun 4 system. ANNs however would for the same problem require memory in relation to the compression required. For if N inputs were to be reduced to M hidden layers, then $2 \times N \times M$ connections would be required. Thus as the compression rates increase ANN are increasingly more efficient than PCA.

Cottrell et al[40] raster scanned images with an 8 by 8 window, and put the 64 values of each scan into an input layer connected to a hidden layer of 16 units, which was connected to a 64 unit output layer. Figure 7.1 shows the network structure. The ANN was trained to learn the identity mapping, with a 4:1 compression from input to middle layer. This study used a similar network architecture, and also experimented with segments of 32 by 16 pixels, which formed a 512 input layer, connected to a 128 unit hidden layer (ie. also 4:1 compression), which was connected to a 512 unit output layer.

Figure 7.1 Compression Network Topology



7.2. Experimental Methodology

7.2.1. Acquisition of Images

The images were obtained by "frame grabbing" analogue images generated from an Elscint gamma camera system using a video camera and a digitiser board. A light box illuminated the image posteriorly, and a subtraction from an image of the light box alone allowed the effects of unequal illumination to be corrected.

7.2.2. Image Sets

Since the utility of compressing images in this study would be to detect features in normal and abnormal images, a set of images with both clinically normal, and with common defects, were used as test data. 20 lung scintigram images were used, one half contained PE, the remaining images were typical non PE examples, of which 5 were clinically normal, and 5 had COAD. Since COAD can be confused with PE, a mixture of PE, COAD and normal images would constitute a good test of a classification network.

In clinical practice perfusion and ventilation scans are used as it is a mis-match between these two studies which indicates PE, matched defects indicating typically COAD. An automatic system may therefore use both studies, or may subtract one from the other. The latter technique has certain difficulties due to problems of registration. Ventilation, perfusion and subtraction images of ventilation minus perfusion were used, and three views, posterior, left and right posterior oblique were given for each image.

7.2.3. Networks Used

An MLP using error back propagation as described by Rumelhart and McLelland[73] was used in a similar fashion as in[40].

PCA analysis was achieved using MATLAB, which is a matrix manipulation package[74]. The macro files used are listed in Appendix 2. The image files were converted to PDP input and target data, and to MATLAB format using the image processing suite, whose source code is listed in Appendix 1, and described in Appendix 4. Transfer between MATLAB and PDP, and between other representations of data, was done using utility C programs listed in Appendix 3.

All images were pre-processed using a 2 by 2 median filter[54] to compress the image and reduce noise. A 2 by 2 filter was employed as this reduction transformed the images to a suitable size for viewing on an IBM compatible computer screen with EGA graphics. Pixel values were scaled to lie in the interval [0,1.0].

Lung fields were obtained interactively by delineating a region of interest, which was the whole of both lungs, and these fields were scaled using the median filter technique, described earlier, to make each image 64 by 32 pixels in size. The lung fields could have been obtained using the automatic segmentation technique used earlier, but it was not important for the purpose of demonstrating data compression how this was done. Cottrell et al employed quantisation of outputs of hidden layer units in order that the bits per pixel may be calculated. Ie. the output from a unit was split into several ranges, and all outputs within a range were set to the same value. This was not done in this study as we wished to explore how ANNs compared in normal practice with other methods. The compression ratio might not strictly speaking thus be 4:1, as the degrees of freedom in non-linear nets tends to increase as learning progresses, because the activation values of units tend to move away from mid-range, where they are approximately linear, to the more non-linear extremes of their range[75]. Empirically the quantisation has been found to show a relationship with tsse such that as one increases quantisation the tsse converges to a steady value[62], and it was found that using 8 bit input data, a quantisation of 4 in the hidden layer was close to optimal, and so the compression factor could be 8:1 rather than 4:1.

Various learning rates were tried on similar images, and a rate of 0.01 was found to be satisfactory, high learning rates caused "flooding" of hidden units. Using a momentum term (α) helps learning by damping convergence. A high momentum allows a higher learning rate and faster convergence[16]. A learning rate η of 0.01 and α of 0.9 was used in all the experiments described in this section. The learning rate was found by trial and error. Essentially the higher the number of units, the lower η should be to avoid "flooding" of hidden units, whereby the hidden units become stuck at a value. A workable η was generally found to be of the order of the reciprocal of the number of input units. A high α helps speed learning, as[16] pointed out a high α of 0.9 allows a low η to be used which achieves convergence more quickly than a higher η with no α term. Furthermore as stated in[25] and[76] when convergence is being approached, the α term has the effect of raising the η term by a factor of $\frac{1}{1-\alpha}$. Clearly α of 0.9 gives an equivalent η an order of magnitude higher near convergence. However at very large values of α close to unity, for some values of η convergence becomes more difficult.

7.3. Image Groups

The images were used to create three non-overlapping sets, A, B and C. Three images not allocated into groups were discarded to keep the number of images in each group the same. Group A consisted of one randomly allocated subject (ie. 9 images), Groups B and C both contained 8 randomly allocated subjects.

Groups A and B were rasterised into non-overlapping 8 by 8 pixel segments, and Groups B and C were also rasterised into non-overlapping 32 by 16 pixel segments. All groups produced 288 segments. In the case of group A using 8 by 8 pixel segments, and Groups B and C using 32 by 16 pixel segments all non-overlapping segments were used. For Group B, which contained many more images than group A, to keep the number of segments the same as in group A, 288 8 by 8 segments were randomly selected from all possible non-overlapping segments, and no segment was allowed to be chosen more than once. If all segments had been used, Group B would have had a much larger training set, which would have made any comparison meaningless as larger training sets may give better results.

7.4. Experiments

PCA and ANNs were used to compress 8 by 8, and 32 by 16 pixel segments of the three experimental groups A, B and C. In the case of PCA, the first 128 or the first 16 PCs were used for 512 to 128 dimensional, or 64 to 16 dimensional reduction respectively. For the ANN approach the architecture described above as used in Cottrell et al was employed, and 1000 epochs were used to train the networks.

PCA analysis is achieved by transforming the data basis to a smaller basis, which contains a linearly optimal amount of information. The transformation from the original basis onto the full PC basis is :-

$$B = PA \tag{7.1}$$

Consequently to go back to original data representation. the segments were expanded back to full size using an inverse of the PCA matrix on the reduced data set.

$$A = P^{-1}B$$

When compressing the data equation 7.1 is used, but only a portion of the PCs are used, ie the matrix P is reduced to (say) P', and hence when expanding back the matrix A is not fully recovered.

$$B' = P'A$$
$$A' = P'^{-1}B'$$

In the case of ANNs expansion was achieved by the output of the output layer, and compression by taking the values of the hidden units. After expansion to original size, the segments were tested for error. For the compression technique the fidelity of the compression was tested using the total sum square error (tsse).

7.5. Results

7.5.1. Data Compression Using PCA On 64 To 16 Dimensions

To clarify discussion PCs computed using Group A will be called PC_A and those computed using Group B PC_B etc. Similarly the network trained on Group A will be N_A and one trained on A and further trained on Group B would be N_{AB} etc. Table 7.1 shows the results of compression using PCA of Groups A and B in various combinations. It is seen that the tsse is lower for the group that has been used to obtain the PCs. However the tsse using PCs obtained on another group is still comparable. The sign test was used to measure significant differences in all comparisons in this study, with a $p < 0.05$ level being considered significant. This test makes very few assumptions about the data set, and in particular does not require normally distributed data, see[77] for a description. Using this test there was a significant difference between Group B using PC_B and Group A using PC_B, ($p < 0.001$). There was no significant difference between Group B using PC_A and Group A using PC_A.

Table 7.1 Group A and B 64 to 16 Unit Compression using PCA

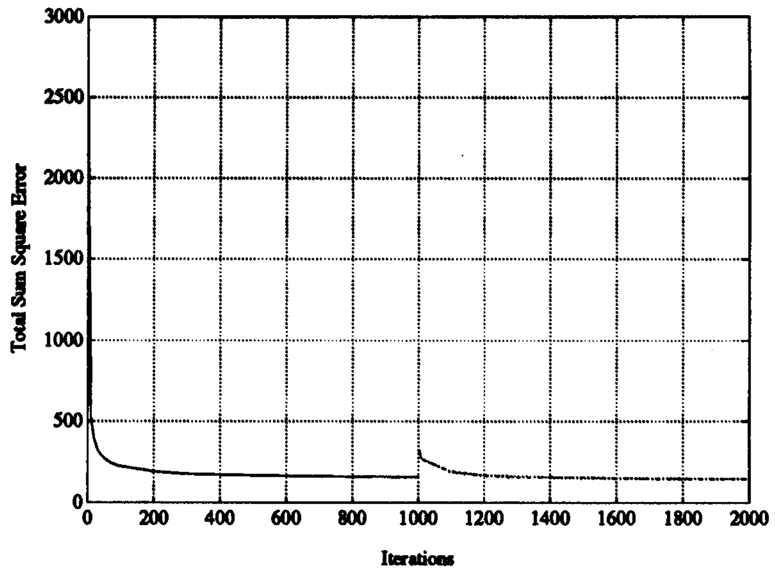
Test Group	PCA Group	tsse	% error per output
A	A	181.028	0.98
B	A	200.825	1.09
A	B	242.645	1.32
B	B	152.592	0.83

7.5.2. Data Compression Using Neural Networks With 64 Inputs To Hidden Layer With 16 Units

Convergence was achieved within 100 iterations for both groups A and B, ie. the tsse asymptoted to some value much lower than the starting tsse with random weights, and the outputs for the training set were similar to the target values. Groups A and B had virtually identical curves for tsse against epoch number, and several repeated simulations gave virtually identical results (Group B was repeated 5 times, Group A 3 times). As the network weights were reset to different random values for each simulation, clearly the networks were not sensitive to initial weights. In each case the training for a group was 1000 epochs, and where more than one group was used in training each group was given 1000 epochs of training.

If a network has been trained on a set of data, it would seem plausible that subsequent training using another group would be faster than if one started from scratch. To test this networks were trained on one group, and then further trained on a second group. Using a network pre-trained on one group, and giving further training epochs on the other group did not show much advantage. Convergence took roughly the same time (100 iterations) although the network started at a lower initial error. Figure 7.2 shows the network convergence for Group A, and then Group B. The graph for Group B followed by Group A was very similar.

Figure 7.2 64-16-64 Network Topology: Trained on Group A, then Group B



Legend
First Run Group B ...
Second Run Group B -.-.

Table 7.2 Group A and C 64 to 16 compression using Neural Networks

Test Group	Training Group	No. Epochs	tsse	% error/output
A	A	1000	157.906	0.86
A	B	1000	332.997	1.81
A	B then A	2000	134.314	0.73
A	A then B	2000	354.916	1.83
B	B	1000	139.952	0.76
B	A	1000	345.624	1.86
B	A then B	2000	150.404	0.82
B	B then A	2000	134.314	0.73

7.5.3. Comparison Between PCA And ANN Compression For 64 To 16 Dimensions Or Units

The results of using the network trained on a group to compress that same group are slightly lower than for PCA analysis, though these differences were not significant at $p < 0.05$ using the sign test. When compressing images that the network has not "seen" however ANNs gave much worse values than PCA analysis (37% - 72% higher tsse), and these differences were significant for Group B using N_A compared with Group B using PC_A ($p < 0.0001$) though Group A using N_B compared with Group A using PC_B was not significant at the $p < 0.05$ level (though it was at $p < 0.1$).

7.5.4. Data Compression Using PCA On 512 To 128 Dimensions

Table 7.3 shows the results of using PCA analysis on 32 by 16 pixel segments. It is seen that much better results are obtained if PCs are used which were calculated using the group under test than the other group. This contrasts with the 8 by 8 pixel segments where the difference was much less pronounced. Between 95 and 99% of the variance of the group used to compute the PCs was accounted for in one quarter of the 512 pixel segment PCs. This compares with about 90% of the variance being accounted for with about one quarter of the PCs in 64 pixel segments. The error per pixel is higher in unseen segments for the larger segments compared with smaller segments. These results suggest that specific rather than general compression mappings are being used for the larger segments.

Table 7.3 Group B and C 512 to 128 Dimension Compression Using PCA

Test Group	PCA Group	tsse	% error per output
B	B	747.92	0.51
C	B	2320.98	1.57
B	C	2406.67	1.63
C	C	759.141	0.51

7.5.5. Data Compression Using Neural Networks With 512 Inputs Connected To Hidden Layer Of 128 Units

The time taken to converge in the 512 to 128 unit network is again about 100 epochs. The solution is then approached asymptotically. Again the groups show very similar curves, and repeating the simulation, which was done for both groups, showed almost identical results. The tsse for "unseen" segments is again much higher, and proportionately higher than for smaller segments.

Table 7.4 Group B and C 512 to 128 compression using Neural Networks

Test Group	Training Group	No. Epochs	tsse	% error per output
B	B	300	987.112	0.67
B	C	300	5702.012	3.86
C	B	300	4347.607	2.95
C	C	300	888.015	0.60

7.5.6. Comparison Between PCA and ANN compression for 512 to 128 Dimensions or Units

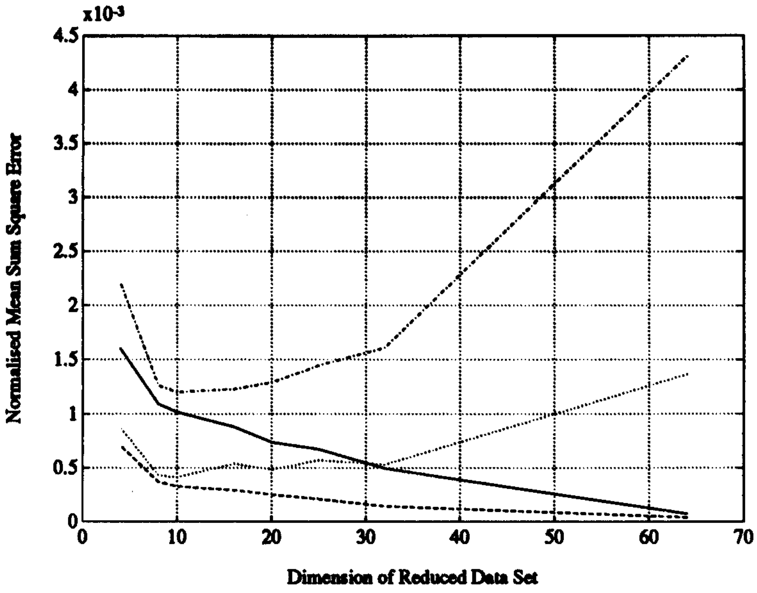
Larger segments showed similar trends to the 8 by 8 pixel segments. PCA scored slightly better for compression of "seen" segments, and though "unseen" segments scored worse in both techniques, this deterioration was more marked in ANNs, which had a tsse of about twice that of PCA.

7.5.7. Effect of Raster Scan Size

As a result of previous experiments which showed a substantial effect on error using different raster scan sizes, varying the segment sizes in PCA was further explored. Using different raster scan sizes will effect the compression performance. If PCA is used to compress data to one fourth of its original dimensions, and a raster scan size of 2 by 2 pixels is used, one will use only the first PC, which will be very close to a mean value of the pixels. Larger scan sizes will allow the statistical nature of the images to be better represented, and some optimal value may be expected that is less than the full image size. To test whether the raster scans used were appropriate, Groups A and B were raster scanned with a range of scan sizes. The results of normalised mean sum square error (nmsse), as described in Cottrell et al[40] were plotted where the dimensions of the data were reduced to one quarter of the original data.

It is clear from the graph of nmsse against dimension of reduced data set shown in Figure 7.3 that when "unseen" data is reduced (e.g. Group A data using PC_B) there is a wide plateau where the nmsse is similar, between 8 and 32 dimensions (original data dimensions of 32 and 128 respectively) and that outside this range the error increases, especially at high dimensions.

Figure 7.3 nmsse Group A and B using PCA Reduction



Legend

Group A using PCs from A ___

Group B using PCs from A ...

Group A using PCs from B -.-.

Group B using PCs from B - -

7.6. Observations

PCA analysis is similar to neural networks in data compression of segments that have been "seen", but is superior in compressing "unseen" images in these tests. The difference between "seen" and "unseen" images with respect to tsse is more pronounced in 32 by 16 pixel segments than 8 by 8 segments in PCA compression. Further experimentation showed that the 8 by 8 pixel segments are optimal with regard to tsse on "unseen" data. ANNs also seem to generalise less accurately on larger segments.

Since the time taken for neural network compression learning is about an order of magnitude higher than PCA, and PCA is more repeatable in terms of the error magnitude, and produces lower error for "unseen" segments, it would seem preferable to use PCA analysis than neural network methods to produce the reduced dimensional input to a diagnostic network.

It is possible that other network paradigms would show better results than those given above, in particular it has been suggested[78] that linear nets would be an improvement as they do not suffer local minima problems. This is addressed in chapter 12.

8. ANNs USING PRINCIPAL COMPONENTS

8.1. Introduction

ANN have been used as diagnostic aids in back pain [43], dyspepsia [44] and headache [45] where symptoms are fed into an input layer, and the various possible diagnoses form an output layer. It has been shown above that raw data from scintigrams is unsatisfactory for input to an ANN, and that PCA is more efficient than ANNs in data compression of scintigrams[71]. Furthermore several workers have suggested the need to pre-process ANN inputs, e.g. Hallam et al [3] and Hutchinson et al[79].

An ANN using error back propagation for learning was used to classify images into those with, and those without PE and/or COAD, using compressed input data based on PCA analysis. The ANN was trained to detect the view of the image (anterior, posterior etc) and abnormalities other than PE. The accuracy with which it gives its output for "unseen" images was compared for perfusion, ventilation, and images composed by subtracting perfusion from ventilation images[80].

8.2. Experimental Methods

Two non-overlapping random groups were created, A and B, each containing scintigraphic images from 10 subjects. There were 3 views per subject, posterior, left and right oblique posterior, and there was 3 types of image per view, perfusion, ventilation, and ventilation minus perfusion.

The perfusion images are capable of excluding PE if they are normal, and PE may be diagnosed with some degree of confidence using perfusion alone. However as previously noted there is improved diagnostic capability if ventilation images are used together with perfusion. Thus the subtracted images may provide a better input as it is the mismatching that is important. Ventilation images alone should not be capable of diagnosing PE.

Each group contained 90 images. In all cases a diagnosis had been given by a radiologist expert in the area of nuclear images.

8.3. Method of Image Acquisition

The images were obtained in the same way as in chapter 7.

PCA was used to compress all the images of each group. Each image, which was standardised in size by a median filter technique to 64 by 32 pixels, was rasterised into 8 by 8 segments, and principal components (PCs) of these segments for each of the groups were calculated. Using the first 4 PCs, each 8 by 8 segment was reduced from 64 dimensions to 4 dimensions. The 32 segments forming each image were thus reduced from 2048 to 128 dimensions. The values from Group A were then used as input to an MLP using the error backpropagation learning rule. Outputs were constructed such that a training value of 1.0 constituted the presence of that feature, and 0.0 its absence. There were 8 outputs which were :-

1. PE output (ie. set to 1.0 if PE present, 0 otherwise).
2. COAD output.
3. Perfusion output.
4. Ventilation output.
5. Subtraction output, ie. ventilation minus perfusion.
6. Posterior output.
7. Left posterior oblique output.
8. Right posterior oblique output.

The compressed values of Group A were used to train the network using a learning rate of 0.06 and a momentum (α) of 0.09, using a proprietary network package (NeuralWare) running under MS DOS. The reason for using this package in preference to the PDP was purely pragmatic. There were several micros available, and therefore several experimental runs could be conducted simultaneously. The reduced size of the networks due to the input data compression allowed the use of the micros, where previous experiments using raw data in Chapter 6 were not practicable on micros. The learning rate η was found empirically by trial and error, though a dynamic method of parameter adjustment is tested later in Chapter 9. Momentum values were described in the NeuralWare user handbook as optimal in the ratio of 0.6:0.9 for $\eta:\alpha$. This is not what [16] suggest, but the parameters chosen allowed convergence is nearly all cases, and were accordingly not changed. (A higher α would have given faster convergence but the end result would have probably been similar). Training continued until convergence was reached, or until 50,000 iterations had been completed, whichever was the sooner. Convergence was considered to have been reached when the network correctly identified the images with respect to type of view, image type, and presence of PE and COAD. The criterion for convergence to a particular output was that the output was greater than or equal to 0.8 for an output that should be 1.0, or less than or equal to 0.2 for an output that should be 0.0.

8.4. Results

8.4.1. Convergence on Training Set

Several network designs were tested, all with 128 inputs and 8 outputs as discussed above, but with a varied number of hidden units, in one hidden layer. In all 3, 10 and 20 units were used in the hidden layer. The networks using 10 or 20 hidden units converged on the training set to give the desired outputs. The network with 3 units converged to give the correct desired outputs for the image type with respect to whether the image was ventilation, perfusion or subtraction, but gave outputs for the view type that were mostly in the range 0.2 - 0.4, with a few true negatives, and no true positive values. The outputs for PE and COAD in the 3 unit hidden layer network did not converge satisfactorily, even after 50,000 iterations, and while virtually all PE images were correctly positively identified, so were the non-PE images (falsely) identified as PEs.

The fact that 3 is too few hidden units is in agreement with[48] who state the number of hidden units should be at least as many as the number of classes to be described (which here is 8).

8.4.2. Performance on Unseen Data

No network gave totally acceptable outputs for "unseen" data. The results for PE from the unseen data are given in Table 8.1. This shows that although in all cases except the 3 unit hidden layer network, convergence was achieved, consistent learning for PE occurrence had not occurred. The results for COAD were similar.

The networks gave the correct results for the view of the image, and for the type of image (perfusion, ventilation etc) in almost all cases, but seemed unable to determine with complete accuracy whether the image contained an abnormality. The perfusion and subtraction images had made a reasonable classification in the 10 and 20 unit hidden layer networks. In each case the true positives outnumbered the false positives by 5 or 4 to 1, and the true negatives were about 2:1 over false negatives for perfusion, though subtraction images showed a high false negative rate.

Table 8.1 Network Results All Patterns for PE (Ventilation, Perfusion, Subtraction)

Type images	Hidden units	Output Units	True +	False +	True -	False -
Perf	3	8	7	9	3	3
Perf	10	8	4	1	11	5
Vent	10	8	5	4	2	3
Sub	10	8	5	1	7	5
Perf	20	8	5	1	11	4
Vent	20	8	7	7	1	5
Sub	20	8	4	1	2	4

8.4.3. Restricting Learning to Perfusion Images

One should not be able to determine the presence of PE from a ventilation image, and the results above are consistent in that respect. Further the inclusion of ventilation images may have made it more difficult for the network to converge to a correct solution with respect to PE. Therefore a network was trained on only perfusion images, with a hidden layer of 10 units and 8 outputs. This also however failed to consistently detect PE on "unseen" images, though it converged on the training set to give the desired output in every case. The performance was reduced to 2:1 for true positives against false positives. The ratio for true to false negatives was not altered. The reduction in performance may indicate that the subtraction images were giving useful information to the original ANN.

8.4.4. Reducing the Number of Outputs

The number of outputs was reduced to one, namely that for PE, to test whether conflict with other outputs was causing the inability to give the correct response. This would also reduce the total number of connections in the network, which as previously stated is advantageous. With a hidden layer of 20 units and an output layer of 1 unit the results were only slightly improved compared with the 8 output case, indicating that this was not the problem. As a final test a network with 100 units in the hidden layer and one output was tried, with worse results. Table 8.2 contains the results for PE from unseen data for both 8 and 1 output cases.

Table 8.2 Network Results Perfusion Patterns Only for PE

Type images	Hidden units	Output Units	True +	False +	True -	False -
Perf	1	1	7	6	4	2
Perf	10	1	6	3	5	4
Perf	20	1	6	2	5	2
Perf	100	1	6	5	4	2

8.5. Discussion

These results are encouraging. The classification into different views, or into perfusion/ventilation is not an interesting one, in that this information is known already by the clinician, but it is shown that the ANN can distinguish this much at least in the images. The diagnosis of PE is the main reason for performing lung scintigrams, and any future automated system will need to be able to detect PE. COAD may be diagnosed by other techniques and clinical examination, however its diagnosis may still be useful, given that one needs to perform scintigraphy for some other purpose.

The classification for PE, while not perfect, showed some ability as true positives and true negatives consistently scored higher than false positives and negatives. These results held for several different network architectures, and with a training set of perfusion alone, and additionally with subtraction and ventilation images. Increasing the hidden layer units above a certain number adds no further accuracy to the system. This is to be expected as an optimal number for the hidden layer will be found that adequately represent the data, and an increase over this simply allows a higher probability of uninteresting one-to-one mappings.

The network may have had suboptimal performance for several reasons :

1. Too few training patterns

2. An unacceptable image noise level

3. Too few inputs to give the necessary information from which to abstract the classification.

In order to rectify the noise problem, directly acquired digital data is required. A larger dimension input set is problematic as a larger input set increasingly slows the network. A larger number of training patterns also slows the network, but only in a linear fashion.

8.6. Observations

A direct data transfer system should be used in subsequent simulations. A larger training set should be given to an ANN, utilising inputs found to be most useful for classification of PE.

The defects are seen in some views but not all views necessarily, the network may have been more successful if only views with a noted abnormality were used. Clinical information available to a doctor, such as X Ray or clinical history was not available to the network. This information could be mixed with an embedded ANN output in an expert system using possibly conventional AI techniques. This sort of marriage of conventional AI and ANNs has been done in other areas e.g. [3]

9. DYNAMIC TUNING OF PARAMETERS APPLIED TO IMPEDANCE IMAGING AND PIXEL PREDICTION

9.1. Introduction

In previous chapters all learning in MLPs used the standard technique as used by Rumelhart et al[16]. There are however many potential methods of accelerating learning, e.g[25]. In a comparison of several methods, it was found[81] that adaptive training, where momentum (α) and learning rate (η) are dynamically changed to be faster than the two other methods tested (conjugate gradient and delta-bar method).

Images used in a related study on impedance imaging were found to be difficult to obtain convergence on. A dynamic parameter method was explored to solve this problem, and as the methods could be used also on the data from the lung scintigrams, the results are reported here, and compared with the scintigram data. In particular the impedance data provided a known difficult test case for the dynamic method, which should it succeed may well offer advantage in converging the scintigram data nets.

9.2. Impedance Tomography

This section uses data from work done by an MSc student[11]. No work has been done by the author on impedance imaging practically, but the data supplied by[11] has been analysed by the author, and the experiments on dynamic training and data reduction by PCA were done by the author.

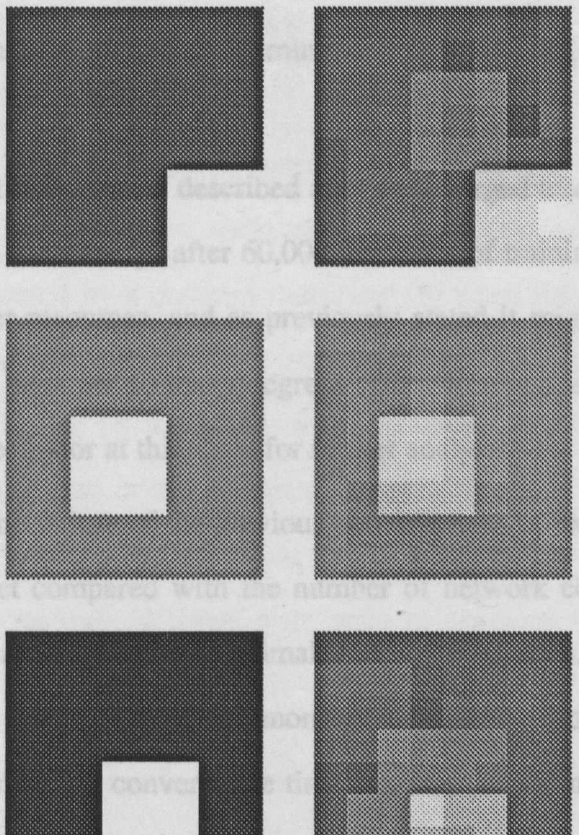
Impedance tomography is concerned with determining the conductivity distribution of a medium from peripheral voltage measurements for a given injection current[10]. A constant current source is applied between two electrodes and differential voltage measurements are taken at the remaining electrodes. The current source is then moved to the adjacent pair of electrodes and the measurement repeated. This process is repeated until all adjacent pairs have been used. For 16 electrodes this gives 208 voltage measurements. Due to symmetry only 104 measurements are independent, but in reality superimposed noise would make it worthwhile to include all measurements. This measurement set is not unique, but there is both experimental and theoretical evidence to suggest that this is a preferred set to provide a unique solution to the problem[10].

For a known conductivity distribution, solution for the peripheral voltage measurements is termed the forward problem. In practice, however, we know the peripheral voltages but require the conductivity distribution. This is referred to as the inverse problem (an excellent review of reconstruction algorithms is given by Yorkey and Webster[82]).

For a given conductivity distribution and injection current, the forward problem can be solved using the finite element technique[83].

In the work of [11] two training sets were produced, a set of 60 random conductivity distributions (Set 1) and a set of 73 rectangular conductivity distributions (Set 2, where the grey scale represents normalised conductivity) with the square at different positions and/or different background/foreground conductivity ratios. Figure 9.1 shows some of the rectangular conductivity distribution data (reproduced by permission of P. Neaves).

Figure 9.1 Unseen Conductivity Distributions (left) and the ANN Restored Images (right)



The problem was to get an artificial neural network (ANN) to solve the inverse problem, using the known conductances as the training set and the known outputs as the target set. In the setup described here 208 inputs mapped to 64 outputs with a single hidden layer of 100 units. All values were normalised to lie between 0 and 1. While this training set is too small to guarantee learning, it might converge, albeit to a possibly incorrect value.

It was found that the network described above converged after 27,000 iterations for Set 2, but Set 1 failed to converge after 60,000 iterations of training. It was clearly very expensive in computer resources, and as previously stated it might converge to a local minimum anyway, as there are so many degrees of freedom in a large network. The data was passed over to the author at this stage for further analysis.

One reason for the failure of the previous network may be expected to be the small size of the training set compared with the number of network connections. While this may as stated above cause a local (suboptimal) minimum solution, it is unlikely to cause reduced convergence speed. For as[84] demonstrated, in situations where the network is incapable of generalising, the convergence time increases as the number of training patterns increases (in that particular case, with a $\frac{4}{3}$ power dependence), and perfect generalisation should give a constant convergence time regardless of the number of patterns. Rather than consider increasing the training patterns for raw data, one must therefore look for a different solution. Changing the architecture and/or reducing the input data set are obvious possibilities.

Principal component analysis (PCA, see Jolliffe [68] for a good introduction) allows information in correlated data to be represented more efficiently by re-expressing the data with a reduced number of orthogonal vectors, called principal components (PCs). 13 PCs contained 100% of the data in Set 2, and 20 PCs contained over 98% of the data in Set 1.

A second set of networks using the first 20 PCs of the data, and 10 units in a hidden layer, with 64 outputs were trained. These were orders of magnitude faster, and contained a much reduced connectivity due to the reduction in inputs and hidden units. However they still sometimes failed to converge in a satisfactory manner (see below for details).

9.3. The Accelerated Convergence Method

The method described by Vogl et al [85] has been used to produce convergence by dynamically adjusting the parameters of momentum and learning rate. It has been shown that this allows convergence where static parameters failed, and speeds convergence. Essentially one increases η by a constant multiplier (θ) when the total sum square error (tsse) is decreasing from one epoch to the next, and decrease it by a different value (β) when the tsse is increasing by more than a few percent (they suggest 1-5%), setting momentum to zero until tsse decreases again. In particular they stressed the utility of making weight changes at the end of epochs (ie. after every pattern has been presented) rather than after each pattern. Weights were reset to previous values every time tsse increased.

9.4. Use of the Method on Impedance Training Set

The following terms will be used :

tsse = total sum square error

η = learning rate

η_i = initial learning rate

η_l = lowest bound for learning rate

θ = scalar to multiply η by when tsse decreasing

β = scalar to multiply η by when tsse increasing by more than 1%

Various learning regimes were applied to a PCA reduced Set 1 (Data Set A), and PCA reduced Set 2 (Set B). 10,000 epochs of training were given in each experimental run.

9.5. Updating Weights After Each Pattern

Using this standard technique Data Set A quickly converged to a tsse of around 200 with $\eta = 0.005$ (Figure 9.2), and slightly more rapidly to the same value for $\eta = 0.1$ (Figure 9.3), though the curve here is not as smooth. This may be caused by a higher η allowing the network to move from side to side across an error ravine. For if the error function contains a ravine, and the current error is placed on one side of it, the largest gradient is straight down the ravine, but if one proceeds in that direction too quickly, one ends up on the other side of the ravine, and not at its base.

Momentum was fixed at 0.9 (and was kept at this value in all subsequent experiments using static parameters, or switched between 0.0 and 0.9 in dynamic parameter training), as according to [16] a high momentum allows a high η and faster convergence. Data Set B however showed convergence at $\eta = 0.005$ (Figure 9.4) and no convergence at $\eta = 0.1$ (Figure 9.5).

Figures 9.2 Pattern Learning Data tsse Set A $\eta = 0.005$

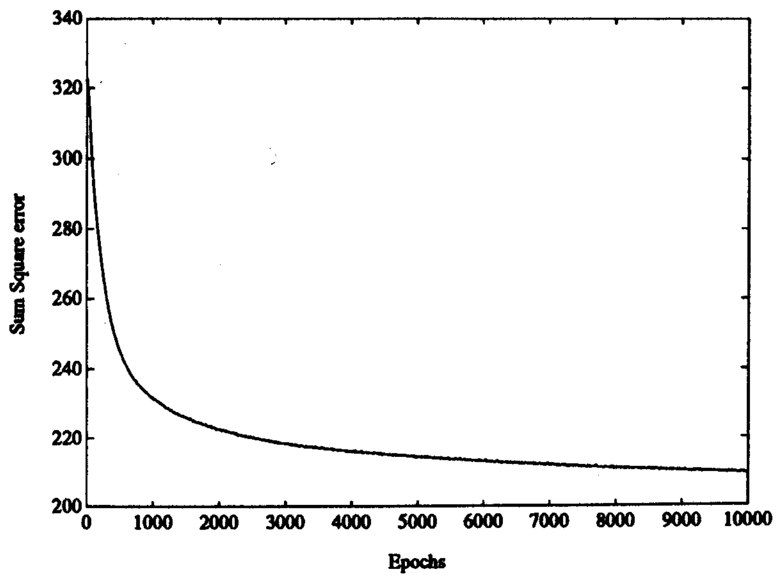


Figure 9.3 Pattern Learning Data tsse Set A $\eta = 0.1$

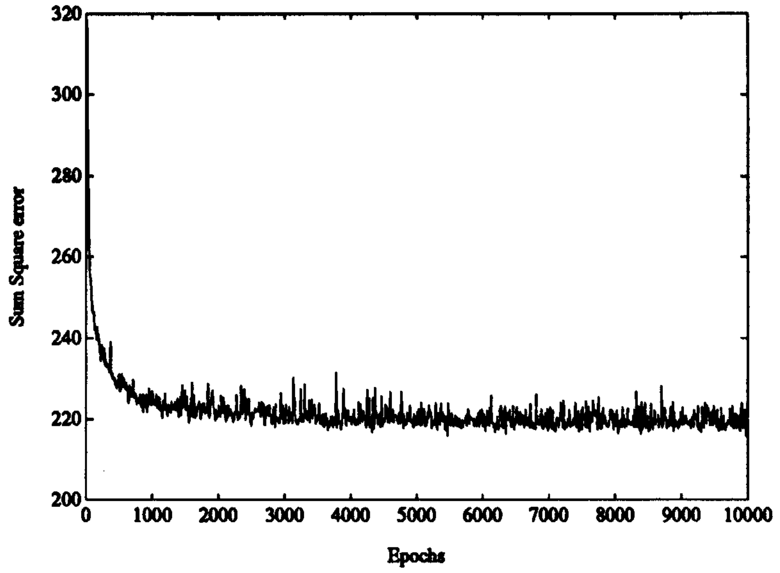


Figure 9.4 Pattern Learning Data Set B tsse $\eta = 0005$

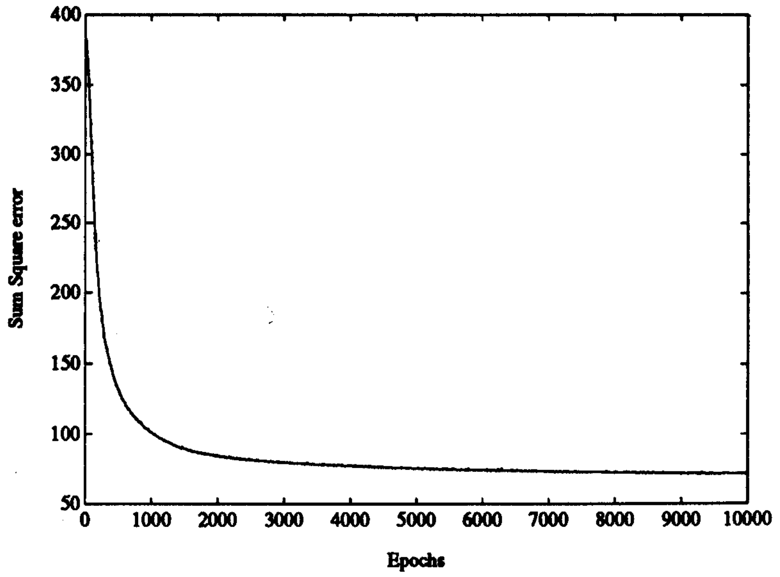
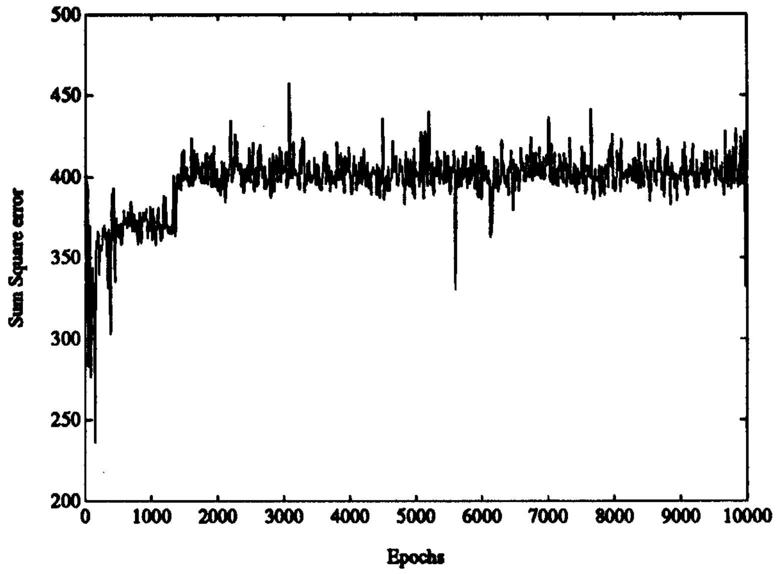


Figure 9.5 Pattern Learning Data Set B tsse $\eta = 0.1$



9.6. Updating Weights After Each Epoch

Data Set A converged to a similar tsse with an $\eta = 0.005$ as in pattern updating, Figure 9.6). An η of 0.1 failed to converge. An η of 0.05 converged, but only after 6,500 iterations (Figure 9.7), the tsse was still fluctuating considerably.

Figure 9.6 Data Set A Epoch Learning tsse $\eta = 0.005$

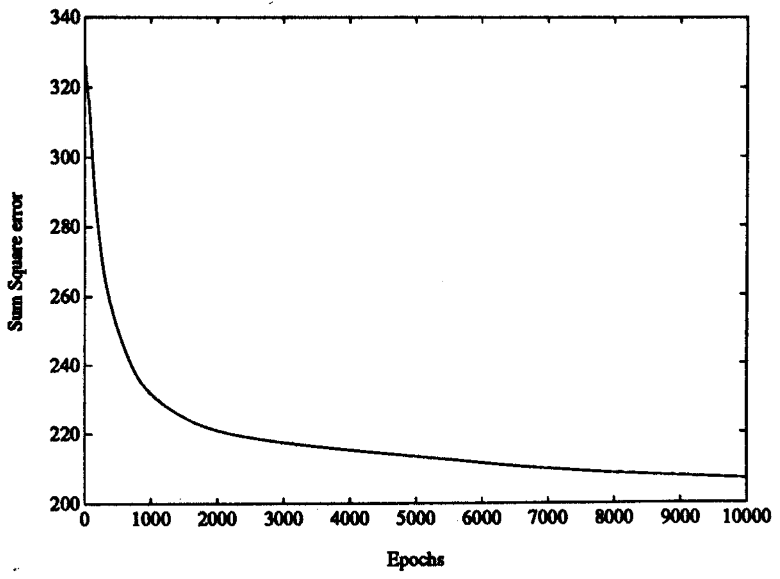
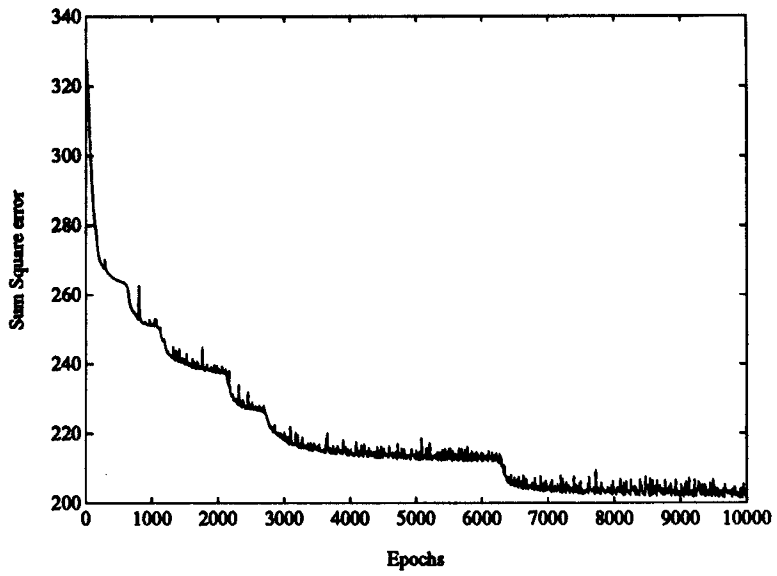
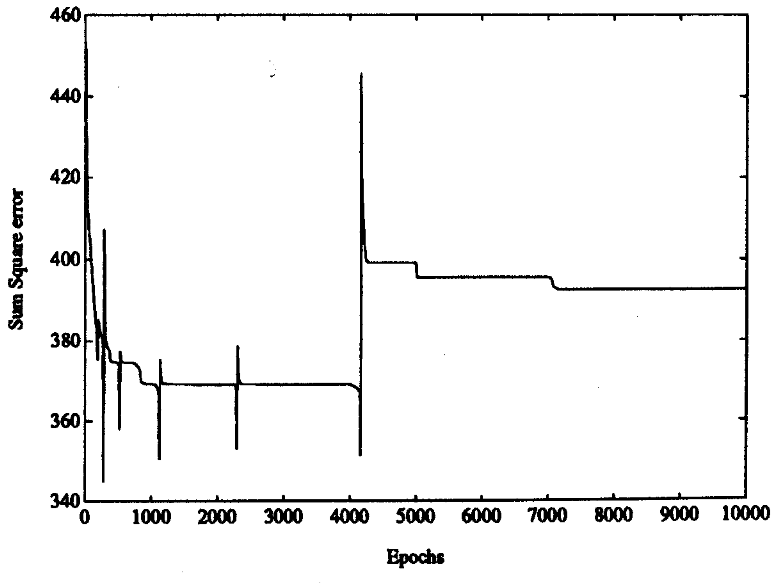


Figure 9.7 Data Set A Epoch Learning tsse $\eta = 0.05$ (lower)



Data Set B failed to converge at any of the η values tried, 0.1, 0.05 and even 0.005 failed to produce a tsse below the random starting value (Figure 9.8), and oscillated wildly.

Figure 9.8 Epoch Learning Data Set B $\eta = 0.005$



9.7. Dynamic Learning Rate Change After Epochs

The dynamic method as described by Vogl et al was then applied, where any increase above 1% of the tsse caused a reduction in η. As in the previous study θ was set to 1.05, and β to 0.7, and these values were used for all experiments.

The learning rate would on occasions dive to such low levels that the network stopped learning and remained static, To avoid this a low cutoff η was set, A cutoff limit of 0.001 was then tried for the lower learning rate limit (η_l). and the algorithm adapted to revert to normal back propagation and maintain a steady η rather than go outside this range, but keep momentum to zero until the tsse went down again, weights being updated as normal. A low η_i was tried to rule out the possibility of initial saturation of the network.

High η, significantly above one, have been shown to be effective in some cases[86]. It has been reported that genetic algorithms (GAs) trained to give an η for a network at a given point in the convergence[87], sometimes gave an η significantly higher than unity, up to 12.8. The η here never reached very high values, and no upper range was allocated for its value.

Using Data Set A network with an η_i of 0.05 convergence was achieved (Figure 9.9a). The η value (Figure 9.9b) reduced from an η_i of 0.05 to a stable value around 0.005. The convergence is however no improvement over the static epoch learning with the η of 0.005 (Figure 9.6). It is better than the the static η of 0.05 (Figure 9.7). Reducing the η_i to 0.001 (Figure 9.10a) gave much faster convergence, and the learning rate oscillated between values close to zero and about 0.025 (Figure 9.10b). η_i appears to be a critical factor, and should be a low value, the η is then allowed to increase under the algorithm to an optimal value.

Data Set B converged with η_i = 0.001, (Figure 9.11a). The η stabilised at about 0.01, which is twice the η of Figure 9.8 which failed. It is likely that the η of Figure 9.8 was too high to initiate the network learning, sending the network into a local minimum from which it could not escape.

Figure 9.9a Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.05

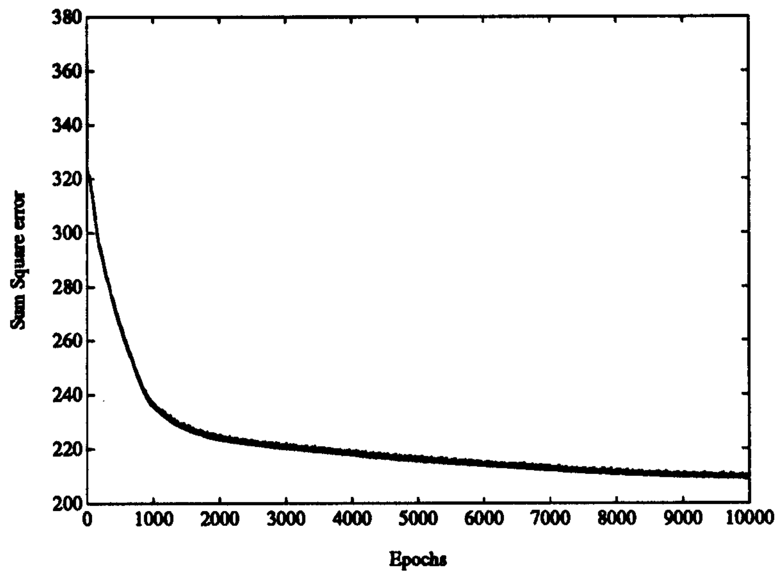


Figure 9.9b Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.05

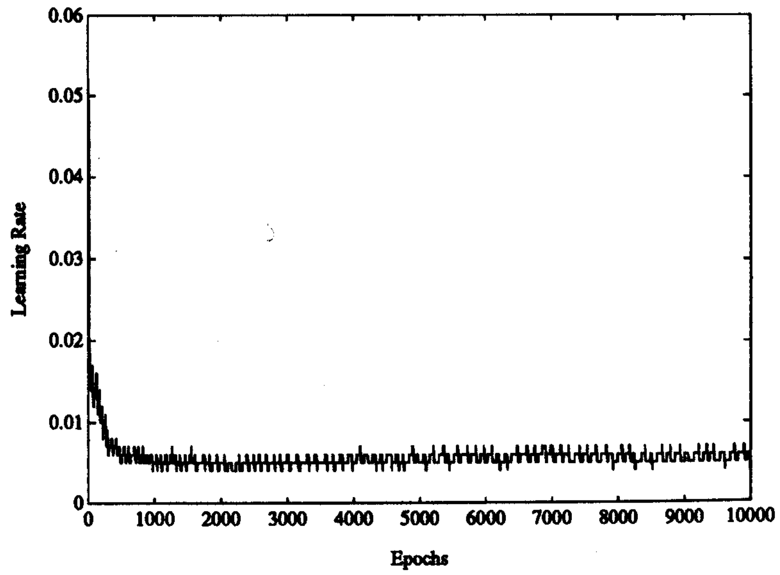


Figure 9.10a Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.001

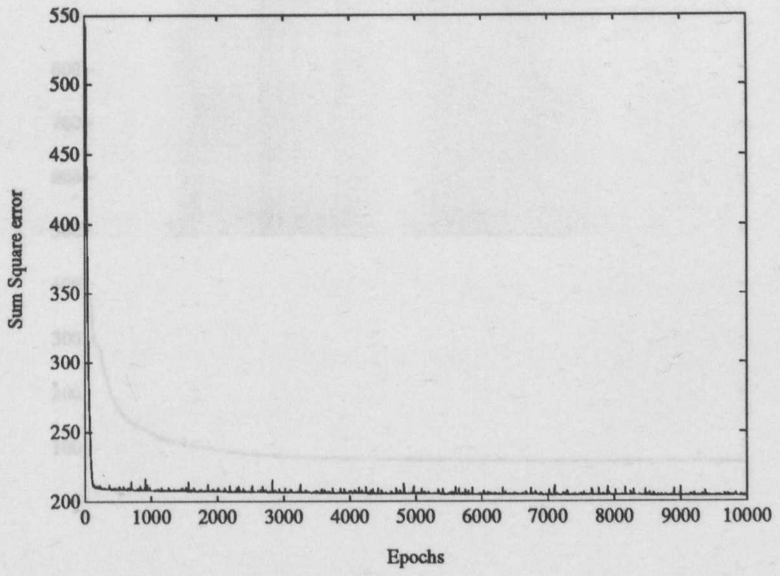


Figure 9.10b Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.001

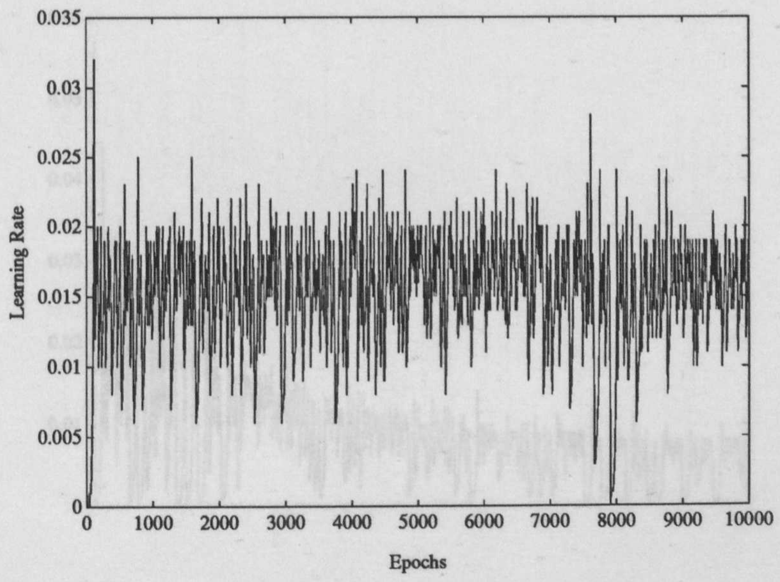


Figure 9.11a Data Set B Epoch Learning Dynamic Parameter Tuning η_i = 0.001

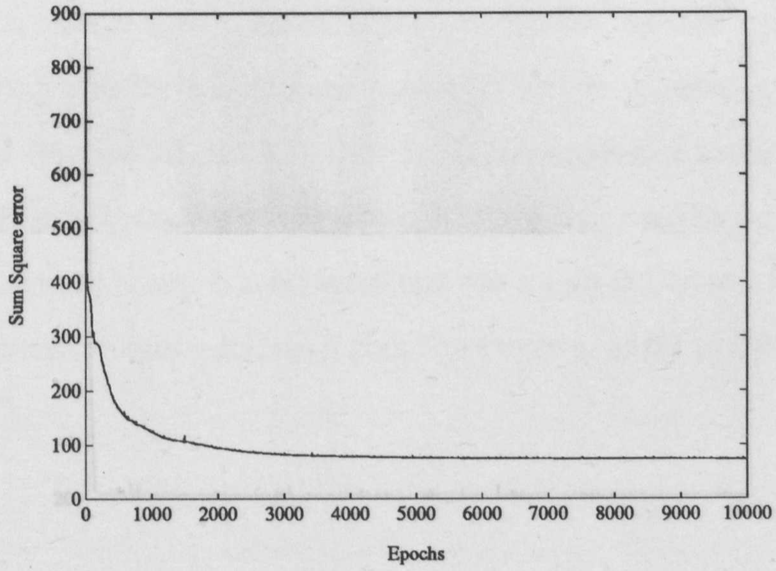
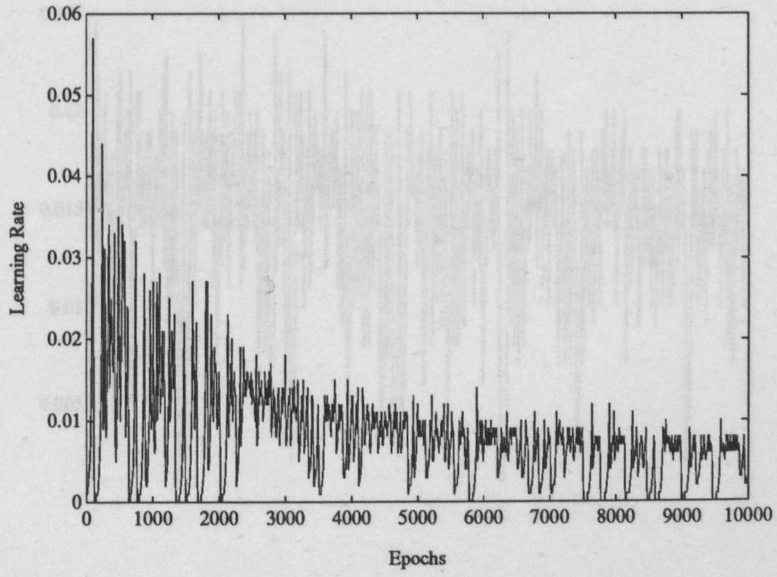


Figure 9.11b Data Set B Epoch Learning Dynamic Parameter Tuning η_i = 0.001



Increasing the η_i for Data Set A to 0.005 (Figure 9.12a) gave a near identical result as for η_i of 0.001 (Figure 9.10a), and not intermediate between Figure 9.10a and 9.9a (η_i 0.05), thus the η_i appears to have a threshold value above which it is poorly behaved, and below which it is not critical for convergence speed.

Figure 9.12a Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.005

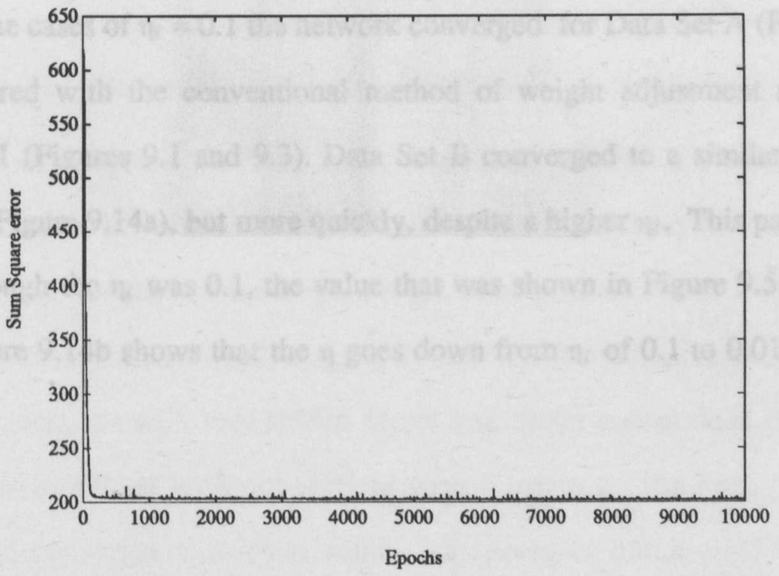
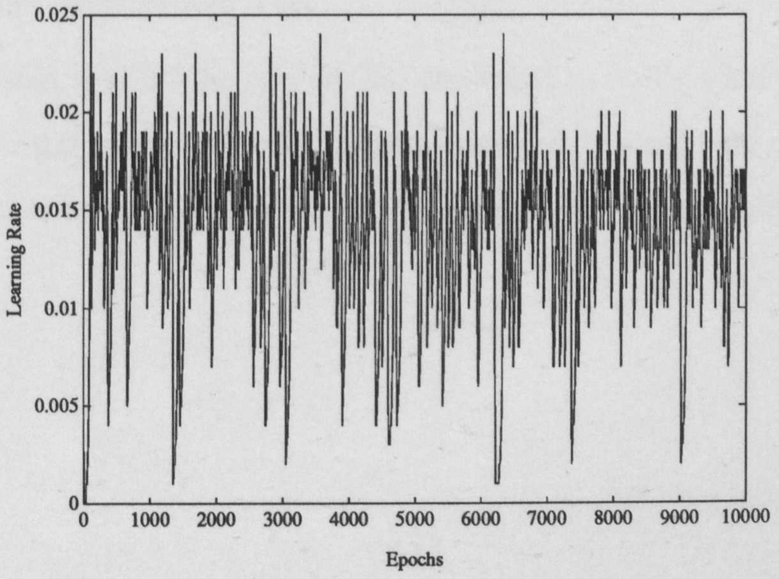


Figure 9.12b Data Set A Epoch Learning Dynamic Parameter Tuning η_i = 0.005



9.8. Adaptive Learning Rate After Pattern Presentation

The algorithm of Vogl et al was adapted then for pattern learning, ie. where weights are updated after each pattern presentation as opposed to each epoch. The weights at each epoch were stored, and after another epoch, if the tsse was increased, the weights were restored to their values during the previous epoch and η was reduced and momentum stopped, or if decreased η was scaled upwards and momentum restarted. One could have stored weights after each pattern, but this would have slowed the learning process down much more. In the cases of $\eta_i = 0.1$ the network converged for Data Set A (Figure 9.13a) similarly compared with the conventional method of weight adjustment and an η of 0.005, or $\eta = 0.1$ (Figures 9.1 and 9.3). Data Set B converged to a similar value as in epoch learning (Figure 9.14a), but more quickly, despite a higher η_i. This particular case worked even though the η_i was 0.1, the value that was shown in Figure 9.5 not to work for static η. Figure 9.14b shows that the η goes down from η_i of 0.1 to 0.01, at which it stabilises.

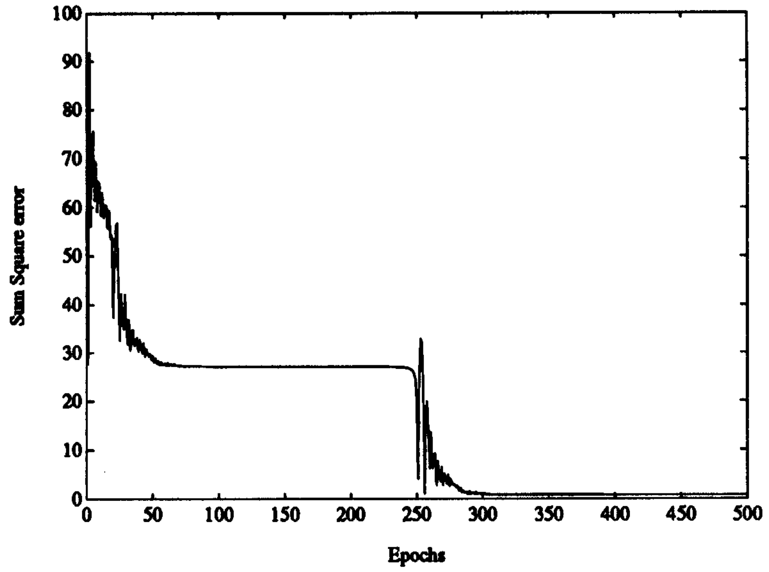
PAGES
MISSING
IN
ORIGINAL

these results were the exception rather than the rule. One could have added noise to the data synthetically, and this would have the additional advantage of increasing the training set (by adding noise on the same pattern several times). However impedance images are not a primary focus of this thesis, and the experiments of [11] were not repeated.

Rather a set of noisy images (Data Set C) were used to create a prediction network as described in Chapter 10, where 4 pixels were to be predicted from their nearest neighbours. The images in this study were lung scintigram images, which are subject to noise, as the original source is a radio-active agent. It had been found experimentally[88] that fully connected networks with two hidden layers and direct connections from input to output in addition to hidden layer connections gave a low tsse. The learning curve had been smooth, and convergence reached within 3-5 epochs of training (127 patterns) in many consecutive runs using pattern learning. The tsse returned by an unseen test set was similar to the training set. It was of interest to see whether the technique would improve convergence on this "well behaved" data.

With a constant η of 0.1 the network had converged smoothly within 300 epochs using epoch training (Figure 9.15), and a test set not previously seen by the network gave similar tsse. This is orders of magnitude slower than pattern training with identical η (Figure 9.17).

Figure 9.15 Epoch Learning Data Set C $\eta = 0.1$



In all cases using dynamic η adjustment the networks converged. Figure 9.16a shows a greatly improved convergence time compared with the static technique. The η curve (Figure 9.16b) shows a reduction in η to a very low rate after convergence appears to have been obtained, and an increase towards the end of the training which had little effect.

Figure 9.16a Epoch Learning Data Set C Dynamic Parameter Tuning η_i = 0.1

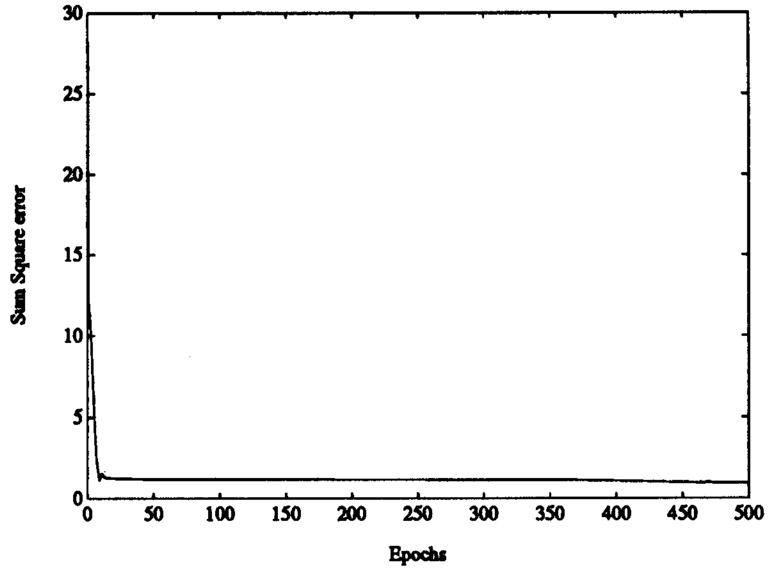


Figure 9.16b Epoch Learning Data Set C Dynamic Parameter Tuning η_i = 0.1

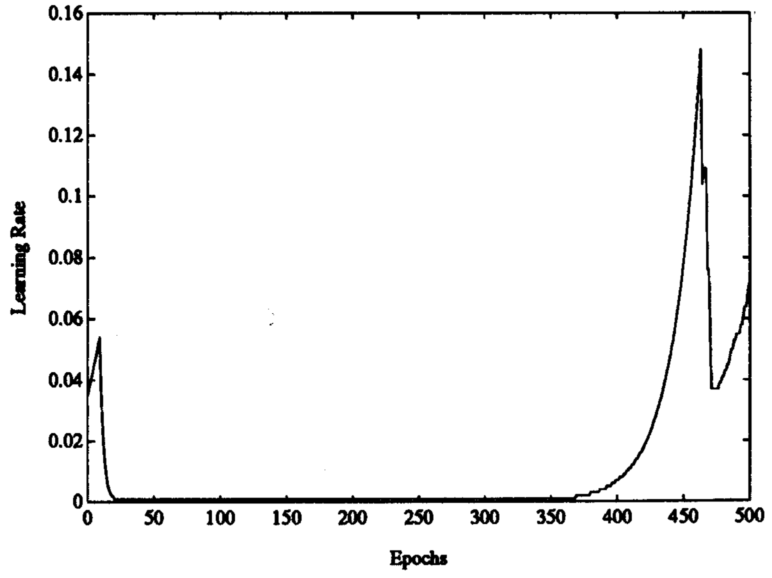


Figure 9.17 Pattern Learning Data Set C $\eta = 0.1$

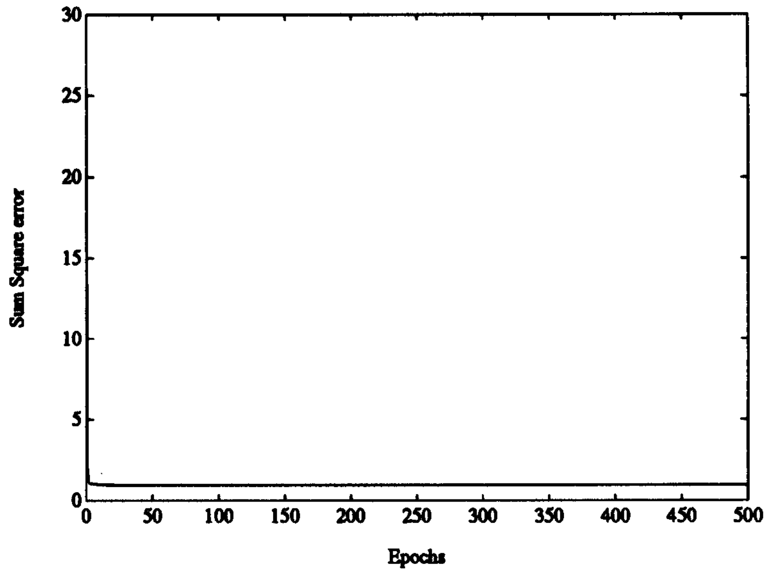


Figure 9.18a Pattern Learning Data Set C Dynamic Parameter Tuning η_i = 0.1

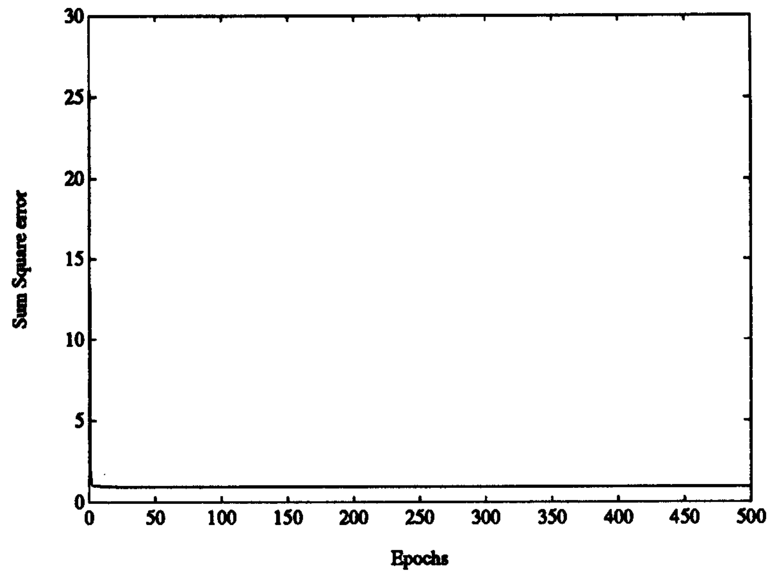
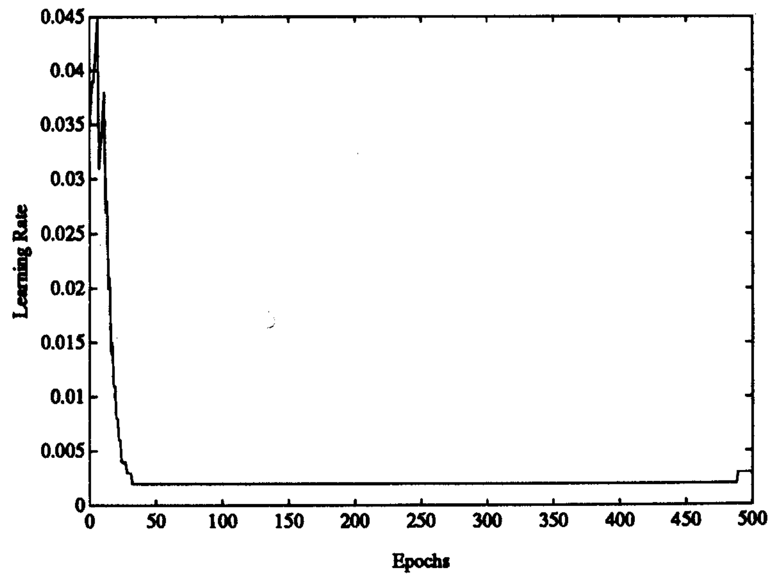


Figure 9.18b Pattern Learning Data Set C Dynamic Parameter Tuning η_i = 0.1



Looking at adjustment of weights after each pattern, with adjustment to η after epochs as described above, there is little discernible difference between the two slopes for static and dynamic parameter learning (Figures 9.17 and 9.18a), and both are an improvement on either of the epoch training methods,

As in the epoch dynamic learning, the pattern dynamic learning showed a rapid reduction in η, but there was only a minimal final rise.

9.10. Conclusion

These two sets of data, one exact, the other with noise, exhibited different behaviour. The scintigram data showed little difference for constant η against the dynamic η adjustment method for pattern learning, though epoch learning was improved with the dynamic method. The epoch method was no improvement over pattern learning. However it should be noted that in these simulations all patterns were presented in random order, and if many patterns of the same class were presented in batches the results may have been very different.

The impedance image data was significantly better using dynamic η adjustment. A cutoff of η_l may be necessary. An initial η_i needs to be chosen with care as the networks are sensitive to high η_i. Updating the weights after each epoch does not seem necessarily to be an improvement to updating after each pattern, quite the reverse in some of these particular cases.

The impedance data (Data Sets A and B) seemed much more difficult to obtain convergence on than the prediction data (Data Set C), and it was on this more "difficult" data that the accelerated method was an improvement.

The networks were sensitive to η_i. This latter point arises since on the first epoch of the network learning, there is no previous tsse against which to compare, and the network is forced to accept the result and update the weights, which may lead to "flooding" of the units if too high an η_i is used.

Accordingly in the remaining chapters dynamic training has not been implemented, as it appears to offer no particular advantage over the standard method with the data used in this study, though under other circumstances it may be advantageous. One particular use would be to start the network at a very low η_i, and use the dynamic method as a means of establishing an appropriate η, where a priori it is not known at what level to set the learning rate.

10. LOCAL PREDICTION TECHNIQUES

10.1. Introduction

As in previous chapters for the experiments described below one paradigm was used only. This was the multi-layered perceptron (MLP) using error back propagation as described in Rumelhart and McLelland[73].

The point of implementing a prediction network is to provide a pre-processor for a subsequent classifying network. Prediction may be achieved by a variety of methods, linear methods being the most common, but nonlinear methods (e.g.[89]) allow more general cases to be analysed (e.g. chaotic time series)[42]. Neural nets have been suggested[42] in prediction problems.

The method described here is to train a network on normal images, where some group of pixels locally predict a neighbouring group of pixels. It may be the case that when images with abnormalities are presented to such a network, that the predicted pixels and the actual pixels of the abnormal area will be very different, and produce a high tsse, which could be used to flag areas of potential further classification.

Thus the prediction network should output a larger error if it is trained on normal images and then given images with abnormalities, since it has not "seen" the abnormal areas. The error should be localised to the portions of the image that contain these abnormalities. Such segments could be fed into a classifying ANN to distinguish between (say) PE, chronic obstructive airways disease (COAD) and artifacts. An assessment of such nets for prediction is presented in[90] and in this chapter.

In all cases the networks consisted of 3 layers, an input layer, an output layer, and between them the "hidden" layer. The layers were fully connected to the layer below or above, ie. all units were connected to all other units in the adjacent layers. The particular net in Figure 10.1b was one of the net used, where the 12 input units are the pixels of a 4 by 4 segment with the middle 4 pixels removed, the output unit is trained to give the missing pixels (see Figure 10.1a).

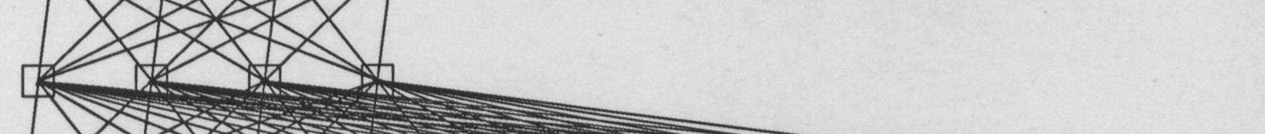
Figure 10.1a Prediction of Inner Segment from Outer Segment ANN

Input	Input	Input	Input
Input	Target	Target	Input
Input	Target	Target	Input
Input	Input	Input	Input

Input = Input pixel value to ANN

Target = Target pixel value to ANN

Figure 10.1b Prediction of Inner Segment from Outer Segment ANN : Network Topology



10.2. Local Prediction of SubSegments

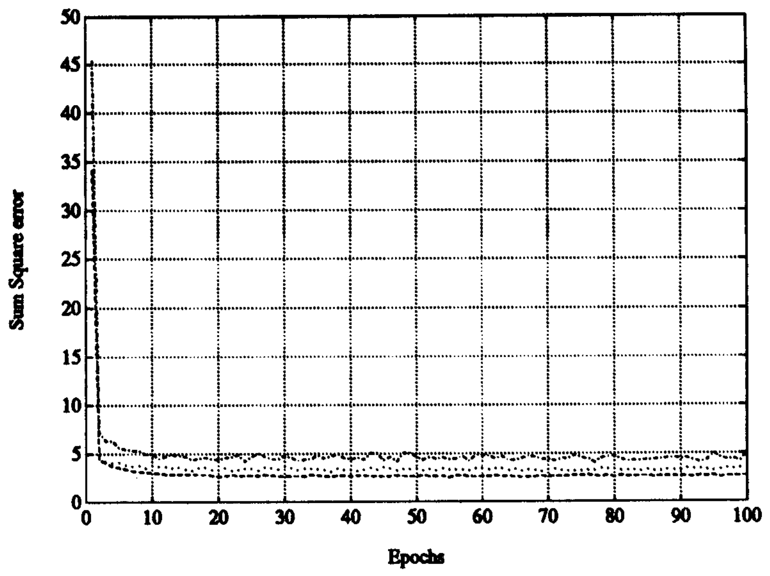
Where chaotic data is used, local methods may provide greater accuracy than global ones. Nonlinear mappings have been "learned" using a local approximation with success by Farmer and Sidorowitch, [89] where they suggested ANNs could be employed to produce a similar mapping. The data used in their study were time series, but similar methods could be used on images, where one part of the image is approximated by considering neighbouring points. Furthermore feeding part of an image to a network, scanning the image section by section, offers the possibility of stating where an abnormality exists in the image. Experiments were performed to test the viability of such an approach.

10.2.1. Video Acquired Images

Using an atlas of nuclear images, two lung scintigram images were "frame grabbed" using a video camera. One was used to create a training set, and the other to create a test set. The images were standardised using a median filter technique to 64 by 32 pixels in size. The pixel values were scaled to lie in the range [0,1], Segments of an image of various square sizes with values in square subsegments removed, were presented to a MLP in an attempt to predict the missing sub-segment. The results were compared to a prediction method based on using the mean of the larger segment to predict the missing data points. The networks gave much better results than a simple mean value, except for the one pixel case surrounded by only its nearest neighbour. Here the network gave similar results, and is probably simply computing an average value.

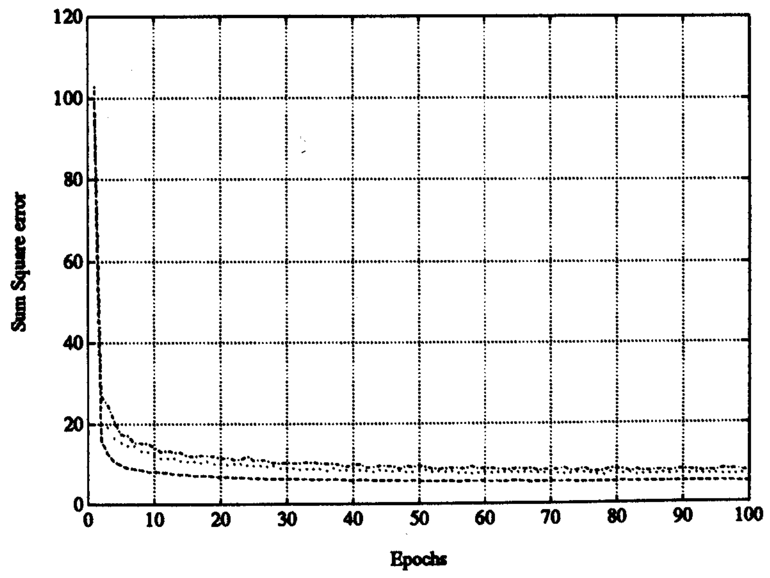
From Figure 10.2-5, which show the total sum square error against epochs, it is evident that convergence was achieved within a few epochs of training.

Figure 10.2 Network Convergence Training Predicting 1 Pixel



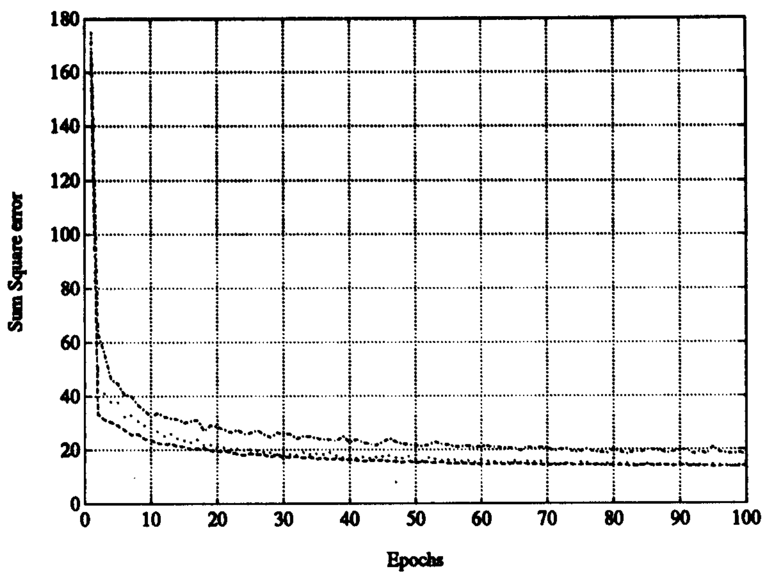
Legend
Segment Surrounding Subsegment by One Pixel _ _ .
Segment Surrounding Subsegment by Two Pixels ...
Segment Surrounding Subsegment by Three Pixels -.-.

Figure 10.3 Network Convergence Training Predicting 4 Pixels



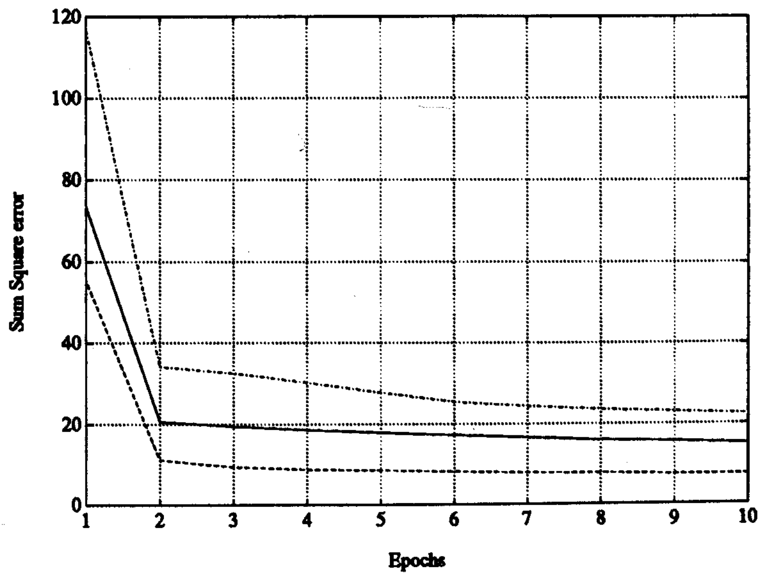
Legend
Segment Surrounding Subsegment by One Pixel _ _
Segment Surrounding Subsegment by Two Pixels ...
Segment Surrounding Subsegment by Three Pixels -.-.

Figure 10.4 Network Convergence Training Predicting 9 Pixels



Legend
Segment Surrounding Subsegment by One Pixel _ _ ·
Segment Surrounding Subsegment by Two Pixels ...
Segment Surrounding Subsegment by Three Pixels -.-.

Figure 10.5 Network Convergence Training Predicting Pixels Using Digital Data



Legend
Segment Surrounding Subsegment by One Pixel _ _
Segment Surrounding Subsegment by Two Pixels ____
Segment Surrounding Subsegment by Three Pixels -.-.

It is seen from Tables 10.1-10.3 that the network gives similar values for training and test data. It is stated from empirical studies [57] that the number of training patterns should be at least of the order of the number of connections in the network. Smaller sized training sets allow the network to obtain one to one mappings which do not allow the network to "generalise". The largest network had 657 connections, (72 - 9 - 9 network) and the number of training patterns were 1680, ie. comfortably in the range O(N), N = connections.

Table 10.1 Errors of Neural Net on Training Set After 100 Epochs
Nuclear Atlas Data

No. Inputs	No. Hidden	No. Output	Tot sum sq. Error
8	1	1	2.685
24	1	1	3.257
48	1	1	5.176
12	4	4	5.258
32	4	4	7.331
60	4	4	7.751
16	9	9	16.165
40	9	9	13.876
72	9	9	18.362

Table 10.2 Errors of Neural Net on Test Set
Nuclear Atlas Data

No. Inputs	No. Hidden	No. Output	Tot sum sq. Error
8	1	1	2.190
24	1	1	6.812
48	1	1	2.174
12	4	4	6.830
32	4	4	5.872
60	4	4	8.186
16	9	9	11.917
40	9	9	13.759
72	9	9	20.759

Table 10.3 Errors Using Mean Value Prediction Method on Test Set
Nuclear Atlas Data

No. Inputs	No. Outputs	Tot sum sq. Error
8	1	3.248
24	1	8.179
48	1	16.215
12	4	36.582
32	4	60.182
60	4	96.763
16	9	154.159
40	9	293.670
72	9	407.723

The error for networks using increasing numbers of neighbouring points does not improve, but rather slowly increases. Consequently there seems no obvious advantage to using larger segments surrounding a given subsegment size, especially as these add to the connections thus requiring larger training sets, and the nets take longer to compute errors.

10.2.2. Directly Acquired Digital Images

Using a recently acquired RS232 interface and supporting software, images were taken directly from the gamma camera system, thus avoiding the problem of addition of noise. The experiments for prediction were repeated with some amendments :-

1) Subsegments were predicted from the next larger segment that surrounds the subsegment only. Three tests were conducted, 2 by 2 subsegment from a 4 by 4 segment, 3 by 3 from 5 by 5 and 4 by 4 from 6 by 6.

2) The images were not standardised for size. The training image was the left lung and the test image the right lung of the same image, thus standardisation was not required.

3) Not all segments were used, as this would have produced too large a training set. One in 10 segments were randomly chosen to go into the training and test sets.

4) Only 10 epochs of training were employed, because the previous experiments showed this to be a sufficient number of epochs.

The results of the networks are shown in Tables 10.4 and 10.5. The numbers of training patterns used were 1482, 1413 and 1451 for the 2 by 2, 3 by 3 and 4 by 4 pixel subsegments respectively, and so were of the order of the largest number of connections in any network. The test set contained 1218, 1221 and 1178 patterns.

Table 10.4 Errors of Neural Net on Training Set After 10 Epochs
Directly Transferred Digital Data

No. Inputs	No. Hidden	No. Output	Tot sum sq. Error	% Error/Pixel
12	4	4	7.418	0.125
16	9	9	15.032	0.118
20	16	16	22.386	0.096

Table 10.5 Errors of Neural Net on Test Set
Directly Transferred Digital Data

No. Inputs	No. Hidden	No. Output	Tot sum sq. Error	% Error/Pixel
12	4	4	6.103	0.125
16	9	9	14.636	0.133
20	16	16	49.146	0.260

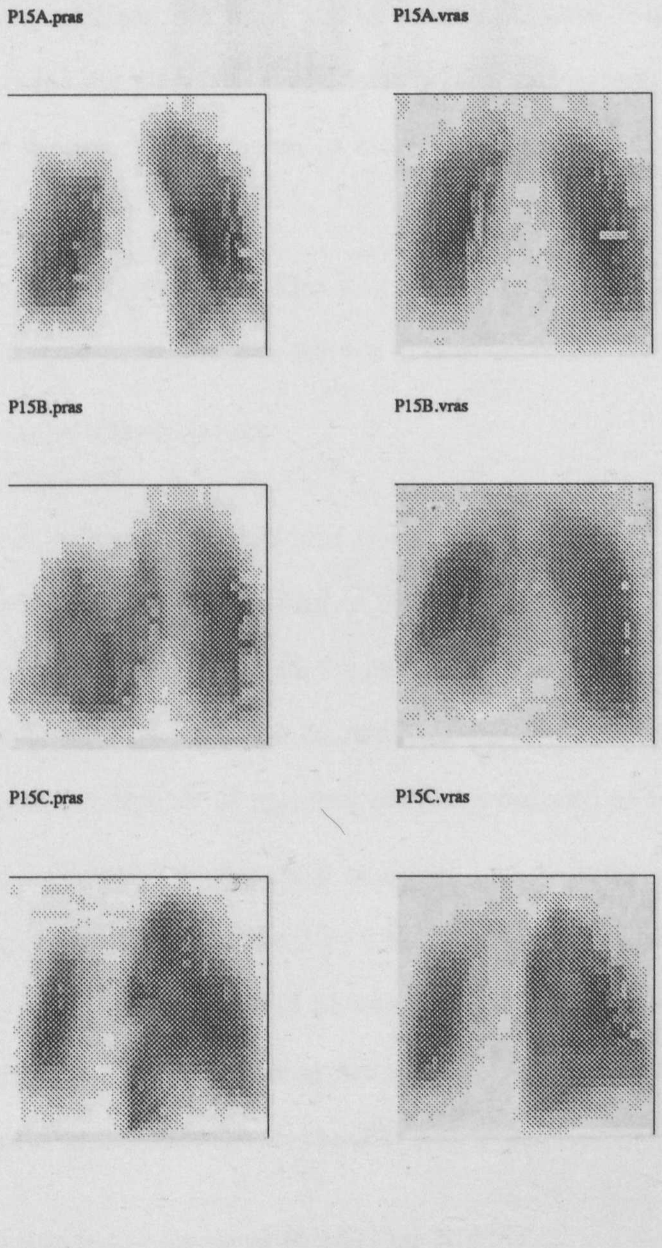
Errors per pixel were calculated for these experiments, and it may be seen that the error in the test patterns rose slightly as the subsegment size increased. Since this did not occur in the training set it seems that an incomplete generalistion of the prediction method has occurred, despite the large training set. However the error per pixel for the largest subsegments is still quite low.

10.3. Images Generated by the Prediction Network

A subjective assessment of the prediction technique may be made by comparing the output of the prediction net with the original images. An example of the images used in the study are displayed in Figure 10.6. These images were "frame grabbed" images. The network trained for prediction with normal images was presented with all the PE perfusion images as a test set. Using 4 by 4 pixel segments, the inner 2 by 2 pixels were predicted by the network, and these were output to create a "predicted" image. Similarly error images were created. For each output pixel the pattern sum square error (psse) for the 2 by 2 pixel subsegment was calculated. This is simply the sum of the square differences between the target values and the output values. By normalising the errors so that the maximum error was given a value corresponding to white (255 in this case as 8 bits/pixel are used) and linearly interpolating all over values, an image is produced which shows high error as bright and low error as dark. Figure 10.7 shows both error images (left) and restored images (right). It is clear the output images are close to the original (Figures 10.7 cf 10.6), and that the error image does not appear to show areas of potential PE. The figures show the effect on one image, all other images showed very similar results.

Figure 10.6 Pulmonary Embolus Image: Perfusion Images (left) and ventilation Images (right)

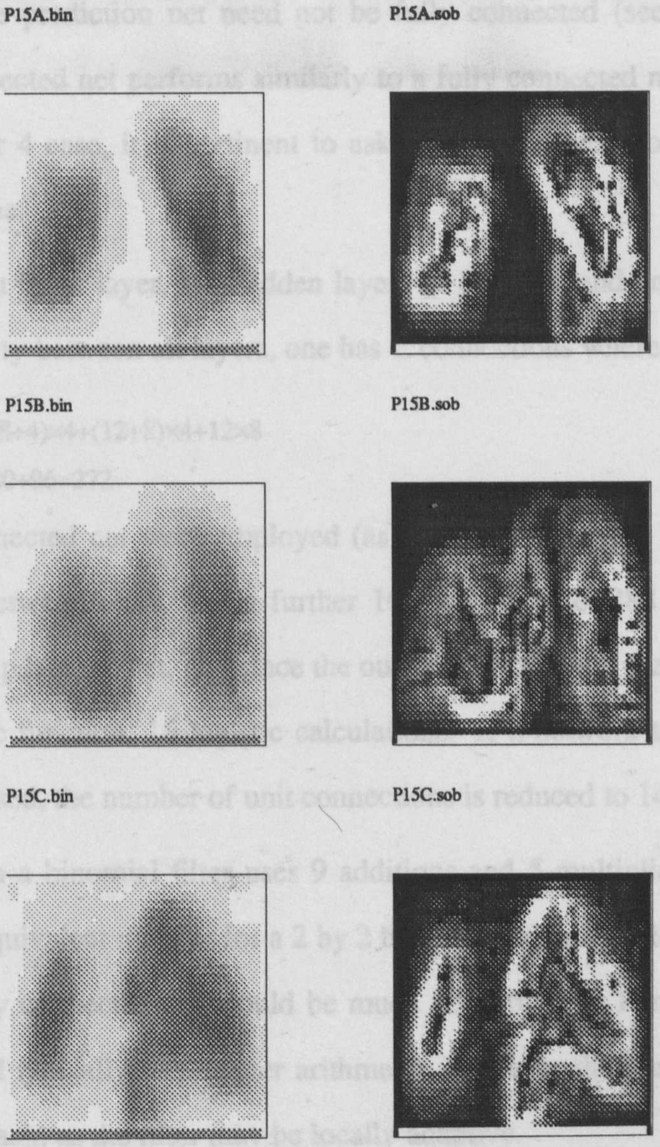
Top Images - posterior-anterior
Middle Images - Left Posterior Oblique
Bottom Images - Right Posterior Oblique



PAGE
MISSING
IN
ORIGINAL

Figure 10.8 Pulmonary Embolus Image Perfusion Image: Binomial Smooth (left) and Sobel Filtered (right)

Top Images - posterior-anterior
Middle Images - Left Posterior Oblique
Bottom Images - Right Posterior Oblique



10.3.1. Smoothing Effect of Prediction Net

It seems that prediction nets act as a smoothing filter, and indeed if one compares the original images with the restored images using the prediction nets much of the noise has been removed. As a comparison with a standard technique, the binomial smooth was applied to the same images (left images in Figure 10.8), and it appears that the resultant images are very similar.

Given that the prediction net need not be fully connected (see Chapter 12), and indeed a 20% connected net performs similarly to a fully connected net with the 12 pixels predicting inner 4 case, it is pertinent to ask whether the net would be an efficient smoothing technique.

With a 12 unit input layer, and hidden layers of 8 and 4, and an output layer of 4, with full connectivity between all layers, one has C connections where :-

$$C=(12+8+4)\times4+(12+8)\times4+12\times8$$
$$C=96+80+96=272$$

If a fully connected net were employed (as is done in Chapter 11) there would be 272 connections between units, and a further 16 bias links, ie. 288 additions and 288 multiplications to update the net, and since the outputs from all but the input layer are put through the logistic function, 16 logistic calculations. If a network topology as used in this chapter were used, the number of unit connections is reduced to 144.

In comparison a binomial filter uses 9 additions and 5 multiplications to compute each pixel, so an equivalent number for a 2 by 2 block would be 36 additions and 20 multiplications. A fully connected net would be much less efficient. Furthermore the binomial smooth would typically use integer arithmetic. However there may still be a place for the ANN approach, as the filter may be locally adaptive.

10.3.2. Edge Detection using Error of Prediction Net

The error of the prediction net has been used to restore images after applying the prediction net, and an example of this is shown in Figure 10.7. As a comparison, the Sobel filter, a standard spatial domain edge detector, is shown in Figure 10.8 (right images). It is clear that while the prediction error image is a sort of edge detector, it is not a particularly good one compared with the Sobel.

10.4. Conclusion

Prediction nets are not useful pre-processors for classification nets. They make smoothing filters however that subjectively appear similar to binomial smoothing filters, and the error gives a crude edge detector.

11. OPTIMISING NETWORK STRUCTURES WITH GENETIC ALGORITHMS

11.1. Introduction

A problem with neural network design is optimising the connections between units, and the number of hidden units. For any input or output unit numbers there are an infinity of ways of joining them via arbitrary numbers of hidden units. Even if the number of hidden units required is known, and currently this is a matter of rule of thumb, there are many permutations of connectivity. In the previous chapters, experiments were undertaken where total connectivity was assumed, however the layout of retinal cells does not have this pattern, and restricting connections from certain inputs to one or a few hidden units, and similarly from hidden to output units, may be more appropriate. Furthermore the reduced networks would run more quickly.

As[91] has explained, many complex systems are sub-optimally designed using a priori assumptions, and it is suggested that Genetic algorithms (GAs) and other adaptive strategies may help reduce this problem.

11.2. Genetic Algorithms

GAs are algorithms used to optimise a set of functions, based on ideas taken from genetics and evolution. Binary strings are used to represent the functions, and an evaluation of the functions determines "fitness". A population of initially randomised strings are mutated (some bits switched in a random manner), and portions of strings are interchanged between strings. The new population (next generation) is further tested. A general overview is given in [92] , for a discussion of GAs and classifier systems see[93]

11.3. Need for Optimisation

Work has shown that using principal components (PCs) of lung scintigrams one may classify the images with a limited accuracy into those with specific abnormalities (pulmonary embolism (PE) and chronic obstructive airways disease (COAD))[94]. In an attempt to improve the capability of the system, it was decided to scan the image, pick out the areas which were possibly abnormal, and use these in a subsequent classification algorithm.

The technique would involve feeding in the pixels of a square area minus an inner subsegment as the input layer, and giving the inner segment as the training layer. Thus the network would be predicting the inner segment from the outer segment. This is a form of pattern completion, and where the predicted segment is in error in excess of some stated threshold, one could use this area as a pattern for the next (classification of abnormalities) ANN.

One can use auto-associative networks such as the distributed memory model [50], but it was considered worthwhile to use the multi-layered perceptron (MLP) using error back propagation as it can map non-linearities.

As was shown in Chapter 10, these nets did not perform the function they were designed for, rather they acted as edge detectors and smoothing filters. However large training sets may readily be made for such nets, they make good test sets for exploring optimising network architecture.

A problem of ANNs is to determine an optimum network design. The number of different networks that can be built on even a small problem is forbidding. Even in the smallest network worth considering, which has four inner pixels, determined by the 12 pixels which surround it, there is a large number of permutations, and indeed if any arbitrary number of hidden layers and/or units is considered it is infinite.

There are good reasons for being concerned about the specific network structure used. It is not sufficient to use a net structure obtained that converges on a training set. The point has previously been made (e.g. Chapter 6) that the size of the network affects performance, speed and accuracy on a test set. In addition it has been shown that over-training of a network, which is characterised by overfitting of data, is less critical in a reduced network size[75]. This is related to the Runge effect, where counter-intuitively adding data points to interpolate between allows in extremis arbitrarily large errors for the interpolated points. In networks the error is actually bounded because the outputs of units are bounded, a less extreme effect is therefore seen. If a network is trained beyond the point of convergence, overtraining may occur. The tsse continues to be reduced, but the error of particular patterns or particular outputs actually increases.

Since it is not known a priori when a network will converge, network structures that minimise this problem are advantageous. It has been shown[75] empirically that the smaller a network in terms of connectivity, the less overtraining to be a problem.

11.4. Genetic Algorithms and Network Design

GAs can search solution spaces efficiently and quickly, and are suited to high dimensional systems[87]. GAs have been used to evolve network architectures. Harp et al[87] used a GA to design ANNs. This system used genes to map to learning rate, connection density and other parameters, in a network trained to learn digits from their binary images. Previous work has shown that GAs can improve network construction. Whitley [95] experimented with GAs learning the connection weights in the XOR problem, a 424-encoder and an addition problem. Genotypes represented connection weights, and 8 bits were used to represent each weight. These weights were used in a network, and the network generated an error in the same way that an MLP using back-propagation would. He found that very small errors could be generated (e.g. 0.00001 average in XOR after 1500 recombinations and a population of 200). Smaller populations gave faster and more accurate results, but were less reliable.

Network architecture has been optimised using GAs. Schaffer et al [96] used a GA to learn the momentum, learning rate, initial weight range, and the connectivity between units. The problem addressed in their work was the minimum interesting coding problem, where the first two bits of a four bit string are noise. The evolved networks were sparsely connected and gave a lower total sum square error (tsse) than a fully connected network.

Miller et al [97] designed networks for the XOR problem, the four quadrant problem, and pattern copying, where the connectivities between units were switched on or off by the genes. They found additional connections were made between input and output, which increased the rate of learning in the MLP, and they stated the solutions were non-intuitive.

de Garis [98] has built a genetically programmed neural network which teaches a pair of sticks to walk. The GA learns the weights in a feedback time dependent network. Dodd[99] showed GAs out-performed pseudo-gradient descent and random search for two problems, one a "toy" problem and the other a large problem (5,000 connections) concerning identification of dolphin sounds.

11.5. Initial Experiments

To gain an idea of what a standard network could do, an MLP was constructed with full connectivity between the input layer, two hidden layers and an output layer. Initially all layers were connected to all other layers (net A), then only connections between adjacent layers were allowed (net B), then the connections between the hidden layers and the output were cut (net C), and finally a network with only input and output layers was constructed (net D, a purely linear network). The rationale for testing net C is that it was desirable to determine whether hidden units unconnected to outputs reduced performance.

These nets were tested on two different problems, the prediction of 2 by 2 pixel segments from the concentric ring surrounding them, ie. 12 units predicting 4, and the prediction of 5 by 5 pixel blocks from the two surrounding concentric rings, ie. 56 units predicting 25. A learning rate of 0.05 was used for the larger network, and 0.1 for the smaller one. In all cases the networks converged.

The number of training patterns used in both training and testing were 115. They were generated from two separate lung scintigram images, where randomly about 1 in a 100 segments were chosen. The number of patterns is small is comparison to the number of connections, and as empirically it is known that pattern number should be of the same order as connectivity[57] , there is a risk of falling into uninteresting one to one mappings. It is clear that on these runs this did not happen as the tsse reported from the test image, which the network had not previously seen is similar to the training error. Hence the "worst possible case" scenario was not encountered.

In the small network there is little difference in the tsse for the different net designs, and it may be that the relationship is linear, though slightly lower errors were obtained using hidden units where no connection was made directly from input to output.

In the larger network, designs with hidden units performed much better than the two linear networks, the fully connected network performing best. It seems likely that large subsegments are related to surrounding areas in a more non-linear fashion. In every case the test data supported the training data.

The convergence rate was fastest for the fully connected network in both small and large networks. The slope was smooth in all cases except in the case of networks with adjacent layers only connected. In both cases there was a period of slow convergence, and in the larger network a sharp kink into what may have been a local minimum.

Table 11.1 : 2 by 2 Pixel Target Network
Errors on Training Data (100 epochs)
12 inputs, 8 in First Hidden Layer
4 in Second Hidden Layer, 4 in Output Layer

Network	tsse
A	0.94
B	0.83
C	1.06
D	1.02

Table 11.2 : 2 by 2 Pixel Target
Network Errors on Test Data (100 epochs)
12 inputs, 8 in First Hidden Layer
4 in Second Hidden Layer, 4 in Output Layer

Network	tsse
A	1.02
B	0.83
C	1.12
D	1.12

Table 11.3 : 5 by 5 pixel Target Network
Errors on Training Data (100 epochs)
56 inputs, 44 in First Hidden Layer
25 in Second Hidden Layer, 25 in Output Layer

Network	tsse
A	3.15
B	4.09
C	4.95
D	5.00

Table 11.4 : 5 by 5 pixel Target Network
Errors on Test Data (100 epochs)
56 inputs, 44 in First Hidden Layer
25 in Second Hidden Layer, 25 in Output Layer

Network	tsse
A	4.26
B	5.15
C	5.94
D	5.77

11.6. Genetic Approach

It has been suggested that GAs work better given a priori knowledge about the domain in which they seek a solution [100], and that such knowledge may not need to be substantial. An example they quote is the traveling salesman problem, where the heuristic added is that typically "good" tours have short edges.

The assumptions built into the GA should not be such as to stop the GA from exploring reasonable solutions. A weak specification has the advantage of greater speed and compactness, but may suffer from this problem. Miller et al [97] noted that strong network specifications have an obvious advantage of reducing human design bias, and a strong specification was used in this work, whereby every connection could be independently set by the GA. As in their study, the software used was the PDP package by Rumelhart and McLelland [50] combined with Grenfenstatte's [92] genetic algorithm system. This GA system uses biologically inspired techniques. A "population" of "genomes" (bit strings) are mapped to "phenotypes" (some user defined mapping, which here is network structure) and "fitter" genomes are allocated more "offspring". The population is changed over time by "recombination" which is a mixture of random mutation of bits by flipping certain locations, and by "crossover", where sections of strings are swapped.

This system uses the improvement offered by Baker[101].

It is known that hidden units are necessary to solve non-linearities, and it is suggested [56] that two hidden layers may have some advantage over one layer. It was decided therefore to allow two hidden layers in the networks to be designed, but arbitrary mapping were to be allowed, with the exception that connections within a layer are not allowed. This allows the MLP error back propagation technique to be used. As shown previously direct links from input to output may be beneficial, and these are permitted in this model. It was also demonstrated that unconnected hidden units will not necessarily impair the net performance, and a specific check on redundant connections while useful to reduce connectivity, is not required for optimising performance.

The error used as a criterion of fitness by the GA may vary according to the requirements of the user. Shaffer et al used the tsse of the network. Harp et at [87] used the integral area of the tsse curve. Here the tsse was used, but it was modified to reduce connectivity, as described below.

11.7. Genetic Experiments

The genes were decoded to provide a connection with a variable weight, or one fixed at zero, the latter being identical to no connection. For the large network the input layer was fixed by the pixel numbers at 56, and likewise the outputs at 25 units. The number allocated to the first hidden layer was 44, and the second 25. The small network had 12 inputs, 8 in the first hidden layer, 4 in the second and 4 in the output.

The GA could in principle reduce any hidden units to redundancy by setting all their connections off. Connections from inputs or to outputs can be also set to zero. Thus after a series of generations one would expect a near optimal number of units and connectivity to be learned. However as [87] noted GAs tend to allocate connections readily if some check is not put on them. It has been suggested that several parameters might be added to produce an overall fitness criterion, of which tsse is but one. Clearly one would wish to have a network with a low connectivity, both for speed of operation, and for a better ability to generalise.

Accordingly the tsse was weighted to reflect connectivity. One required a function that encourages low connectivity. One might scale the error by the ratio of total possible connections to actual connections. This might however give unacceptably large weightings. In the extreme a network with a few connections would have two to three orders of magnitude advantage over a fully connected one. This is comparable with the observed difference between an error generated by an untrained network and a converged one.

A solution is to use the logistic function as used in the MLP error back propagation. Thus the following formula for error was employed.

$$error = tsse \times \frac{1}{1+e^{\frac{-C_a}{C_t}}} \tag{11.1}$$

where C_t = total possible connections and C_a is the actual connections in the network. This was compared with using the more crude :-

$$error = tsse \times \frac{C_a}{C_t} \tag{11.2}$$

and also against using the raw tsse score :-

$$error = tsse \tag{11.3}$$

The larger network is more interesting in that it appears to exhibit non-linearity, but the time consuming nature of testing hundreds of large networks made it useful to test the smaller networks for response to error scaling. The smaller networks showed that the best function to use was surprisingly that described by equation 2.

Using this error scaling method two large network simulations were run. In one the error returned was the training error, which may have a tendency to lead the network weights into a local minimum. In the second a test set was used after the network had run through its epoch of training. The tsse from the test was reported back to the GA to be used in evaluation, which may be expected to give more optimal nets than the training error.

In all the experiments described below a population of 50 was used, with a mutation rate of 0.01, and a crossover rate of 0.6. One could have explored changing these parameters. From previous work though it is known that a mutation rate of the order of 0.01 or less, and a population size of 30-100 have been effective[87]. The number of trials required is not known a priori, and a large number was chosen to best test whether the GA could succeed in this problem. The GA was tested until 10,000 trials had been achieved for the smaller network, and 4,000 for the larger network (a lower figure simply as the computational resources required were larger for the larger network).

11.8. 4 Pixel From 12 network

11.8.1. tsse Used as Error

The GA produced a best individual of error 14.76 (using tsse as a test criterion) in the first generation, this fell over 68 generations (2123 trials) to 11.49, after which it remained static till generation 188 (4007 trials) where it fell to 11.14, which remained the best until generation 973 (9606 trials), falling slightly to 11.05. Thus very small changes were made for a lot of computation. The GA was ended at generation 1000 (10,000 trials). Of a total possible connectivity of 272, the GA initially started with approximately half the connections switched on, and ended with between 149 and 151 connections in the last generation. All the values of this generation were in the range 11.14 to 16.25.

11.8.2. Logistic of tsse Used as Error

Starting at a best error of 8.25, over 41 generations (1321 trials) this was reduced to 7.514, where it remained static until 10,000 generations were completed. The connectivity was between 125 and 126, and the error ranged from 7.514 to 11.84 in the final generation.

11.8.3. Ratio of Connectivity to Total Allowed Connectivity Times tsse Used as Error

Starting at 7.30 the best error fell to 4.44 over 128 generations (3305 trials), where it remained till 400 generations (5500 trials) where it sharply fell to 1.60, and it terminated at 1.55 after 10,000 trials. The connectivity was between 67 and 78, and the error ranged from 1.55 to 42.82.

11.9. 25 Pixel from 56 Network

Only equation 2 was used in these experiments. This had been shown in the previous experiments to be the most effective in reducing connectivity, and yet did stabilise at a reasonable number of connections.

11.9.1. Using Training Data Error

In 14 generations (735 trials) the error fell from 28.82 to 26.28, where it stayed until the experiment ended after 4,000 trials. Of a total possible connectivity of 8089, the last population had between 3969 and 4017 connections, with an error ranging from 26.2 to 38.9.

11.9.2. Using test data Error

The error fell from 31.42 to 25.97 in 32 generations (1612 trials) where it remained static till 4,000 trials were completed. The final error ranged from 25.97 to 42.07, with a connectivity of 3923 to 3977. Thus there is no apparent advantage in using test data over training data for error reporting.

Table 11.5 : 2 by 2 Pixel Net (10,000 Trials)

Error	Best	No. Connections	Average
Eq 1	7.514	125	8.054
Eq 2	1.55	73	13.12
Eq 3	11.05	150	12.71

Table 11.6 : 5 by 5 Pixel Net (4,000 Trials)

Error Used	Best	No. Connections	Average
Training tsse, Eq 2	26.28	4002	35.15
Test tsse, Eq 2	25.97	3969	36.84

The results are summarised in Tables 11.5 and 11.6. It appears that the more fully connected networks have little variance, as one might expect. Lower connectivities show wide variance in the final population, where one surmises that changing a few connections would have greater effect. This may explain the sudden jumps seen in the 2 by 2 network trained by the GA using equation 2. Note that the errors quoted in Table 11.5 are scaled by different amounts due to the three different equations used, and a direct comparison is not appropriate.

11.10. Comparison of Genetically Designed Nets and Fully Connected Nets

The best fully connected network was the one where all layers were connected to each other. This was compared for both networks, the 4 pixel and the 25 pixel output nets. Convergence was much more quickly reached with the GA designed nets. They both converge in a fewer number of epochs than the fully connected ones. Furthermore since the number of connections is far fewer (272 compared with 73 for the smaller net, and 8089 compared with 4002) the net would take less time to compute an epoch.

11.11. Conclusion

Using GAs to design ANNs can produce low connectivity networks which give similar or better tsse than fully connected networks. There is obvious advantage in having such reduced networks as they run faster, and are less prone to produce one to one mappings. The reduction in error produced by a GA with no check on connectivity is very moderate, and it tends to increase connections. The equation 1 error seems to keep the connectivity roughly at the starting value of 50% in the smaller network, and using the equation 2 error reduces it to about 25 - 30%. The larger network remains at 50% connectivity even with the equation 2 error.

There is little difference in the results obtained using the test set error against the training set error.

Regarding the lack of improvement after a given point in the GA simulations, it has been suggested[102] that mutation rates should be increased when GAs cease giving substantial improvement. This may be related to the biological effect of punctuated equilibria, where organisms evolve rapidly and then remain static for long periods, and after (typically) a change in the environment, evolve quickly again in a subsequent period. A technique employing this idea[103] has been evaluated, whereby several competing populations occasionally send individuals to other groups. This has shown improved performance, and thus the notion of punctuated equilibrium applied to GAs may not be fanciful.

It has been suggested [104] that crossover may be unnecessary or even counter-productive in GAs learning highly non-linear structures such as ANN networks. The effect of random connectivities is explored in a further chapter (Chapter 12).

12. REDUCING CONNECTIVITY IN COMPRESSION AND PREDICTION NETWORKS

12.1. Introduction

Discussed above and in [94] was the use of artificial neural networks (ANNs) compared with principal components analysis (PCA) for data compression. It was found that using a multilayer perceptron (MLP) with error back propagation, PCA gave a lower total sum square error (tsse) than ANNs for "unseen" data.

A review of this work [78], discusses a weakness in using MLPs, which is the well known problem of local minima. In this review it is stated that the increase in tsse shown in the ANNs over PCA is probably due to local minima. The reviewer suggests a linear auto-associator may be a better option as it does not suffer this effect, and refers to previous work using a linear network in reducing the dimensionality of speech spectrum data[105].

An alternative approach to avoiding local minima might be to change the connectivity of units in an MLP. As previously stressed the connectivity of a net affects the learning rate, and the number of patterns required to avoid one to one mappings[57]. Reducing connectivity speeds up the network convergence and reduces the number of training patterns required.

The degrees of freedom of an MLP network are not the same as the number of units. As [75] has pointed out, the degrees of freedom in nonlinear nets change as the network learns, since the activations tend to move away from mid-range, and hence from the linear part of the activation function (assuming the logistic function is used). This can have the effect of overfitting of data, and is referred to as overlearning. Stopping training at convergence would avoid this effect, but one does not necessarily know in advance when convergence will occur. Backtracking from an overtrained net is not practicable as at each pattern presentation, the weights would need to be saved, and one would very quickly run out of disk space on all but the most trivial problem. It has been shown [75] that lowering the connectivity reduces the overfitting problem, and allows the network to continue training beyond convergence with little ill-effect.

As[106] has pointed out, in real nervous systems, the space occupied by neurons is negligible compared with that occupied by the connections. As the number of neurons increase, the number of connections increases exponentially if full connectivity is assumed, and in the brain partial connectivity is observed. Apart from being necessary in large networks, there are advantages in partial connectivity. Using a Hopfield net, it was found that a partially connected network stores more information per connection than a fully connected one[106]. As the network size increases, the partially connected network becomes increasingly efficient in its use of space, communication times and storage capacity[106].

The networks in [94] was allowed to proceed beyond convergence, and suffered from the disadvantage of being fully connected. The reason a fully connected network topology was used is that it is not known a priori the connectivity that is optimal.

Reducing the structural size of a network has been attempted by several methods. One may use relevance of units[107], calculating the importance of individual units on the outcome, and deleting those with low relevance. As pointed out in that study, one can not use the absolute output value of a unit, for a high value output may be fed into a unit which is saturated, and consequently makes little impact. The value $\rho_i = E_{\textit{without the unit}} - E_{\textit{with the unit}}$ where E is the error and ρ the measure of relevance was used. PCA has been used[108] on weight vectors to determine those carrying most information in the Shannon sense, and to delete connections which are low in information passing.

12.2. Genetic Algorithms

Genetic algorithms (GAs) have been used by a variety of workers, for example[87] to optimise network design, and above in Chapter 11. Good results have been quoted on small "toy" problems, e.g. XOR, and a minimal encoding problem. Good results have also been quoted on larger problems[99].

12.3. Compression Networks

By this term is meant the auto-associative MLPs used in Chapter 7 and in[40].

A GA was employed to look at reducing the connectivity of compression networks, by biasing the error towards lower connectivity. This was done simply by multiplying the tsse after one epoch by the ratio of connections to total possible connections, a method used in[88].

Training and test sets of 3 views of 3 images, each set consisting of 576 patterns were used. A GA was used to optimise the network structure, where only connections between adjacent layers were permitted. It was found that the decrease in error was not very substantial, after one generation the best structure gave an error of 539.8, but after 47 generations (2001 trials) the error was 453.7. Raising the mutation rate from 0.001 to 0.01, did not improve the best error. The final connectivities were close to 40% for mutation of 0.001, and 48% for mutation rate = 0.01.

Compression nets with randomly allocated connections were created and tested. For each potential connection a pseudo-random number generator was used to determine whether the connection was present or not. If N% of the connections were to be created, then if the random number returned was R, then if (R mod 100) < N the connection is made.

50 such nets were created. Results are shown in Tables 12.1 and reported in [109], where the best and worst net (as measured by the error criteria) after one epoch were further trained to 100 epochs. The fully connected network figures are shown in Table 12.2. Results are given for test and training data sets. GA designed nets are shown in Table 12.3.

Table 12.1 Compression Network tsse after 100 epochs with nets of best and worst initial tsse (8 by 8 Segments)

Connectivity	Worst Initial tsse	% Error/Pixel	Best Initial tsse	% Error/Pixel
Training				
1%	1248.91	3.39	1211.09	3.29
5%	1190.00	3.23	729.29	1.98
10%	1000.76	2.71	353.21	0.96
20%	232.50	0.63	158.58	0.43
30%	440.01	1.19	101.90	0.28
50%	269.50	0.73	67.06	0.18
Test				
1%	1768.50	4.80	1712.14	4.64
5%	1677.36	4.55	1016.40	2.76
10%	1390.93	3.77	496.93	1.35
20%	822.59	2.23	614.26	1.67
30%	592.75	1.61	156.26	0.42
50%	352.93	0.96	113.52	0.31

Table 12.2 Compression Networks Fully Connected (8 by 8 Segments)

Data	tsse with Trained Network	% Error/Pixel
Training	52.29	0.14
Test	81.45	0.22

Table 12.3 GA Designed Compression Networks (8 by 8 Segments)

mutation rate	Data	tsse with Trained Network	% Error/Pixel
0.01	Training	68.98	0.19
0.001	Training	75.18	0.20
0.01	Test	110.29	0.30
0.001	Test	122.32	0.33

nb. Lower errors were here obtained than found in Chapter 7, for similar networks, however the training sets were here larger.

The GA designed nets were disappointing, the 48% net (mutation rate = 0.001) being worse than a random 50% net, and the 40% net (mutation rate 0.01) about the same as a 50% random net.

The compression nets showed a substantial difference between best and worst network, typically by a factor of 3 to 4. The actual connectivities are shown in Table 12.4, and it is clear that the number of connections is not the only factor, as in some cases the worst structure has more connections than the best, and in others it has less. However from Table 12.1, there is an obvious increase in error with decreasing connectivity.

Table 12.4 Actual Connectivity of Compression Networks

Random % Connections	Actual Connections Worst	Connections Best
1%	12	16
5%	102	107
10%	183	217
20%	390	374
30%	592	621
50%	982	951

12.4. Prediction Networks

Similar experiments with prediction networks (where a rectangular group of pixels had an inner segment removed, and the inner segment predicted from the outer pixels, ie. the inner segment was the target, other pixels in the rectangle were inputs) have been performed. It was found that there was little difference between the best and worst randomly generated network for a particular connectivity value. Thus a GA would not be able to optimise the performance other than by changing the connectivity.

Using data from nuclear medical images a GA was employed to look at reducing the connectivity of prediction networks, by biasing the error towards lower connectivity. This was done simply by multiplying the tsse after one epoch by the ratio of connections to total possible connections, a method used above in Chapter 11, and in[88].

Two such networks have been built, using a 4 by 4 pixel segment predicting an inner 2 by 2 segment, (ie. 12 surrounding pixels predicting the 4 central ones) and an 8 by 8 segment with a 4 by 4 inner segment (48 pixels predicting inner 16). Both nets used the same image data as a training set (3 views of a lung scintigram standardised to 64 by 32 pixels), the smaller network using 768 patterns and the larger 192 patterns. A test set of equal size was also created with data the net had not been trained on.

An MLP with 12 inputs, a hidden layer of 8 units, a second hidden layer of 4 units, and 4 outputs, was randomly given connections from input to hidden and output, and hidden to output layers. Connections were allowed between all layers as this has been found to give faster convergence and lower tsse[88]. Initially 50% connectivity was allocated. From an initial population of 50 such networks, a GA then generated a new generation, selecting the better network designs. Crossover was 0.6 in all experiments below, mutation rates were either 0.001 or 0.01.

Although the GA produced networks that were similar with respect to performance to fully connected networks, the successive improvement over generations was not substantial. At a mutation rate of 0.001 the best error after one generation was 25.6, and dropped to 23.7 after 44 generations (2010 trials). The connectivity had been reduced from 50% to 30%, and the reduced error can be explained by the lower connectivity alone. The mutation rate was raised from 0.001 to 0.01 to provide greater variability of the populations, and this improved the best error to 17.5, however it reached this stage at 12 generations, and remained there till 145th generation (2001 trials).

To explore whether GAs were any better than randomly generated nets, an initial randomly generated population of 50 small prediction nets was examined. The network structure with the largest error and that with the smallest error after one epoch, were trained to convergence and compared. Fully connected nets were at convergence within 3-5 epochs. 100 epochs was chosen to train the nets, to ensure that if the nets failed to converge, they did so orders of magnitude after fully connected nets were successful. Various connectivities were tried, and the results are shown in Table 12.5, results for a fully connected network are shown in Table 12.6. These are compared with the performance of nets designed by GAs in Table 12.7

It can be seen that the best GA network is better than a similarly connected (30%) best of 50 random network. With respect to test tsse the 20% connected network is the best of all the sparsely connected nets, and is better than the GA designed ones.

Table 12.5 Small Prediction Network tsse after 100 epochs with nets of best and worst initial tsse

WorstInitialtsse BestInitailtsse				
Connectivity	tsse	% Error/Pixel	tsse	% Error/Pixel
Training				
10%	41.52	0.0135	8.88	0.29
20%	13.77	0.45	3.01	0.10
30%	6.48	0.21	6.51	0.21
50%	5.30	0.17	4.28	0.14
Test				
10%	55.19	0.0180	11.36	0.37
20%	17.81	0.50	4.09	0.13
30%	9.13	0.30	9.26	0.30
50%	7.43	0.24	5.96	0.19

Table 12.6 Small Prediction Networks Fully Connected (4 by 4 to 2 by 2 Segments)

Data	tsse with Trained Network	% Error/Pixel
Training	1.67	0.05
Test	2.78	0.09

Table 12.7 Small Prediction Networks (4 by 4 to 2 by 2 Segments)

mutation rate	Data	tsse with Trained Network	% Error/Pixel
0.01	Training	4.29	0.14
0.001	Training	4.32	0.14
0.01	Test	6.09	0.20
0.001	Test	5.93	0.19

nb. The error results are better than found in similar nets in Chapter 11, but again the training sets were larger.

It seems that up to a critical point of about 20% connectivity there is little difference in the tsse for randomly generated networks, ie. the best and worst networks after one epoch are not very different from each other after 100 epochs. Below this point however large differences are noted. The larger network (192 patterns) showed a similar pattern, but a lower connectivity was possible, see Tables 12.8 and 12.9, and the nets designed by GAs in Table 12.10. The optimal sparse network appeared to be at 30%, and only below 5% was it really poor. Unlike the smaller net no critical point seemed to exist, where best and worst were very different.

Table 12.8 Large Prediction Network tsse after 100 epochs with nets of best and worst initial tsse

WorstInitialtsseBestInitailtsse				
Connectivity	tsse	% Error/Pixel	tsse	% Error/Pixel
Training				
1%	71.04	2.31	97.26	3.17
5%	9.59	0.31	8.75	0.28
10%	11.86	0.39	6.44	0.21
20%	7.25	0.24	5.62	0.18
30%	6.94	0.23	5.45	0.18
50%	6.31	0.21	5.56	0.18
Test				
1%	92.23	3.00	124.63	4.06
5%	17.76	0.58	14.14	0.46
10%	17.82	0.58	10.61	0.35
20%	12.16	0.40	8.75	0.28
30%	10.96	0.36	6.94	0.23
50%	9.95	0.32	7.96	0.26

Table 12.9 Large Prediction Networks Fully Connected (8 by 8 to 4 by 4 Segments)

Data	tsse with Trained Network	% Error/Pixel
Training	1.98	0.06
Test	4.11	0.13

Table 12.10 Large Prediction Networks (8 by 8 to 4 by 4 Segments)

mutation rate	Data	tsse with Trained Network	% Error/Pixel
0.01	Training	3.12	0.10
0.001	Training	5.85	0.19
0.01	Test	5.27	0.17
0.001	Test	8.87	0.29

12.5. Discussion

In small networks, such as that solving the XOR problem, the specific connections are important, but as the network size increases this may not be the case. An analogy may be drawn with statistical dynamics, where the overall behaviour of large scale systems is not dependent upon the states of individual entities, but exhibit stability due to the large number of such entities. It is suggested that in the compression network of the size described here (64 inputs, 16 hidden units, 64 outputs) that the specific connectivity is less important than the number of connections.

It is instructive to compare the results given here with the paper by Dodd[99] where in a large problem (5,000 connections) GAs were significantly better than random search strategies for network connections design. It may be that some large networks are more sensitive to precise topology than suggested by the above simulations. It is not clear from the Dodd paper quite how the genotypes (bit strings) were mapped to the phenotypes (network structure). Further it was stated in[99] that fitness was proportional to error and network complexity, but did not give details of how this was achieved. Thus it is possible the simulations were not comparable with those undertaken in this study.

If these moderately sized networks are dependent upon the connectivity per se, rather than the precise topology, it will be more constructive to determine the level of connectivity that is compatible with the error requirement, and choose the best of a randomly generated population.

Lower connectivity networks are faster than fully connected nets, and would be expected to give fewer local minima problems. For the prediction nets, and especially for the larger of the nets, quite substantial reduction in connectivity can be obtained with minimal increase in tsse for a test set.

The difference between the behaviour of the networks is partly determined by size, the larger prediction network allowing much lower connectivities than the smaller. However the nature of the problem or the type of structure may be as important. The compression network is between the two prediction networks in size (maximum number of connections in larger prediction net 5928, small prediction net 272 and compression net 2048), but the prediction net has two hidden layers in comparison to the one layer of the compression network, and connections are allowed between all layers, as opposed to adjacent layers only.

The reason GAs were of no great utility in prediction nets was presumably because there is little difference in the random nets, and so all a GA can do is reduce the connectivity. This may help to explain why [98] found crossover of no utility when designing nets.

12.6. Conclusion

There is a limited capability for reduction of connections in compression networks. If a higher error can be tolerated this may be a useful approach, otherwise linear nets may as[78] suggests be more appropriate.

In prediction networks of the type described above randomly connected network structures may be used to speed up convergence and reduce the possibilities of one to one mappings. In smaller networks a critical point may occur below which the precise connections become important. Compression networks seem more dependent upon the actual connectivity, though this may be because of the different structure type rather than the nature of the problem being solved. For compression networks there is a more limited capability for reduction of connections.

13. CLASSIFICATION OF DEFECTS USING LOCAL TECHNIQUES

13.1. Multi-Network Systems

Rather than build one network to solve a whole problem, many workers are building systems composed of several subnets, each trained separately to do a particular task. This speeds up training, as each net in isolation may be quite small. Cottrell[60] has built ANNs for face recognition, where a first hidden layer extracts features for a second output layer to classify the faces into different groups. Initially a network with input, output and one hidden layer was trained to learn the identity mapping. The output layer was then replaced with a new set of output units for classification, the weights from input to hidden clamped, and the system trained to learn classification.

Similarly one could train one network to recognise abnormal areas, and separately to train another network to classify abnormal areas. The two networks could then be linked such that the first network filtered input images, and only passed on abnormal areas to the second network. Specialised network designs have been described for classification purpose e.g[110]. but the MLP approach was used in this study as an MLP simulator was immediately available, and it is easy to connect MLPs together.

A network designed to perform the first task, that of picking out the potential segments, was tested in Chapter 10. However the prediction network used there acted as a smoothing filter, and the error returned gave an edge detector, and did not give appropriate segments for classification.

One might try other techniques for extracting the relevant areas, but given the time scale of the project it was considered advantageous to press on with the second problem, that of classification. Accordingly areas have had to be manually delineated for further classification.

Such an approach was tested to determine whether it would give satisfactory results. If this were so then networks could be trained on other specific abnormalities. One advantage would be that a smaller set of images would be needed, as many images would contain several defects.

A classifier network would be trained on local segments. This would help avoid the problem of majority bias, where a network is trained on a large proportion of normal cases, and only a few abnormal ones. In that case the network can give a low tsse while essentially ignoring the abnormal cases, since they effect the tsse to a limited extent. In the following experiments the classifier was given training and test sets with 50% PE and 50% non PE segments.

The local nature of the networks would allow the abnormality to be located spatially. In addition the size of the abnormality may be seen. This would be most useful for showing serial changes in patients, to monitor improvement or deterioration following clinical intervention. This is particularly pertinent to anti-coagulant therapy following PE. The networks should be capable of showing more than one disease or artifact on an image due again to the fact that local areas are used, and not the whole image. Colour coding for different abnormalities could be employed to show the different areas in cases of complicated pathology.

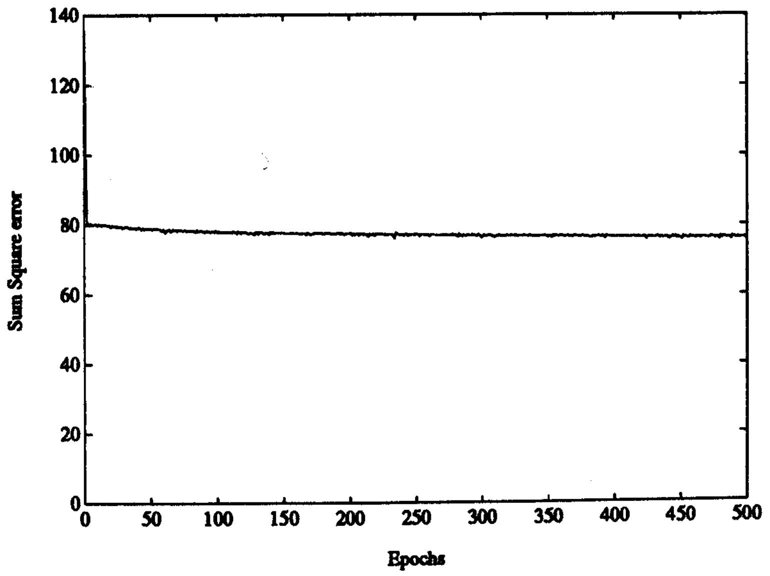
In clinical practice perfusion and ventilation images are used, and by looking at both images, a mismatch where a perfusion defect is not seen on the ventilation image indicates PE. However limiting the network to perfusion images avoids problems of registration. Networks were trained on both perfusion, and perfusion plus ventilation images.

13.2. Experimental Methodology

A medical physicist picked out areas of pulmonary scintigrams which contained PE. A transparency with a grid splitting the 64 by 32 image into 32 8 by 8 segments was overlaid printed images of the scintigrams. Examining the original analogue images the physicist determined which segments contained PE. These segments were used to produce a training and test set of PE images. COAD and normal images which were previously paired with the PE images had the same segments extracted and placed in the training and test sets. The desired outputs were set to 1.0 or 0.0. These were two in number, one for PE and the other for COAD. Experiments were executed on perfusion images, and with data from paired perfusion and ventilation images combined.

A network was trained with 64 inputs and 2 outputs with a hidden layer of 16 units, adjacent layers being fully connected. This network was only given perfusion images, as ventilation images cannot discern PE. The results of an attempt to converge the network is shown in fig 13.1. It clearly does not converge to a satisfactory error, since the converged error is only slightly below the initial error, which was that obtained with randomised weights. Various learning rates were tried with little difference in the result, Figures 13.1 was obtained using a η of 0.005. A file was next set up which contained paired perfusion and ventilation data, this had 128 inputs, a hidden layer of 32 and 2 outputs. The error curve obtained was very similar as in the perfusion case, and little improvement in tsse was seen.

Figure 13.1 Convergence of Perfusion Classification Net



Both experiments were repeated with a network fully connected between all layers. No improvement was noted with respect to tsse.

The input data set was reduced to 4 from an original 64 dimensions for each 8 by 8 segment, using PCA. A net was constructed with 4 inputs, 2 outputs and a hidden layer of 4 units. Ordinary error back propagation using a learning rate of 0.1 and 0.01 were tried, both produced poor results.

The learning rate η was reduced to 0.00002, and a similar convergence was achieved as in the previous simulations, with no improvement in tsse. In an attempt to speed convergence dynamic parameter tuning as described in Chapter 9 was employed. Epoch learning with parameter tuning produced no improvement in tsse.

A second network with combined ventilation and perfusion data was trained. The network had 8 inputs, 8 hidden units and 2 outputs. Both pattern and epoch learning were attempted with similar results. Pattern and epoch learning with parameter tuning both failed to improve the tsse.

13.3. Accuracy of ANN Classification

An ROC curve is constructed by plotting the true positive against false positive values. The percentage of true positives above a given conficence level is plotted against the percentage of false positives above the same level. Typically about 5 levels are used which may be for example : very confident, confident, not sure, confidently not, and very confidently not. A set of points will be plotted which will, if the classification is random, form a straight line going from left lower corner to top right corner, ie. at each level the ratio of true positive to false positive is constant. If however the classification is near perfect, most of the true positive values occur at very high confidence levels, and most of the false positives occur at low confidence levels, and so the ratio changes with confidence, and a curve is traced which lies to the left of the diagonal formed by random guessing. The further away from this diagonal, the better the classification. For an introduction to ROC analysis see[111]).

Since the uncompressed data network did not even converge to a solution at all, there was obviously no point in assessing the accuracy of the results given from it. The compressed data network, using 4 PCs of segments showed a high error. It may be that 4

PCs are simply too few, and therefore a higher number was tried. Using 20 PCs of a perfusion/ventilation input data set did reduce its tsse substantially, and the results from these networks were explored. Using the fully connected network, with 20 inputs, of which PCs of an original 128 raw data inputs (64 perfusion and 64 ventilation, from the 8 by 8 segments) and a hidden layer of 10 units, and 2 outputs, test images were evaluated.

The outputs for each image segment were placed in one of 5 groups, viz the intervals [0.8,1.0], [0.6,0.8], [0.4,0.6], [0.2,0.4], and < 0.2. The numbers of images in each group can be considered to belong to confidence levels, where the higher the value, the more confident one is about the output. True positive/false positive and true negative/false negative ratios can be built from these values, which may be used to construct an ROC curve.

The values from the training set are shown in Table 13.1 and 13.2. In the below tables the figures are given separately for PE, COAD and normal images. Ie. the 3 columns represent respectively the numbers at each output level of the subsegments with PE, COAD and those that are normal.

Table 13.1 Numbers of Segments at Various Confidence Levels for PE Output

Output range	PE		
> 0.8	12	0	3
[0.6,0.8[	7	0	0
[0.4,0.6[	1	0	2
[0.2,0.4[	0	0	2
< 0.2	2	14	11

Table 13.2 Numbers of Segments at Various Confidence Levels for COAD Output

Output range	PE		
> 0.8	0	8	0
[0.6,0.8[	0	7	0
[0.4,0.6[	0	0	0
[0.2,0.4[	4	0	2
< 0.2	18	1	16

These value look encouraging as the number of true positives /false positives and true negatives / false negatives are high. E.g. at above the 0.8 level for PE output the true positives /false positives ratio is 12:3, and below 0.2 the true negatives / false negatives ratio is 25:2. However this was the training set on which convergence may be achieved without true learning. A test set gave very different results, see Tables 13.3 and 13.4.

Table 13.3 Numbers of Segments at Various Confidence Levels for PE Output

Output range	PE		
> 0.8	6	3	1
[0.6,0.8[	0	2	1
[0.4,0.6[	0	0	3
[0.2,0.4[	2	2	0
< 0.2	14	5	21

Table 13.4 Numbers of Segments at Various Confidence Levels for COAD Output

Output range	PE		
> 0.8	4	0	5
[0.6,0.8[	1	0	1
[0.4,0.6[	1	1	2
[0.2,0.4[	1	0	2
< 0.2	15	11	16

The results contrast strongly with the training set. The PE images are generally given a low value (< 0.2). The number of true positives /false positives above 0.8 is 6:4. The true negatives / false negatives below 0.2 is 26:14. The results for COAD are worse, 0:9 and 31:11 respectively. ROC curves for PE and COAD have been computed for the training and test data, and these are shown in Figure 13.2- 13.5.

Figure 13.2 ROC Curve Training Data PE Classification

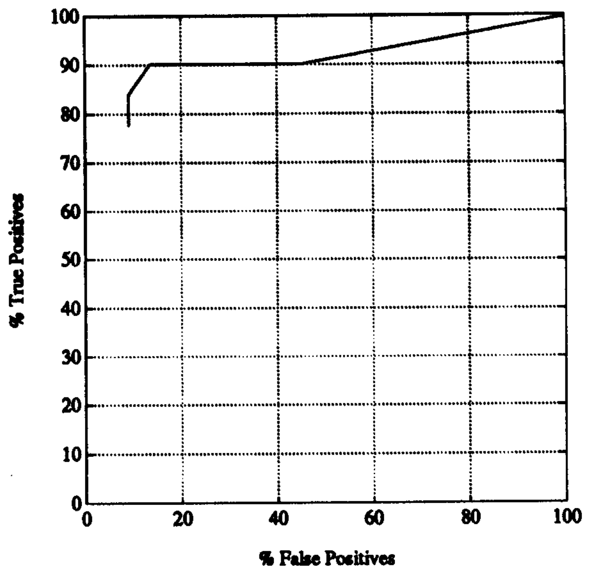


Figure 13.3 ROC Curve Training Data COAD Classification

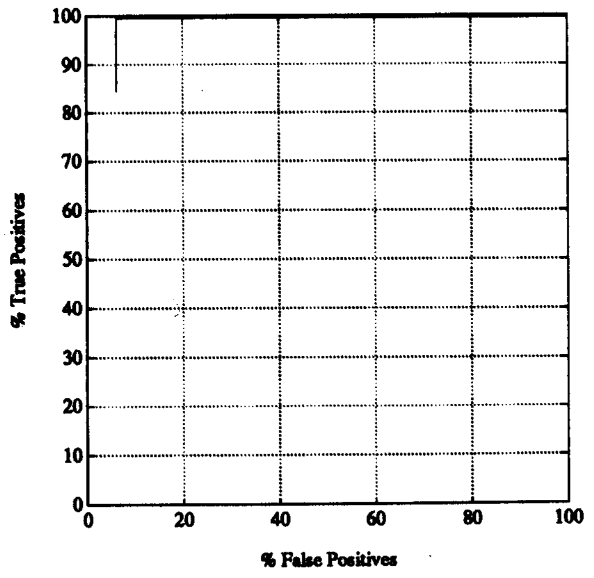


Figure 13.4 ROC Curve Test Data PE Classification

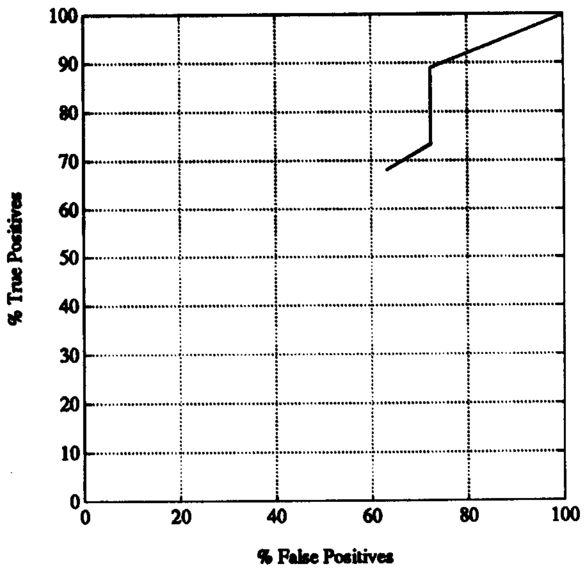
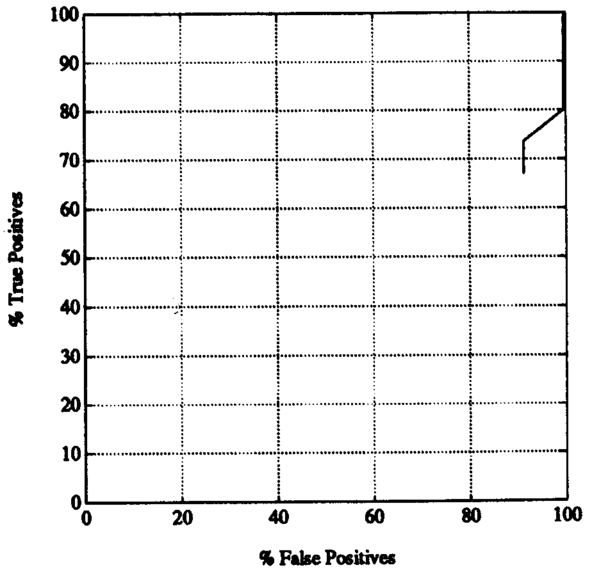


Figure 13.5 ROC Curve Test Data COAD Classification



13.4. Global Learning Revisited

Clearly the local methods using raw data are not suitable as the error scarcely reduces below its initial random level, and fails to converge using raw data. Networks using PCs do converge, but to a local minimum as the test set gives much poorer results than the training set.

It may be the case that global information is required, and that local techniques can not produce a solution. Initial results for raw data reported above were quite hopeless, but this used no data compression. Results from PCs of 8 by 8 segments were more encouraging, but still most images were mis-classified. If one assumes that the improvement in using PCs was due to the reduction in data, and that 8 by 8 segments do not allow classification as they are too local in nature, then it follows that PCs of whole images may be of utility.

The set of training images was used to produce PCs. The computational load of obtaining PCs from 2048 dimensional data is too great for the computing resources, however 32 by 32 segments (ie. half of an image, corresponding roughly to a single lung field) were able to have their PCs computed. It was found that over 95% of the variance was in the first 8 PCs. 10 PCs were used to create a training set, and nets of 10 inputs, 8 hidden units and 2 outputs were constructed. The network was fully connected between all layers. A low learning rate of 0.01 was given. The network performed poorly for perfusion data alone, that is the networks did not converge.

14. CONCLUSION

In this study, a novel approach to classification of pulmonary scintigrams is presented. Such an approach has not previously been used in pulmonary nuclear images.

14.1. Use of Raw Data

The data, compressed by means of a median filter to standardise for size, did not allow correct classification of lung scintigrams into PE, COAD and normal images. It did not even allow detection of the view (PA, LPO, RPO). It did however successfully rule out artifact regions output from a segmentation procedure. A simple raw data network could be used therefore to determine which segments are suitable for further analysis. In lung scintigrams this is not very useful, as the lung fields are going to be in roughly the same screen area from image to image.

14.2. Data Compression

ANNs can be used to significantly compress raw data. In a comparison with PCA though, PCs were found to give better reconstruction of data than ANN networks on identical unseen data. No previous experimental study is known comparing PCA with ANNs.

14.3. ANNs with Principal Components

PE and COAD could be classified using PCs of image segments. The results were still poor, but a true positive to false positive of 4-5:1 showed the networks were capable of a limited classification capability. The main problem was that in many images the network did not give any meaningful output. The use of PCs as inputs to an ANN is not known to have been previously published.

14.4. Parameter Tuning

To speed up the network convergence, parameter tuning of the learning rate and momentum was tested. The results on scintigram data were not promising. In particular epoch learning (updating the network weights after presentation of the whole epoch) did not seem to be an advance over pattern learning (updating after each pattern presentation). These results are at variance with previously published papers using "toy problem" data. Parameter tuning did help in impedance image data, where the learning rate stabilesed to a value that was at least as good as any static learning rate.

14.5. Local Prediction of Subsegments

A prediction network, which was expected to be useful in picking out segments which were abnormal and suitable for further classification was not found to be useful. The network acted as an edge detector and smoothing filter, and did not output areas of known abnormalities. No previous published work is known demonstrating nets acting as smoothing filters or edge detectors.

14.6. Genetic Algorithms for Optimising Networks

Initially encouraging results for reducing connectivity of networks was found using a GA approach. Reduced connectivity networks gave similar errors as fully connected, and ran much faster. They would be less susceptible to local minima and acting as look-up tables. These results initially suggested a confirmation of previous work using GAs to design networks in other unrelated areas.

14.7. Reducing Connectivity Randomly

Prediction and Compression networks with connectivities reduced randomly were found to be as good as those generated by GA techniques. It is likely that in these moderately large networks the precise topology is less important than the amount of connectivity. Prediction networks were capable of much reduced connectivity, with an optimal range of connections which was less than full connectivity. Compression networks deteriorated with respect to tsse as the connectivity was reduced. Previous published work comparing GA designed and randomly generated nets showed improved performance using GAs. The size of the nets used in this study are moderately large, and this may have significant bearing on the reported differences.

14.8. Local Classification Techniques

Using areas of scintigrams picked out as containing PE, and an identical number of non PE segments in other images, classification was attempted using an ANN. This was found to be completely unsuccessful in PE or COAD.

14.9. Discussion

The classification of lung scintigrams by automatic methods seems to be a non-trivial problem. Simple ANN techniques using raw data are hopeless, some data compression seems to be essential. The results of some feature detector, in this study PCs, shows promise in a classification network. The problems of network design may be addressed by randomly cutting connections until performance deteriorates to an unacceptable level.

The noise added to the data due to the acquisition technique may have caused the networks to be unable to classify the scintigrams, whereas "clean" data may have worked. A digital link now exists from the gamma camera to a PC, and this recent facility should be used in any future projects of this type. However the images were capable of being diagnosed by a human observer even with this excess noise. Thus data reduced in noise fed into a network as explored above is unlikely to result in a system which classifies correctly. Unfortunately the lung scintigram images at the Walsgrave Hospital are not available via the digital link, as they reside on a separate system, and the experiments detailed in this study can not be repeated with "clean" data at the present.

As a result of this study one may be guardedly optimistic that ANNs using compressed input data from scintigrams are capable of being classified. The classification is not as good as human observers at present, and techniques will need to be refined and developed.

References

1. Anthony D.M, Hines E.L, Taylor D, and Barham J, "An Investigation into the Use of Colour and Image Processing Techniques in Nuclear Medicine," *Medical & Biological Engineering & Computing*, vol. 28, pp. 489-492, 1990.
2. Sagerer G, "Automatic Interpretation of Medical Image Sequences," *Pattern Recognition Letters*, vol. 8 NO 2, pp. 87-102, 1988.
3. Hallam N.J, Hopgood A.A, and Woodcock N, "Defect Classification in Welds Using a Feedforward Network Within a Blackboard System," *INNS Int. Neural Networks Conference*, vol. 1, pp. 353-356, Paris, 1990.
4. Schneider P.B, Treves S,, *Nuclear Medicine in Clinical Practice,* Elsevier/North Holland Biomedical Press, 1978.
5. Todd-Pokropek A.E,, "The Comparison of a Black and White and a Color Display : An Example of the Use of Receiver Operating Characteristic Curves," *IEEE Transactions*, vol. MI-2, pp. 19-23, 1983.
6. He L.J, "Computer Aided Diagnoses of Lung Diseases Through Radiographs," *SPIE Close-Range Photogrammetry Meets Machine Vision*, vol. 1395, pp. 1220-1226, 1990.
7. Burton G.H, Vernon P, and Seed W.A, "An Automated Quantitative Analysis of Ventilation-Perfusion Lung Scintigrams," *Journal of Nuclear Medicine*, vol. 25, pp. 564-570, 1984.
8. Burton G.H, Seed W.A, and Vernon P, "Computer Analysis of Ventilation-Perfusion Scans for Detection and Assessment of Lung Disease," *Thorax*, vol. 40, pp. 519-525, 1985.
9. Niemann H, Bunke H, Hofmann I, Sagerer G, Wolf F, and Feistel H, "A Knowledge Based System for Analysis of Gated Blood Pool Studies," *IEEE Transactions*, vol. PAMI-7:3, pp. 246-259, 1985.
10. Barber D.C, and Brown B.H, "Applied Potential Tomography," *Physical Electronics:Scientific Intrumentation*, vol. 17, pp. 723-732, 1984.
11. Neaves P, *Design and Development of a Complex Impedance Measurement System for Impedance Tomograhpy (MSc Thesis, Warwick University)*, 1991.
12. Fogelman I, and Maisey M, *An Atlas of Clinical Nuclear Medicine,* Martin Dunitz, London, 1988.
13. Holman J.G, and Cookson M.J, "Expert Systems for Medical Applications," *J of Medical Engineering & Technology*, vol. 11 NO 4, pp. 151-159, 1987.
14. Josin G, "Neural-Network Heuristics," *Byte*, vol. OCT, pp. 183-192, 1987.
15. Kohonen T, "An Introduction to Neural Computing," *Neural Networks*, vol. 1 NO 1, pp. 3-16, 1988.
16. Rumelhart D.E, Hinton G.E, and Williams N, *Learning Internal Representations by Error Propagation (in Parallel Distributed Processing ed Rumelhart and McLelland)*, 1, pp. 318-362, 1986.
17. Carnevali P, and Patarnello S, *Neural Networks "Living" in an External Environment (in Parallel Architectures and Neural Networks, ed. Cianiello E.R, World Scientific)*, pp. 77-88, 1988.
18. Touretzky D.S, "Representing Conceptual Structures in a Neural Network," *IEEE Proc. Int. Conf. Neural Network* , vol. II, pp. 279-286, San Diego, 1987.
19. Parisi D, and Nolfi S, *Some Mental Abilities that Can be Learned by Neural Networks (in Parallel Architectures and Neural Networks, ed. Cianiello E.R, World Scientific)*, pp. 177-187, 1988.

20. Voevodsky J, "Plato/Aristotle: A Neural Net Knowledge Processor," *IEEE Proc. Int. Conf. Neural Network* , vol. II, pp. 399-407, San Diego, 1987.

21. Buttner W, "Neural Network Research at Siemens," *2nd European Seminar on Neural Computing*, London, 1989.

22. Hebb D.O, *The Organization of Behavior*, Wiley, New York, 1949.

23. Rumelhart D.E, Hinton G.E, and Williams N, *A General Framework for Parallel Distributed Processing (in Parallel Distributed Processing ed Rumelhart and McLelland)*, 1, pp. 45-76, 1986.

24. McLelland J.G, Rumelhart D.E, Hinton G.E, *The Appeal of Parallel Distributed Processing (in Parallel Distributed Processing ed. Rumelhart and McLelland)*, 1, pp. 3-44, 1986.

25. Chan L, *Adaptive and Invariant Connectionist Models for Pattern Recognition (PhD Thesis, cambridge University Engineering Department, 1989)*, 1989.

26. Garey M.R, and Johnson D.S, *Computers and Intractability (W.H. Freeman)*, 1979.

27. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt Rinehart and Winston, 1976.

28. Aarts E, and Korst J, *Simulated Annealing and Boltzmann Machines*, John Wiley, 1989.

29. Kirkpatrick S, Gelatt C.D, and Vecchi M.P, "Optimisation by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

30. Parberry I, and Schnitger G, "Relating Boltzmann Machines to Conventional Models of Computation," *Neural Networks*, vol. 2, pp. 59-67, 1989.

31. Kolonay M.A, and Klimasauskas C.C, *An Introduction to Neural Computing*, NeuralWare Professional, 1987.

32. Minsky M.L, and Papert S.A, *Perceptrons (Expanded edition)*, MIT Press, 1988.

33. Pao Y, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, 1989.

34. Kohonen T, "Self Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

35. Ultsch A, Sieman H.P, "Kohonen's Self Organising Feauture Maps for Exploratory Data Analysis," *INNS Int. Neural Networks Conference*, vol. 1, pp. 305-308, Paris, 1990.

36. Aleksander I, "Exploding the Engineering Bottleneck in Neural Computing," *2nd European Seminar on Neural Computing*, pp. 1-7, London, 1989.

37. Stanley J, *Introduction to Neural Networks*, California Scientific Software, 1989.

38. Guiver J, *Networks 2 (In NeuralWorks Manual)*, pp. 435-499, 1988.

39. Widrow B, and Hoff M.E, "Adaptive Switching Circuits," *Inst. Radio Eng.,W. Electronic Show & Conv., Conv. Record*, vol. 4, pp. 96-104, 1960.

40. Cottrell G.W, Munro P, and Zipser D, "Learning Internal Representations from Gray-Scale Images: An Example of Extensional Programming," *Cognitive Science Soc. Annual Conference*, vol. 9TH PROCS., pp. 461-473, Seattle, 1987.

41. Glover D.E, "An Optical/Electronic Neurocomputer Automated Inspection System," *IEEE Proc. Int. Conf. Neural Network* , vol. I, pp. 569-576, San Diego, 1988.

42. Farmer J.D, and Sidorowitch J.J, *Exploiting Chaos to Predict the Future and Reduce Noise (Technical Report, Theoretical Division and Center for Nonlinear Studies, Los Alamos National Laboratory)*, pp. 1-54, 1988.

43. Bounds D.G, and Lloyd P.J, "A Multi Layer Perceptron Network for the Diagnosis of Low Back Pain," *IEEE Proc. Int. Conf. Neural Network* , pp. II-481-489, San Diego, 1988.

44. Scalia F, Marconi L, Ridella S, Arrigo P, Mansi C, and Mela G.S, "An Example of Back Propagation: Diagnosis of Dyspepsia," *IEE Int. Conference on Neural Networks*, pp. 332-336, London, 1989.

45. Saito K, and Nakano R, "Medical Diagnostic Expert System Based on PDP Model," *IEEE Proc. Int. Conf. Neural Network* , vol. I, pp. 255-262, San Diego, 1988.

46. Bradshaw G, Fozzard R and Ceci L, *A Connectionist Expert System that Actually Works (In Advances in Neural Information Processing Systems 1 ed Touretzku D.S)*, pp. 248-255, Moran Kaufman, 1989.

47. Saito K, and Nakano R, "Rule Extraction from Facts and Neural Networks," *INNS Int. Neural Networks Conference*, vol. 1, pp. 379-382, Paris, 1990.

48. Gallinari P, Thiria S and Fogelman F, "Multilayer Perceptrons and Data Analysis," *IEEE Proc. Int. Conf. Neural Network* , vol. I, pp. 391-399, San Diego, 1988.

49. Boone J.M, Sigillito V.G, Shaber G.S, "Neural Networks in Radiology: An Introduction and Evaluation in a Signal Detection Task," *Medical Physics*, vol. 17:2, pp. 234-241, 1990.

50. McLelland J.L, and Rumelhart D.E, *Explorations in Parallel Distributed Processing*, MIT Press, 1988.

51. Sochor H, Dorffner G, Porenta G, "Classification of Thallium-201 Scintigrams Using a Neural Network Trained by Back Propagation," *Journal of Nuclear Medicine*, vol. JULY, p. 1314, 1988.

52. Porenta G. Dorffner G. Schedlmayer J. Sochor H., "Parallel Distributed Processing as a Decision Support Approach in the Analysis of Thallium-201 Scintigrams," in *Proc. Computers in Cardiology*, pp. 259-262, Washington, 1988.

53. DeLand F.H, and Wagner H.N, *Atlas of Nuclear Medicine, Vol 2, Lung and Heart*, W.B. Saunders, 1970.

54. Gonzalez R.C, Wintz P, *Digital Image Processing*, Addison Wesley, 1986.

55. Castleman K.R, *Digital Image Processing*, Prentice Hall, 1979.

56. Lippmann R.P, *IEEE ASSP Magazine*, vol. APRIL, 1987.

57. Hinton G.E, "Connectionist Learning Procedures," *Artificial Intelligence*, vol. 40, pp. 185-234, 1989.

58. Barber D.C, "The Use of Principal Components in the Quantitative Analysis of Gamma Camera Dynamic Studies," *Physics in Medicine & Biology*, vol. 25 NO 2, pp. 283-292, 1980.

59. Nijran K.S, and Barber D.C, "Towards Automatic Analysis of Dynamic Radionuclide Studies Using Principal-Components Factor Analysis," *Physics in Medicine & Biology*, vol. 30 NO 12, pp. 1315-1325, 1985.

60. Cottrell G.W, "Face Recognition Using Feature Extraction," *INNS Int. Neural Networks Conference*, vol. 1, pp. 322-325, Paris, 1990.

61. Nakane K, "Image Data Compression using a Neural Network Model," *IEEE Proc. Int. Conf. Neural Networks*, vol. II, pp. 35-41, San Diego, 1989.

62. Sicuranza G.L, and Ramponi G, "Artificial Neural network for Image Compression," *Electronic Letters*, vol. 26:7, pp. 477-479, 1990.

63. Sirat J.A, Viala J.R, and Remus C, "Image Compression with Competing Multilayer Perceptrons," *IEE Int. Conference on Neural Networks*, pp. 404-408, London, 1989.

64. Naillon M, Theeten J.B, and Krauth W, "Self Organising Hopfield Network (SOHN) Application to TV Image Compression," *IEE Int. Conference on Neural Networks*, London, 1989.

65. Anthony D.M, Hines E.L, Taylor D, and Barham J, "An Investigation into the Use of Neural Networks for an Expert System in Nuclear Medicine Image Analysis," *IEE Conference on Image Processing*, pp. 338-342, Warwick University, Coventry, 1989.

66. Anthony D, Barham J, Hines E.L, Taylor D, , *IJCNN International Joint Conference on Neural Networks*, 1, pp. 339-344, San Diego, 1990.

67. Baldi P, and Hornik K, "Neural Networks and Principal Component Analysis: Learning from Examples without Local Minima," *Neural Networks*, vol. 2, pp. 53-58, 1989.

68. Jolliffe I.T, *Principal Component Analysis*, Spinger-Verlag, 1986.

69. Oja E, "A Simplified Neuron Model as a Principal Component Analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267-273, 1982.

70. Oja E, "Neural Networks, Principal Components, and Subspaces," *Int. Journal of Neural Systems*, vol. 1 NO 1, pp. 61-68, 1989.

71. Anthony D.M, Hines E.L, Taylor D, and Barham J, *A Study of Data Compression using Neural Networks and Principal Component Analysis (IEE Coloquiuum on Biomedical Applications of Digital Signal Processing)*, pp. 2/1-2/4, London, 1989.

72. Bourland H, and Kamp Y, "Auto-Association by Multilayer Perceptrons and Singular Value Decomposition," *Biological Cybernetics*, vol. 59, pp. 291-294, 1988.

73. Rumelhart D.E, Hinton G.E, and Williams N, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323 NO 9, pp. 533-536, 1986.

74. Moler C, Little J and Bangert S, *Pro-Matlab for Sun Workstations. User's Guide*, The MathWorks Inc, 1987.

75. Chauvin Y, *Generalization Performance of Overtrained Back-propagation Networks (in Neural Networks ed Almeida L.B and Wellekens C.J, Springer-Verlag)*, pp. 46-55, 1987.

76. Chan L, Fallside F, "An Adaptive Training Algorithm for Back Propagation Networks," *Computer Speech and Language*, vol. 2, pp. 205-218, 1987.

77. Bajpai A.C, Calus I.M, Fairley J.A, *Statistical Methods for Engineers and Scientists*, John Wiley, 1978.

78. Leen T.K, "Neural Network Data Encoding and PCA," *Neural Network Review*, 1990.

79. Hutchinson R.A,, "Development of an MLP Feature Location Technique Using Preprocessed Images," *INNS Int. Neural Networks Conference*, vol. 1, pp. 67-70, Paris, 1990.

80. Anthony D.M, Hines E.L, Taylor D, and Barham J, "The Use of Neural Networks to Classify Lung Scintigrams," *IASTED Conference on Applied Informatics*, pp. 240-242, 1990.

81. Chan L, "Efficacy of Different Learning Algorithms of the Back Propagation Network (in Conference Proc. Computer and Communication Systems, TENCON 90)," *Proceedings of the IEEE Region 10*, 1990.

82. Yorkey T.J, and Webster J.G, "A Comparison of Impedance Tomographic Reconstruction Algorithms," *Physical Physiology Measurement*, vol. 8 SUPPL. A, pp. 55-62, 1987.

83. Murai T, and Kagawa Y, "Electrical Impedance Computed Tomography Based on a Finite Element Model," *IEEE Transactions*, vol. BME 32, pp. 177-184, 1985.

84. Tesauro G, "Scaling Relationships in Back-Propagation Learning: Dependence on Training Set Size," *Complex Systems*, vol. 1:2, pp. 367-372, 1987.

85. Vogl T.P, Mangis J.K, Rigler A.K, Zink W.T, and Alkon D.L, "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, vol. 59, pp. 257-263, 1988.

86. Cater J.P, "Successfully Using Peak learning Rates of 10 (and Greater) in Back-propagation Networks with the Heuristic Learning Algorithm," *IEEE Proc. Int. Conf. Neural Network* , vol. II, pp. 645-651, San Diego, 1987.

87. Harp S.A, Samad T, and Guha A, "Towards the Genetic Synthesis of Neural Networks," *Int. Conference on Genetic Algorithms*, pp. 360-369, MIT, Cambridge, Massachusetts, 1989.

88. Anthony D.M, Hines E.L, Taylor D, and Barham J, "The Use of Genetic Algorithms to Learn the Most Appropriate Inputs to a Neural Network," *IASTED Conf. Artificial Intelligence App. & Neural Networks*, pp. 223-226, Zurich, 1990.

89. Farmer J.D, and Sidorowitch J.J, "Predicting Chaotic Time Series," *Physical Review Letters*, vol. 59 NO 8, pp. 845-848, 1987.

90. Anthony D, Barham J, Hines E.L, Taylor D, "The Use of Artificial Neural Networks in Diagnosis of Lung Scintigrams," *Royal Society of Medicine Meeting on Computers in Medicine, Chester, Great Britain*, 1990.

91. De Jong K, "Adaptive System Design : A Genetic Approach," *IEEE Transactions*, vol. SMC-10:9, pp. 566-574, 1980.

92. Grenfenstette J.J, *A User's Guide to GENESIS (Technical Report CS-84-11, Computer Science Dept., Vanderbilt Univ, Nashville)*, 1984.

93. Holland J, "Genetic Algorithms and Classifier Systems: Foundations and Future Directions," *Proc. Int. Conf. on Genetic Algorithms*, pp. 82-89, 1987.

94. Anthony D.M, Hines E.L, Taylor D, and Barham J, "The Use of Neural Networks in Classifying Lung Scintigrams," *INNS Int. Neural Networks Conference*, vol. 1, pp. 71-74, Paris, 1990.

95. Whitley D, "Applying Genetic Algorithms to Neural Network Learning (in Proceedings of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour, ed Cohn A)," *Robotics Neural Networks and Vision*, pp. 137-144, 1989.

96. Schaffer J.D, Caruana R.A, and Eshelman L.J, *Using Genetic Search to Exploit the Emergent Behavior of Neural Networks (Technical Report, Phillips Labs, New York)*, pp. 1-8, 1989.

97. Miller G.F, Todd P.M, and Hegde S.U, "Designing Neural Networks Using Genetic Algorithms," *Int. Conference on Genetic Algorithms*, pp. 379-384, MIT, Cambridge, Massachusetts, 1989.

98. De Garis H, *WALKER, A Genetically Programmed, Time Dependent, Neural Net Which Teaches a Pair of Sticks to Walk (Technical Report, Center for AI, George Mason Univ, Virginia)*, 1989.

99. Dodd N, "Optimisation of Network Structure using Genetic Algorithms," *INNS Int. Neural Networks Conference*, vol. 2, pp. 693-696, Paris, 1990.

100. Suh J.Y, van Gucht D, "Incorporating Heuristic Information into Genetic Search," *Proc. Int. Conf. on Genetic Algorithms*, pp. 100-107, 1987.

101. Baker J.E, "Reducing Bias and Inefficiency in the Selection Algorithm," *Proc. Int. Conf. on Genetic Algorithms*, pp. 14-21, 1987.

102. Penfold H.B, Diessel O.F and Bentink M.W, "A Genetic Breeding Algorithm Which Exhibits Self-Organizing in Neural Networks," *IASTED Conf. Artificial Intelligence App. & Neural Networks*, pp. 293-296, Zurich, 1990.

103. Cohoon J.P, Hegde S.U, Martin W.N, and Richards D, "Puntuated Equilibria: A Parallel Genetic Algorithm," *Proc. Int. Conf. on Genetic Algorithms*, pp. 148-154,

1987.

104. De Garis H, *Genetic Programming, Modular Neural Evolution for Darwin Machines (Technical Report, Machine Learning and Inference Laboratory, George Mason University)*, 1989.

105. Leen T.K, Rudnick M and Hammerstrom D, "Hebbian Feature Discovery Improves Classifier Efficency," *IEEE Proc. Int. Conf. Neural Network* , vol. 1, pp. 51-56, San Diego, 1990.

106. Canning A, *Neural Network Models with Partial Connectivity (in Neural Networks from Models to Applications ed Personnaz L and Dreyfus G)*, pp. 326-335, 1988.

107. Mozer M.C, and Smolensky P, "Using Relevance to Reduce Network Size Automatically," *Connection Science*, vol. 1:1, pp. 3-16, 1989.

108. Darbel N, Jutland F, and Trotin A, "A Neural net Sized by Data," *IASTED Conf. Artificial Intelligence App. & Neural Networks*, pp. 45-49, Zurich, 1990.

109. Anthony D.M, "Reducing Connectivity in Compression Networks," *Neural Network Review*, 1990.

110. Compiani M, Montanari D, Serra R, and Valastro G, *Classifier Systems and Neural Networks (in Parallel Architectures and Neural Networks, ed. Cianiello E.R, World Scientific)*, pp. 105-118, 1988.

111. Swets J.A, "ROC Analysis Applied to the Evaluation of Medical Imaging Techniques," *Investigative Radiology*, vol. 14 PT 2, pp. 109-121, 1979.

The Use of Artificial Neural Networks in Classifying Lung Scintigrams Volume 2 (of 2)

Denis Anthony

Thesis Submitted for the Award of PhD in Engineering
Department of Engineering
Warwick University
February 1991
Supervised by Dr. E. L. Hines

APPENDICES

Contents

Appendix 1 Source Code of Image Processing System

This appendix gives the source for the header files and functions used in the image processing suite. All the code is written in the C language, and is compiled under Borland (trademark) Turbo C. As it uses extended Turbo C functions and MSDOS calls it will not transfer directly to another processor environment, and may need modification if another compiler is used.

```
/* PHD.H */

/* This header file contains defintions for use in all parts of the program. It includes relevant header files.
definitions are given, and all the functions are listed under the file that they reside in, and with the callin
parameters required. */

#include <dos.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>
#include <dos.h>
#include <dir.h>
#include <graphics.h>
#include <ctype.h>
#include <string.h>
#include <fcntl.h>

/* #define EVOR 1 */  /* This means 512 by 512 images for Evor Hines */
/* remove above line if images 768 by 288 (Digit form) */
#include <graphics.h>
#define DELAY 2000 /* time for delay of messages on screen before moving on */
#define VIDEO 0x10
#define MOUSE 1
#define MSDOS 0x21
#define KEYBOARDCALL 0x16
#define MSYSCALL 0x33
#define MAX_MENU_ITEMS 20
#define MAX_FILES 99
#define MAX_SEGMENTS 255
#define THRESHOLD 10
#define NO_TARGETS 12
#define FROM_FILE 0
#define OK 1
#define TRUE 1
#define FALSE 0
#define MAX_MEDIAN_FILTER_SIZE 6
#define PIXEL_BITS 8
#define MONO 0 /* non zero for monochrome, 0 for colour */
#define PIXEL_SIZE 16
#define FILE_BYTE_SIZE 256
#ifdef EVOR
#define XSIZE 512
#define YSIZE 512
#define PC_X_SIZE 640 /* size of cols on PC */
#define PC_Y_SIZE 200 /* 200 size of rows on PC */
#else
#define XSIZE 768     /* 768 Horizontal (row) size of image file */
#define YSIZE 288      /* 288 Vertical (column) size of image file */
#define PC_X_SIZE 640 /* size of cols on PC */
#define PC_Y_SIZE 200 /* 200 size of rows on PC */
#endif
#define SpecialKey 0
#define LOOK '1'
```

```
#define ESC   0x1b
#define CR       13
#define F1    0x3b
#define F2    0x3c
#define F3    0x3d
#define F4    0x3e
#define F5    0x3f
#define F6    0x40
#define F7    0x41
#define F8    0x42
#define F9    0x43
#define F10   0x44
#define F11   0x54
#define CURS_PLUS 0x2b
#define CURS_MINUS 0x2d
#define CURS_UP    0x48
#define CURS_DOWN  0x50
#define CURS_LEFT  0x4b
#define CURS_RIGHT 0x4d

typedef char FileName[13];
typedef char ShStr[20];
typedef char LongStr[40];

typedef struct {
        int left;
        int right;
        int bottom;
        int top;
        } corners;

typedef struct {
        corners c;
        FileName name;
        int LowThreshold;
        int HighThreshold;
        struct palettetype palette;
        } imagetype;

typedef struct {
        int noitems;
        int position;
        char *banner;
        char *item[MAX_MENU_ITEMS];
        char *choice;
        } menutype;

typedef struct {
        FileName search,infile,outfile,dfile,tfile;
        FileName *RandFiles;
        char *buffer;
        struct ffblk ffblk,d_ffblk;
        int TotFilesUsed,NoRectX,NoRectY,NoFiles;
        int width,height;
        int NW,UseDFile,done,d_done;
        corners c;
        } ConvType;

typedef struct {
        struct ffblk ffblk,ffblkS;
        int done,doneS,index,NoSubs,NoFiles;
        long int *fsize;
        FileName *list;
        int CharHeight,CharWidth,FilesLine,NoLines;
        int FilesPage,back,fore,OptionColour;
        int FiNaLen;
        void *buffer;
        size;
        } dirtype;

typedef struct {
        FileName search,infile,outfile,tfile;
        char *StanBuffer;
        int XStan,YStan,width,height,rows,cols,ROW,COL;
        int done;
        FILE *InFilePtr,*OutFilePtr;
        char temp;
        long int position;
        char *buffer;
        struct ffblk ffblk;
        corners c;
        } stantype;


/* phdconc.c */
int CtoPDP(ConvType *p);
int SetConvert();
int SetPhil2Mat(ConvType *p);
int SetCtoMAT(ConvType *p);
int SetCtoNW(ConvType *p);
int SetCtoPDP(ConvType *p);
int SetRanCtoNW(ConvType *p);
int SetNWorPDPtoC(ConvType *p);
int CtoMAT(ConvType *p);
int CtoNW(ConvType *p);
int Phil2Mat(ConvType *p);
int PDPtoC(ConvType *p);


/* phdgraph.c */
int WritePixel(int screen,int x,int y,int z);
int SetGraphMode();
int SetGraphMode();
int OpenGraphics();
int TestGraphics();
float TimeSec(struct time *t);
int PrTime(struct time *t);
int CloseGraphics();
int SetAcquire(int DisplayType);
int ShowCorners(FileName *infile);
int acquire(int screen,FileName *infile);
char ReadPixel(int screen,int x,int y);
int SetClear();
int Write256Pixel(int x,int y,int z);
int acquire256(FileName *infile);
int VgaMode(int mode);
int clear(int screen);
int SetAlternateScreen();
int SetPage(int screen);
int SetPixelDisplay();
int PixelDisplay(int screen);
int print(char *s);
int GrWait(int message);
int SetCopyScreen();
int CopyScreen( int screen1, int screen2);
int AcquireCompressedImage(int screen,FileName *infile);
int SaveCompressedImage(int screen,FileName *outfile);
int CopyCorners(corners *from,corners *to);
int CheckCorners(int screen);


/* phdlut.c */
int VgaRGB();
int VgaMono();
int hot_body();
int SetRedGreenBlue();
int RedGreenBlue(int screen);
int SetColour();
int Colour(int screen);
int SetMono();
int Mono(int screen);

/* phdmask.c */
int SetFilterMask();
int FilterMask(FileName *infile,FileName *outfile,int filt_type);

/* phdmed.c */
int SetMedian();
int median(int compress,char *suffix,int ESPFile,FileName *infile,int XFiltWidth,int YFiltWidth);

/* phdmouse */
int InitMouse();
int ClearMouse();
int MouseButton();
int MouseGet(int *dx,int *dy);
int SetMouse(int x,int y,int z);
int GetMouseCh(int *x,int *y);
int MouseKeyGet();

/* phdnworks */
int colors(int fore,int back);
int DispMenu(menutype *menu);
int phdnwork();
int ManipulateMenu();
int SetDir();
int dir(char *spath,FileName *file,int GetFile,char *string);
int GetNextDirScreen(dirtype *p);
int CheckRoom(void *pointer,char *function);

/* phdrast.c */
int raster();
int save_rasters(corners *r,int x_size,int y_size,int x_disp,int y_disp,FileName *infile);

/* phdrect */
int SetDrawRect();
int DrawRect(int screen,corners *c);
int GetIncForRect(int *x_increment,int *y_increment,corners *c,int loop,int key);
int SetSubtractRect();
int MoveRect(int screen,corners *current,corners *from);
int SubtractRect(FileName file1,FileName file2,FileName file3,int XDiff,int YDiff);
int ShowSubtractParameter(int screen,int *MinScore,corners *from,corners *current);
int SetSaveRect();
int SaveRect(int screen,corners *rect,FileName *outfile);

/* phdseg.c */
int SetSegment();
int segment(FileName *infile);
int grow_region(FileName *infile,FILE *OutFilePtr);
int Link1Pixels(int x,int y,int *SegNo);
int Link2Pixels(int x,int y);
int save_seg(FileName *infile,FileName *outfile,corners seg,int *LowThreshold,int *HighThreshold);
int ClipScreen(int screen);

/* phdsob */
int SetSobel();
int sobel(FileName *in_file,FileName *out_file);

/* phdstan */
int SetStan();
int stan1(stantype *p);
int stan2(stantype *p);
int ExpandFile(FileName *infile,FileName *outfile,int XScale,int YScale,corners *c);


/* phdsub.c */
int SetSubFile();

/* phdthre */
int SetThreshold();
int GetThreshold(int screen);

/* phdturn */
int SetTurnRound();
int TurnRoundImage(FileName *infile,FileName *outfile);
int SetXReflect();
int SetYReflect();
int SetExpand();
int LeftToRight();
int UpsideDown();
```

```
/* phdutil.c */
int IsGetCharCR();
int WriteCorners(FILE *OutFilePtr,corners *c);
int ReadCorners(FILE *InFilePtr,corners *c);
int putsuffix(char *infile,char *suffix);
int SaveFile(FileName *outfile);
int contour();
int file_check(FileName *infile,FILE *in_file_ptr);
int cls();
int wait();
int exit1(int value);
int dummy();
int matches(char *s1,char *s2,int n);
int CheckSuffix(char *infile,char *string);

/* PHDNW.H */
#define MAX_FILES 99
#define MAX_SEGMENTS 99
#define FILE_NAME_LEN 12
#define OK 0
#define TRUE 1
#define FALSE 0
#define FILE_BYTE_SIZE 256
#define PC_X_SIZE  256 /* 640 size of cols on PC */
#define PC_Y_SIZE  96 /* 200 size of rows on PC */
#define THRESHOLD 192
#define SHOW_GRAPHICS 1
#define PIXEL_SIZE 16
#define SpecialKey 0
#define LOOK 'l'
#define ESC    0x1b
#define CURS_PLUS 0x2b
#define CURS_MINUS 0x2d
#define CURS_UP    0x48
#define CURS_DOWN   0x50
#define CURS_LEFT   0x4b
#define CURS_RIGHT  0x4d
```

```
/* PHDCONV.C */
#include <phd.h>
#include <phdext.h>

SetConvert()
{
/* PHDCONV.C */
/* Converts integer file for entry to NWORKS  or PDP */
/* definitions */
menutype menu = {
        8,
        0,
        "CONVERSIONS to or from various NETWORK programs",
        "C from NWORKS/PDP",
        "HELP",
        "MATLAB from C",
        "NWORKS from C",
        "PDP from C",
        "Philip files (NWORKS) to MATLAB",
        "Random files to NWORKS from C (Max 100 files)",
        "Restore Image from PDP .log file",
        "","",
        "","","","","","","","","","",
        "    ",
        };
ConvType *p;

/* working */
strcpy(menu.choice,"    "); /* should not be necessary, just to make sure */
p = malloc(sizeof(ConvType));
if (CheckRoom(p,"SetConvert")) {
        while (strncmpi(menu.choice,"quit",4) ) {
                setgraphmode(0);
                DispMenu(menu);
                if (!strncmpi(menu.choice,"c fro",4))
                        SetNWorPDPtoC(p);
                else if (!strncmpi(menu.choice,"help",4))
                        SetHelp(menu);
                else if (!strncmpi(menu.choice,"matl",4))
                        SetCtoMAT(p);
                else if (!strncmpi(menu.choice,"nwor",4))
                        SetCtoNW(p);
                else if (!strncmpi(menu.choice,"pdp ",4))
                        SetCtoPDP(p);
                else if (!strncmpi(menu.choice,"phil",4))
                        SetPhil2Mat(p);
                else if (!strncmpi(menu.choice,"rand",4))
                        SetRanCtoNW(p);
                else if (!strncmpi(menu.choice,"rest",4))
                        PDPtoC(p);
                }
        }
free(p);
setgraphmode(0);
return(0);
```

```
}


int SetPhil2Mat(p)
ConvType *p;
{
/* definitions */
int c;

/* working */
while ((c = dir(spath,p->infile,"File to transfer to MATLAB")) != ESC & c != CR)
        ;
if (c != ESC) {
        printf("\nOutput file (will be appended to if already exists\n");
        scanf(" %s", p->outfile);
        p->done = findfirst(p->search,p->ffblk,0);
        while (!p->done) {
                strcpy(p->infile,p->ffblk.ff_name);
                p->done = findnext(p->ffblk);
                Phil2Mat(p);
                }
        }
}


int SetCtoMAT(p)
ConvType *p;
{
/* definitions */
int c;

/* working */
while ((c = dir(spath,p->search,"File to tranfer to MATLAB")) != ESC & c != CR)
        ;
if (c != ESC) {
        strcpy(p->infile,p->ffblk.ff_name);
        printf("\nOutput file (will be appended to if already exists\n");
        scanf(" %s", p->outfile);
        p->done = findfirst(p->search,p->ffblk,0);
        while (!p->done) {
                CtoMAT(p);
                p->done = findnext(p->ffblk);
                }
        }
free(p);
}


int SetCtoNW(p)
ConvType *p;
{
/* definitions */
int c;

/* working */
```

```
while ((c = dir(spath,p->search,"File to tranfer to NWORKS")) != ESC & c != CR)
          ;
if (c != ESC) {
          p->done = findfirst(p->search,p->ffblk,0);
          printf("\np->Use Desired Output file (0 no, 1 yes)\n");
          scanf(" %d", p->UseDFile);
          if (p->UseDFile) {
                    while ((c = dir(spath,p->search,"Desired output file for NWorks")) != ESC & c != CR)
                              ;
                    if (c != ESC)
                              p->UseDFile = 0;
                    }
          else
                    strcpy(p->dfile," ");
          printf("\nFile for output (appended to if already existing)");
          scanf(" %s", p->outfile);

          if (p->UseDFile)
                    p->d_done = findfirst(p->search,p->d_ffblk,0);
          else
                    p->d_done = 0;
          while (!p->done & !p->d_done) {
                    strcpy(p->infile,p->ffblk.ff_name);
                    if (p->UseDFile) {
                              strcpy(p->dfile,p->d_ffblk.ff_name);
                              CtoNW(p);
                              p->d_done = findnext(p->d_ffblk);
                              }
                    else
                              CtoNW(p);
                    p->done = findnext(p->ffblk);
                    }
          }
}


int SetCtoPDP(p)
ConvType *p;
{
/* definitions */
int c;

/* working */
while ((c = dir(spath,p->search,"Input file to PDP")) != ESC & c != CR)
          ;
if (c != ESC) {
          p->done = findfirst(p->search,p->ffblk,0);
          printf("Use a desired output file, (default yes)");
          p->UseDFile = IsGetCharCR();
          if (p->UseDFile) {
                    while ((c = dir(spath,p->search,"Desired output for PDP")) != ESC & c != CR)
                              ;
                    if (c == ESC)
                              p->UseDFile = 0;
                    else
                    p->d_done = findfirst(p->search,p->d_ffblk,0);
                }
        else
                p->d_done = 0;
        printf("\nFile for output (appended to if already existing)");
        scanf(" %s", p->outfile);
        while (!p->done & !p->d_done) {
                strcpy(p->infile,p->ffblk.ff_name);
                if (p->UseDFile) {
                        strcpy(p->dfile,p->d_ffblk.ff_name);
                        CtoPDP(p);
                        p->d_done = findnext(p->d_ffblk);
                        }
                else
                        CtoPDP(p);
                p->done = findnext(p->ffblk);
                }
        }
}


int SetRanCtoNW(p)
ConvType *p;
{
/* definitions */
int c,i;

/* working */
while ((c = dir(spath,p->search,"Random Input file to PDP")) != ESC & c != CR)
        ;
if (c != ESC) {
        printf("\nFile for output (in transfer from C appended to if already existing)\n");
        scanf(" %s", p->outfile);
        p->done = findfirst(p->search,p->ffblk,0);
        p->NoFiles = 0;
        while (!p->done) {
                p->done = findnext(p->ffblk);
                p->NoFiles++;
                }
        p->RandFiles = malloc(sizeof(FileName) * (p->NoFiles + 1));
        if (CheckRoom(p->RandFiles,"SetRandCtoNW") ) {
                p->done = findfirst(p->search,p->ffblk,0);
                i = 0;
                while (!p->done) {
                        strncpy(p->RandFiles[i],p->ffblk.ff_name,12);
                        i++;
                        p->done = findnext(p->ffblk);
                        }
                printf("\nNumber of Files to choose randomly from is %d",p->NoFiles + 1);
                printf("\nHow many files required ");
                printf("\n(Unlimited number as files can be used more than once\n");
                scanf(" %d", p->TotFilesUsed);
                printf("\nUse Same File as Desired Output  (default yes)\n");
                scanf(" %d", p->UseDFile);
                randomize();
                for (i = 0;i < p->TotFilesUsed;i++) {
                        strcpy(p->infile,p->RandFiles[random(p->NoFiles)]);
                        printf("\nFile %s",p->infile);
                        if (p->UseDFile)
                                CtoNW(p);
                        else
                                CtoNW(p);
                        }
                }
        }
free(p->RandFiles);
}


int SetNWorPDPtoC(p)
ConvType *p;
{
/* definitions */
int c;

/* working */
while ((c = dir(spath,p->infile,"C image file to tranfer into PDP, NO Wildcard allowed"))
!= ESC & c != CR)
        ;
if (c != ESC) {
        printf("\nFile for output \n");
        scanf(" %s",p->outfile);
        printf("\nThis routine recovers an image file rasterised for entry to NWORKS/PDP");
        printf("\nIt is assumed that non-overlapping regular rectangles have beed used");
        printf("\nThe following propts find out the size and number of thse rectangles");
        printf("\nWidth of rectangles used\n");
        scanf(" %d", p->width);
        printf("\nHeight of rectangles used\n");
        scanf(" %d", p->height);
        printf("\nNumber of rectangles used in X axis\n");
        scanf(" %d", p->NoRectX);
        printf("\nNumber of rectangles used in Y axis\n");
        scanf(" %d", p->NoRectY);
        p->buffer = malloc(sizeof(char) * p->width * p->height * p->NoRectX * p->NoRectY);
        if (CheckRoom(p->buffer,"SetNWorPDPtoC") ) {
                printf("\nNWorks or PDP used as source (1 for NWORKS (.nni files) 0 for PDP (.pat files))\
                scanf(" %d", p->NW);
                if (p->NW)
                        NWtoC(p);
                else
                        PDPtoC(p);
                }
        }
free(p->buffer);
}


int CtoMAT(p)
ConvType *p;
```

```
{
/* definitions */
int i;
FILE *InFilePtr,*OutFilePtr;
int c;
int x,y;

/* workings */
printf("\nConversion of file for entry to MATLAB software ");
printf("\nOpening file %s for inputs, %s for outputs",p->infile,p->outfile);
InFilePtr = fopen(p->infile,"rb");
if (file_check(p->infile,InFilePtr) == 0)
        return(0);
OutFilePtr = fopen(p->outfile,"at");
if (file_check(p->outfile,OutFilePtr) == 0)
        return(0);
ReadCorners(InFilePtr,p->c);
for (y = p->c.bottom;y < p->c.top;y++) {
        for (x = p->c.left;x < p->c.right;x++) {
                if ((c = getc(InFilePtr)) == EOF)
                        c = 0;
                fprintf(OutFilePtr," %d",c);
                }
        }
fprintf(OutFilePtr,"\n");
fclose(InFilePtr);
fclose(OutFilePtr);
return(0);
}


int CtoNW(p)
ConvType *p;
{
/* definitions */
int i;
FILE *InFilePtr,*OutFilePtr;
int c;
int x,y;
float temp;
int AllSame;

/* workings */
printf("\nConverting file %s for entry to NWORKS",p->infile);
InFilePtr = fopen(p->infile,"rb");
file_check(p->infile,InFilePtr);
OutFilePtr = fopen(p->outfile,"at");
file_check(p->outfile,OutFilePtr);
printf("\nOpened file inputs %s desired outputs %s",p->infile,p->dfile);
fprintf(OutFilePtr,"\n* BEGIN %s",p->outfile);
ReadCorners(InFilePtr,p->c);
fprintf(OutFilePtr,"\n* CORNERS %d %d %d %d",p->c.left,p->c.right,p->c.bottom,p->c.top);
fprintf(OutFilePtr,"\ni ");
i = 0;
```

```
for (y = p->c.bottom;y < p->c.top;y++) {
        for (x = p->c.left;x < p->c.right;x++) {
                if ((c = getc(InFilePtr)) == EOF)
                        c = 0;
                if (i >= 8) {
                        fprintf(OutFilePtr,"\n");
                        i = 0;
                        }
                temp = (float) c / (float) FILE_BYTE_SIZE;
                fprintf(OutFilePtr," %0.3f",temp);
                i++;
                }
        }
fprintf(OutFilePtr,"\n* END INPUT %s",p->infile);
if (p->dfile[0] != ' ') {
        fprintf(OutFilePtr,"\n* START DESIRED OUTPUT %s",p->dfile);
        fprintf(OutFilePtr,"\nd ");
        fclose(InFilePtr);
        InFilePtr = fopen(p->dfile,"rb");
        file_check(p->dfile,InFilePtr);
        ReadCorners(InFilePtr,p->c);
        for (y = p->c.bottom;y < p->c.top;y++) {
                for (x = p->c.left;x < p->c.right;x++) {
                        if ((c = getc(InFilePtr)) == EOF)
                                c = 0;
                        if (i >= 8) {
                                fprintf(OutFilePtr,"\n");
                                i = 0;
                                }
                        temp = (float) c / (float) FILE_BYTE_SIZE;
                        fprintf(OutFilePtr," %0.3f",temp);
                        i++;
                        }
                }
        fprintf(OutFilePtr,"\n* END DESIRED OUTPUT %s",p->dfile);
        }
fclose(InFilePtr);
fclose(OutFilePtr);
return(0);
}


int CtoPDP(p)
ConvType *p;
{
/* definitions */
int i;
FILE *InFilePtr,*OutFilePtr;
int c;
int x,y;
float temp;
ShStr patt_name;
FileName s;

/* workings */
```

```
printf("\nConversion of file for entry to PDP software ");
printf("\nOpening file %s for inputs, %s for desired outputs",p->infile,p->dfile);
InFilePtr = fopen(p->infile,"rb");
if (file_check(p->infile,InFilePtr) == 0)
        return(0);
OutFilePtr = fopen(p->outfile,"at");
if (file_check(p->outfile,OutFilePtr) == 0)
        return(0);
strcpy(patt_name,"p"); /* to make sure name does not begin with a number */
strcat(patt_name,p->infile);
ReadCorners(InFilePtr,p->c);
itoa(p->c.left,s,10);
strcat(patt_name,"L");
strcat(patt_name,s);
strcat(patt_name,"R");
itoa(p->c.right,s,10);
strcat(patt_name,s);
strcat(patt_name,"B");
itoa(p->c.bottom,s,10);
strcat(patt_name,s);
strcat(patt_name,"T");
itoa(p->c.top,s,10);
strcat(patt_name,s);
fprintf(OutFilePtr,"%s",patt_name);
i = 0;
for (y = p->c.bottom;y < p->c.top;y++) {
        for (x = p->c.left;x < p->c.right;x++) {
                if ((c = getc(InFilePtr)) == EOF)
                        c = 0;
                if (i >= 10) {
                        fprintf(OutFilePtr,"\n");
                        i = 0;
                        }
                temp = (float) c / (float) FILE_BYTE_SIZE;
                fprintf(OutFilePtr," %0.3f",temp);
                i++;
                }
        }
i = 0;
if (p->dfile[0] != ' ') {
      fprintf(OutFilePtr,"\n");
        fclose(InFilePtr);
        InFilePtr = fopen(p->dfile,"rb");
        if (file_check(p->dfile,InFilePtr)) {
                ReadCorners(InFilePtr,p->c);
                for (y = p->c.bottom;y < p->c.top;y++) {
                        for (x = p->c.left;x < p->c.right;x++) {
                                if ((c = getc(InFilePtr)) == EOF)
                                        c = 0;
                                if (i >= 10) {
                                        fprintf(OutFilePtr,"\n");
                                        i = 0;
                                        }
                                temp = (float) c / (float) FILE_BYTE_SIZE;
                                fprintf(OutFilePtr," %0.3f",temp);
                        i++;
                        }
                }
        }
    }
fprintf(OutFilePtr,"\n");
fclose(InFilePtr);
fclose(OutFilePtr);
return(0);
}



NWtoC(p)
ConvType *p;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
float temp;
int x,y;
char s[17]; /* itoa only returns max 17 */
int i,j;
int c;

/* workings */
printf("\nConverting file %s for entry to C program",p->infile);
InFilePtr = fopen(p->infile,"rb");
if (file_check(p->infile,InFilePtr) == 0)
        return(0);
OutFilePtr = fopen(p->outfile,"wb");
if (file_check(p->outfile,OutFilePtr) == 0)
        return(0);
p->c.left = 0;p->c.bottom = 0;
p->c.right = p->width * p->NoRectX;
p->c.top = p->height * p->NoRectY;
WriteCorners(OutFilePtr,p->c);
x = 0;y = 0;
for (i = 0;i < p->NoRectX;i++) {
        for (j = 0;j < p->NoRectY;j++) {
                while (fscanf(InFilePtr,"%s",s) != EOF
                & (strncmp(s,"r",1))) /* next r */
                        ;
                for (y = j * p->height;y < (j + 1) * p->height;y++) {
                        for (x = i * p->width;x < (i + 1) * p->width;x++) {
                                fscanf(InFilePtr,"%s",s);
                                c = (int) (atof(s) * FILE_BYTE_SIZE);
                                if (c < 0)
                                        c = 0;
                                p->buffer[(y * p->width * p->NoRectX) + x] = (char) c;
                                }
                        }
                }
        }
for (y = 0;y < p->NoRectY * p->height;y++) {
        for (x = 0;x < p->NoRectX * p->width;x++) {
                c = p->buffer[y * p->NoRectX * p->width + x];
                putc(c,OutFilePtr);
                }
        }
for (y = 0;y < p->NoRectY * p->height;y++) {
        printf("\n");
        for (x = 0;x < p->NoRectX * p->width;x++) {
                printf("%d",p->buffer[y * p->NoRectX * p->width + x]);
                }
        }
wait();
fclose(InFilePtr);
fclose(OutFilePtr);
}




int Phil2Mat(p)
ConvType *p;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
int c;
char s[20];

/* workings */

printf("\nConverting Philip file %s to %s for entry to MATLAB",p->infile,p->outfile);
InFilePtr = fopen(p->infile,"rb");
file_check(p->infile,InFilePtr);
OutFilePtr = fopen(p->outfile,"at");
file_check(p->outfile,OutFilePtr);
while (fscanf(InFilePtr,"%s",s) != EOF) {
        if (strncmp(s,"i",1) == 0) {
                while ((fscanf(InFilePtr,"%s",s) != EOF)
                & (atof(s) > 0))
                        fprintf(OutFilePtr," %s",s);
                fprintf(OutFilePtr,"\n");
                }
        }
fclose(InFilePtr);
fclose(OutFilePtr);
}

int PDPtoC(p)
ConvType *p;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
char temp[80],s[5];
int i,j,k,c;
float f;
int result = 0;
```

```
/* workings */
while ((c = dir(spath,p->infile,"PDP to C image file")) != ESC & c != CR)
        ;
if (c != ESC)
        return(0);
printf("\nOutput file (Numbers from 0 will be  added to this, you may put '.'");
printf("\nat end of name if an extension of numbers is wanted");
scanf(" %s", p->tfile);
if ((InFilePtr = fopen(p->infile,"rb")) == NULL) {
        printf("\nFailed to open input file %s",p->infile);
        exit(0);
        }
temp[0] = ' ';
while  (temp[0] != 'p')
        fscanf(InFilePtr,"%s",temp);
i = 0;
while  (result != EOF) {
        fclose(OutFilePtr);
        if (temp[0] != 'p') {
                fclose(InFilePtr);
                return(1);
                }
        strcpy(p->outfile,p->tfile);
        itoa(i,s,36);
        strcat(p->outfile,s);
        if ((OutFilePtr = fopen(p->outfile,"wb")) == NULL) {
                printf("\nFailed to open output file %s",p->outfile);
                wait();
                return(0);
                }
        if (temp[0] != 'p')
                return(0);
        j = 0;
        while (temp[j] != '.')
                j++;
        while (temp[j] != 'L')
                j++;
        k = 0;
        while (temp[j] != 'R')
                s[k++] = temp[++j];
        s[k] = ' ';
        p->c.left = atoi(s);
        k = 0;
        while (temp[j] != 'B')
                s[k++] = temp[++j];
        s[k] = ' ';
        p->c.right = atoi(s);
        k = 0;
        while (temp[j] != 'T')
                s[k++] = temp[++j];
        s[k] = ' ';
        p->c.bottom = atoi(s);
        k = 0;
        while (temp[j] != ' ')
                s[k++] = temp[++j];
        s[k] = ' ';
        p->c.top = atoi(s);
        WriteCorners(OutFilePtr,p->c);
        temp[0] = ' ';
        while  (temp[0] != 'p')  {
                fscanf(InFilePtr,"%s",temp);
                if (temp[0] == '0' || temp[0] == '1') {
                        f = atof(temp);
                        f = f * 255.0 ;
                        c = (int) f;
                        putc(c,OutFilePtr);
                        }
                }
        i++;
        }
fclose(InFilePtr);
fclose(OutFilePtr);
}
```

```
/* PHDDIR.C */
#include <phd.h>
#include <phdext.h>

int colors(fore,back)
int fore,back;
{
setcolor(fore);
setbkcolor(back);
}

int DispMenu(menu)
menutype *menu;
{
/* definitions */
int size,i;
int c,j;
int XMargin,back;
int CharWidth,CharHeight;
signed int x,y;
void *buffer;
int YOffset = 2;
int OptionColour = 3;

/* working */

SetPage(0);
CharHeight = textheight(menu->banner);
CharWidth = textwidth("A");
clear(0);
XMargin = strlen(menu->banner);
for (i = 0;i < menu->noitems;i++)
        if (strlen(menu->item[i]) > XMargin)
                XMargin = strlen(menu->item[i]);
XMargin *= CharWidth;
XMargin = (getmaxx() - XMargin) / 2;
size = imagesize(XMargin,0,getmaxx() - 2 * XMargin,CharHeight - 1);
buffer = malloc(size);
if (CheckRoom(buffer,"dir buffer")) {
        setwritemode(XOR_PUT);
        setfillstyle(SOLID_FILL,OptionColour);
        bar(XMargin,0,getmaxx() - XMargin,CharHeight - 1);
        floodfill(XMargin,0,OptionColour);
        getimage(XMargin,0,getmaxx() - XMargin,CharHeight - 1,buffer);
        outtextxy(XMargin,0,menu->banner);
        for (i = 0;i < menu->noitems;i++)
                outtextxy(XMargin,(i + YOffset) * CharHeight,menu->item[i]);
        if (! mouse & mouse)
                outtextxy(XMargin,(menu->noitems + YOffset + 1) * CharHeight,
                "Make choice using cursor keys and RETURN, ESC to quit");
        else
                outtextxy(XMargin,(menu->noitems + YOffset + 1) * CharHeight,
                "Make choice using mouse and right button, left button to quit");
        c = ' '; /* anything but 13 (return) */
        putimage(XMargin,(menu->position + YOffset) * CharHeight,buffer,XOR_PUT);
```

```
                strncpy(menu->choice,menu->item[menu->position],4);
                if (mouse)
                        ClearMouse();
                while (c != CR & c != ESC) {
                        c = GetMouseCh(x,y);
                        if (c == CURS_UP || c == CURS_DOWN) {
                                putimage(XMargin,(menu->position + YOffset) * CharHeight,buffer,XOR_PUT);
                                if (c == CURS_UP)
                                        menu->position--;
                                else if (c == CURS_DOWN)
                                        menu->position++;
                                if (menu->position < 0)
                                        menu->position = menu->noitems - 1;
                                else if (menu->position >= menu->noitems)
                                        menu->position = 0;
                                putimage(XMargin,(menu->position + YOffset) * CharHeight,buffer,XOR_PUT);
                                strncpy(menu->choice,menu->item[menu->position],4);
                                }
                        }
                clearviewport();
                if (c == ESC) {
                        strcpy(menu->choice,"    ");
                        if (mouse) {
                                ClearMouse();
                                outtextxy(0,0,"Press Right Mouse key or ESC again to quit (Left to continue) ");
                                while ((c = GetMouseCh(x,y)) != CR & c != ESC)
                                        ;
                                }
                        else    {
                                outtextxy(0,0,"Press ESC key again to quit (any other key to continue) ");
                                c = getch();
                                }
                        if (c == ESC)
                                strcpy(menu->choice,"quit");
                        }
                }
        free(buffer);
        return(1);
        }


        int SetDir()
        {
        char s[13]; /* not used, provided for compatibility in calling dir */
        dir(spath,s," ");
        }

        /* DIR */
        int dir(spath,file,string)
        FileName *file; /* used by SetAcquire */
        char *spath,*string;
        {
        /*definitions */
        dirtype *p;
        int c,i;
```

```
FileName temp;
signed int x,y,dx,dy;
/* although x,y are usually non negative, negative values indicate out of range and are therefore used */

/* working */
p = malloc(sizeof(dirtype));
if (CheckRoom(p,"dir p") == 0)
        return(0);
p-> FiNaLen = sizeof(FileName) + 7;
p->CharHeight = textheight("h");
p->CharWidth = textwidth("h"); /* any old letter will do */
p->FilesLine = getmaxx() / (p->CharWidth * p->FiNaLen);
p->NoLines = 20;
p->size = imagesize(0,0,p->CharWidth * p->FiNaLen,p->CharHeight);
p->FilesPage = p->FilesLine * p->NoLines;
p->buffer = malloc(p->size);
p->list = malloc(sizeof(FileName) * p->FilesPage);
p->fsize = malloc(sizeof(long int) * p->FilesPage);
SetPage(0);
clear(0);
c = 0;
setwritemode(XOR_PUT);
p->OptionColour = 3;
setfillstyle(SOLID_FILL,p->OptionColour);
bar(0,0,p->CharWidth * p->FiNaLen,p->CharHeight);
floodfill(0,0,p->OptionColour);
getimage(0,0,p->CharWidth * p->FiNaLen,p->CharHeight - 1,p->buffer);
strcpy(p->list[0],"NEXT");
strcpy(p->list[1],"WILD");
while (c == 0 & CheckRoom(p->buffer,"dir p->buffer") &
CheckRoom(p->list,"dir p->list") & CheckRoom(p->fsize,"dir p->fsize")) {
        clear(0);
        p->doneS = findfirst("*",p->ffblkS,22);
        p->done = findfirst(spath,p->ffblk,0);
        c = 0;
        while ((!p->done || !p->doneS) & c != ESC & c != CR) {
                clear(0);
                outtextxy(0,(p->NoLines + 2) * p->CharHeight,
            "ESC - Quit, RET - choose, space - next screen, w - wild card");
                outtextxy(0,(p->NoLines + 4) * p->CharHeight,string);
                GetNextDirScreen(p);
                for (i = 0;i < p->NoFiles;i++) {
                        x = ((i % p->FilesPage)  % p->FilesLine) * p->CharWidth * p->FiNaLen;
                        y = ((i % p->FilesPage)  / p->FilesLine) * p->CharHeight;
                        strcpy(temp,p->list[i]);
                        if (i < p->NoSubs & i > 1)
                                strcat(temp,"/");
                        outtextxy(x,y,temp);
                        if (i < p->NoSubs)
                                strcpy(temp," ");
                        else
                                ltoa(p->fsize[i],temp,10);
                        outtextxy(x + 13 * p->CharWidth,y,temp);
                        }
                putimage(0,0,p->buffer,XOR_PUT);
```

```
			c = x = y = 0;
			while (c != CR  & c != ESC) {
				c = GetMouseCh(dx,dy);
			   p->index = (y / p->CharHeight) * p->FilesLine;
			   p->index += x / (p->CharWidth * p->FiNaLen);
			   if (c == CURS_UP || c == CURS_DOWN
				|| c == CURS_LEFT || c == CURS_RIGHT) {
					putimage(x,y,p->buffer,XOR_PUT);
					if (c == CURS_DOWN)
						y += p->CharHeight;
					else if (c == CURS_UP)
						y -= p->CharHeight;
					else if (c == CURS_LEFT)
						x -= p->CharWidth * p->FiNaLen;
					else if (c == CURS_RIGHT)
						x += p->CharWidth * p->FiNaLen;
					if  (x >= p->CharWidth * p->FiNaLen * p->FilesLine) {
						x = 0;
						y += p->CharHeight;
						}
					else if (x < 0) {
						x = p->CharWidth * p->FiNaLen * (p->FilesLine - 1);
						y -= p->CharHeight;
						}
					if (y < 0 )
						y = 0;
					if (y >= p->CharHeight * (p->NoLines + 2))
						y = 0;
					putimage(x,y,p->buffer,XOR_PUT);
					} /* end if CURSOR keys */
				}
			if (p->index == 0 & c != ESC)
				c = 0;
			} /* end while (c != CR & c != ESC) */
		if (c != ESC)  {
			if (p->index == 1) {
				clear(0);
				printf("\nWild card chosen, give wild specification\n");
				scanf(" %s", file);
				}
			else if (p->index < p->NoSubs) {
				chdir(p->list[p->index]);
				c = 0;
				}
			else
				strcpy(file,p->list[p->index]);
			}

		} /* end CheckRoom */
	free(p->buffer);
	free(p->fsize);
	free(p->list);
	free(p);
	clear(0);
	return(c);
```

```
}

int GetNextDirScreen(p)
dirtype *p;
{
int i = 2;
while (!p->doneS & i < p->FilesPage)  {
        if (p->ffblkS.ff_attrib  16) {
                strcpy(p->list[i],p->ffblkS.ff_name);
                i++;
                }
        p->doneS = findnext(p->ffblkS);
        } /* end if !p->doneS */
p->NoSubs = i;
while (!p->done & i < p->FilesPage) {
        strcpy(p->list[i],p->ffblk.ff_name);
        p->fsize[i] = (p->ffblk.ff_fsize);
        p->done = findnext(p->ffblk);
        i++;
        } /* end if !p->done */
p->NoFiles = i;
}
```

```
/* PHDGRAPH.C */
#include <phd.h>
#include <phdext.h>

/* WRITEPIXEL */
int WritePixel(screen,x,y,z)
int screen,x,y,z;
{
setactivepage(screen);
putpixel(x,y,z);
}

/* SET_GRAPH_MODE */
int SetGraphMode()
{
/* definitions */
int max_col,max_x,max_y;
struct viewporttype view;

/* working */
max_col = (int) getmaxcolor();
max_x = (int) getmaxx();
max_y = (int) getmaxy();
getviewsettings(view);
printf("\nleft %d bottom %d right %d top %d clip
%d",view.left,view.top,view.right,view.bottom,view.clip);
printf("\nNumber of colours %d",max_col);
printf("\nMax x %d",max_x);
printf("\nMax y %d",max_y);
printf("\nCurrently in mode %d",getgraphmode());
printf("\nMax mode %d",getmaxmode());
printf("\nChange to mode\n");
scanf(" %d", graphmode);
setgraphmode(graphmode);
}

/* OPEN_GRAPHICS ***************************************************************
* Open and initialize image processing hardware                              *
*******************************************************************************/

/*    Purpose - to set up graphics environment for pluto or IBM
      compatible.

      Method - If the pluto board is used then the board is initialised
      and low resolution is chosen, and a partition is allocated. If
      IBM compatible is used then the colours are re-allocated to give
      a red-green-blue display which is easier to view with than
      the default.
*/
int OpenGraphics()
{
int graphmode;
int errorcode;
int graphdriver;
int c,i;
/* workings */

graphdriver = DETECT;          /* detect and initialize graphix */
initgraph(graphdriver, graphmode, ".");
errorcode = graphresult();
if (errorcode != grOk) {
        printf("graphics error: %s\n",grapherrormsg(errorcode),1);
        delay(2000);
        return(0);
        }
setgraphmode(0);
setviewport(0,0,639,199,1); /* modes 0 and 1 are not recognised properly
        and give wrong viewport settings, hence need to set manulally */
for (i = 0;i < 4;i++) {
        im[i].c.left = im[i].c.bottom = 0;
        im[i].c.right = getmaxx();
        im[i].c.top = getmaxy();
        im[i].LowThreshold = 0;
        im[i].HighThreshold = getmaxcolor();
        getpalette(im[i].palette);
        }
cor.left = 0;cor.bottom = 0;
cor.right = getmaxx();cor.top = getmaxy();
return(0);
}


/* WRITEPIXEL */
int writepixel(x,y,z)
int x,y,z;
{
union REGS regs;
regs.h.dh = regs.h.ch = 0;
regs.h.dl = x;
regs.h.cl = y;
regs.h.al = z;
regs.h.ah = 0x0c;
int86(0x10,regs,regs);
}

/* CLOSE_GRAPHICS ***********************************************************
* Close image processing hardware                              *
****************************************************************************/

/*      Purpose - Close down graphics environment

        Method - Pluto requires no specific close-down. IBM compatibles
        are returned to text display.
*/

int CloseGraphics()
{
restorecrtmode();
return(0);
}
```

```
/* SET_ACQUIRE ************************************************************
* get an image file for diplay                                          *
*************************************************************************/

/*      Purpose - to find out which image is to be displayed

        Method - The file name, scrren to display on are entered and
        acquire is executed.
*/
int SetAcquire(DisplayType)
int DisplayType; /* 0 - normal file, 1 - 256 file, 2 - show corners only */
{
/* definitions */
struct ffblk ffblk;
FileName infile;
int c,done;
int screen;

/* working */
while ((c = dir(spath,infile,"Choose File for Display")) != CR & c != ESC)
        ;
if (! DisplayType & c != ESC) {
        printf("\nFile Chosen %s\n",infile);
        printf("\nOn page (screen) ?\n");
        scanf(" %d", screen);
        SetPage(screen);
        }
else if (DisplayType == 2)
        clear(0);
if (done = findfirst(infile,ffblk,0)) {
        printf("\nFile %s not found",infile);
        c = DelayOrKeyRead(DELTIME);
        }
while (!done) {
        if (DisplayType == 1)
                acquire256(ffblk.ff_name);
        else if (!DisplayType)
                acquire(screen,ffblk.ff_name);
        else ShowCorners(ffblk.ff_name);
        done = findnext(ffblk);
        if (!done)
                c = DelayOrKeyRead(DELTIME);
        if (c == ESC)
                done = 1;
        }
if (DisplayType == 1)
        setgraphmode(graphmode);
else if (DisplayType == 2)
        wait();
SetPage(0);
return(0);
}

int ShowCorners(infile)
FileName *infile;
```

```
{
corners c;
FILE *InFilePtr;
InFilePtr = fopen(infile,"rb");
if (!file_check(infile,InFilePtr))
        return(0);
SetPage(0);
if (ReadCorners(InFilePtr,c)) {
        printf("\nFile %13s Left %3d",infile,c.left);
        printf(" Right %3d",c.right);
        printf(" Bottom %3d",c.bottom);
        printf(" Top %3d",c.top);
        printf(" Width %3d Height %3d",c.right - c.left,c.top - c.bottom);
        }
fclose(InFilePtr);
return(OK);
}


int acquire256(infile)
FileName *infile;
{
/* definitions */
FILE *InFilePtr;
int x,y;
char far *buffer;

/* workings */
buffer = MK_FP(0xA000,0);
InFilePtr = fopen(infile,"rb");
if (! file_check(infile,InFilePtr))
        return(0);
strcpy(im[0].name,infile);
if (ReadCorners(InFilePtr,im[0].c)) {
        VgaMode(0x13);
        for (y = im[0].c.bottom;y < im[0].c.top;y++) {
                for (x = im[0].c.left;x < im[0].c.right;x++)
                        buffer[y * 320 + x] = getc(InFilePtr);
                }
        GrWait(0);
        VgaMono();
        GrWait(0);
        VgaRGB();
        GrWait(0);
        cor.left = im[0].c.left;cor.right = im[0].c.right;
        cor.bottom = im[0].c.bottom;cor.top = im[0].c.top;
        }
fclose(InFilePtr);
free(buffer);
return(OK);
}

int Write256Pixel(x,y,z)
int x,y,z;
{
```

```
union REGS regs;
regs.ah = 0x0c;
regs.al = z;
regs.cx = x;
regs.dx = y;
int86(VIDEO,regs,regs);
}

int VgaMode(mode)
int mode;
{
union REGS regs;
regs.ah = 0;
regs.al = mode;
int86(VIDEO,regs,regs);
}


/* ACQUIRE */
int acquire(screen,infile)
FileName *infile;
int screen;
{
/* definitions */
FILE *InFilePtr;
int x,y;
 int scale;

/* workings */
InFilePtr = fopen(infile,"rb");
if (!file_check(infile,InFilePtr))
        return(0);
strcpy(im[screen].name,infile);
if (ReadCorners(InFilePtr,im[screen].c)) {
        scale = FILE_BYTE_SIZE / PIXEL_SIZE;
        SetPage(screen);
        for (y = im[screen].c.bottom;y < im[screen].c.top;y++)
                for (x = im[screen].c.left;x < im[screen].c.right;x++)
                        putpixel(x,y,getc(InFilePtr)/scale);
        }
fclose(InFilePtr);
return(OK);
}


/* READ_PIXEL ***************************************************************
* Read a single pixel from image memory location x,y                  *
***************************************************************************/

char ReadPixel(screen,x,y)
int screen,x,y;
{
setactivepage(screen);
return(getpixel(x,y));
```

```
}


int SetClear()
{
int page;
printf("\nPage to clear");
scanf(" %d", page);
clear(page);
return(0);
}

/* CLEAR */
int clear(screen)
int screen;
{
SetPage(screen);
clearviewport();
SetPage(0);
}



int SetAlternateScreen()
{
/* definitions */
int screen;
int i;
int dx,dy;

/* working */
SetPage(0);
clear(0);
screen = 0;

for (i = 0;i < 4;i++)
        printf("\nScreen %d image %s last pasted in",i,im[i].name);
if (mouse) {
        printf("\nUse left button to advance page, right to escape, or
number for screen");
        while ((i = GetMouseCh(dx,dy)) != ESC) {
                if (i == CR) {
                        screen = ++screen;
                        screen %= 4;
                        SetPage(screen);
                        if (mouse)
                                ClearMouse();
                        }
                else if (i >= '0' & i <= '4')
                        SetPage(i - '0');
                }
        }
else {
        printf("\nPress number 0-3 for screen required, or ESCAPE to
return");
        while ((i = getch()) != ESC) {
                screen = i - '0';
                SetPage(screen % 4);
                }
        }
SetPage(0);
return(0);
}




/* SETPAGE */
int SetPage(screen)
int screen;
{
setactivepage(screen);
setvisualpage(screen);
setallpalette(im[screen].palette);
return(0);
}

/* SET_PIXEL_DISPLAY */
int SetPixelDisplay()
{
/* definitions */
int screen;

/* workings */
printf("\nScreen to look at\n");
scanf(" %d", screen);
PixelDisplay(screen);
}


int PixelDisplay(screen)
int screen;
{
/* definitions */
int x,y;
int dx = 0;
int dy = 0;
int x_increment,y_increment;
unsigned char c;

/* working */
setwritemode(XOR_PUT);
SetPage(screen);
x = y = 2;
y_increment = 1;x_increment = 1;
c = ' ';
while (c != ESC) {
    c = GetMouseCh(dx,dy);
```

```
        if ((c > '0') & (c <= '9'))
                x_increment = y_increment = c - '0';
        if (c == CURS_LEFT || c == CURS_RIGHT || c == CURS_UP
          || c == CURS_DOWN || c == CR) {
                rectangle(x - 1,y - 1,x + 1,y + 1);
             x += dx;
             y += dy;
                if (dx == 0 & dy == 0) {
                        switch(c) {
                                case CURS_LEFT :
                                        x -= x_increment;;
                                        break;
                                case CURS_RIGHT :
                                        x += x_increment;
                                        break;
                                case CURS_UP :
                                        y -= y_increment; /* screen goes
from 0 at top to PC_Y_SIZE at base */
                                        break;
                                case CURS_DOWN :
                                        y += y_increment;
                                        break;
                                }
                        }
                if (c == CR) {
                        SetPage(0);
                        printf("\nValue of pixel at x %d y %d is
%d",x,y,ReadPixel(screen,x,y));
                        wait();
                        SetPage(screen);
                        break;
                        }
             if (x < 0 || x > getmaxx())
               x = 0;
             if (y < 0 || y > getmaxy())
               y = 0;
                rectangle(x - 1,y - 1,x + 1,y + 1);
                }
        }
SetPage(0);
}


int print(s)
char *s;
{
/* definitions */
static int y;
int CharSize;

/* workings */
CharSize = textheight(s);
if ((y + 0) * CharSize > getmaxy())
        y = 0;
```

```
else
        y++;
outtextxy(0,y * CharSize,s);
moveto(0,(y + 1) * CharSize);
}

int GrWait(message)
int message;
{
if (message)
        outtext("Press a key to continue");
while (! getch())
        ;
}

int SetCopyScreen()
{
int screen1,screen2;
printf("\nScreen to copy\n");
scanf(" %d", screen1);
printf("\nScreen to copy to \n");
scanf(" %d", screen2);
CopyScreen(screen1,screen2);
SetPage(0);
}

int CopyScreen(screen1,screen2)
int screen1,screen2;
{
/* definitions */
int x,y;
void *buffer;
int size;

/* working */
SetPage(screen1);
size =
imagesize(im[screen1].c.left,im[screen1].c.bottom,im[screen1].c.right,im[scre
en1].c.top);
buffer = malloc(size);
if (CheckRoom(buffer,"CopyScreen") == NULL) {
        for (x = im[screen1].c.left;x < im[screen1].c.right;x++)
                for (y = im[screen1].c.bottom;y < im[screen1].c.top;y++)
                        WritePixel(screen2,x,y,ReadPixel(screen1,x,y));
        }
else {

getimage(im[screen1].c.left,im[screen1].c.bottom,im[screen1].c.right,im[scree
n1].c.top,buffer);
        im[screen2].c.left = im[screen1].c.left;
        im[screen2].c.right = im[screen1].c.right;
        im[screen2].c.bottom = im[screen1].c.bottom;
        im[screen2].c.top = im[screen1].c.top;
        strcpy(im[screen2].name,im[screen1].name);
        SetPage(screen2);
```

```
          putimage(im[screen2].c.left,im[screen2].c.bottom,buffer,COPY_PUT);
          free(buffer);
          }
}


AcquireCompressedImage(screen,infile)
int screen;
FileName *infile;
{
/* definitions */
FILE *bit_file_ptr;
 int size;
void *buffer;

/* workings */
size =
imagesize(im[screen].c.left,im[screen].c.bottom,im[screen].c.right,im[screen]
buffer = malloc(size);
if (CheckRoom(buffer,"AcquireCompressedImage") == NULL)
          return(0);
bit_file_ptr = fopen(infile,"rb");
if (bit_file_ptr == NULL)
          return(0);
read(bit_file_ptr,buffer,size);
putimage(im[screen].c.left,im[screen].c.bottom,buffer,COPY_PUT);
free(buffer);
fclose(bit_file_ptr);
return(1);
}

SaveCompressedImage(screen,outfile)
int screen;
FileName *outfile;
{
/* definitions */
FILE *bit_file_ptr;
struct ffblk ffblk;
void *buffer;
 int size;

/* workings */
if (! findfirst(outfile,ffblk,0))
          return(0); /* it already exists */
size =
imagesize(im[screen].c.left,im[screen].c.bottom,im[screen].c.right,im[screen]
buffer = malloc(size);
if (CheckRoom(buffer,"SaveCompressedImage") == NULL)
          return(0);
bit_file_ptr = fopen(outfile,"wb");
if (bit_file_ptr != NULL) {

getimage(im[screen].c.left,im[screen].c.bottom,im[screen].c.right,im[screen].
c.top,buffer);
```

```
        write(bit_file_ptr,buffer,size);
        }
fclose(bit_file_ptr);
free(buffer);
return(1);
}

CopyCorners(from,to)
corners *from,*to;
{
to->left = from->left;
to->right = from->right;
to->bottom = from->bottom;
to->top = from->top;
}

CheckCorners(screen)
int screen;
{
SetPage(0);
if (im[screen].c.left < 0 || im[screen].c.bottom < 0 ) {
        printf("\nNegative screen values encountered");
        return(0);
        }
if (im[screen].c.left >= im[screen].c.right || im[screen].c.bottom >=
im[screen].c.top) {
        printf("\nLeft >= right OR bottom >= top ");
        return(0);
        }
if (im[screen].c.right > getmaxx() || im[screen].c.top > getmaxy()) {
        printf("\nOutside Screen Range");
        return(0);
        }
SetPage(screen);
return(1);
}
```

```
/* PHDHELP.C */
#include <phd.h>
#include <phdext.h>

int SetHelp(menu)
menutype *menu;
{
/* definitions */
int position;
LongStr banner;

/* workings */
position = menu->position; /* store main menu defaults */
strcpy(menu->choice,"    ");
clear(0);
printf("\nEntering Help on %s ,choose menu item for help");
wait();
if (!strncmpi(menu->banner,"main",4))
        MainHelp(menu);
else if (!strncmpi(menu->banner,"conv",4))
        ConvHelp(menu);
else if (!strncmpi(menu->banner,"rast",4))
        RastHelp(menu);
strcpy(menu->choice,"help");
menu->position = position;
wait();
return(1);
}

int MainHelp(menu)
menutype *menu;
{
printf("\nHelp on Main Menu Not Written yet");
}

int ConvHelp(menu)
menutype *menu;
{
printf("\nHelp on Conv Menu Not Written yet");
return(0);
while (strncmpi(menu->choice,"quit",4) ) {
        setgraphmode(0);
        DispMenu(menu);
        }
}

int RastHelp(menu)
menutype *menu;
{
printf("\nHelp on Raster Menu Not Written yet");
}
```

```
/* PHDLUT.C */
#include <phd.h>
#include <phdext.h>


int VgaRGB()
{
/* sets up vga red-green-blue LUT */
/* definitions */
union REGS regs;
int red,green,blue,i;

/* workings */
regs.ax = 0x1010; /* call for VGA pallete */
for (i = 0;i < 256;i++) {
        if (i < 64 || i >= 196) {
                red = i / 4;
                green = i / 4;
                blue = i / 4;
                }
        else  if (i >= 64 & i < 128) {
                red = (96 - i);
                green = (i - 32);
                blue = 0;
                }
        else if (i >= 128 & i < 196) {
                red = 0;
                green = (160 - i);
                blue = (i - 96);
                }

        regs.bx = i; /* colour number */
        regs.dh = red; /* red component */
        regs.cl = blue; /* blue component */
        regs.ch = green; /* green component */
        int86(VIDEO,regs,regs);
        }
}

int VgaMono()
{
/* sets up vga mono LUT */
/* definitions */
union REGS in,out;
int i;

/* workings */
in.ax = 0x1010; /* call for VGA pallete */
for (i = 0;i < 256;i++) {
        in.bx = i; /* colour number */
        in.dh = i / 4; /* red component */
        in.cl = i / 4; /* blue component */
        in.ch = i / 4; /* green component */
        int86(VIDEO,in,out);
        }
```

```
}

/* COLOUR */
int Colour(screen)
int screen;
{
/* workings */
im[screen].palette.colors[0] = EGA_BLACK;
im[screen].palette.colors[1] = EGA_BLUE;
im[screen].palette.colors[2] = EGA_GREEN;
im[screen].palette.colors[3] = EGA_CYAN;
im[screen].palette.colors[4] = EGA_RED;
im[screen].palette.colors[5] = EGA_MAGENTA;
im[screen].palette.colors[6] = EGA_LIGHTGRAY;
im[screen].palette.colors[7] = EGA_BROWN;
im[screen].palette.colors[8] = EGA_DARKGRAY;
im[screen].palette.colors[9] = EGA_LIGHTBLUE;
im[screen].palette.colors[10] = EGA_LIGHTGREEN;
im[screen].palette.colors[11] = EGA_LIGHTCYAN;
im[screen].palette.colors[12] = EGA_LIGHTRED;
im[screen].palette.colors[13] = EGA_LIGHTMAGENTA;
im[screen].palette.colors[14] = EGA_YELLOW;
im[screen].palette.colors[15] = EGA_WHITE;
setallpalette(im[screen].palette);
return(0);
}


/* HOT_BODY *************************************************************
* set LUT to hot body scale                                     *
**********************************************************************/

int hot_body()
{
/*
      Purpose - To set look-up table to hot body scale

      Method - Pluto routines are used to set individual pixel values
      from 0 to maximum value in such a way that the display does from
      hot to cold colours, black is cold, red throughrange and finally
      white for hottest.
*/
printf("\nHot body not written");
return(0);
}


/* SETREDGREENBLUE */
int SetRedGreenBlue()
{
int screen;

/* working */
printf("\nScreen to set to RedGreenBlue\n");
```

```
scanf(" %d", screen);
RedGreenBlue(screen);
}

/* RED_GREEN_BLUE ***********************************************************
* set LUT to red green blue colour scale                                *
***************************************************************************/
int RedGreenBlue(screen)
int screen;
{
/*
        Purpose - Toset a Look-up table so that the red-green-blue colour
        system is employed.

        Method - Low pixel values are set to red with increasing ammounts
        of green added as the value increases toa mid-point of green and then
        blue is added till the highest pixel value is pure blue.

*/

/* workings */
im[screen].palette.colors[0] = EGA_BLACK;
im[screen].palette.colors[1] = EGA_DARKGRAY;
im[screen].palette.colors[2] = EGA_BROWN;
im[screen].palette.colors[3] = EGA_RED;
im[screen].palette.colors[4] = EGA_LIGHTRED;
im[screen].palette.colors[5] = EGA_MAGENTA;
im[screen].palette.colors[6] = EGA_LIGHTMAGENTA;
im[screen].palette.colors[7] = EGA_GREEN;
im[screen].palette.colors[8] = EGA_LIGHTGREEN;
im[screen].palette.colors[9] = EGA_CYAN;
im[screen].palette.colors[10] = EGA_LIGHTCYAN;
im[screen].palette.colors[11] = EGA_BLUE;
im[screen].palette.colors[12] = EGA_LIGHTBLUE;
im[screen].palette.colors[13] = EGA_YELLOW;
im[screen].palette.colors[14] = EGA_LIGHTGRAY;
im[screen].palette.colors[15] = EGA_WHITE;
setallpalette(im[screen].palette);
return(0);
}


/* SETCOLOUR */
int SetColour()
{
int screen;

/* working */
printf("\nScreen to set to default colour\n");
scanf(" %d", screen);
Colour(screen);
}

/* SETMONO */
int SetMono()
```

```
{
int screen;

/* working */
printf("\nScreen to set to mono\n");
scanf(" %d", screen);
Mono(screen);
}


/* MONO  ************************************************************
* sets up display to monochrome                                    *
*******************************************************************/

int Mono(screen)
int screen;
{
/* definitions */
int colour,i;

/* working */
for (i = 0;i < getmaxcolor();i++) {
        if (i < getmaxcolor() / 4)
                im[screen].palette.colors[i] = EGA_BLACK;
        else if (i < getmaxcolor() / 2)
                im[screen].palette.colors[i] = EGA_DARKGRAY;
        else if (i < (3 * getmaxcolor()) / 4)
                im[screen].palette.colors[i] = EGA_LIGHTGRAY;
        else
                im[screen].palette.colors[i] = EGA_WHITE;
      setallpalette(im[screen].palette);
        }
return(0);
}
```

```
/* PHDMASK.C */
#include <phd.h>
#include <phdext.h>

int SetFilterMask()
{
/*
        Purpose - to allow one of several spatial masks to be used to
        filter an image

        Method - If Pluto board used then screens are used to
        tranfer transformed pixels. Original image on one screen is sent
        after filtering to another screen. This allows faster processing
        and portions of a screen may be processed using the square routine.
        If no pluto board then files are used as input and output.
*/

/* definitions */
int choice;
FileName infile,outfile;

/* workings */
printf("\n1. Binomial\n2. Sharpen mask\n3. Blurring mask \n4. Laplacian\n");
scanf(" %d", choice);
printf("File from");
scanf(" %s", infile);
printf("File to");
scanf(" %s", outfile);
FilterMask(infile,outfile,choice);
}


int FilterMask(infile,outfile,filt_type)
int filt_type;
FileName *infile,*outfile;
{
/*
        Purpose - To use a spatial mask on a file or screen and produce
        output on a file or screen

        Method - A matrix is chosen dependant upon the choice variable.
        The elements of that matrix are copied into a work matrix.
        The pixels input from a screen or file are operated upon by this
        matrix, and the size of the matrix is used as a divisor to
        normalise the resulting pixel.
*/


/* definitions */
FILE *InFilePtr,*OutFilePtr;
int buffer_size,x,y,i,j,k;
int buffer[3][XSIZE];
int out_buffer[XSIZE + 1];
int matrix[3][3];
int kersharp[]   = {-1, -1, -1,           /* Kernel for sharpening */
```

```
                -1, 9, -1,
                -1, -1, -1,};

int kerlapla[] = {-1, -1, -1,           /* Kernel for laplacian */
                -1, 8, -1,
                -1, -1, -1,};

int kerblur[] = { 1, 1, 1,
                1, 1, 1,
                1, 1, 1,};

int kerbin[] = { 1, 2, 1,          /* Kernel for binomial */
                2, 4, 2,
                1, 2, 1,};

/* workings */
InFilePtr = fopen(infile,"rb");
file_check(infile,InFilePtr);
OutFilePtr = fopen(outfile,"wb");
file_check(outfile,OutFilePtr);
for (i = 0;i < 3;i++) {
        for (j = 0;j < 3;j++) {
                if (filt_type == 1)
                matrix[i][j] = kerbin[i * 3 + j];
                else if (filt_type == 2)
                        matrix[i][j] = kersharp[i * 3 + j];
                else if (filt_type == 3)
                        matrix[i][j] = kerblur[i * 3 + j];
            else if (filt_type == 4)
                        matrix[i][j] = kerlapla[i * 3 + j];
                else {
                        printf("\nIncorrect option mask routine filter_mask");
                        exit(0);
                        }
                }
        }
buffer_size = 0;
for (i = 0;i < 3;i++) {
        printf("\n");
        for (j = 0;j < 3;j++) {
                buffer_size += matrix[i][j];
                printf("%d ",matrix[i][j]);
                }
        }
printf("\nBuffer size %d",buffer_size);

for (x = 0;x < XSIZE;x++)
        out_buffer[x] = 0;
for (y = 0;y <= YSIZE;y++) {
        for (x = 0;x <= XSIZE;x++)
                buffer[y % 3][x] = getc(InFilePtr);
        for (x = cor.left;x <= cor.right; x++) {
                out_buffer[x + 1] = 0;
                for (j = 0;j < 3;j++)
                        for (k = 0;k < 3;k++)
```

```
                    out_buffer[x + 1] += buffer[(y + j) % 3][x + k] * matrix[k][j];
                out_buffer[x + 1] /= buffer_size;
              }
          for (x = cor.left;x <= cor.right;x++)
                putc(out_buffer[x],OutFilePtr);
          if (y % (YSIZE / 10) == 0)
                printf("\nSpacial mask %f percent finished",
              (float) 100 * (y - cor.bottom) / (cor.top - cor.bottom));
          }
    fclose(InFilePtr);
    fclose(OutFilePtr);
    }
```

```
/* PHDMED.C */
#include <phd.h>
#include <phdext.h>

int SetMedian()
{
/* definitions */
FileName infile,outfile;
FILE *InFilePtr,*OutFilePtr;
struct ffblk ffblk;
int done;
char suffix[5];
int XFiltWidth,YFiltWidth;
int ESPFile,c,compress;

/* workings */
XFiltWidth = 1 + XSIZE / PC_X_SIZE;
YFiltWidth = 1 + YSIZE / PC_Y_SIZE;
printf("\nThis utility works on .ESP files or image files with no");
printf("\nleft, right, bottom or top parameters in file");
printf("\nand converts it into a file with these parameters");
printf("\nOR on images with corners defined in first 8 bytes of file");
printf("\nIt assumes that files with .ESP as suffix are raw images with");
printf("\nNO corners defined at start of file, and that any other suffix");
printf("\nmeans file HAS corners in first 8 bytes of file");
printf("\nSuffix to use to store median filtered file \n");
scanf(" %s", suffix);
printf("\nMedian filter defaults to width %d by height %d",XFiltWidth,YFiltWidth);
if (c = IsGetCharCR()) {
	printf("\nX Filter Width Size\n");
	scanf(" %d", XFiltWidth);
	printf("\nY Filter Height Size\n");
	scanf(" %d", YFiltWidth);
	}
printf("\nCompression will reduce image by %d by %d",XFiltWidth,YFiltWidth);
printf("\nCompress image (RETURN for yes, any other key for no)\n");
while (! (c = MouseKeyGet()))
	;
if (c == CR)
	compress = 1;
else
	compress = 0;
printf("\nResultant image will be in image format, with first 8 bytes corners of image");
while ((c = dir(spath,infile,"Median Filtering")) != ESC & c != CR)
	;
if (c != ESC) {
	done = findfirst(infile,ffblk,0);
	while (!done) {
		ESPFile = CheckSuffix(ffblk.ff_name,".ESP");
		if (! CheckSuffix(ffblk.ff_name,suffix))
			median(compress,suffix,ESPFile,ffblk.ff_name,XFiltWidth,YFiltWidth);
		else {
			printf("\nFile not filtered as %s would be overwritten",ffblk.ff_name);
			DelayOrKeyRead(DELTIME);
			}
```

```
                done = findnext(ffblk);
                }
        }
}


/* MEDIAN ******************************************/
/*
Applies median filter of XFiltWidth by YFiltWidth pixels and
returns the median value. This gives one pixel for each rectangle
pixels entered, cf median() which moves the filter mask over each pixel in
turn. Median2 in contrast moves the mask from one rectangle
to the next saving one
pixel each time. This has the effect of compressing the image by a factor
of XFiltWidth times YFiltWidth as well as smoothing it.
*/

int median(compress,suffix,ESPFile,infile,XFiltWidth,YFiltWidth)
char *suffix;
int compress,ESPFile;
FileName *infile;
int XFiltWidth,YFiltWidth;
{

/* definitions */
int xsize,ysize;
FileName outfile;
int x_screen,y_screen;
int no_members,middle_member;
int i,j,y2,x,y,y1,c;
char *buffer,*med_buffer;
FILE *InFilePtr,*OutFilePtr;
int XInc,YInc;
long int position;

/* workings */

strcpy(outfile,infile);
putsuffix(outfile,suffix);
printf("\nMedian filtering file %s and results in %s",infile,outfile);
no_members = XFiltWidth * YFiltWidth;
middle_member = no_members / 2;
InFilePtr = fopen(infile,"rb");
if (file_check(infile,InFilePtr) == 0)
        return(0);
OutFilePtr = fopen(outfile,"wb");
if (file_check(outfile,OutFilePtr) == 0)
        return(0);
if (! ESPFile) {
        ReadCorners(InFilePtr,cor);
        xsize = cor.right - cor.left;
        ysize = cor.top - cor.bottom;
        }
else    {
        xsize = XSIZE;
        ysize = YSIZE;
        }
    buffer = malloc(sizeof(char) * YFiltWidth * xsize);
    if (CheckRoom(buffer,"median") == NULL)
        return(0);
    med_buffer = malloc(sizeof(char) * YFiltWidth * XFiltWidth);
    if (CheckRoom(med_buffer,"median") == NULL)
        return(0);
    cor.left = cor.bottom = 0;
    if (compress)  {
        cor.right = xsize / XFiltWidth;
        cor.top = ysize / YFiltWidth;
        }
    else   {
        cor.right = xsize;
        cor.top = ysize;
        }
    WriteCorners(OutFilePtr,cor);
    if (compress) {
        YInc = YFiltWidth;
        XInc = XFiltWidth;
        }
    else
        YInc = XInc = 1;
    for (y = 0;y < ysize;y += YInc) {
        if (ESPFile)  {
            position = (long) (sizeof(char) * y);
            position *= (long) xsize;
            }
        else {
            position =  (long) (sizeof(char) * y);
            position *= (long) xsize;
            position += (long) (sizeof(int) * 4);
            }
        if (fseek(InFilePtr,position,SEEK_SET) != 0) {
            printf("\nFile pointer to %s failed",infile);
            return(0);
            }
        for (i = 0;i < YFiltWidth;i++) {
            for (j = 0;j < xsize;j++) {
                c = getc(InFilePtr);
                if (c != EOF)
                    buffer[i * xsize + j] = c;
                else
                    buffer[i * xsize + j] = buffer[(i - 1) * xsize + j];
                }
            }
        for (x = 0;x < xsize;x += XInc) {
            for (i = 0;i < XFiltWidth;i++) {
                for (j = 0;j < YFiltWidth;j++)
                    med_buffer[i * XFiltWidth + j] =
                    buffer[j * xsize + x + i];
                }
            qsort(med_buffer,no_members,sizeof(char),strcmp);
            putc(med_buffer[middle_member],OutFilePtr);
```

```
                }
                printf("*");
        }
    free(buffer);
    free(med_buffer);
    fclose(InFilePtr);
    fclose(OutFilePtr);
    return(1);
    }
```

```
/* PHDMOUSE.C */
#include <phd.h>
#include <phdext.h>

test()
{
int x,y,z,dx,dy;
z = 64;
while (1) {
      printf("x %d y %d z %d",x,y,z);
      GrWait(0);
      if (x == 1)
        x = 32000;
      else
         x = 1;
      if (y == 1)
          y = 32000;
      else y = 1;
      SetMouse(x,y,z);
      PixelDisplay(1);
      }
}

/* INITMOUSE */
int InitMouse()
{
/* definitions */
union REGS in,out;

/* workings */
in.x.ax = 0x3500 + MSYSCALL;
int86(MSDOS,in,out);
in.x.ax = 0;
int86(MSYSCALL,in,out);
if (out.x.ax != 0)
        return(1);
else
        return(0);
}

int ClearMouse()
{
if (mouse)
        while (MouseButton() != 0) /* to clear button */
                ;
}


/* MOUSEBUTTON */
int MouseButton()
{
/* definitions */
union REGS in,out;

/* workings */
```

```
in.h.ah = 0;
in.h.al = 5;
in.h.bl = in.h.bh = 0;
int86(MSYSCALL,in,out);
return(out.h.al);
}

/* MOUSEGET */
int MouseGet(dx,dy)
int *dx,*dy; /* relative movement in x and y respectively */
{
/* definitions */
union REGS in,out;

/* workings */
in.x.ax = 11;
int86(MSYSCALL,in,out);
*dx = out.x.cx;
*dy = out.x.dx;
}



int SetMouse(x,y,z)
int x,y,z;
{
/* definitions */
union REGS in;

/* workings */
in.x.ax = 0x1a;
in.x.bx = x;
in.x.cx = y;
in.x.dx = z;
int86(MSYSCALL,in,in);
}



int GetMouseCh(x,y)
signed int *x,*y;
/* returns a char from mouse, rather in same way as getch() does */
{
/* definitions */
int c = 0;
signed int dx,dy;
int XSen = 20;
int YSen = 10;

/* workings */
while (c == 0) {
        if (! mouse)
                c = MouseKeyGet();
        else {
                if ((c = MouseButton()) == 1)
```

```
                        c = CR;
                else if (c == 2)
                        c = ESC;
                else {
                        MouseGet(dx,dy);
                        *x += dx;*y += dy;
                        if (*x < - XSen)
                                c = CURS_LEFT;
                        else if (*x > XSen)
                                c = CURS_RIGHT;
                        else if (*y > YSen)
                                c = CURS_DOWN;
                        else if (*y < - YSen)
                                c =  CURS_UP;
                        if (abs(*x) > XSen)
                                *x = 0;
                        else if (abs(*y) > YSen)
                                *y = 0;
                        }
                if (! c)
                        c = MouseKeyGet();
                }
        }
return(c);
}

int MouseKeyGet()
{
/* definitions */
union REGS in;
int c = 0;

/* workings */
in.h.al = 0;
in.h.ah = 1;
int86(KEYBOARDCALL,in,in);
if (in.h.al || (in.h.ah > 1)) {
        in.h.ah = 0;
        int86(KEYBOARDCALL,in,in);
        if (in.h.al == 0)
                c = in.h.ah;
        else
                c = in.h.al;
        }
return(c);
}
```

```
/* PHDNWORK.C */
#include <phd.h>
#include <phdnwork.h>

main()
{
int stat;
strcpy(ScratchFile,"qqq.qqq"); /* will be used as temporary file only */
strcpy(spath,"*.*");
mouse = InitMouse();
phdnwork();
strcpy(spath,"    cenis");
chdir(spath);
printf("\nPROGRAM ENDED");
}


/* PHDNWORK ****************************************************************
* main calling program for nuclear medicine neural network          *
* image processing system.                                          *
* Copyright D.M. Anthony for Warwick University.                    *
****************************************************************************/

/*     Purpose - to give menu and sub-menus of nuclear imaging neural network
       program and proceed to subroutines.

     Method - Graphics station is opened and then an infinite for-loop
       displays the main menu, conditional if statements determine which
       sub-routine to execute.
*/


int phdnwork()
{
/* definitions */
int stat;
ShStr s;
int i;
menutype menu = {
       19,
       0,
       "MAIN MENU",
       "256 colours file display",
       "Alternate screen",
       "Acquire image",
       "Clear a screen",
       "Colour",
       "Convert Files to/from NWORKS, PDP, MATLAB ",
       "Copy Screen",
       "Corners of image show (left,right,bottom,top)",
       "Graphics Mode Change",
       "HELP",
       "Manipulate image directly (cut,negative,subtract etc)",
       "Median filter /median filter compression",
        "Monochrome",
        "Raster Scan (or any regular segmentation of image",
        "Red Green Blue Display",
        "Search Criteria for Files Change",
        "Segment Image",
        "Sobel Filter",
        "Standardise a file or files",
        "   ",
        };

/* workings */
OpenGraphics();
for (i = 0;i < 4;i++)
        RedGreenBlue(i);
SetPage(0);
clearviewport();
outtextxy(0,0,"Main Nuclear Medicine Neural Network Image Processing Menu");
outtextxy(0,10,"Copyright D.M. Anthony for Warwick University 1988");
#ifdef EVOR
        outtextxy(0,20,"Version for 512 by 512 (original) images");
#else
        outtextxy(0,20,"Version for 768 by 288 (original digit) images");
#endif
if (mouse)
        outtextxy(0,30,"Mouse installed, use mouse OR cursor keys in menus and routines");
else
        outtextxy(0,30,"Mouse not installed, use cursor keys in menus and routines");
outtextxy(0,40,"ESC key returns from any menu to previous menu, RETURN to choose item");
outtextxy(0,50,"Press a key to continue");
while (! GetMouseCh(i,i))
        ;
cor.left = 0;cor.right = 8;cor.bottom = 0;cor.top = 8;
/* setup to some initial value */
while (strncmpi(menu.choice,"quit",4)) {
        DispMenu(menu);
        if (!strncmpi(menu.choice,"256 ",4))
                SetAcquire(1);
        else if (!strncmpi(menu.choice,"alte",4))
                SetAlternateScreen();
        else if (!strncmpi(menu.choice,"acqu",4))
                SetAcquire(0);
        else if (!strncmpi(menu.choice,"clea",4))
                SetClear();
        else if (!strncmpi(menu.choice,"colo",4))
                SetColour();
        else if (!strncmpi(menu.choice,"conv",4))
                SetConvert();
        else  if (!strncmpi(menu.choice,"copy",4))
                SetCopyScreen();
        else if (!strncmpi(menu.choice,"com",4))
                SetAcquire(2);
        else if (!strncmpi(menu.choice,"grap",4))
                SetGraphMode();
        else if (!strncmpi(menu.choice,"help",4))
                SetHelp(menu);
        else  if (!strncmpi(menu.choice,"mani",4))
                ManipulateMenu();
        else  if (!strncmpi(menu.choice,"medi",4))
                SetMedian();
        else  if (!strncmpi(menu.choice,"mono",4))
                SetMono();
        else  if (!strncmpi(menu.choice,"mous",4))
                test();
        else if (!strncmpi(menu.choice,"rast",4))
                raster();
        else  if (!strncmpi(menu.choice,"red ",4))
                SetRedGreenBlue();
        else  if (!strncmpi(menu.choice,"sear",4)) {
                printf("\nOld Criteria %s \nNew Search Criteria\n",spath);
                scanf(" %s", spath);
                }
        else  if (!strncmpi(menu.choice,"segm",4))
                SetSegment();
        else  if (!strncmpi(menu.choice,"sobe",4))
                SetSobel();
        else if (!strncmpi(menu.choice,"stan",4))
                SetStan();
        else if (!strncmpi(menu.choice,"turn",4))
                SetTurnRound();
        }
CloseGraphics();
return(1);
}


/*      Purpose - to display sub-menu for look up tables of pixel values

        Method - same as msc.c
*/



int ManipulateMenu()
{
/*      Purpose - to display sub-menu for performing simple direct operations
        on image

        Method - same as msc.c
*/

/* definitions */
menutype menu = {
        13,
        0,
        "MANIPULATE MENU",
        "Draw Rectangle on Screen",
        "Expand image",
        "HELP",
        "Left to right image",
        "Pixel value display",
                "Save rectangle of image",
                "Subtract rectangle",
                "Take one file from another (Background removal e.g.)",
                "Threshold image",
                "Turn image 180%",
                "X axis Reflection",
                "Y axis Reflection",
                "Upside down image",
                " "," "," "," "," "," "," "," ",
                " "};

        /* working */
                while (strncmpi(menu.choice,"quit",4) ) {
                        DispMenu(menu);
                        if (!strncmpi(menu.choice,"draw",4))
                                SetDrawRect();
                        else if (!strncmpi(menu.choice,"expa",4))
                                SetExpand();
                        else if (!strncmpi(menu.choice,"help",4))
                                SetHelp(menu);
                        else if (!strncmpi(menu.choice,"left",4))
                                LeftToRight();
                        else if (!strncmpi(menu.choice,"pixe",4))
                                SetPixelDisplay();
                        else if (!strncmpi(menu.choice,"save",4))
                                SetSaveRect();
                        else if (!strncmpi(menu.choice,"subt",4))
                                SetSubtractRect();
                        else if (!strncmpi(menu.choice,"take",4))
                                SetSubFile();
                        else if (!strncmpi(menu.choice,"thre",4))
                                SetThreshold();
                        else if (!strncmpi(menu.choice,"turn",4))
                                SetTurnRound();
                        else  if (!strncmpi(menu.choice,"x ax",4))
                                SetXReflect();
                        else  if (!strncmpi(menu.choice,"y ax",4))
                                SetYReflect();
                        else  if (!strncmpi(menu.choice,"upsi",4))
                                UpsideDown();
                        }
        }
```

```
/* RASTER.C */
#include <phd.h>
#include <phdext.h>

int raster()
{
/* definitions */
int c,x,y;
int x_start,x_end,y_start,y_end,x_size,y_size,x_disp,y_disp;
FileName infile;
struct ffblk ffblk;
int colour,done;
int Keyboard = 1;
int WholeImageRasterScan = 1;
corners raster;

menutype menu = {
        9,
        0,
        "RASTER SCANNING of images",
        "Choose Image Files for Scanning",
        "Define Raster Scan Size (size of individual raster scans)",
        "Displacements Define (of consecutive rasters in X,Y directions)",
        "HELP",
        "Keyboard Entry for Scan Size (default ON)",
        "Random Raster Scans",
        "Show Defaults fo Raster Scans",
        "Raster Scan Images",
        "Whole Image Scan Set (default ON)",
        " ",
        " "," "," "," "," "," "," "," "," "," ",
        "  ",
        };
unsigned int screen = 1;

/* working */
SetPage(0);
strcpy(menu.choice,"   "); /* should not be necessary, just to make sure */
while (strncmpi(menu.choice,"quit",4) ) {
        setgraphmode(0);
        DispMenu(menu);
        clear(0);
        if (!strncmpi(menu.choice,"help",4))
                SetHelp(menu);
        else if (!strncmpi(menu.choice,"choo",4)) {
                while ((c = dir(spath,infile,
                "Raster Scan File Selection")) != ESC & c != CR)
                        ;
                /* default for when WholeImage */
                }
        else if (!strncmpi(menu.choice,"defi",4))  {
                if (!Keyboard) {
                        DrawRect(1,cor);
                        x_size = cor.right - cor.left;
                        y_size = cor.top - cor.bottom;
                        }
                else {
                        printf("\nWidth of raster scans\n");
                        scanf(" %d", x_size);
                        printf("\nHeight of raster scans\n");
                        scanf(" %d", y_size);
                        }
                }
        else if (!strncmpi(menu.choice,"disp",4)) {
                printf("\nDisplacement in x direction between images ");
                printf("\n(1 for raster scan) ");
                scanf(" %d", x_disp);
                printf("\nDisplacement in y direction between images");
                printf("\n(1 for raster scan) ");
                scanf(" %d", y_disp);
                printf("\nWhole Image Scan (default yes)?");
                }
        else if (!strncmpi(menu.choice,"keyb",4)) {
                Keyboard = !Keyboard;
                if (Keyboard)
                        printf("Entry from Keyboard for Scan Size");
                else
                        printf("\nEntry from Mouse for Scan Size");
                wait();
                }
        else if (!strncmpi(menu.choice,"ra",2)) {
                if (!strncmpi(menu.choice,"rand",4)) {
                        printf("\nHow many random rasters required\n");
                        scanf(" %d", y_disp);
                        x_disp = 0;
                        }
                done = findfirst(infile,ffblk,0);
                while (!done) {
                        acquire(screen,ffblk.ff_name);
                        CopyCorners(im[1],c,raster);
                        if (! WholeImageRasterScan)
                                DrawRect(1,raster);
                        save_rasters(raster,x_size,y_size,x_disp,y_disp,screen);
                        done = findnext(ffblk);
                        }
                }
        else if (!strncmpi(menu.choice,"show",4)) {
                printf("\nDefault Values are:\n");
                printf("\nStart at x %d y %d End at %d %d",x_start,y_start,x_end,y_end);
                printf("\nSize of rasters in x direction %d y direction %d",x_size,y_size);
                printf("\nDisplacement between rasters in x direction %d y direction %d",x_disp,y_disp);
                printf("\nUsing file(s) %s",infile);
                wait();
                }
        else if (!strncmpi(menu.choice,"whol",4)) {
                WholeImageRasterScan = !WholeImageRasterScan;
                if (WholeImageRasterScan)
                        printf("\nWhole Image Scan");
                else
                        printf("\nPartial Image Scan");
                wait();
                }
        }
}


int save_rasters(r,x_size,y_size,x_disp,y_disp,screen)
corners *r;
int x_size,y_size,x_disp,y_disp;
unsigned int screen;
{
/* definitions */
FileName outfile;
int c,i,j,k,x,y;
long int MaxFiles;
int temp;
char s1[4],s2[4];
long int position;
int RandomFiles;
corners segment; /* used as corners for each raster segment to save to file */
corners *CornerBuffer;
int GotCornersAlready = 0;
int radix = 36;

/* workings */
MaxFiles = (long) radix * (long) radix * (long) radix;
SetPage(0);
strcpy(outfile,im[screen].name);
i = 0;
/* r are corners of file to raster */
/* x_size and y_size are width and height of raster scans */
/* x_disp and y_disp are displacements made after each raster scan saved */
/* but if x_disp = 0 (silly) then RandomFiles is true and MaxFiles = y_disp */
RandomFiles = !x_disp;
if (RandomFiles) {
        MaxFiles = y_disp;
        y_disp = 0;
        CornerBuffer = malloc(sizeof(corners) * MaxFiles);
        }
if (RandomFiles & !CheckRoom(CornerBuffer,"save_rasters"))
        return(0);
randomize();
for (y = r->bottom;y <= r->top - y_size;y += y_disp) {
        for (x = r->left;x <= r->right - x_size;x += x_disp) {
                if (RandomFiles) {
                        x = r->left + random(r->right - r->left - x_size);
                        y = r->bottom + random(r->top - r->bottom  - y_size);
                        }
                if (i >= MaxFiles) {
                        printf("\nMaximum of  %d files exceeded",MaxFiles);
                        x = r->right;
                        y = r->top;
                        }
                else {
                        itoa(i,s1,radix);
                        if (i < radix)
                                strcpy(s2,"00");
                        else if (i < radix * radix)
                                strcpy(s2,"0");
                        strcat(s2,s1);
                        putsuffix(outfile,s2);
                        printf("\nSaving file %s",outfile);
                        segment.left = x;segment.right = x + x_size;
                        segment.bottom = y;segment.top = y + y_size;
                        for (j = 0;j < i;j++) {
                                GotCornersAlready = CompareCorners(segment,CornerBuffer[j]);
                                if (GotCornersAlready)
                                        j = i;
                                }
                        if (!RandomFiles || !GotCornersAlready) {
                                SaveRect(screen,segment,outfile);
                                i++;
                                }
                        if (RandomFiles) {
                                x = r->left;
                                y = r->bottom;
                                }
                        }
                }
        }
SetPage(0);
if (RandomFiles)
        free(CornerBuffer);
return(1);
}
```

```
/* PHDRECT.C */
#include <phd.h>
#include <phdext.h>

int SetDrawRect()
{
/* definitions */
int screen;

/* workings */
printf("\nScreen\n");
scanf(" %d", screen);
DrawRect(screen,im[screen].c);
}



/* DRAW_RECT ***************************************************************
* puts a rectangle on the screen, allows translations and scaling of    *
* to give boundaries for subsequent operations or to mark an            *
* area                                                    *
*************************************************************************/

int DrawRect(screen,c)
int screen;
corners *c;
{
/*
        Purpose - To draw a rectangle or square and keep the original
        image under the rectangle for screen re-freshing. Expansion
        of the square, retraction and movement of the square are all allowed.

        Method -
        Rectangle is drawn using function rectangle and XORing pixels.
        A character input determines whether one of several options
        is accomplished.
*/

/* definitions */
int colour,loop;
int key,temp;
int x_increment,y_increment;
int width,height;
int *dx,*dy;
int button;


/* workings */

SetPage(screen);
x_increment = 1;
y_increment = 1;
setwritemode(XOR_PUT);
if (c->left >= c->right || c->left < 0)
```

```
        c->left = 0;
if (c->right <= c->left || c->right >= PC_X_SIZE)
        c->right = PC_X_SIZE - 1;
if (c->bottom >= c->top || c->bottom < 0)
        c->bottom = 0;
if (c->top <= c->bottom || c->top >= PC_Y_SIZE)
        c->top = PC_Y_SIZE - 1;
rectangle(c->left,c->bottom,c->right,c->top);
for (loop = 1;loop <= 2;loop++) {
        if (mouse)
                ClearMouse();
        key = 0; /* or anything but ESC */
        while (key != CR & key != ESC) {
                key = GetMouseCh(dx,dy); /* dx,dy not used */
                if (key == CURS_LEFT || key == CURS_RIGHT ||
                key == CURS_UP || key == CURS_DOWN ||
                (key >= '0' & key <= '9')) {
                        CopyCorners(c,cor);
                        key = GetIncForRect(x_increment,y_increment,c,loop,key);
                        rectangle(cor.left,cor.bottom,cor.right,cor.top);
                        rectangle(c->left,c->bottom,c->right,c->top);
                        } /* end if Cursor keys */
                if (key == 'l') {
                        SetPage(0);
                        printf("\nLeft %d Right %d Bottom %d Top %d",c->left,c->right,c->bottom,c->top);
                        wait();
                        SetPage(screen);
                        break;
                        }
                } /* end while */
        } /* end for loop = 1 to 2 */
rectangle(c->left,c->bottom,c->right,c->top);
return(0);
}


int GetIncForRect(x_increment,y_increment,c,loop,key)
corners *c;
int *x_increment,*y_increment;
int loop;
int key;
{
if ((key > '0') & (key <= '9')) {
        *y_increment = (key - '0');
        *x_increment = (*y_increment * getmaxx()) / getmaxy();
        }
else {
        switch(key) {
                case CURS_LEFT :
                        if (loop == 1)
                                c->left -= *x_increment;
                        else if (loop == 2)
                                c->right -= *x_increment;
                        break;
                case CURS_RIGHT :
                    if (loop == 1)
                         c->left += *x_increment;
                    else if (loop == 2)
                         c->right += *x_increment;
                    break;
               case CURS_UP :
                    if (loop == 1)
                         c->top -= *y_increment;
                    else if (loop == 2)
                         c->bottom -= *y_increment;
                    break;
               case CURS_DOWN :
                    if (loop == 1)
                         c->top += *y_increment;
                    else if (loop == 2)
                         c->bottom += *y_increment;
                    break;
               case CURS_PLUS :
                    c->top += *y_increment;
                    c->bottom -= *y_increment;
                    c->left -= *x_increment;
                    c->right += *x_increment;
                    break;
               case CURS_MINUS :
                    c->top -= *y_increment;
                    c->bottom += *y_increment;
                    c->left += *x_increment;
                    c->right -= *x_increment;
                    break;
               } /* end switch */
          if (c->bottom >= c->top) {
               c->bottom -= *y_increment;
               c->top += *y_increment;
               }
           if (c->left >= c->right) {
               c->left -= *x_increment;
               c->right += *x_increment;
               }
           if (c->left < 0)
               c->left = cor.left;
           if (c->right >= getmaxx())
               c->right = getmaxx();

           if (c->top > getmaxy())
               c->top = getmaxy();
           if (c->bottom < 0)
               c->bottom = 0;

           }
     return(key);
     }

     int SetSubtractRect()
     {
     /* definitions */
```

```
corners from,to,current;
FileName infile,sfile;
FileName pfile,vfile;
int XDiff,YDiff,screen;

/* workings */
clear(0);
SetPage(0);
printf("\nThis routine assumes thresholding has been done, or it assumes");
printf("\ndefault theshold values");
printf("\nName for images, give root only, images will be :\n");
printf("\n.u (for s(U)b), .p (perfusion) .v (ventilation)\n");
scanf(" %s", infile);
strcpy(vfile,infile);strcpy(pfile,infile);strcpy(sfile,infile);
putsuffix(vfile,"v");putsuffix(pfile,"p");putsuffix(sfile,"u");
printf("\nScreen to subtract rectangle from \n");
scanf(" %d", screen);
printf("\nYou will be asked to put a rectangle around each lung for reference");
printf("\nFirst outline ventilation image");
printf("\nThen the perfusion image which will be moved to cover ventilation");
wait();
CopyCorners(cor,to);
DrawRect(screen,to);
SaveRect(screen,to,vfile);
rectangle(to.left,to.bottom,to.right,to.top);
CopyCorners(to,from);
DrawRect(screen,from);
CopyCorners(from,current);
SaveRect(screen,from,pfile);
if (MoveRect(screen,current,from) == 0)
      return(0);
rectangle(to.left,to.bottom,to.right,to.top);
XDiff = current.left - from.left;YDiff = current.bottom - from.bottom;
if (SubtractRect(pfile,vfile,sfile,XDiff,YDiff) == 0)
      return(0);
return(1);
}

/* MOVERECT */
int MoveRect(screen,current,from)
int screen;
corners *current,*from;
{
/* definitions */
void *buffer;
int size;
signed int x_increment = 1;
signed int y_increment = 1;
int *dx,*dy;
int c,temp;
int button;
int MinScore;

/* working */
MinScore = getmaxy();
```

```
SetPage(screen);
size = imagesize(current->left,current->bottom,current->right,current->top);
buffer = malloc(size);
if (CheckRoom(buffer,"MoveRect") == NULL) {
        printf("\nInsufficient room for size of rectangle chosen");
        printf("\nYou will need to return and choose a smaller rectangle");
        GrWait(1);
        return(0);
        }
c = 0; /* or anything but ESC */
getimage(current->left,current->bottom,current->right,current->top,buffer);
putimage(current->left,current->bottom,buffer,XOR_PUT);
if (mouse)
        ClearMouse();
while (c != ESC) {
        c = GetMouseCh(dx,dy); /* dx,dy not used */
        if ((c > '0') & (c <= '9')) {
                y_increment = (c - '0');
                x_increment = (y_increment * getmaxx() / getmaxy();
                }
        else if ((c == CURS_LEFT) || (c == CURS_RIGHT) ||
        (c == CURS_DOWN) || (c == CURS_UP) || (c == CR)) {
                putimage(current->left,current->bottom,buffer,XOR_PUT);
                switch(c) {
                        case CURS_LEFT :
                                current->left -= x_increment;
                                current->right -= x_increment;
                                break;
                        case CURS_RIGHT :
                                current->left += x_increment;
                                current->right += x_increment;
                                break;
                        case CURS_UP :
                                current->top -= y_increment;
                                current->bottom -= y_increment;
                                break;
                        case CURS_DOWN :
                                current->top += y_increment;
                                current->bottom += y_increment;
                                break;
                        }
                if (c == CR)
                        ShowSubtractParameter(screen,MinScore,from,current);
                if (current->left < 0)
                        current->left += x_increment;
                if (current->right >= PC_X_SIZE)
                        current->right -= x_increment;
                if (current->left > current->right) {
                        temp = current->left;
                        current->left = current->right;
                        current->right = temp;
                        }
                if (current->top > PC_Y_SIZE)
                        current->top -= y_increment;
                if (current->bottom < 0)
```

```
                    current->bottom += y_increment;
                if (current->bottom > current->top) {
                    temp = current->bottom;
                    current->bottom = current->top;
                    current->top = temp;
                    }
                putimage(current->left,current->bottom,buffer,XOR_PUT);
                } /* end if c = .. etc */
        } /* end while */
putimage(current->left,current->bottom,buffer,XOR_PUT);
free(buffer);
return(1);
}


/* SUBTRACTRECT */
int SubtractRect(file1,file2,file3,XDiff,YDiff)
char *file1,*file2,*file3;
int XDiff,YDiff;
{
/* definitions */
FILE *in_ptr_1,*in_ptr_2,*OutFilePtr;
int x,y;
int c;
corners f1,f2,f3;
int temp,temp1,temp2,temp3;

/* working */
/*
File 2 is subtracted from file 1, with an offset of XDiff and YDiff
File 3 is created such that it contains both files in its total area, and
the offset area. This allows fields that overlap incompletely to be
created.
*/

in_ptr_1 = fopen(file1,"rb");
if (file_check(file1,in_ptr_1) == 0)
        return(0);
in_ptr_2 = fopen(file2,"rb");
if (file_check(file2,in_ptr_2) == 0)
        return(0);
OutFilePtr = fopen(file3,"wb");
if (file_check(file3,OutFilePtr) == 0)
        return(0);
ReadCorners(in_ptr_1,f1);
ReadCorners(in_ptr_2,f2);
f1.bottom += YDiff;
f1.top += YDiff;
f1.left += XDiff;
f1.right += XDiff;
f3.left = min(f1.left,f2.left);
f3.right = max(f2.right,f2.right);
f3.bottom = min(f1.bottom,f2.bottom);
f3.top = max(f1.top,f2.top);
f3.right = f3.right - f3.left;
```

```
f3.top = f3.top - f3.bottom;
f3.bottom = 0;
WriteCorners(OutFilePtr,f3);
for (y = f3.bottom;y < f3.top;y++) {
        for (x = f3.left;x < f3.right;x++) {
                if (y >= f1.bottom & y < f1.top & x >= f1.left & x < f1.right)
                        temp1 = getc(in_ptr_1);
                else
                        temp1 = 0;
                if (y >= f2.bottom & y < f2.top & x >= f2.left & x < f2.right)
                        temp2 = getc(in_ptr_2);
                else
                        temp2 = 0;
                temp3 = temp1 - temp2;
                temp3 += FILE_BYTE_SIZE;
                temp3 /= 2;
                putc(temp3,OutFilePtr);
                }
        }
fclose(in_ptr_1);
fclose(in_ptr_2);
fclose(OutFilePtr);
return(1);
}

int ShowSubtractParameter(screen,MinScore,from,current)
int screen;
int *MinScore;
corners *from,*current;
{

/* For given rectangle with corners at left,right,bottom,top, works out
normalised pixel score of image, by subtraction of same size
rectangle with top (of screen, bottom y value) left corner at
from->left,from->bottom */

/* definitions */
int x,y;
int XDiff,YDiff;
long int score = 0L;
int maxx,maxy;
int temp;
int rect[] = {0,0,0,0,0,0,0,0};

/* working */
maxx = getmaxx();
maxy = getmaxy();
XDiff = from->left - current->left;
YDiff = from->bottom - current->bottom;
for (y = current->bottom;y < current->top;y++) {
        for (x = current->left;x < current->right;x++) {
                if ((getpixel(x,y) >= im[screen].LowThreshold
                & getpixel(x,y) <= im[screen].HighThreshold)
                || (getpixel(x + XDiff,y + YDiff) >= im[screen].LowThreshold
                & getpixel(x + XDiff,y + YDiff) <= im[screen].HighThreshold)) {
                        temp = abs(getpixel(x,y) - getpixel(x + XDiff,y + YDiff));
                        score += (long) temp;
                        }
                }
        }
score *= ((long) maxy / (long) (1 + im[screen].HighThreshold - im[screen].LowThreshold));
/* divide to normalise colour range, but times maxy to normalise for height */
score /= (long) ((current->top - current->bottom) * (current->right - current->left));
 /* Now normalised to between 0 and max height */
rect[0] = rect[6] = maxx - 10;
rect[2] = rect[4] = maxx;
rect[7] = rect[5] = maxy;
setfillstyle(SOLID_FILL,im[screen].LowThreshold);
fillpoly(sizeof(rect) / (2 * sizeof(int)),rect);
if (score <= *MinScore) {
        *MinScore = score;
        setfillstyle(SOLID_FILL,im[screen].LowThreshold);
        }
else
        setfillstyle(EMPTY_FILL,im[screen].LowThreshold);
fillpoly(sizeof(rect) / (2 * sizeof(int)),rect);
rect[1] = rect[3] = (int) score;
setfillstyle(SOLID_FILL,im[screen].HighThreshold);
fillpoly(sizeof(rect) / (2 * sizeof(int)),rect);
}

int SetSaveRect()
{
/* definitions */
int screen;
FileName outfile;
int XDiff,YDiff;

/* working */
SetPage(0);
printf("\nThis routine takes a screen, and saves a rectangle");
printf("\nwhere the pixel values are set to 0 (below low threshold) ");
printf("\nmax colour (Above threshold) or the actual pixel value");
printf("\nThresholds will be set by default to 0 and Max Colour ");
printf("\nUnless thresholds set specifically");
printf("\nScreen to take rectangle from\n");
scanf(" %d", screen);
printf("\nFile to store rectangle in\n");
scanf(" %s", outfile);
DrawRect(screen,im[screen].c);
SaveRect(screen,im[screen].c,outfile);
}

int SaveRect(screen,rect,outfile)
int screen;
corners *rect;
FileName *outfile;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
```

```
int x,y;
int temp;
corners f;
long int position;
int LowThreshold,HighThreshold;
int MaxColour;

/* working */
InFilePtr = fopen(im[screen].name,"rb");
OutFilePtr = fopen(outfile,"wb");
if (! file_check(im[screen].name,InFilePtr))
        return(0);
if (! file_check(outfile,OutFilePtr))
        return(0);
MaxColour = getmaxcolor();
LowThreshold = (im[screen].LowThreshold * FILE_BYTE_SIZE)/ MaxColour;
HighThreshold = (im[screen].HighThreshold * FILE_BYTE_SIZE)/ MaxColour;
WriteCorners(OutFilePtr,rect);
ReadCorners(InFilePtr,f);
for (y = rect->bottom;y < rect->top;y++) {
        position = 4 * sizeof(int) + sizeof(char) *
        ((y - f.bottom) * (f.right - f.left) + rect->left - f.left);
        fseek(InFilePtr,position,SEEK_SET);
        for (x = rect->left;x < rect->right;x++) {
                temp = getc(InFilePtr);
                if ((temp == EOF) || (temp < LowThreshold))
                        temp = 0;
                else if (temp > HighThreshold)
                        temp = 0;
                putc(temp,OutFilePtr);
                }
        }
fclose(InFilePtr);
fclose(OutFilePtr);
}
```

```
/* PHDSEG.C */
#include <phd.h>
#include <phdext.h>

/* PHDSEG.T ****************************/
/*
One image file is loaded for transfer to NWORKS.
The areas grown are saved
to disk, after combining overlapping areas. These files may then be
standardised for NWORKS entry.
*/



/* SET_SEGMENT */
int SetSegment()
{
/* definitions */
FileName infile;
struct ffblk ffblk;
int done;

/* working */

printf("\nFile to segment (Wildcard choice allowed, ");
printf("\nin which case several files may be segmented)\n");
scanf(" %s", infile);
done = findfirst(infile,ffblk,0);
while (!done) {
        segment(ffblk.ff_name);
        done = findnext(ffblk);
        }
SetPage(0);
}

/* SEGMENT */
segment(infile)
FileName *infile;
{
/*
PSEUDO-CODE
Begin

        Send each pair of centres to grow_region. This routine finds
        connected points and returns via a pointer the left, right, bottom
        and top of each connected image segment

        The images are combined if they overlap.

        The images are saved to disk
End
*/

/* definitions */
int file_no;
int i,x,y;
FILE *OutFilePtr;

/* workings */
printf("\nSegment  program started");
printf("\nAbout to segment");
grow_region(infile,OutFilePtr);
fclose(OutFilePtr);
}




/* GROW_REGION */
int grow_region(infile,OutFilePtr)
FILE *OutFilePtr;
FileName *infile;
{
/*
PSEUDO-CODE
Algorithm from K.R. Castleman "Digital Image Processing", 1979, Prentice Hall,
p 317-319
Begin
        Segment Number = 1
        For each line
                First_Seg_Val = 0;
                First = True (means that whatever segment found
                above current one connected to it is first)
                For each pixel on line
                        If within threshold
                                If pixel to left within threshold
                                        Allocate left segment
                                        to current pixel segment
                                Else
                                        Allocate left segment + 1 to current
                                        pixel
                                If pixel above within threshold
                                        If first or above = First_seg_val
                                                Put Segment value into
                                                variable First_Seg_Val
                                                Make current pixel in same
                                                segment as above segment
                                                Make pixels to left of current
                                                pixel same as above segment
                                                until a pixel found not
                                                in threshold
                                        Else
                                                Do LinkSegmentAbove
                                                (in effect makes above segment
                                                same as current segment)
                                Else (Pixel above not in segment)
                        Else (not within threshold)
                                Allocate no segment
                Next column
        Next Row
End
*/
/* definitions */
int LColour,RColour;
FileName sfile,tfile;
char temp[4];
int colour;
int c,x,y,i,j;
int FirstAboveSeg,SegNo;
int first;
corners pic_edges[MAX_SEGMENTS];


/* working */
acquire(1,infile);
cor.left = im[1].c.left;
cor.right = im[1].c.right;
cor.bottom = im[1].c.bottom;
cor.top = im[1].c.top;
SetPage(0);
printf("\nChoose area to be segmented");
ClipScreen(1);
SetPage(1);
for (i = 0;i < MAX_SEGMENTS;i++) {
        pic_edges[i].left = PC_X_SIZE;
        pic_edges[i].bottom = PC_Y_SIZE;
        pic_edges[i].right = pic_edges[i].top = 0;
        }
SegNo = 0;
SetPage(0);
printf("\nSegmentation started");
clear(2);
clear(3);
GetThreshold(1);
SetPage(1);
for (x = cor.left;x < cor.right;x++)
        for (y = cor.bottom;y < cor.top;y++)
                if (ReadPixel(1,x,y) > im[1].HighThreshold)
                        WritePixel(1,x,y,PIXEL_SIZE - 1);
                else if (ReadPixel(1,x,y) < im[1].LowThreshold)
                        WritePixel(1,x,y,0);
SetPage(2);
for (y = cor.bottom;y < cor.top;y++) {
        for (x = cor.left;x < cor.right;x++) {
                if (ReadPixel(1,x,y) >= im[1].LowThreshold &
                ReadPixel(1,x,y) <= im[1].HighThreshold)
                        Link1Pixels(x,y,SegNo);
                } /* end x loop */
        } /* end for y loop */
for (y = cor.bottom;y < cor.top - 1;y++) {
        for (x = cor.left;x < cor.right;x++) {
                if (ReadPixel(1,x,y) >= im[1].LowThreshold &
                ReadPixel(1,x,y) <= im[1].HighThreshold)
                        Link2Pixels(x,y);
                } /* end x loop */
        } /* end for y loop */
for (y = cor.bottom;y < cor.top - 1;y++) {
        for (x = cor.left;x < cor.right;x++) {
                SegNo = ReadPixel(2,x,y) + ReadPixel(3,x,y) * PIXEL_SIZE;
                if (pic_edges[SegNo].left > x)
                        pic_edges[SegNo].left = x;
                if (pic_edges[SegNo].right < x)
                        pic_edges[SegNo].right = x + 1;
                if (pic_edges[SegNo].bottom > y)
                        pic_edges[SegNo].bottom = y;
                if (pic_edges[SegNo].top < y)
                        pic_edges[SegNo].top = y + 1;
                } /* end x loop */
        } /* end for y loop */
j = 1;
for (i = 1;i <= MAX_SEGMENTS;i++) {
        cor.left = pic_edges[i].left;
        cor.right = pic_edges[i].right;
        cor.bottom = pic_edges[i].bottom;
        cor.top = pic_edges[i].top;
        if (cor.left >= 0 & cor.right <= PC_X_SIZE & cor.bottom >= 0
        & cor.top < PC_Y_SIZE & cor.right > cor.left + 5 & cor.top > cor.bottom + 5) {
        /* ignore tiny segments less than 5 pixels wide */
                strcpy(tfile,infile);
                itoa(j,temp,10);
                strcat(tfile,temp);
        /*      cor.left = pic_edges[i].left;
                cor.right = pic_edges[i].right;
                cor.bottom = pic_edges[i].bottom;
                cor.top = pic_edges[i].top;
                DrawRect(1); */
                save_seg(infile,tfile,pic_edges[i],im[1].LowThreshold,im[1].HighThreshold);
                j++;
                }
        }
}


int Link1Pixels(x,y,SegNo)
int x,y;
int *SegNo;
{
int tx;
int SegL,SegA,SegAL,SegAR,Seg;

/* working */
SegA = ReadPixel(2,x,y - 1) + ReadPixel(3,x,y - 1) * PIXEL_SIZE;
if (x > cor.left)
        SegL = ReadPixel(2,x - 1,y) + ReadPixel(3,x - 1,y) * PIXEL_SIZE;
else
        SegL = 0;
if (SegA > 0) {
        WritePixel(2,x,y,ReadPixel(2,x,y - 1));
        WritePixel(3,x,y,ReadPixel(3,x,y - 1));
        tx = x - 1;
        while (tx >= cor.left & ReadPixel(2,tx,y) > 0 ) {
              WritePixel(2,tx,y,ReadPixel(2,x,y));
              WritePixel(3,tx,y,ReadPixel(3,x,y));
              tx--;
              } /* end while loop */
        } /* end  if current not same as above */
else if (SegL > 0) {
        WritePixel(2,x,y,ReadPixel(2,x - 1,y));
        WritePixel(3,x,y,ReadPixel(3,x - 1,y));
        }
else {
        (*SegNo)++;
        if ((*SegNo) % PIXEL_SIZE == 0)
              (*SegNo)++;
        WritePixel(2,x,y,(*SegNo) % PIXEL_SIZE);
        WritePixel(3,x,y,(*SegNo) / PIXEL_SIZE);
        }
}

int Link2Pixels(x,y)
int x,y;
{
/* definitions */
int SegB,Seg,SegC,tx,ty;

/* working */
Seg = ReadPixel(2,x,y) + ReadPixel(3,x,y) * PIXEL_SIZE;
SegB = ReadPixel(2,x,y + 1) + ReadPixel(3,x,y + 1) * PIXEL_SIZE;
if (SegB == 0 || SegB == Seg)
        return(0);
for (tx = cor.left;tx < cor.right;tx++) {
        for (ty = cor.bottom;ty < cor.top;ty++) {
              SegC = ReadPixel(2,tx,ty) + ReadPixel(3,tx,ty) * PIXEL_SIZE;
              if (SegC == SegB) {
                    WritePixel(2,tx,ty,ReadPixel(2,x,y));
                    WritePixel(3,tx,ty,ReadPixel(3,x,y));
                    }
              }
        }
SetPage(2);
}




/* SAVE_SEG ***************************************************************
* saves a portion of an image file to a file                              *
***************************************************************************/
int save_seg(infile,outfile,seg,LowThreshold,HighThreshold)
int *LowThreshold,*HighThreshold;
FileName *infile,*outfile;
corners seg;
```

```
{
/*
        Purpose - To save an image to a file.

        Method - A buffer is accepted of pixels and this is output to a file
        pixel by pixel as characters.
*/

/* definitions */
int c,x,y;
FILE *InFilePtr,*OutFilePtr;
int scale = FILE_BYTE_SIZE / PIXEL_SIZE;

/* working */
SetPage(0);
printf("\nSaving File %s",outfile);
printf("\nLeft %d Right %d Bottom %d Top %d",seg.left,seg.right,seg.bottom,seg.top);
InFilePtr = fopen(infile,"rb");
if (file_check(infile,InFilePtr) == 0)
        return(0);
OutFilePtr = fopen(outfile,"wb");
if (file_check(outfile,OutFilePtr) == 0)
        return(0);
ReadCorners(InFilePtr,cor);
WriteCorners(OutFilePtr,seg);
for (y = cor.bottom;y < cor.top;y++)  {
        for (x = cor.left;x < cor.right;x++) {
                c = getc(InFilePtr);
                if (x >= seg.left & x < seg.right & y >= seg.bottom
                   & y < seg.top) {
                        if (c < *LowThreshold * scale)
                                c = 0;
                        else if (c > *HighThreshold * scale)
                                c = FILE_BYTE_SIZE - 1;
                        putc(c,OutFilePtr);
                        }
                }
        }
fclose(OutFilePtr);
}



ClipScreen(screen)
int screen;
{
/* definitions */
int x,y;

/* working */
DrawRect(screen,im[screen].c);
for (x = im[screen].c.left;x < im[screen].c.right;x++) {
        for (y = im[screen].c.bottom;y < cor.bottom;y++)
        WritePixel(screen,x,y,0);
        for (y = cor.top;y < im[screen].c.top;y++)
```

```
                WritePixel(screen,x,y,0);
                for (y = cor.bottom;y < cor.top;y++)
                        if (x < cor.left || x > cor.right)
                                WritePixel(screen,x,y,0);
                }
        SetPage(0);
        }
```

```
/* PHDSOB.C */
#include <phd.h>
#include <phdext.h>

int SetSobel()
{
/* definitions */
int sq_type;
int screen1,screen2;
FileName in_file,out_file;

/* workings */
printf("File from which to sobel filter\n");
scanf(" %s", in_file);
printf("File for sobel image to go to\n");
scanf(" %s", out_file);
sobel(in_file,out_file);
return(1);
}

/* SOBEL ******************************************************************
* apply sobel operators to determine gradient                       *
*************************************************************************/

int sobel(in_file,out_file)
FileName *in_file,*out_file;
{
/*
      Purpose - To use the edge detection process of sobel filtering.

      Method - The sobel matrices forvertical and horizontal gradient
      are applied to every pixel and its neighbours. The results of both
      gradients are added and the sum is used to generate a new image.
      If Pluto board used then result goes to a different screen, other-
      wise files are used as input and output. Buffers are employed to keep
      three rows of pixels so that the algorithm is the same once the buffer
      is loaded for file or screen access.
*/
/* definitions */
FILE *InFilePtr,*OutFilePtr;
int i,j,x,y;
int mask[3][3];
int width,height,temp,temp1,temp2;
int *buffer;

/* workings */
InFilePtr = fopen(in_file,"rb");
if (file_check(in_file,InFilePtr) == 0)
      return;
OutFilePtr = fopen(out_file,"wb");
if (file_check(out_file,OutFilePtr) == 0)
      return(0);
ReadCorners(InFilePtr,cor);
width = cor.right - cor.left;
height = cor.top - cor.bottom;
buffer = malloc(sizeof(int) * 3 * width);
if (CheckRoom(buffer,"sobel") == NULL)
        return(0);
WriteCorners(OutFilePtr,cor);
for (i = 0;i < 3;i++)
        for (j = 0;j < 3;j++)
                mask[i][j] = 0;
temp1 = 0;temp2 = 0;temp = 0;
for (x = 0;x < width;x++)
        for (i = 0;i < 3;i++)
                buffer[i * width + x] = 0;
for (y = 0;y < height;y++) {
        for (x = 0;x < width;x++)
                buffer[(y % 3) * width + x] = getc(InFilePtr);
        for (x = 0;x < width;x++) {
                if ((x == 0) || (x == width - 1) || (y == 0) || (y == height - 1))
                        temp = 0;
                else {
                        for (i = 0;i < 3;i++) {
                                for (j = 0;j < 3;j++)
                                        mask[i][j] =
                                        buffer[((y + j) % 3) * width + x + i];
                                }
/* Gx sobel */                  temp1 = (mask[0][2] + 2 * mask[1][2] + mask[2][2]);
                        temp1 -= (mask[0][0] + 2 * mask[1][0] + mask[2][0]);
/* Gy sobel */          temp2 = (mask[2][0] + 2 * mask[2][1] + mask[2][2]);
                        temp2 -= (mask[0][0] + 2 * mask[0][1] + mask[0][2]);
                        temp = abs(temp1) + abs(temp2);
                        }
                if (temp > FILE_BYTE_SIZE)
                        temp = FILE_BYTE_SIZE - 1;
                putc(temp,OutFilePtr);
            }
                printf("*");
        }
fclose(InFilePtr);
fclose(OutFilePtr);
free(buffer);
return(1);
}
```

```
/* PHDSTAN.C */

#include <phd.h>
#include <phdext.h>

/* SET_STAN */
int SetStan()
{
/* definitions */
stantype *p;
int c;

/* working */
p = malloc(sizeof(stantype));
if (! CheckRoom(p,"SetStan"))
        return(0);
strcpy(p->search,NULL);
printf("\nThis routine standardises images files to a stated size");
printf("\nIf the standard size is LARGER than the image file it will");
printf("\nbe saved to a temporary file of suffix .q");
printf("\nwhich should then be standardised");
printf("\nYou should delete these files later or rename them");
wait();
while ((c = dir(spath,p->search,"Standardise")) != ESC & c != CR)
        ;
if (CheckSuffix(p->search,".s")) {
        printf("\nFile %s with .s as suffix not used for standardising");
        printf("\nas it would be overwritten Choose another file");
        wait();
        }
if (c != ESC) {
        printf("\nStandardise to number of pixels in X direction \n");
        scanf(" %d", p->XStan);
        printf("\nStandardise to number of pixels in Y direction \n");
        scanf(" %d", p->YStan);
        p->done = findfirst(p->search,p->ffblk,0);
        p->StanBuffer = malloc(sizeof(char) * p->XStan * p->YStan);
        if (CheckRoom(p->StanBuffer,"SetStan")) {
                while (!p->done) {
                        strcpy(p->infile,p->ffblk.ff_name);
                        stan1(p);
                        p->done = findnext(p->ffblk);
                        }
                }
        p->done = findfirst("*.q",p->ffblk,0);
        if (CheckRoom(p->StanBuffer,"SetStan")) {
                while (!p->done) {
                        strcpy(p->infile,p->ffblk.ff_name);
                        stan1(p);
                        p->done = findnext(p->ffblk);
                        }
                }
        }
free(p->StanBuffer);
free(p);
```

```
}


int stan1(p)
stantype *p;
{
/* definitions */
int i,j;

/* working */
printf("\nSTAN 1");
strcpy(p->tfile,p->infile);
putsuffix(p->tfile,"q");
printf("\nStandardising file %s",p->infile);
strcpy(p->outfile,p->infile);
putsuffix(p->outfile,"S");
printf("\ninfile %s outfile %s",p->infile,p->outfile);
p->InFilePtr = fopen(p->infile,"rb");
p->OutFilePtr = fopen(p->outfile,"wb");
if ((file_check(p->infile,p->InFilePtr))
& (file_check(p->outfile,p->OutFilePtr))) {
        ReadCorners(p->InFilePtr,p->c);
        p->width = p->c.right - p->c.left;
        p->height = p->c.top - p->c.bottom;
        if (p->width < p->XStan || p->height < p->YStan) {
                ExpandFile(p->infile,p->tfile,1 + p->XStan / p->width,1 + p->YStan / p->height);
                strcpy(p->infile,p->tfile);
                }
        else {
                p->buffer = malloc(sizeof(char) * (1 + p->width / p->XStan) * (1 + p->height / p->YStan));
                if (CheckRoom(p->buffer,"stan1"))
                        stan2(p);
            free(p->buffer);
                }
        }
fclose(p->InFilePtr);
fclose(p->OutFilePtr);
printf("\nFinished standardisation\n");
return(OK);
}


int stan2(p)
stantype *p;
/* called from standardise if all checks are OK */
{
/* definitions */
int a,b,x,y;
int result;

/* workings */
printf("\nSTAN 2 Standardising using input file %s output file %s",p->infile,p->outfile);
cor.left = cor.bottom = 0;
cor.right = p->XStan;
cor.top = p->YStan;
```

```
WriteCorners(p->OutFilePtr,cor);
p->width = p->c.right - p->c.left;
p->height = p->c.top - p->c.bottom;
p->ROW = 0;p->COL = 0;
for (y = 0;y < p->YStan;y++) {
        p->rows = ((y + 1) * p->height) / p->YStan - (y * p->height) / p->YStan;
        for (x = 0;x < p->XStan;x++) {
                p->cols = ((x + 1) * p->width) / p->XStan - (x * p->width) / p->XStan;
                a = 0;
                while (a < p->rows) {
                        p->position = sizeof(int) * 4 + sizeof(char) *
                        (p->ROW + a) * p->width + p->COL;
                        result = fseek(p->InFilePtr,p->position,SEEK_SET);
                        b = 0;
                        if (result )
                                printf("\nEOF found");
                        while (b < p->cols)
                                p->buffer[a * p->cols + b++] = getc(p->InFilePtr);
                        a++;
                        }
                qsort(p->buffer,p->rows * p->cols,sizeof(char),strcmp);
                p->temp = p->buffer[(p->rows * p->cols) / 2];
                putc(p->temp,p->OutFilePtr);
                p->COL += p->cols;
                }
        p->COL = 0;
        p->ROW += p->rows;
        printf("*");
        }
}
```

```
/* PHDSUB.C */

#include <phd.h>
#include <phdext.h>

/* SUBTRACT ************************************************************
* Takes one file and subtracts from another. Used to take out noise   *
* added into an image by imaging without the object to view and      *
* subtracting this from image with object                            *
**********************************************************************/

int SetSubFile()
{
/*
        Purpose - To remove the effects of unequal illumination of an image

        Method - Two files are used. One is the original image file, the
        other is the image collected when the object imaged previously has
        been removed. Ie. it is the image of the background only. If the
        background is removed from the total image then only the object should
        remain. Hence one file is simply subtracted from the other. It is
        assumed that the first file given is the file from which to subtract
        the second.
*/
/* definitions */
int temp,c;
FILE *file1_ptr,*file2_ptr,*out_file_ptr;
FileName infile1,infile2,outfile;

/* workings */
printf("\nFile to subtract (foreground)");
scanf(" %s", infile2);
printf("\nFile to subtract from (background)");
scanf(" %s", infile1);
printf("\nFile to create (background - foreground)");
scanf(" %s", outfile);
if ((file1_ptr = fopen(infile1,"rb")) == NULL) {
        printf("\nFile error %s",infile1);
        exit(0);
        }
if ((file2_ptr = fopen(infile2,"rb")) == NULL) {
        printf("\nFile error %s",infile2);
        exit(0);
        }
if ((out_file_ptr = fopen(outfile,"wb")) == NULL) {
        printf("\nFile error %s",outfile);
        exit(0);
        }
c = 0;
while (c != EOF) {
        temp = getc(file1_ptr);
        c = getc(file2_ptr);
        temp -= c;
        if (temp < 0)
                temp = FILE_BYTE_SIZE - 1;
        else
            temp = FILE_BYTE_SIZE - 1 - temp;
        putc(temp,out_file_ptr);
        }
    fclose(file1_ptr);
    fclose(file2_ptr);
    fclose(out_file_ptr);
    }
```

```
 /* PHDSUN.C */
 #include <stdio.h>
 #include <pixrect/pixrect_hs.h>

 struct pixrect *pr;
 int left,right,top,bottom;


 main()
 {
 /* do nothing */
 }

 /* PHDSUN ******************************
 * routines for SUN graphics board    *
 ****************************************/




/* NEGATIVE *********************
* Change LUT to negate image       *
*********************************/

negative()
{
/* definitions */
unsigned char red[PIXEL_SIZE],green[PIXEL_SIZE],blue[PIXEL_SIZE];
int index,count,i;
unsigned char temp;

/* workings */
count = PIXEL_SIZE;
index = 0;
pr_getcolormap(pr,index,count,red,green,blue);
for (i = 0;i < PIXEL_SIZE / 2;i++) {
        temp = red[i];
        red[i] = red[PIXEL_SIZE - 1 - i];
        red[PIXEL_SIZE - 1 - i] = temp;
        temp = green[i];
        green[i] = green[PIXEL_SIZE - 1 - i];
        green[PIXEL_SIZE - 1 - i] = temp;
        temp = red[i];
        blue[i] = blue[PIXEL_SIZE - 1 - i];
        blue[PIXEL_SIZE - 1 - i] = temp;
        }
pr_putcolormap(pr,index,count,red,green,blue);
}


/* HOT_BODY *****************************
* set LUT to hot body scale           *
****************************************/

hot_body()
```

```
{
/*
        Purpose - To set look-up table to hot body scale

         Method - Sun routines are used to set
       individual pixel values from 0 to
       maximum value in such a way that the display
       does from hot to cold colours,
       black is cold, red through range and finally
         white for hottest.
*/

/* definitions */
unsigned char red[PIXEL_SIZE],green[PIXEL_SIZE],blue[PIXEL_SIZE];
int index,count,i;
unsigned char temp;

/* workings */
index = 0;
count = PIXEL_SIZE;
for (i = 0;i < PIXEL_SIZE;i++) {
        if (i >= (9 * PIXEL_SIZE) / 10) {
                red[i] = PIXEL_SIZE - 1;
                green[i] = (i - (9 * PIXEL_SIZE) / 10) * 10;
                blue[i] = PIXEL_SIZE - 1;
                }
        else if (i < PIXEL_SIZE / 10) {
                red[i] = 0;
                green[i] = 0;
                blue[i] = 10 * i;
                }
        else {
                red[i] = (10 * i) / 9;
                green[i] = 0;
                blue[i] = PIXEL_SIZE - 1;
                }
        }
pr_putcolormap(pr,index,count,red,green,blue);
}


/* RED_GREEN_BLUE ********************************
* set LUT to red green blue colour scale        *
************************************************/

red_green_blue()
{
/*
        Purpose - To set a Look-up table so that the
      red-green-blue colour system is employed.

        Method - Low pixel values are set to red with
      increasing ammounts of green added as the
      value increases toa mid-point of green and then
        blue is added till the highest pixel value is pure blue.
```

```
 */

 /* definitions */
 unsigned char red[PIXEL_SIZE],green[PIXEL_SIZE],blue[PIXEL_SIZE];
 int index,count,i;
 unsigned char temp;

 /* workings */
 index = 0;
 count = PIXEL_SIZE;
 red[0] = green[0] = blue[0] = 0;
 red[PIXEL_SIZE - 1] = green[PIXEL_SIZE - 1] = blue[PIXEL_SIZE - 1] = PIXEL_SIZE - 1;
 for (i = 1;i < PIXEL_SIZE;i++) {
 	if (i <= PIXEL_SIZE / 2) {
 		red[i] = 2 * (i - 1);
 		green[i] = 0;
 		blue[i] = PIXEL_SIZE - 2 * i;
 		}
 	else {
 		blue[i] = (PIXEL_SIZE - 1) - 2 * (i - PIXEL_SIZE / 2);
 		green[i] = 2 * (i - PIXEL_SIZE / 2);
 		blue[i] = 0;
 		}
 	}
pr_putcolormap(pr,index,count,red,green,blue);
}


/* MONO   **********************************
* sets up display to monochrome         *
*****************************************/

mono()
{

/* definitions */
unsigned char red[PIXEL_SIZE],green[PIXEL_SIZE],blue[PIXEL_SIZE];
int index,count,i;
unsigned char temp;

/* workings */
count = PIXEL_SIZE;
index = 0;
for (i = 0;i < PIXEL_SIZE;i++)
	red[i] = green[i] = blue[i] = i;
pr_putcolormap(pr,index,count,red,green,blue);
}
```

```
/* PHDTHRES.C */
#include <phd.h>
#include <phdext.h>


/* SETTHRESHOLD */
int SetThreshold()
{
/* definitions */
int screen;

/* working */
clear(0);
printf("\nScreen to threshold\n");
scanf(" %d", screen);
GetThreshold(screen);
}

/* GET_THRESHOLD */
int GetThreshold(screen)
int screen;
{
/* get a threshold for subsequent analysis */
/*
PSEUDO-CODE
Begin
        If not upper
                Starting from lowest pixel value, set LUT value to zero,
                saving old LUT in array
        else
                Starting from highest pixel value, set LUT value to zero,
                saving old LUT in array
        When user request end, return last pixel value as threshold
End
*/
/* definitions */
int threshold;
int colour;
int i,c;

/* working */
SetPage(screen);
getpalette(im[screen].palette);
im[screen].LowThreshold = 0;
im[screen].HighThreshold = getmaxcolor();
for (i = 0;i < 2;i++) {
        c = ' '; /* anything but ESC */
        if (i == 0)
                colour = im[screen].LowThreshold;
        else
                colour = im[screen].HighThreshold;
        while ((c = getch()) != ESC) {
                switch(c) {
                        case CURS_MINUS:
                        if (i == 1) {
```

```
                                        if (colour > 0) {
                                                setpalette(colour,EGA_WHITE);
                                                colour--;
                                                }
                                        }
                                else {
                                        if (colour > 0) {
                                                colour--;
                                                setpalette(colour,im[screen].palette.colors[colour]);
                                                }
                                        }
                                break;
                                case CURS_PLUS:
                                if (i == 1) {
                                        if (colour < getmaxcolor()) {
                                                colour++;
                                                setpalette(colour,im[screen].palette.colors[colour]);
                                                }
                                        }
                                else {
                                        if (colour < getmaxcolor()) {
                                                setpalette(colour,EGA_BLACK);
                                                colour++;
                                                }
                                        }
                                break;
                                }
                        }
                if (i == 0)
                        im[screen].LowThreshold = colour;
                else
                        im[screen].HighThreshold = colour;
                }
        for (i = 0;i < getpalettesize();i++) {
                if (i < im[screen].LowThreshold)
                        im[screen].palette.colors[i] = EGA_BLACK;
                else if (i > im[screen].HighThreshold)
                        im[screen].palette.colors[i] = EGA_WHITE;
                }
        setallpalette(im[screen].palette);
        return(0);
        }
```

```
/* PHDTURN.C */

#include <phd.h>
#include <phdext.h>

/* SET_TURN_ROUND */
int SetTurnRound()
{
/*definitions */
int done,c;
turntype *p;

/* workings */
p = malloc(sizeof(turntype));
if (! CheckRoom(p,"SetTurnRound"))
        return(0);
while ((c = dir(spath,p->searchfile,"Turning Image (will have suffix .t)"))
!= ESC & c != CR)
        ;
if (c != ESC) {
        done = findfirst(p->searchfile,p->ffblk,0);
        while (!done) {
                strcpy(p->outfile,p->ffblk.ff_name); .
                putsuffix(p->outfile,"t");
                if (! CheckSuffix(p->ffblk.ff_name,"t"))
                        TurnRoundImage(p->ffblk.ff_name,p->outfile);
                else {
                        printf("\nFile not turned as %s would be overwritten",p->ffblk.ff_name);
                        DelayOrKeyRead(DELTIME);
                        }
                done = findnext(p->ffblk);
                }
        }
free(p);
}

/* TURN_ROUND_IMAGE */
int TurnRoundImage(infile,outfile)
FileName *infile,*outfile;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
unsigned char *buffer;
long int position;
int c,x,y;

/* digit captures images upside down and inside out */
/* workings */
printf("\nTurning image %s 180 degs to form image %s",infile,outfile);
InFilePtr = fopen(infile,"rb");
OutFilePtr = fopen(outfile,"wb");
buffer = malloc(sizeof(char) * (getmaxx() + 1));
if (CheckRoom(buffer,"TurnRoundImage") & file_check(infile,InFilePtr)
    & file_check(outfile,OutFilePtr)) {
    ReadCorners(InFilePtr,cor);
```

```
        WriteCorners(OutFilePtr,cor);
        for (y = cor.bottom;y < cor.top;y++) {
                position = (cor.top - 1 - y) * (cor.right - cor.left) +
                4 * sizeof(int);
                fseek(InFilePtr,position,SEEK_SET);
                for (x = cor.left;x < cor.right;x++) {
                        if ((c = getc(InFilePtr)) == EOF)
                                c = 0;
                        buffer[cor.right - 1 - x] = c;
                        }
                for (x = cor.left;x < cor.right;x++)
                        putc(buffer[x],OutFilePtr);
                }
        }
free(buffer);
fclose(InFilePtr);
fclose(OutFilePtr);
return(0);
}



/* SETXREFLECT */
int SetXReflect()
{
/*definitions */
int done,c;
turntype *p;

/* workings */
p = malloc(sizeof(turntype));
if (! CheckRoom(p,"SetTurnRound"))
        return(0);
while ((c = dir(spath,p->searchfile,"X Axis Reflect (will have .x suffix"))
!= ESC & c != CR)
        ;
if (c != ESC) {
        done = findfirst(p->searchfile,p->ffblk,0);
        while (!done) {
                strcpy(p->outfile,p->ffblk.ff_name);
                putsuffix(p->outfile,"x");
                if (! CheckSuffix(p->ffblk.ff_name,"x"))
                        XReflect(p->ffblk.ff_name,p->outfile);
                else {
                        printf("\nFile not turned as %s would be overwritten",p->ffblk.ff_name);
                        DelayOrKeyRead(DELTIME);
                        }
                done = findnext(p->ffblk);
                }
        }
}


/* SETYREFLECT */
int SetYReflect()
```

```
{
/*definitions */
int done,c;
turntype *p;

/* workings */
p = malloc(sizeof(turntype));
if (! CheckRoom(p,"SetTurnRound"))
        return(0);
while ((c = dir(spath,p->searchfile,"Y Axis Reflect (will have .y suffix)"))
!= ESC & c != CR)
        ;
if (c != ESC) {
        done = findfirst(p->searchfile,p->ffblk,0);
        while (!done) {
                strcpy(p->outfile,p->ffblk.ff_name);
                putsuffix(p->outfile,"y");
                if (! CheckSuffix(p->ffblk.ff_name,"y"))
                        YReflect(p->ffblk.ff_name,p->outfile);
                else {
                        printf("\nFile not turned as %s would be overwritten",p->ffblk.ff_name);
                        DelayOrKeyRead(DELTIME);
                        }
                done = findnext(p->ffblk);
                }
        }
}

/* XReflect */
XReflect(infile,outfile)
char *infile,*outfile;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
unsigned char *buffer;
long int position;
int c,x,y;


/* workings */
printf("\nReflecting image %s in X axis to form image %s",infile,outfile);
buffer = malloc(sizeof(char) * (getmaxx() + 1));
InFilePtr = fopen(infile,"rb");
OutFilePtr = fopen(outfile,"wb");
if (CheckRoom(buffer,"XReflect") & file_check(infile,InFilePtr)
        & file_check(outfile,OutFilePtr)) {
        ReadCorners(InFilePtr,cor);
        WriteCorners(OutFilePtr,cor);
        for (y = cor.bottom;y < cor.top;y++) {
                position = (cor.top - 1 - y) * (cor.right - cor.left)
                + 4 * sizeof(int);
                fseek(InFilePtr,position,SEEK_SET);
                for (x = cor.left;x < cor.right;x++) {
                        if ((c = getc(InFilePtr)) == EOF)
                                c = 0;
                        buffer[x] = c;
                        }
                for (x = cor.left;x < cor.right;x++)
                        putc(buffer[x],OutFilePtr);
                }
        }
fclose(InFilePtr);
fclose(OutFilePtr);
free(buffer);
return(0);
}



/* YReflect */
int YReflect(infile,outfile)
char *infile,*outfile;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
unsigned char *buffer;
long int position;
int c,x,y;


/* workings */
printf("\nReflecting image %s in Y axis to form image %s",infile,outfile);
buffer = malloc(sizeof(char) * (getmaxx() + 1));
InFilePtr = fopen(infile,"rb");
OutFilePtr = fopen(outfile,"wb");
if (file_check(infile,InFilePtr) & file_check(outfile,OutFilePtr)) {
        ReadCorners(InFilePtr,cor);
        WriteCorners(OutFilePtr,cor);
        for (y = cor.bottom;y < cor.top;y++) {
                for (x = cor.left;x < cor.right;x++) {
                        if ((c = getc(InFilePtr)) == EOF)
                                c = 0;
                        buffer[cor.right - 1 - x] = c;
                        }
                for (x = cor.left;x < cor.right;x++)
                        putc(buffer[x],OutFilePtr);
                }
        }
fclose(InFilePtr);
fclose(OutFilePtr);
free(buffer);
return(0);
}


/* SETEXPAND */
int SetExpand()
{
/* definitions */
int screen,XScale,YScale;
```

```
FileName outfile;
corners c;

/* working */
printf("\nThis routine expands/contracts image, do not use file based routines");
printf("\nSuch as subtract as the image on the screen will not then");
printf("\ncorrespond to the file");
printf("\nScreen to expand/contract \n");
scanf(" %d", screen);
printf("Factor of expansion in X axis\n");
scanf(" %d", XScale);
printf("Factor of expansion in Y axis\n");
scanf(" %d", YScale);
printf("\nFile to save to \n");
scanf(" %s", outfile);
CopyCorners(im[screen].c,c);
DrawRect(screen,c);
SaveRect(screen,c,ScratchFile);
ExpandFile(ScratchFile,outfile,XScale,YScale);
}


int ExpandFile(infile,outfile,XScale,YScale)
FileName *infile,*outfile;
int XScale,YScale;
{
/* definitions */
FILE *InFilePtr,*OutFilePtr;
unsigned char *xbuffer;
int x,y,i,j;
corners c;

/* workings */
SetPage(0);
printf("\nExpanding file %s to %s",infile,outfile);
InFilePtr = fopen(infile,"rb");
OutFilePtr = fopen(outfile,"wb");
if (file_check(outfile,OutFilePtr) & file_check(infile,InFilePtr)) {
        ReadCorners(InFilePtr,c);
        cor.left = cor.bottom = 0;
        cor.right = XScale * (c.right - c.left);
        cor.top = YScale * (c.top - c.bottom);
        WriteCorners(OutFilePtr,cor);
        xbuffer = malloc(sizeof(char) * (c.right - c.left));
        if (CheckRoom(xbuffer,"ExpandFile")) {
                for (y = c.bottom;y < c.top;y++) {
                        for (x = c.left;x < c.right;x++)
                                xbuffer[x - c.left] = getc(InFilePtr);
                        for (i = 0;i < YScale;i++)
                                for (x = c.left;x < c.right;x++)
                                        for (j = 0;j < XScale;j++)
                                                putc(xbuffer[x - c.left],OutFilePtr);
                        }
                }
        }
```

```
fclose(InFilePtr);
fclose(OutFilePtr);
free(xbuffer);
return(1);
}

/* LEFT_TO_RIGHT *********************************************************
* turns image left to right                                    *
*************************************************************************/

/*      Purpose - to mirror an image accross vertical axis

        Method - a buffer stores a vertical line, the vertical line
        mirroring this vertical is copied into it, and the buffer is
        written into the mirror line. This is repeated for half of the image
        as on each occasion 2 lines are dealt with.
*/
int LeftToRight()
{
/* definitions */
int x,y,xmax,ymax,screen;
unsigned char *buffer;

/* workings */
buffer = malloc(sizeof(char) * YSIZE);
if (CheckRoom(buffer,"LeftToRight") == 0)
        return(0);
printf("\nScreen to invert\n");
scanf(" %d", screen);
SetPage(screen);
xmax = getmaxx();
ymax = getmaxy();
for (x = 0;x <= xmax / 2;x++) {
        for (y = 0;y <= ymax;y++)
                buffer[y] = ReadPixel(screen,x,y);
        for (y = 0;y <= ymax;y++) {
                WritePixel(screen,x,y,ReadPixel(screen,xmax - x,y));
                WritePixel(screen,xmax - x,y,buffer[y]);
                }
        }
free(buffer);
return(0);
}

/* UPSIDE_DOWN ***********************************************************
* turns image upside down                                      *
*************************************************************************/

/*      Purpose - to invert an image by rotating 180 degrees.

        Method - A horiziontal buffer is used to store the current
        horizontal line. The mirror line along the horizontal axis is
        written to the current line in reverse order. The buffer is
        then written in reverse order to the mirro line. This is repeated
        for each line upt to the centre line, by which time all lines
        will have been swapped.
*/
int UpsideDown()
{
/* definitions */
int x,y,xmax,ymax,screen;
unsigned char *buffer;

/* workings */
buffer = malloc(sizeof(char) * XSIZE);
if (CheckRoom(buffer,"LeftToRight") == 0)
        return(0);
printf("\nScreen to invert\n");
scanf(" %d", screen);
SetPage(screen);
xmax = getmaxx();
ymax = getmaxy();
for (y = 0;y <= ymax / 2;y++) {
        for (x = 0;x <= xmax;x++)
                buffer[x] = ReadPixel(screen,x,y);
        for (x = 0;x <= xmax;x++) {
                WritePixel(screen,x,y,ReadPixel(screen,xmax - x,ymax - y));
                WritePixel(screen,xmax - x,ymax - y,buffer[x]);
                }
        }
free(buffer);
return(0);
}
```

```
/* PHDUTIL.C */
#include <phd.h>
#include <phdext.h>

int DelayOrKeyRead(delay)
int delay;
{
time_t time1,time2;
int c;
int dx,dy;

time(time1);
time(time2);
if (mouse)
        ClearMouse();
while  ((c = GetMouseCh(dx,dy)) != ESC & c != ' '
        & difftime(time2,time1) < (double) delay)
        time(time2);
return(c);
}

int CheckRoom(pointer,function)
void *pointer;
char *function;
{
if (pointer == NULL) {
        printf("\nInsufficient Memory for variable creation in function %s",function);
        DelayOrKeyRead(DELTIME);
        return(0);
        }
else
        return(1);
}



int find(infile)
FileName *infile;
{
struct ffblk ffblk;
if (findfirst(infile,ffblk,0))
        return(1);
else
        return(0);
}

int IsGetCharCR()
{
int c;
printf("\nEnter RETURN for no change, or other key to change\n");
while (! (c = MouseKeyGet()))
        ;
if (c == CR)
        return(1);
else
```

```
        return(0);
}


int CompareCorners(c1,c2)
corners *c1,*c2;
{
if (c1->left == c2->left & c1->right == c2->right
& c1->bottom == c2->bottom  c1->top == c2->top)
        return(1);
else
        return(0);
}

int ReadCorners(InFilePtr,c)
FILE *InFilePtr;
corners *c;
{
rewind(InFilePtr);
fread(c,sizeof(corners),1,InFilePtr);
if ((c->left > c->right) || (c->bottom > c->top) || (c->bottom < 0) || (c->left < 0)) {
        printf("\nCorners outside allowed range");
        printf("\nCorners are :");
        printf("\nleft %d right %d bottom %d top %d",c->left,c->right,c->bottom,c->top);
        printf("\nRoutine ReadCorners has returned a failure");
        DelayOrKeyRead(DELTIME);
        return(0);
        }
else
        return(1);
}

int WriteCorners(OutFilePtr,c)
FILE *OutFilePtr;
corners *c;
{
rewind(OutFilePtr);
if (c->left > c->right || c->bottom > c->top || c->bottom < 0 || c->left < 0) {
        printf("\nCorners outside allowed range");
        printf("\nFile still written to, corners written are :");
        printf("\nleft %d right %d bottom %d top %d",c->left,c->right,c->bottom,c->top);
        DelayOrKeyRead(DELTIME);
        }
fwrite(c,sizeof(corners),1,OutFilePtr);
return(1);
}

int putsuffix(infile,suffix)
char *infile,*suffix;
{
/* definitions */
int i = 0;

/* working */
while (infile[i] != ' ') {
        if (infile[i] == '.')
              infile[i] = '  ';
        else
              i++;
        }
 if (suffix[0] != '.')
        strcat(infile,".");
 strcat(infile,suffix);
 }


/*
SetSaveFile()
{
FileName outfile;
scanf(" %s", outfile);
square(1);
SaveFile(outfile);
return(OK);
}
*/

/* SAVE_FILE ************************************************************
* saves a screen to a file                              .      *
***********************************************************************/
int SaveFile(outfile)
FileName *outfile;
{
/*
       Purpose - To save an image onthe Pluto board to a file.

       Method - A buffer is accepted of pixels and this is outputtoa file pixel
       by pixel as characters.
*/

/* definitions */
char *buffer;
int colour,x,y;
FILE *out_file_ptr;

/* working */
buffer = malloc(sizeof(char) * PC_X_SIZE);
if (! CheckRoom(buffer,"SaveFile"))
      return(0);
out_file_ptr = fopen(outfile,"wb");
file_check(outfile,out_file_ptr);
fwrite(cor.left,sizeof(int),1,out_file_ptr);
fwrite(cor.right,sizeof(int),1,out_file_ptr);
fwrite(cor.bottom,sizeof(int),1,out_file_ptr);
fwrite(cor.top,sizeof(int),1,out_file_ptr);
for (y = cor.bottom;y < cor.top;y++)
      for (x = cor.left;x < cor.right;x++)
          putc(getpixel(x,y),out_file_ptr);
fclose(out_file_ptr);
free(buffer);
```

```
}




/* FILE_CHECK *****************************************************************
* sees that a file can be opened properly                *
*****************************************************************************/

int file_check(infile,in_file_ptr)
FileName *infile;
FILE *in_file_ptr;
{
if (in_file_ptr == NULL) {
        SetPage(0);
        printf("\nFile access error on file %s",infile);
        DelayOrKeyRead(DELTIME);
        }
return(OK);
}




/* CLS ************************************************************************
* Clear the screen                             *
*****************************************************************************/

int cls()
{
printf(        " 33[2J");
return(OK);
}


/* WAIT ***********************************************************************
* wait for an imput so screen may be viewed or to halt program for  *
* a while.                                     *
*****************************************************************************/

int wait()
{
union REGS in;
printf("\nPress a key to continue\n");
in.h.ah = 0;
int86(KEYBOARDCALL,in,in);
return(OK);
}

/* EXIT1 **********************************************************************
* Tailor made exit function.                           *
*****************************************************************************/

int exit1(value)
int value;
{
printf("\nExit %d",value);
switch (value) {
        case 2 :
                printf("\nFile access error");
                break;
        defaullt :
                printf("\nUnknown error");
                break;
        }
return(OK);
}

/* DUMMY *******************************************************************
* If a function is required but no action needed.                *
*************************************************************************/

int dummy()
{
/* Do nothing */
return(OK);
}


CheckSuffix(infile,suffix)
char *infile;
char *suffix;
{
int i = 0;
while (infile[i] != '.' & infile[i] != ' ')
        i++;
if (suffix[0] != '.')
        i++;
return(! strcmpi(infile[i],suffix));
}
```

Appendix 2 Source Code of MATLAB Functions Created for Study

The macros below are Matlab macros, which are largely devoted to PCA analysis and presentation of data.

```
% PCA.M
function [P,N] = pca(A)
% *** PCA ***
% This function returns principal components analysis matrix and
% the number of components required for different amounts of variance
% Format [p,n] = pca(a)
% Where p = pca matrix, each column is the next most important component
% n = matrix of variance for each number of PCs, ie. n(1) is
% variance of 1 PC, n(2) that of first 2 PCs etc
% A is a matrix with rows of observations, columns of which
% are the variables
cov_matrix = cov(A) ;
[no_entries,dummy] = size(cov_matrix);
if dummy < no_entries  | dummy > no_entries
        disp('Correlation matrix for pca analysis must be square')
        return
        end
[v,d] = eig(cov_matrix);
clear cov_matrix;
for i = 1:no_entries
        evalue(i) = d(i,i);
        end
clear d ;
[sorted,index] = sort(evalue) ;
clear evalue ;
tot_ev = 0.0;
for i = 1:no_entries
        tot_ev = tot_ev + abs(sorted(i)) ;
        end
N(1) = abs(sorted(no_entries)) / tot_ev;
for i = 2:no_entries
        N(i) = N(i - 1) + abs(sorted(1 + no_entries - i)) / tot_ev;
        end
clear sorted ;
for i = 1:no_entries
        norm_eig = v(:,index(no_entries - i + 1));
        divisor = norm(norm_eig,'fro');
        norm_eig = norm_eig / divisor;
        P(:,i) = norm_eig;
        end
clear v ;
clear norm_eig;
clear divisor;
clear index;
```

```
% PCARED.M
function [r] = pcared(a,p,n)
% *** PCARED ***
% This function returns n principal components of matrix a using
% pca matrix p, equivilent to putting other components to zero
% p is assumed to be columns of eigenvectors, with most important first
% followed by next most impotant in adjacent columns
% matrix a is rows of observations, columns of variables
% returning matrix r is similarly rows of observations, and n variables
% Format [r] = pcared(a,p,n)
a = a' ;
p = p' ;
r = p(1:n,:) * a;
r = r' ;

% PCAREST.M
function [a] = pcarest(r,p)
% *** PCAREST ***
%Format [a] = pcarest(r,p)
%This function returns restored components of matrix a using
%pca matrix p
%p is assumed to be columns of eigenvectors, with most important first
%followed by next most impotant in adjacent columns
%matrix a is rows of observations, columns of variables
%matrix r is similarly rows of observations, and n variables
%routine is in effect an approximation to an inverse of pcared
%though the degree to which this is attained depends on the variance
%captured in the pcared routine
[no_rows,no_cols] = size(r) ;
a = r * (p(:,1:no_cols))';

% PCA2PAT.M
function [p,n] = pca2roch(a,file,no_red_dim)
%*** PCA2ROCH (or GENETIC algorithms) ***
%This routine takes .mat files and finds principal components
%using pca function
%a = matrix of rows (observations) and columns (variables)
%file = file to store results into (PCs)
%no_red_dim = number of PCs to use of reduced data set
%R, is reduced matrice to some number of most important PCAs
%returned value p matrix of PCs and n is vector of variance of PCs
[p,n] = pca(a);
R = pcared(a,p,no_red_dim) ;
[no_patts,no_cols] = size(a);
for i = 1: no_patts
        fprintf(file,'0%g0,i)
        for j = 1:no_red_dim
        fprintf(file,'%g ',R(i,j))
        end
      end
```

```
% PCAERR.M
function [D,E,R,A,S] = pcaerr(X,P,N)
%*** PCAERR ***
%Format [D,E,R,A] = pcaerr(X,P,N)
%This function gives the normalised mean square error (nmse)
%and sum of squares of error (sse) of sucessive rows of
%a matrix X which is original data, rows observations, cols variables
%and matrix A, similarly formatted, of restored values by some
%compression of data using PCA
%A is made within the routine by using matrix P (PCA components)
%and N, where N = number of components to consider
%The returned values are :
%D - vector of nmse, one for each observation (row) of X,A
%E - vector of sse, one for each observation (row) of X,A
%R - reduced martix made by transforming X using P for N dimensions
%A - restored observations after compression by PCA
%S - sum of square of X
[no_rows,no_cols] = size(X);
R = pcared(X,P,N) ;
A = pcarest(R,P) ;
for i = 1:no_rows
        [D(i),E(i),S(i)] = nmse(X(i,:),A(i,:)) ;
        end
return
```

Appendix 3 Source Code for C Unix Environment Programs

These functions are simple utilities mainly concerned with translation of data formats.

```
/* MakeNet.C */
main(argc,argv)
int argc;
char *argv[];
{
FILE *fp;
int NoInputs,NoHidden,NoOutputs;

if (argc != 5) {
        printf("\nFormat MakeNet <NetName> <NoInputs> <NoHidden> <NoOutputs>");
        printf("\nCreates network with one hidden layer");
        printf("\nand fully connected inputs - hidden - outputs\n");
        return;
        }
fp = fopen(argv[1],"wb");
if (fp == NULL)
        printf("\nUnable to open file %s",argv[1]);
NoInputs = atoi(argv[2]);
NoHidden = atoi(argv[3]);
NoOutputs = atoi(argv[4]);
fprintf(fp,"definitions:\n");
fprintf(fp,"nunits %d \n",NoHidden + NoInputs + NoOutputs);
fprintf(fp,"ninputs %d\n",NoInputs);
fprintf(fp,"noutputs %d\n",NoOutputs);
fprintf(fp,"end\nnetwork:\n");
fprintf(fp,"%%r %d %d %d %d\n",NoInputs,NoHidden,0,NoInputs);
fprintf(fp,"%%r %d %D %d %d\n",NoInputs + NoHidden,NoOutputs,NoInputs,NoHidden);
fprintf(fp,"end\nbiases:\n");
fprintf(fp,"%%r %d %d\n",NoInputs,NoHidden + NoOutputs);
fprintf(fp,"end");
}
```

```
#include <stdio.h>
/* combine_pca_pat.c */

main(argc,argv)
int argc;
char *argv[];
{
FILE *in1,*in2,*out;
int i,j,result;
char str[20];
int NoInputs;

/* working */
if (argc != 5) {
        printf("\nCombines targets of PDP file and PCs in MATLAB format to PDP file");
        printf("\ninfile1 is PDP file created by pca2roch, infile2 is PDP file from which the matlab file ");
        printf("\nwas created to make infile1");
        printf("\n*** NB. Pattern names are assumed to begin with p ie. p1, p2 etc");
       printf("\nNoInputs is number of inputs in original PDP file (infile2)");
       printf("\nFormat pdp2mat <infile1> <infile2> <outfile> <NoInputs>\n");
        exit(0);
        }
if ((in1 = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((in2 = fopen(argv[2],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
if ((out = fopen(argv[3],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[3]);
        exit(0);
        }
NoInputs = atoi(argv[4]);
i = 0;
j = 0;
while ((result = fscanf(in1,"%s",str)) > 0 && str[0] != 'p') /* get first PCA pattern */
        ;
fprintf(out,"%s ",str);
while ((result = fscanf(in2,"%s",str)) > 0 && str[0] != 'p') /* get first raw data pattern */
        ;
while (result > 0) {
        while ((result = fscanf(in1,"%s",str)) > 0 && str[0] != 'p') /* get next PCA pattern */
                fprintf(out,"%s ",str);
        fprintf(out,"\n");
        for (i = 0;i < NoInputs;i++)
                fscanf(in2,"%s",str); /* ignore inputs */
      while ((result = fscanf(in2,"%s",str)) > 0 && str[0] != 'p') /* get next raw data pattern */
                fprintf(out,"%s ",str);
        fprintf(out,"\n%s ",str);
     }
fprintf(out,"\n");
fclose(in1);
fclose(in2);
fclose(out);
}
```

```
/* im2matseg.c */
#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int NoRows,NoCols,ImX,ImY,result;
char str[40];
int i,j,k,x,y;
int *buffer;

/* working */
if (argc != 7) {
        printf("\nConvert image file of several images to segments in");
        printf("\nMATLAB format");
      printf("\nFormat im2segmat <infile> <outfile> <No. Cols> ");
        printf("\n<No. Rows> <Im X size> <Im Y size>\n");
        exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
NoRows = atoi(argv[3]);
NoCols = atoi(argv[4]);
ImX = atoi(argv[5]);
ImY = atoi(argv[6]);
printf("\nNoRows %d NoCols %d Original Size of file x %d by y %d \n",NoRows,NoCols,ImX,ImY);
buffer = (int *) malloc(sizeof(int) * ImX * ImY);
if (buffer == NULL) {
        printf("\nUnable to allocate enough memory for buffer");
        exit(0);
        }
i = j = k = x = y = 0;
while ((result = fscanf(in,"%s",str)) != EOF) {
        if (result > 0) {
                if (str[0] == '*') {
                        printf("\nSegmenting file %s",str);
                        for (y = 0;y < ImY;y++)
                                for (x = 0;x < ImX;x++) {
                                        result = fscanf(in,"%d",&buffer[(y * ImX) + x]);
                                        }
                for (y = 0;y < ImY;y += NoRows) {
                        for (x = 0;x < ImX;x += NoCols) {
                                for (i = y;i < y + NoRows;i++) {
                                        for (j = x;j < x + NoCols;j++) {
                                                fprintf(out,"%d ",buffer[i * ImX + j]);
                                                }
                                        }
```

```
                                    fprintf(out,"%c%c",13,10);
                              }
                        }
            }
      }
      }
free(buffer);
fclose(in);
fclose(out);
}
```

```
/* im2raster.c */

#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define     NULL      0
#define CMS_SIZE 256

unsigned char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];

main(argc,argv)
int argc;
char *argv[];
{
if (argc != 7) {
printf("\nFormat ct2raster <infile> <outfile> <width> <height> <depth> <ColourScale>");
printf("\nWhere ColourScale is (r)edgreenblue (m)onochrome (h)otbody (g)rey scale/n");
     exit(0);
     }
ct_to_rasterfile(argv[1],argv[2],atoi(argv[3]),atoi(argv[4]),atoi(argv[5]),argv[6]);

}

ct_to_rasterfile(in_file, out_file, width, height, depth,ColourScale)
char  in_file[20];
char  out_file[20];
int   width, height, depth;
char *ColourScale;
{
      register int i,c;
      int x,y;
      FILE *in,*out;
      struct rasterfile rh;
      rh.ras_magic = RAS_MAGIC;
      rh.ras_width = width;
      rh.ras_height = height;
      rh.ras_depth = depth;
      rh.ras_length = (width*height);
      rh.ras_type = RT_STANDARD;
      rh.ras_maptype = RMT_EQUAL_RGB;
      rh.ras_maplength = 768;
      if ((in = fopen(in_file, "rb")) == NULL) {
            printf("Can't open %s\n", in_file);
            }
      else if ((out = fopen(out_file, "wb")) == NULL) {
            printf("Can't open %s\n", out_file);
            }
    if (ColourScale[0] == 'r')
            red_green_blue();
      else if (ColourScale[0] == 'h')
          hot_body();
      else if (ColourScale[0] == 'g' )
            grey_scale();
      else
            mono();
```

```
	fwrite(&rh,sizeof(rh),1,out);
	 if (ColourScale[0] != 'm') {
		fwrite(red,sizeof(char),CMS_SIZE,out);
		fwrite(green,sizeof(char),CMS_SIZE,out);
		fwrite(blue,sizeof(char),CMS_SIZE,out);
		}
	 else
		fwrite(red,sizeof(char),1,out);
	for (y = 0;y < height;y++)
	 for (x = 0;x < width;x++)
			putc(getc(in),out);
	fclose(in);
	 fclose(out);
}


/* NEGATIVE **********************
* Change LUT to negate image       *
**********************************/

negative()
{
/* definitions */
int index,i;
unsigned char temp;

/* workings */
index = 0;
for (i = 0;i < CMS_SIZE / 2;i++) {
	temp = red[i];
	red[i] = red[CMS_SIZE - 1 - i];
	red[CMS_SIZE - 1 - i] = temp;
	temp = green[i];
	green[i] = green[CMS_SIZE - 1 - i];
	green[CMS_SIZE - 1 - i] = temp;
	temp = red[i];
	blue[i] = blue[CMS_SIZE - 1 - i];
	blue[CMS_SIZE - 1 - i] = temp;
	}
}


/* HOT_BODY ******************************
* set LUT to hot body scale             *
*****************************************/

hot_body()
{
/*
	Purpose - To set look-up table to hot body scale

	 Method - Sun routines are used to set
	individual pixel values from 0 to
	maximum value in such a way that the display
	does from hot to cold colours,
        black is cold, red through range and finally
          white for hottest.
*/

/* definitions */
int index,i;

/* workings */
for (i = 0;i < CMS_SIZE;i++) {
        if (i >= (9 * CMS_SIZE) / 10) {
                red[i] = CMS_SIZE - 1;
                green[i] = (i - (9 * CMS_SIZE) / 10) * 10;
                blue[i] = CMS_SIZE - 1;
                }
        else if (i < CMS_SIZE / 10) {
                red[i] = 0;
                green[i] = 0;
                blue[i] = 10 * i;
                }
        else {
                red[i] = (10 * i) / 9;
                green[i] = 0;
                blue[i] = CMS_SIZE - 1;
                }
        }
}


/* RED_GREEN_BLUE ********************************
* set LUT to red green blue colour scale        *
*************************************************/

red_green_blue()
{
/*
          Purpose - To set a Look-up table so that the
        red-green-blue colour system is employed.

          Method - Low pixel values are set to red with
        increasing ammounts of green added as the
        value increases toa mid-point of green and then
          blue is added till the highest pixel value is pure blue.

*/

/* definitions */
int i;

/* workings */
red[0] = green[0] = blue[0] = 0;
red[CMS_SIZE - 1] = green[CMS_SIZE - 1] = blue[CMS_SIZE - 1] = CMS_SIZE - 1;
for (i = 1;i < CMS_SIZE;i++) {
        if (i <= CMS_SIZE / 2) {
                red[i] = 2 * (i - 1);
                green[i] = 0;
                blue[i] = CMS_SIZE - 2 * i;
                }
        else {
                blue[i] = (CMS_SIZE - 1) - 2 * (i - CMS_SIZE / 2);
                green[i] = 2 * (i - CMS_SIZE / 2);
                blue[i] = 0;
                }
        }
}


/* MONO  **********************************
* sets up display to monochrome           *
*******************************************/
mono(pr)
{
red[0] = 1;
return(0);
}

/* GREY_SCALE **************************/

grey_scale(pr)
{

/* definitions */
int index,i;

/* workings */
index = 0;
for (i = 0;i < CMS_SIZE;i++)
        red[i] = green[i] = blue[i] = i;
}
```

```
/* make_nni.c */

#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *in1,*in2,*out;
int i,j,result;
char str1[40],str2[40],str3[40];

/* working */
if (argc != 4) {
        printf("\nConvert PDP and .nni file made from MATLAB ");
        printf("\nto full .nni file by adding required outputs");
       printf("\nFormat make_nni <PDP file> <.nni file> <output file>");
        printf("\nThis program assumes a file (.nni file) has been created") ;
        printf("\nwhich contains inputs to NeuralWare") ;
        printf("\nThe corresponding pattern file from which it was created") ;
        printf("\nis scanned until a pattern name is found.");
        printf("\nThe pattern name is used to determine the required output");
        printf("\n32 segments are assumed to complete each image") ;
        printf("\nIe. after 32 sets of numbers from .nni file, a required");
        printf("\noutput is added");
        exit(0);
        }
if ((in1 = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((in2 = fopen(argv[2],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
if ((out = fopen(argv[3],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[3]);
        exit(0);
        }
while ((result = fscanf(in1,"%s",str1)) != EOF) {
        if (result > 0) {
                while ((result = fscanf(in1,"%s",str1))
                 != EOF && str1[0] != 'p' && str1[0] != 'P')
                        ;
                /* get next pattern name */

        printf("\n* input pattern starting at segment %s\n",str1);
        fprintf(out,"%c%c* input pattern starting at segment %s%c%ci",13,10,str1,13,10);
                for (i = 0;i < 32;i++) {
                        while ((result = fscanf(in2,"%s",str2))
                        != EOF && str2[0] != '-' && str2[0] != '.' && (str2[0] < '0' || str2[0] > '9'))
                                ;
                        for (j = 0;j < 4;j++) {
                                printf(" %s",str2);
                                fprintf(out," %s",str2);
                        result = fscanf(in2,"%s",str2);
                                }
                printf("\n");
                        fprintf(out,"%c%c",13,10);
                }
                /* get 32 segments of reduced PCs */
        if (str1[1] == 'P')
                        strcpy(str3,"d 1.0 ");
                else
                        strcpy(str3,"d 0.0 ");
                if (str1[1] == 'C')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[4] == 'P')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[4] == 'L')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[4] == 'R')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[5] == 'P')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[5] == 'V')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                if (str1[5] == 'U')
                        strcat(str3,"1.0 ");
                else
                        strcat(str3,"0.0 ");
                printf("\n%s",str3);
                fprintf(out,"%c%c%s",13,10,str3);
                /* put in desired output */
                for (i = 1;i < 32;i++)  {
                        while ((result = fscanf(in1,"%s",str1))
                        != EOF && str1[0] != 'p' && str1[0] != 'P')
                                ;
                        /* ditch next 31 pattern names */
                }
        }
        }
fprintf(out,"%c%c",13,10);
fclose(in1);
fclose(in2);
fclose(out);
}
```

```
/* mat2pdp.c */

#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int i,result;
typedef char string[12];
char str[20];
string *buffer;
int patt_no,j,no_cols;

/* working */
if (argc != 4) {
        printf("\nFormat pdp2mat <infile> <outfile> <No. Cols>\n");
        exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
buffer = malloc(sizeof(char) * atoi(argv[3]));
if (buffer == NULL) {
        printf("\nFailed to allocate room for buffer");
        exit(0);
        }
i = 0;
patt_no = 0;
no_cols = atoi(argv[3]);
while ((result = fscanf(in,"%s",str)) != EOF) {
        if (result > 0) {
                if (i % no_cols  == 0) {
                        fprintf(out,"\np%d ",patt_no);
                        for (j = 0;j < 2;j++)
                                if (j > 0)
                                        fprintf(out,"\n");
                        for (i = 0;i < no_cols;i++)
                                        fprintf(out," %s", buffer[i]);
                                        if (i % 10 == 0)
                                                fprintf(out,"\n");
                        }
        strcpy(buffer[i % atoi(argv[3])],str);
                }
        }
fprintf(out,"\n");
fclose(in);
fclose(out);
}
```

```
#include <stdio.h>
/* n2m.c */

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int i,j,result;
char str[40];

/* working */
if (argc != 4) {
        printf("\nConvert NWORKS file to MATLAB format");
      printf("\nFormat n2m <infile> <outfile> <No. Cols>\n");
        exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
i = 0;
j = 0;
while ((result = fscanf(in,"%s",str)) != EOF) {
        if (result > 0) {
                if (str[0] != 'i' && str[0] != 'd' && str[0] != '*') {
/* as p was used to show pattern names, and * was used in later genetic experiments */
                fprintf(out,"%s ",str);
                   i++;
                if (!(i % atoi(argv[3]))) {
                                i = 0;
                                fprintf(out,"\n");
                                while ((result = fscanf(in,"%s",str))
                                != EOF && str[0] != 'i')
                                        ;
/* because there may be a target vector which we are not interested in */
                        }
                }
        }
        }
fprintf(out,"\n");
fclose(in);
fclose(out);
}
```

```
/* p2m.c */

#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int i,j,result;
char str[40];

/* working */
if (argc != 4) {
        printf("\nConvert PDP file to MATLAB format");
      printf("\nFormat pdp2mat <infile> <outfile> <No. Cols>\n");
        exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
i = 0;
j = 0;
while ((result = fscanf(in,"%s",str)) != EOF) {
        if (result > 0) {
                if (str[0] != 'p' && str[0] != 'P' && str[0] != '*') {
/* as p was used to show pattern names, and * was used in later genetic experiments */
                fprintf(out,"%s ",str);
                  i++;
                if (!(i % atoi(argv[3]))) {
                                i = 0;
                                fprintf(out,"\n");
                                while ((result = fscanf(in,"%s",str))
                                != EOF && str[0] != 'p' && str[0] != 'P' && str[0] != '*')
                                        ;
/* because there may be a target vector which we are not interested in */
                        }
                }
        }
        }
fprintf(out,"\n");
fclose(in);
fclose(out);
}
/* translates training pattern for PDP to recall only, ie gets rid of target vectors to save room */
/* r2tp.c */
#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int i,result,no_inputs;
char str[40];
float *buffer;

/* working */
if (argc != 4) {
        printf("\nFormat pdp2mat <infile> <outfile> <No. Inputs>\n");
        exit(0);
        }
no_inputs = atoi(argv[3]);
buffer = malloc(sizeof(float) * atoi(argv[3]));
if (buffer == NULL) {
        printf("\nUnable to allocate enough room for float buffer ");
        exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
i = 0;
while ((result = fscanf(in,"%s",str))
!= EOF && str[0] != 'p' && str[0] != 'P')
        ;
while (result != EOF) {
        fprintf(out,"%s ",str);
        for (i = 1; i <= no_inputs;i++) {
                result = fscanf(in,"%s",str);
                fprintf(out,"%s ",str);
            buffer[i - 1] = atof(str);
                if ((i % 10) == 0 && i > 0)
                        fprintf(out,"\n");
                }
        fprintf(out,"\n");
        for (i = 1; i <= atoi(argv[3]);i++) {
                fprintf(out,"%f ",buffer[i - 1]);
                if ((i % 10) == 0 && i > 0)
                        fprintf(out,"\n");
                }
        while ((result = fscanf(in,"%s",str))
              != EOF && str[0] != 'p' && str[0] != 'P')
                        ;
/* because there may be a target vector which we are not interested in */
        fprintf(out,"\n");
        }
fclose(in);
fclose(out);
}
```

```
/* raster2im.c */
#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define      NULL      0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];

main(argc,argv)
int argc;
char *argv[];
{
if (argc != 3) {
        printf("\nFormat raster2im <infile> <outfile> \n");
      exit(0);
      }
rasterfile2im(argv[1],argv[2]);
return(TRUE);
}

rasterfile2im(in_file, out_file)
char  in_file[20];
char  out_file[20];
{
        register int i,c;
        int x,y;
        FILE *in,*out;
        struct rasterfile rh;
        if ((in = fopen(in_file, "rb")) == NULL) {
                printf("Can't open %s\n", in_file);
                }
        else if ((out = fopen(out_file, "wb")) == NULL) {
                printf("Can't open %s\n", out_file);
                }
      fread(&rh,sizeof(rh),1,in);
        printf("\nSize of image x %d y %d depth %d",rh.ras_width,rh.ras_height,rh.ras_depth);
        fread(red,sizeof(char),CMS_SIZE,in);
        fread(green,sizeof(char),CMS_SIZE,in);
        fread(blue,sizeof(char),CMS_SIZE,in);
        fwrite(&rh.ras_width,sizeof(int),1,out);
        fwrite(&rh.ras_height,sizeof(int),1,out);
      for (y = 0;y < rh.ras_height;y++)
        for (x = 0;x < rh.ras_width;x++)
                        putc(getc(in),out);
        printf("\n");
      fclose(in);
        fclose(out);
}
```

```
/* raster2pic.c */

#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define     NULL     0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];

main(argc,argv)
int argc;
char *argv[];
{
if (argc != 3) {
        printf("\nFormat raster2pic <infile> <outfile> \n");
        printf("\n<infile> is raster file, <outfile> is ps file\n");
      exit(0);
      }
rasterfile2pic(argv[1],argv[2]);
return(TRUE);
}

rasterfile2pic(in_file, out_file)
char  in_file[20];
char  out_file[20];
{
int i,c;
int x,y;
FILE *in,*out;
struct rasterfile rh;
if ((in = fopen(in_file, "rb")) == NULL) {
        printf("Can't open %s\n", in_file);
        }
else if ((out = fopen(out_file, "wb")) == NULL) {
        printf("Can't open %s\n", out_file);
        }
fread(&rh,sizeof(rh),1,in);
printf("\nSize of image x %d y %d depth %d\n",rh.ras_width,rh.ras_height,rh.ras_depth);
fread(red,sizeof(char),CMS_SIZE,in);
fread(green,sizeof(char),CMS_SIZE,in);
fread(blue,sizeof(char),CMS_SIZE,in);
fprintf(out,"%%!PS\n");
fprintf(out,"/picstr %d string def \n",rh.ras_width);
fprintf(out,"45 140 translate \n");
fprintf(out,"132 132 scale\n");
fprintf(out,"%d %d %d [%d 0 0 %d 0 0] \n",
rh.ras_width,rh.ras_height,rh.ras_depth,rh.ras_width,rh.ras_height);
fprintf(out,"{currentfile picstr readhexstring pop \nimage\n");
for (y = 0;y < rh.ras_height ;y++) {
        for (x = 0;x < rh.ras_width;x++) {
                if ((x % 10) == 0 && x > 0)
                        fprintf(out,"\n");
        c = getc(in);
```

```
                if (c <= 15)
                        fprintf(out,"0");
                fprintf(out,"%x ", c);
                }
        fprintf(out,"\n");
       }
fprintf(out,"image\n");
fprintf(out,"showpage \n");
fclose(in);
fclose(out);
}
```

```
/* seg2pred.c */

#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define     NULL      0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];
char  in_file[20];
char  out_file[20];
struct rasterfile rh;

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int x,y,i,j,result,Ce,Se,pat_no,image_no,BorderX,BorderY;
int RSample,itemp;
char str[40];
float *image;
float border,temp,scale;
int NoPatternsUsed = 0;

/* working */
if (argc != 8) {
        printf("\nConvert raster image file to Rochester/PDP format");
        printf("\nTakes a series of overlapping segments of the image, puts all bar centre segment");
        printf("\nas a training input, centre segment used as desired output");
        printf("\nPurpose is to get surrounding area to predict inner area");
        printf("\n*** Warning, use either both odd, or both even for Se, Ce pair and for Se, Ce pair");
        printf("\nAn outer border is added to the original image, which contains a specified constant");
      printf("\nFormat seg2pred <infile> <outfile> <Ce> <Se> <Border> <Scale> <RSample>\n");
        printf("\nwhere Ce is size (length of sides) of central area to predict");
      printf("\nand Se is size of segments");
        printf("\nand Border is the value to set the border to");
        printf("\nand Scale is the scaling factor for inputs");
        printf("\nand RSample is the randomicity component where");
        printf("\napprox 1 in <RSample> patterns are included");
      printf("\n");
      exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
Ce = atoi(argv[3]);
Ce = atoi(argv[4]);
Se = atoi(argv[5]);
Se = atoi(argv[6]);
itemp = atoi(argv[7]);
border = (float) itemp;
itemp = atoi(argv[8]);
scale = (float) itemp;
RSample = atoi(argv[9]);
BorderX = (Se - Ce) / 2;
BorderY = (Se - Ce) / 2;
fread(&rh,sizeof(rh),1,in);
printf("\nSize of image x %d y %d depth %d",rh.ras_width,rh.ras_height,rh.ras_depth);
fread(red,sizeof(char),CMS_SIZE,in);
fread(green,sizeof(char),CMS_SIZE,in);
fread(blue,sizeof(char),CMS_SIZE,in);
printf("\nCreating training patterns useing segment size of %d by %d and central portion of %d by %d"
,Se,Se,Ce,Ce);
printf("\nand border of %d in x and %d in y axes",BorderX,BorderY);
printf("\nand Scale of %f",scale);
printf("\nWith original image size %d by %d",rh.ras_width,rh.ras_height);
printf("\nsampling about 1 in %d patterns",RSample);
printf("\n");
image = (float *) malloc(sizeof(float) * rh.ras_width * rh.ras_height);
if (image == NULL) {
        printf("\nInsufficient room for image matrix to be built\n");
        exit(0);
        }
srand(getpid());
result = 1;
str[0] = ' ';
image_no = 0;
for (y = 0;y < rh.ras_height;y++) {
        for (x = 0;x < rh.ras_width;x++) {
                image[y * rh.ras_width + x] = (float) (getc(in)) / scale;
                }
        }
for (y = -BorderY,pat_no = 0;y < (rh.ras_height - Se + BorderY);y++) {
        for (x = -BorderX;x < (rh.ras_width - Se + BorderX);x++,pat_no++) {
                if (rand() % RSample == 0) {
                        NoPatternsUsed++;
                        fprintf(out,"*p%d ",pat_no);
                        for (i = 0;i < Se;i++) {
                                for (j = 0;j < Se;j++) {
                                        if ((i < BorderY || i >= (Se + Ce) / 2)
                                        || (j < BorderX || j >= (Se + Ce) / 2)) {
                                                if (x + j < 0 || y + i < 0 || x + j >= rh.ras_width
                                        || y + i >= rh.ras_height) {
                                                        fprintf(out,"%f ",border);
                                                        }
                                                else {
                                                        temp = image[(y + i) * rh.ras_width + (x + j)];
                                                        fprintf(out,"%f ",temp);
                                                        }
                                                }
                                        }
                                }
                        fprintf(out,"\n");
                        for (i = 0;i < Ce;i++)
                            for (j = 0;j < Ce;j++) {
                                temp = image[(y+(Se-Ce)/2 + i) * rh.ras_width + (x + (Se-Ce)/2 +j)];
                                fprintf(out,"%f ",temp);
                                }
                        fprintf(out,"\n");
                        }
                    }
            }
    printf("\n%d patterns used",NoPatternsUsed);
    fprintf(out,"\n");
    fclose(in);
    fclose(out);
    }
```

```
/* seg2predNoBorder.c */
#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define     NULL      0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];
char  in_file[20];
char  out_file[20];
struct rasterfile rh;

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int x,y,i,j,result,Ce,Se,pat_no,image_no,BorderY;
int RSample,itemp;
char str[40];
float *image;
float border,temp,scale;
int NoPatternsUsed = 0;
int Ring,NoOuterRings;

/* working */
if (argc != 7) {
        printf("\nConvert raster image file to Rochester/PDP format");
        printf("\nTakes a series of overlapping segments of the image, puts all bar centre segment");
        printf("\nas a training input, centre segment used as desired output");
        printf("\nPurpose is to get surrounding area to predict inner area");
        printf("\nSerial square perimeters of decreasing size form the input set");
      printf("\nNo outer border is added to the original image");
      printf("\nFormat seg2predNoBorder <infile> <outfile> <Ce> <Se> <Scale> <RSample>\n");
        printf("\nwhere Ce is size (length of sides) of central area to predict");
      printf("\nand Se is size of segments");
        printf("\nand Scale is the scaling factor for inputs");
        printf("\nand RSample is the randomicity component where");
        printf("\napprox 1 in <RSample> patterns are included");
      printf("\n");
      exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
Ce = atoi(argv[3]);
Se = atoi(argv[4]);
if (Se <= Ce || (Se - Ce) % 2) {
        printf("\nSe and Ce must be both odd or even and Se > Ce\n");
```

```
          exit(0);
          }
NoOuterRings = (Se - Ce) / 2 ;
itemp = atoi(argv[5]);
scale = (float) itemp;
RSample = atoi(argv[6]);
fread(&rh,sizeof(rh),1,in);
printf("\nSize of image x %d y %d depth %d",rh.ras_width,rh.ras_height,rh.ras_depth);
fread(red,sizeof(char),CMS_SIZE,in);
fread(green,sizeof(char),CMS_SIZE,in);
fread(blue,sizeof(char),CMS_SIZE,in);
printf("\nCreating training patterns useing segment size of %d by %d and central portion of
%d by %d",Se,Se,Ce,Ce);
printf("\nand Scale of %f",scale);
printf("\nWith original image size %d by %d",rh.ras_width,rh.ras_height);
printf("\nsampling about 1 in %d patterns",RSample);
printf("\n");
image = (float *) malloc(sizeof(float) * rh.ras_width * rh.ras_height);
if (image == NULL) {
          printf("\nInsufficient room for image matrix to be built\n");
          exit(0);
          }
srand(getpid());
result = 1;
str[0] = ' ';
image_no = 0;
for (y = 0;y < rh.ras_height;y++) {
          for (x = 0;x < rh.ras_width;x++) {
                    image[y * rh.ras_width + x] = (float) (getc(in)) / scale;
                    }
          }
for (y = 0,pat_no = 0;y < (rh.ras_height - Se);y++) {
          for (x = 0;x < (rh.ras_width - Se);x++,pat_no++) {
                    if (rand() % RSample == 0) {
                              NoPatternsUsed++;
                              fprintf(out,"*p%d ",pat_no);
                              for (Ring = 0;Ring < NoOuterRings;Ring++) {
                                        j = y + Ring;
                              for (i = x + Ring;i < x + Se - Ring;i++) {
                                                  temp = image[j * rh.ras_width + i];
                                                  fprintf(out,"%f ",temp);
                                        }
                                                  fprintf(out,"\n");
                              for (++j;j < y + Se - Ring;j++) {
                                                  temp = image[j * rh.ras_width + i];
                                                  fprintf(out,"%f ",temp);
                                        }
                                                  fprintf(out,"\n");
                              for (--i,--j;i > x + Ring;i--) {
                                                  temp = image[j * rh.ras_width + i];
                                                  fprintf(out,"%f ",temp);
                                        }
                                                  fprintf(out,"\n");
                              for (j--;j > y + Ring;j--) {
                                                  temp = image[j * rh.ras_width + i];
                                        fprintf(out,"%f ",temp);
                                   }
                                   }
                    fprintf(out,"\n");
                          for (i = 0;i < Ce;i++)
                                for (j = 0;j < Ce;j++) {
                                     temp = image[(y+(Se-Ce)/2 + i) * rh.ras_width + (x + (Se-Ce)/2 + j)];
                                     fprintf(out,"%f ",temp);
                                     }
                          fprintf(out,"\n");
                          }
                    }
          }
     printf("\n%d patterns used",NoPatternsUsed);
     fprintf(out,"\n");
     fclose(in);
     fclose(out);
     }
```

```
/* seg2predaa.c */

#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define      NULL      0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];
char   in_file[20];
char   out_file[20];
struct rasterfile rh;

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int x,y,i,j,result,SeS,pat_no,image_no;
int RSample,itemp;
char str[40];
float *image;
float temp,scale;
int NoPatternsUsed = 0;
int Ring,NoRings,ys,ye,xs,xe;

/* working */
if (argc != 6) {
        printf("\nConvert raster image file to PDP format auto-associative (aa) program");
        printf("\nTakes a series of overlapping segments of the image.");
        printf("\nand creates pattern file as a square matrix, NOT like seg2pred");
      printf("\nFormat seg2pred <infile> <outfile> <Segment Size> <Scale> <RSample>\n");
        printf("\nand Scale is the scaling factor for inputs");
        printf("\nand RSample is the randomicity component where");
        printf("\napprox 1 in <RSample> patterns are included");
      printf("\n");
      exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
SeS = atoi(argv[3]);
SeS = SeS;
itemp = atoi(argv[4]);
scale = (float) itemp;
RSample = atoi(argv[5]);
fread(&rh,sizeof(rh),1,in);
printf("\nSize of image x %d y %d depth %d",rh.ras_width,rh.ras_height,rh.ras_depth);
fread(red,sizeof(char),CMS_SIZE,in);
fread(green,sizeof(char),CMS_SIZE,in);
fread(blue,sizeof(char),CMS_SIZE,in);
printf("\nCreating training patterns useing segment size of %d by %d ",SeS,SeS);
printf("\nand Scale of %f",scale);
printf("\nWith original image size %d by %d",rh.ras_width,rh.ras_height);
printf("\nsampling about 1 in %d patterns",RSample);
printf("\n");
image = (float *) malloc(sizeof(float) * rh.ras_width * rh.ras_height);
if (image == NULL) {
        printf("\nInsufficient room for image matrix to be built\n");
        exit(0);
        }
srand(getpid());
result = 1;
str[0] = ' ';
image_no = 0;
for (y = 0;y < rh.ras_height;y++) {
        for (x = 0;x < rh.ras_width;x++) {
                image[y * rh.ras_width + x] = (float) (getc(in)) / scale;
        }
        }
NoRings = SeS / 2 + SeS % 2;
for (y = 0,pat_no = 0;y < (rh.ras_height -SeS);y++) {
        for (x = 0;x < (rh.ras_width - SeS );x++,pat_no++) {
                if (rand() % RSample == 0) {
                        NoPatternsUsed++;
                        fprintf(out,"*p%d ",pat_no);
                        for (i = y;i < y + SeS;i++) {
                                for (j = x;j < x + SeS;j++) {
                                        temp = image[i * rh.ras_width + j];
                                        fprintf(out,"%f ",temp);
                                        }
                        }
                        fprintf(out,"\n");
                        }
                }
        }
printf("\n%d patterns used",NoPatternsUsed);
fprintf(out,"\n");
fclose(in);
fclose(out);
}
```

```
/* seg2predaaRing.c */
#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define TRUE 1
#define      NULL       0
#define CMS_SIZE 256

char red[CMS_SIZE],green[CMS_SIZE],blue[CMS_SIZE];
char  in_file[20];
char  out_file[20];
struct rasterfile rh;

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int x,y,i,j,result,SeS,pat_no,image_no;
int RSample,itemp;
char str[40];
float *image;
float temp,scale;
int NoPatternsUsed = 0;
int Ring,NoRings,ys,ye,xs,xe;

/* working */
if (argc != 6) {
        printf("\nConvert raster image file to PDP format auto-associative (aa) program");
        printf("\nTakes a series of overlapping segments of the image.");
        printf("\nand creates pattern file with rings of data as in seg2pred");
      printf("\nFormat seg2pred <infile> <outfile> <Segment Size> <Scale> <RSample>\n");
        printf("\nand Scale is the scaling factor for inputs");
        printf("\nand RSample is the randomicity component where");
        printf("\napprox 1 in <RSample> patterns are included");
      printf("\n");
      exit(0);
        }
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
SeS = atoi(argv[3]);
SeS = SeS;
itemp = atoi(argv[4]);
scale = (float) itemp;
RSample = atoi(argv[5]);
fread(&rh,sizeof(rh),1,in);
printf("\nSize of image x %d y %d depth %d",rh.ras_width,rh.ras_height,rh.ras_depth);
fread(red,sizeof(char),CMS_SIZE,in);
fread(green,sizeof(char),CMS_SIZE,in);
fread(blue,sizeof(char),CMS_SIZE,in);
printf("\nCreating training patterns useing segment size of %d by %d ",SeS,SeS);
printf("\nand Scale of %f",scale);
printf("\nWith original image size %d by %d",rh.ras_width,rh.ras_height);
printf("\nsampling about 1 in %d patterns",RSample);
printf("\n");
image = (float *) malloc(sizeof(float) * rh.ras_width * rh.ras_height);
if (image == NULL) {
        printf("\nInsufficient room for image matrix to be built\n");
        exit(0);
        }
srand(getpid());
result = 1;
str[0] = ' ';
image_no = 0;
for (y = 0;y < rh.ras_height;y++) {
        for (x = 0;x < rh.ras_width;x++) {
                image[y * rh.ras_width + x] = (float) (getc(in)) / scale;
        }
        }
NoRings = SeS / 2 + SeS % 2;
for (y = 0,pat_no = 0;y < (rh.ras_height -SeS);y++) {
        for (x = 0;x < (rh.ras_width - SeS );x++,pat_no++) {
                if (rand() % RSample == 0) {
                        NoPatternsUsed++;
                        fprintf(out,"*p%d ",pat_no);
                        for (Ring = 0;Ring < NoRings;Ring++) {
                              j = y + Ring;
                        for (i = x + Ring;i < x + SeS - Ring;i++) {
                                    temp = image[j * rh.ras_width + i];
                                    fprintf(out,"%f ",temp);
                              }
                                    fprintf(out,"\n");
                        for (++j;j < y + SeS - Ring;j++) {
                                    temp = image[j * rh.ras_width + i];
                                    fprintf(out,"%f ",temp);
                              }
                                    fprintf(out,"\n");
                        for (--i,--j;i > x + Ring;i--) {
                                    temp = image[j * rh.ras_width + i];
                                    fprintf(out,"%f ",temp);
                              }
                                    fprintf(out,"\n");
                        for (j--;j > y + Ring;j--) {
                                    temp = image[j * rh.ras_width + i];
                                    fprintf(out,"%f ",temp);
                              }
                                    fprintf(out,"\n");
                        }
                        fprintf(out,"\n");
                        }
                }
        }
printf("\n%d patterns used",NoPatternsUsed);
fprintf(out,"\n");
```

```
fclose(in);
fclose(out);
}
```

```
/* tp2r.c */

/* translates training pattern for PDP to recall only, ie gets rid of target vectors to save room */

#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *in,*out;
int i,j,result,no_inputs;
char str[40];

/* working */
if (argc != 4) {
        printf("\nMaking PDP file with only inputs from one with inputs and desired outputs");
        printf("\nFormat tp2r <infile> <outfile> <No. Inputs>\n");
        exit(0);
        }
no_inputs = atoi(argv[3]);
if ((in = fopen(argv[1],"rb")) == NULL) {
        printf("\nFailed to open %s\n",argv[1]);
        exit(0);
        }
if ((out = fopen(argv[2],"wb")) == NULL) {
        printf("\nFailed to open %s\n",argv[2]);
        exit(0);
        }
i = j = 0;
while ((result = fscanf(in,"%s",str))
!= EOF && str[0] != 'p' && str[0] != 'P')
        ;
while (result != EOF) {
      printf("\nPattern %d being transferred with %d inputs",j,no_inputs);
        j++;
      fprintf(out,"%s ",str);
        for (i = 1; i <= no_inputs;i++) {
                result = fscanf(in,"%s",str);
                fprintf(out,"%s ",str);
        if ((i % 10) == 0 && i > 0)
                        fprintf(out,"\n");
                }
        while ((result = fscanf(in,"%s",str))
                != EOF && str[0] != 'p' && str[0] != 'P')
                                ;
/* because there may be a target vector which we are not interested in */
        fprintf(out,"\n");
        }
fclose(in);
fclose(out);
}
```

Appendix 4 Medical Imaging System

To manipulate and pre-process images an imaging system was built. For speed and compatibility with other programs the language C was chosen. Most image processing systems, ANNs and GAs are built using C. The source code is shown in Appendix 1. This document describes the functions in the package.

1. Main Subroutines

A set of menus are displayed, which may be nested to arbitrary levels, and in which the user chooses an item by use of the cursor key and the RETURN key. All functions which require a filename use a function which displays the files available, allowing directory changes, and selection of the file by use of cursor keys and RETURN key. The mouse is supported as an alternative to cursor keys and RETURN key.

2. Graphics Functions

Images acquired as frame grabbed or directly transferred from the gamma camera may be acquired and stored with a header indicating the dimensions of the image. Median filtering is done to reduce the image where necessary. Images may be displayed in the appropriate mode (EGA, VGA etc) automatically, and 256 VGA is supported by directly writing to screen memory. Except in VGA 256 colours, where one screen is allowed, 4 screens are available so that the user may have up to 3 images ready for display which may be displayed by pressing a number from 1 to 3. Screen 0 is reserved for menu displays, directory information etc.

Images may be displayed as grey scale, red-green-blue or hot body scale.

3. Pre-processing Functions

The following filters have been implemented :-

1. Median
2. Sobel
3. Arbitrary digital filters using a 3 by 3 spatial domain filter.

Standardisation of images or image segments is effected using a variable size rectangle median filter technique.

4. File Conversion Functions

As files may be required in a variety of formats for mathematical analysis, or input to ANNs etc, a suite of transfer functions was provided. Images or portions of images may be transferred into various ANN (PDP, Rochester, NeuralWare) packages. Files may be specified as wild cards or single files. Raster scanning and random selection of raster scans are supported.

5. Image Manipulation

Images may be reflected in X or Y axes, or turned around, or upside down. Image expansion is effected by copying pixels. Rectangular areas may be inserted from other images, or saved to a new filename. A subtraction of one rectangular area with another is achieved by XORing a moving rectangle with the image behind it. This was used to obtain perfusion minus ventilation scans. Segmentation of images is achieved by the technique described in Castleman[1]. Thresholding of images using upper and lower thresholds is allowed. Contouring is available.

6. Utilities

Many small but useful housekeeping functions were created, e.g. to check a variable is not pointing to NULL, checking file existence, setting graphics modes etc.

References

1. Castleman K.R, *Digital Image Processing*, Prentice Hall, 1979.

Appendix 5 Information Taken from Experts in Nuclear Medicine

To build an expert system the relevant rules must be elicited from experts. This was attempted in several visits to medical centres, the details of which are given below.

1. Walsgrave Hospital

A series of lung scans were observed with Mr. John Barham, a medical physicist, and he stated reasons why scintigrams were considered normal or otherwise. The following rules were complied based on these statements :-

1. Lungs should come to a point in the posterior oblique view.
2. Outlines of the lung should be smooth.
3. Lungs should come down to the same level.
4. A reduction in lung area visualised up from the base or down from the top of the lung is abnormal.
5. A reduction in activity inside a lung is abnormal.
6. A defect which is matched on ventilation and perfusion is not a PE
7. A defect which is present in perfusion but not ventilation is a PE.
8. A defect which is present in ventilation but not perfusion is not a PE but might be COAD.
9. A slight diminution of activity in perfusion not matched in ventilation but not seen in any other view is not a definite PE.
10. A PE is usually wedge shaped.
11. Matched raised lung defects are probably effusions.
12. Matched unequal lungs may be old PEs or carcinoma.
13. There should be a straight line across the base of the lungs in the posterior view.
14. An apparent defect in the area of the heart is probably cardiac shadow. If this is seen in oblique view then the anterior and posterior views should be checked to confirm/deny origin as cardiac shadow.
15. An enlarged heart gives a larger cardiac shadow.
16. A white line down the mid-line of a lung is caused by the spinal column, sternum and oesophagus, and is normal.
17. Large breasted females give reduced lower lung uptake.
18. Patchy matched defects or indentations suggest COAD.
19. An indentation may be a defect.
20. A defect in the centre of the lung which is matched may be an enlarged hilium.
21. An apparent matched defect in the top of the lungs may be a gravitational artifact.
22. A matched mid-line ragged edge may be spinal curvature.

2. Visit to Queen Elizabeth II Hospital, Edgbaston, Birmingham,

Further rules were elicited from the Dr. Burroughs, a consultant radiologist specialising in nuclear medicine.

23. A perfusion defect unmatched in ventilation is a vascular defect which is usually but not always a PE Other rarer possibilities include vasculitis, collagen diseases (e.g. systemic lupus erythematous (SLE), polyarteritis nodosa). Non PE defects are only a small minority of cases.
24. A wedge shaped, peripheral perfusion defect with a clear ventilation study is definitely a PE

25. Pulmonary arteriography is used to confirm diagnoses sometimes.
26. About 75% of diagnoses may be made with confidence. This is similar to the estimate of typical nuclear imaging of about 70% given in 1.
27. Mediastinal obstruction causes some 2 cm. to be lost on X-Ray view.
28. Early fluid imbalance is not seen on X-Ray, e.g. pulmonary oedema.

Possible uses of lung scans in addition to diagnosing PE suggested include :

i) Tumours - affect ventilation or perfusion.

ii) Inflamatory states - pneumonia, infection in immunosuppressed patients (e.g. transplants).

Incidental findings of lung scans include :

i) COAD

ii) Abnormal ventilation pattern, which may or may not have clear evidence on X-Ray.

It is noted that the rules given are sometimes inconsistent. For example the rules 7 and 23. In these cases it appears that a rule has been given which is usually correct, but under some circumstances fails. In the 7 versus 23 contradiction, it is seen that unmatched perfusion defects are not always PE, but usually they are, so rule 7 is generally correct, but fails in rare cases.

References

1. Niemann H, Bunke H, Hofmann I, Sagerer G, Wolf F, and Feistel H, "A Knowledge Based System for Analysis of Gated Blood Pool Studies," *IEEE Transactions*, vol. PAMI-7:3, pp. 246-259, 1985.