HIERARCHICAL NEURAL NETWORK SYSTEM FOR IMPROVING EDGE DETECTION

ANTHONY WING KAY SZETO, B.Sc.

HIERARCHICAL NEURAL NETWORK SYSTEM FOR IMPROVING EDGE DETECTION

By

©Anthony Wing Kay Szeto, B.Sc.

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science

Department of Computer Science

Memorial University of Newfoundland

April, 1991

St. John's Newfoundland Canada

ISBN 0-315-78131-9

ABSTRACT

This thesis presents a hierarchical neural network system for improving the edge measurements obtained by an edge operator. The neural network system is designed to adjust the edge measurements based on the information provided by neighbouring edges. The adopted strategy is to analyze the local edge patterns to determine and reinforce edge structures while suppressing unwanted noise and false edges. The hierarchical neural network system is made up of four levels of subnets. The subnet in the first level consists of high-order neural nets to determine the potential adjustment on the element of interest by detecting edge contours according to the selected processes in the neural nets and the input local edge pattern. The second level consists of a cooperative-competitive neural net model to determine the orientation of the strongest edge contour in the local edge pattern. The subnet in the third level consists of two types of neural net models. A high-order neural net ascertains the conditions for adjusting the gradient magnitude and determines the amount of adjustment to the gradient magnitude. A semilinear feedforward net is used to compute the new adjusted gradient magnitude and determines if the element of interest is to be an edge element or a non-edge element. The subnet in level four is a semilinear feedforward net which is used to determine the new orientation for the element of interest. A fast learning algorithm is developed to derive suitable weights for the neural nets to perform efficiently and correctly. Using the hierarchical neural network system for each element in the image, highly parallel processing can be achieved. An iterative approach incorporated into the neural network system has also enabled the application of global analysis in the process of adjusting the edge measurements. As a result, the final edge measurements are more accurate.

ACKNOWLEDGEMENTS

I wish to express my thanks to my supervisor Dr. Siwei Lu for his guidance, interest, constructive criticism and enthusiasm. Without his contribution, it would be impossible to give this thesis its current quality.

I would like to thank the Systems Support staff for providing all the help and assistance during the conduct of my research.

I am also very grateful to the Administrative staff who have helped in one way or another in the preparation of this thesis.

In addition, I would like to acknowledge the financial support received from the Department of Computer Science and the School of Graduate Studies.

Special thanks are due to my fellow graduate students and good friends, and in particular to Todd Wareham, Ren Ying, Sean Hogan and Helmut Roth for their valuable comments and useful suggestions. I would also like to thank Prof. Jane Foltz and Patricia Murphy for their help and assistance.

At last, but not least, I would like to thank Pang Li Lee for her constant encouragement that kept me going throughout my graduate studies, especially during the preparation of this thesis.

This thesis is dedicated to my parents for their support and encouragement throughout the course of my education.

Table of Contents

List of Figures

Chapter 1

Introduction

An important problem in image processing is the detection of edges in a given image. Edges are the consequences of changes in some physical and surface properties, such as illumination, geometry (orientation or depth) or reflectance. An edge image conveys most of the important scene information as there are direct correlations between the edges and the physical properties of a scene. Edge detection is an essential part of many computer vision systems as it plays a key role in early processing. The edge detection process simplifies the analysis of images by drastically reducing the amount of data to be processed, while at the same time preserving useful and important structural information about object boundaries. It is hard to over-emphasize the importance of edge detection in image understanding. Most modules in a vision system depend, directly or indirectly, on the performance of the edge-detector. Edge detection techniques have various applications such as pattern recognition, robot scene analysis, and image coding.

Accurate edge detection is a difficult task [Peli and Malah 1982; Ballard and Brown 1982_a] and there has been a substantial effort to develop the 'ideal' edge detectors or operators. However, each of these edge operators usually embody specific edge models and may perform best only under special circumstances. For example, some operators may find most edges but also respond to noise, while others may be noise-insensitive but miss some crucial

edges. Hence, most edge operators will generally produce imperfect results. Therefore, considering the overall diverse applications for edge detection and the performance of current edge operators, it is better to improve the results of an edge operator rather than to develop the 'ideal' edge operators.

There are different techniques for improving edge operator measurements. Techniques using relaxation labeling are restricted by the limited capacity to accurately represent edge-processes and different labelings. As a result, these techniques lack the ability to detect many different and more complex edge patterns. Some techniques use context information from the image but they lack accuracy as strong noise is also enhanced along with valid edges. Other techniques using contour finding to improve the detected edges perform poorly in noisy images.

As current techniques have limited capability to perform good and accurate edge detection for noise corrupted and degraded images, this thesis presents a new technique for improving the edge measurements. Edge measurements include: (1) the amount of edge strength ascertained by measuring the degree of abruptness in changes in intensities along the edge and (2) the directions of these changes in intensities. A hierarchical neural network system is proposed to accomplish the following tasks : (a) to reinforce or enhance true edges; (b) to recover missing edges; (c) to suppress false, spurious edges; and (d) to eliminate noise. The neural network system is able to achieve four very important objectives :

1. Different types of edge contours (e.g. curvature, linear, corners, etc.) can be accurately detected. An edge contour is the outline that defines an edge. An edge structure on

the other hand is an edge construction through the arrangement of the subcomponents (edge elements) of an edge.

2. More global information is made available for accurate edge detection, interpolation and noise elimination.

3. Highly parallel processing enables high computation speed for real-time application.

4. The designs and architectures of the neural network system enable fast learning and good generalization.

The adjustment of the edge measurement is formulated in a manner such that an edge element may have its gradient magnitude and orientation iteratively altered in agreement with its local surroundings. Hence, the edge image (consisting of edge elements) is updated after each iteration and the new values are fed back to the neural network system for further processing. This iterative approach enables the utilization of global information as the information is 'propagated' to surrounding elements in the edge image after each iteration.

1.1 Organization of the System

The flowchart in Fig. 1.1 gives an overview of the proposed system. Initial edge measurements are obtained using the Sobel operators [Ballard and Brown 1982$_a$]. A global threshold algorithm is used to distinguish edge elements. The information (the orientation and the gradient magnitude) is input to the hierarchical neural network system which iteratively adjusts the edge measurement of each element until convergence is attained. That is, true

edges are detected and enhanced, missing edges or edges not detected by the edge operators are recovered, false and spurious edges detected by the edge operators are suppressed and traces of noise eliminated.

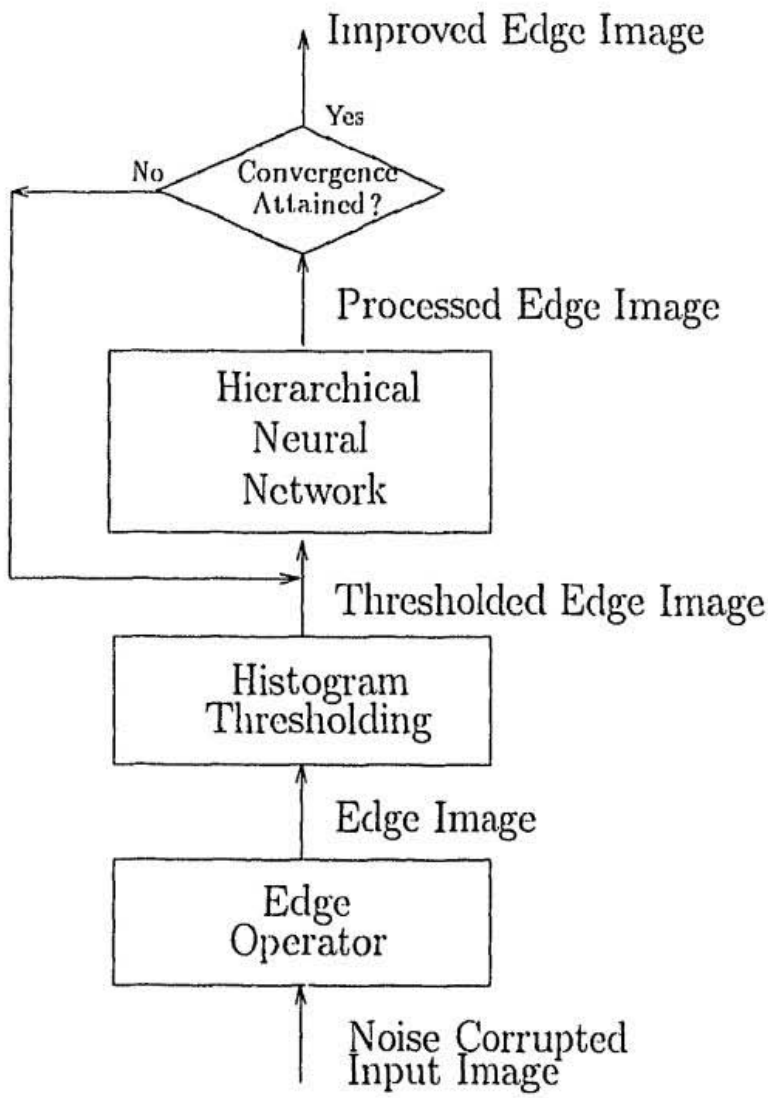


Figure 1.1 An overview of proposed system

1.2 Structure of the Hierarchical Neural Network System

The neural network system consists of four major subnets which perform specific subtasks in order to accomplish the overall task of improving the edge measurements. The interconnections between these subnets are fashioned in a hierarchical manner. The subnet in level one detects the strengths of edge contours according to the gradient magnitudes and orientations of neighbouring edge elements in the local edge pattern. In level two, the subnet determines the orientation of the strongest edge contour in the local edge pattern using the input from the subnet in level one. The subnet in level three uses the information provided by the subnet in level two to adjust the gradient magnitude of the edge element. It will also signal to the subnet in level four to modify the orientation of the central element. The subnet in the final level determines the new orientation for the edge element with the information from the subnets in levels two and three. The hierarchical structure of the neural network system is illustrated in Fig. 1.2.

Each subnet is made up of one or more layers of nodes. The hierarchical neural network system has forward and/or lateral connections between nodes at each level or layer. In the forward connections, the output of nodes at each level (or layer) serves as input to the nodes on the next level (or layer) in the neural network. In the lateral connections between nodes, the emphasis is on lateral inhibition or excitation between nodes on the same layer. The first layer of the subnet in level one is the receptor layer to receive the input data.

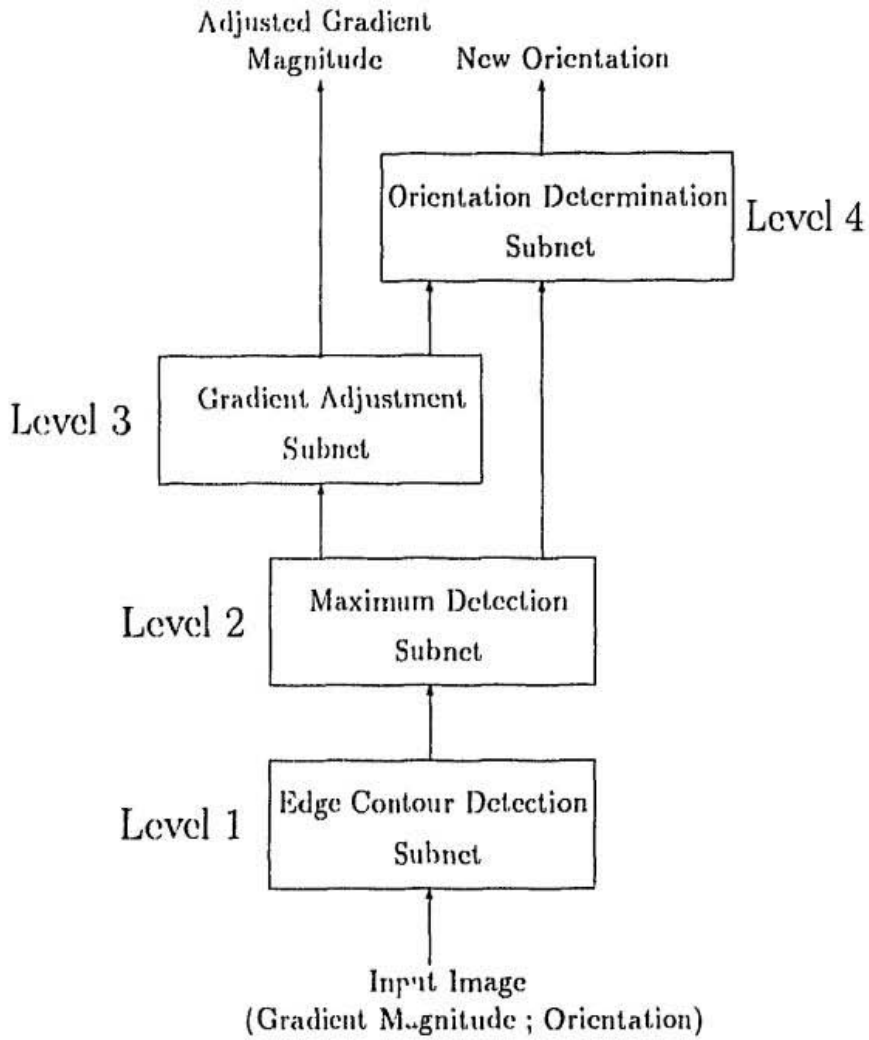


Figure 1.2 Hierarchical structure of neural network system

The output of the subnet in level three is the new adjusted gradient magnitude of the edge element and the signal indicating whether the element is an edge element or a non-edge element. The output of the subnet in level four gives the new orientation for the edge element.

1.3 Organization of the Thesis

This thesis is organized into nine chapters. Chapter two introduces existing techniques for edge detection and enhancement and also reviews the current research and applications of different neural network models. Chapter three describes how the input data to the hierarchical neural network system are generated and formatted. In Chapter four, the design, architecture and mechanism of the subnet in level one of the network are described. The function and purpose of the subnet are also discussed. Chapter five describes the subnet in the second level. The design and concept are adopted from the analogy to the biological neuron. The third subnet in the neural network is described in Chapter six. This subnet consists of a high-order functional linked neural net for deciding the appropriate conditions for edge measurement adjustments and a semilinear feedforward net for modifying the gradient magnitude. Chapter seven describes a semilinear feedforward net for determining the orientation of the edge element. A fast learning algorithm for choosing suitable weights of the neural nets is described in Chapter eight. The different situations are analyzed to find out whether training for the neural nets is required or not. Chapter nine gives the conclusions and possible directions for further research.

Chapter 2

Survey of Edge Detection Techniques and Neural Networks

2.1 Introduction

In this chapter, a brief overview of edge detection techniques, edge enhancement techniques and neural networks is presented. This discussion includes the principles adopted in these techniques, their strengths and weaknesses, as well as focusing on techniques which utilize edge information to enhance the edge image. Finally, the design concepts and applications of the different neural network models are discussed.

2.2 Edge Detection Techniques

There has been tremendous research in the area of edge detection in an attempt to create the 'ideal' edge operator. The following are reviews of the typical edge detection algorithms.

2.2.1 Differential Operators

Most of the earlier edge detection techniques employed first order difference operators. Differential operators include the Roberts 2×2 pixel operator, the Prewitt 3×3 pixel operator and the Sobel 3×3 pixel operator [Ballard and Brown 1982_a]. These operators set

suitable weights over a convenient neighbourhood size to estimate slopes in the 'x' and 'y' directions. First order differential operators are fast edge detection operators. They could sharpen the edge contours, but also inadvertently enhance the noise. The Laplacian of Gaussian operator (Marr-Hildreth method) is used to detect edges at the locations of the zero crossings [Marr and Hildreth 1980]. However, zero-crossings do not always correspond to edges. The Marr-Hildreth method also has poor localization properties and introduces a bias in the edge location estimation [Berzins 1984; Nalwa and Binford 1986]. The smoothing operation with the Gaussian mask tends to blur weak edges [Haralick and Lee 1990], and furthermore, the presence of impulse noise in transmitted images can seriously degrade the performance of the smoothing operator. Another differential operator, Canny edge operator [Canny 1986], uses the first derivative of the filtered image function as its basis for edge detection. Optimal edge operators are then derived for different edge profiles, for example, step edge or ridge edge. These operators are optimal in the sense of jointly maximizing the signal-to-noise ratio and a localization criterion with constraint on multiple responses. Here, smoothing is used to offset the effects of noise before edge detection. The effects of smoothing blurs weak edges. Another drawback involves returning false edges on smoothly shaded surfaces.

2.2.2 Template Matching Edge Operators

The popular edge template matching operators are the Kirsch masks [Ballard and Brown 1982_a], the Robinson masks [Robinson 1977], the Nevatia-Babu masks [Nevatia and Babu

1980], and the Compass Gradient masks [Park and Choi 1989]. By determining the largest response for a set of masks, the edge orientation and magnitude can be rapidly estimated. However, template mask methods give rise to large angular errors and do not give correct values for the gradient. Another type of template matching technique is based on the sum of absolute errors [Strickland, Draelos and Mao 1990]. This technique is effective in detecting edges where the form of the edges to be detected is known. However, there are disadvantages with this technique. Firstly, the technique requires smoothing to remove noise before the edge detection, which affects the detection of weak edges. The performance is also affected if impulse noise is present in the image. Secondly, the form of the edges to be detected must be known in advance. This method also involves the expensive pixel-by-pixel comparison in the image and in the template.

2.2.3 Image Filtering Techniques

Linear filtering techniques were some of the earliest filtering techniques used for edge detection [Modestino and Fries 1977]. In this technique, a stochastic model of edge structure is proposed and the edge detection problem is formulated as one of least mean-square spatial filtering. Edges in noisy digital images are detected using two-dimensional recursive digital filtering. In addition to the noise immunity, the recursive nature of the filtering operation leads to significant computational economies. However, linear filtering techniques are generally very complex and have achieved only moderate success. Nonlinear filtering (e.g. median filtering, order statistics filtering, and nonlinear mean filtering) is able to remove certain

kinds of noise better (e.g. impulse noise) and preserves edge information. The nonlinear filters measure the mean of the luminance. If the dispersion of the luminance within the filter extent is greater than a certain threshold, the center of the filter extent is declared as the edge point [Pitas and Venetsanopoulos 1986]. Such edge detectors have good characteristics only in the presence of uniformly distributed noise. A class of median-type filters [Neuvo, Heinonen and Defee 1987; Neuvo, Nieminen and Heinonen 1987] combines the output of a number of linear spatial filters and the median operation to detect the edges. The computational overhead is low. The filters are capable of retaining edges and removing noise. However, the lines in two-dimensional images do not survive the filtering process [Saito and Cunningham 1990].

2.2.4 Statistical Techniques

A statistical classification technique [Kundu 1990] is used to detect the step and linear edges. This technique is based on two characteristics of natural edges : (1) the pixels near the step or linear edges can be classified into two nearly equal groups with different average intensity values, and (2) the members of each group show strong spatial correlation. The edges are located at points where both these conditions are satisfied with strong statistical evidence. The weak edges are not blurred because no smoothing is involved. However, not all edge elements are detected. Another technique, based on the likelihood ratio test in a given small neighbourhood, derives a decision rule to decide whether there is an edge, a point, a corner edge, or just a smooth region [Huang and Tseng 1988]. More decision rules can

be derived for more complicated situations (neighbourhoods), but they are computationally expensive.

2.3 Edge Improvement (Enhancement) Techniques

There are different techniques for improving the 'raw' edge information obtained by the edge detection. The following is a summary of these techniques for improving the edge image.

2.3.1 Relaxation Labeling

Probabilistic relaxation is a technique for labeling image entities. It relies on iteratively updating the distribution of available probability over a label set. A support function combines evidences from the context-conveying neighbourhood and incorporates prior knowledge of the structure of the labeling task in-hand. One of the earliest probabilistic relaxation labeling method was introduced by [Zucker, Hummel and Rosenfeld 1977; Hummel and Zucker 1983]. It requires the setting of many compatibility weights. The updating process employs only a single formula for all the various, different edge patterns. Convergence can be very difficult since many variables must be optimized simultaneously. With the heuristic nature of the update procedure, there are often internal inconsistencies in the specification of the relaxation scheme, and a limited capacity to accurately represent edge-processes. Another version of relaxation labeling [Prager 1980] allows for more complicated adjustment formulas but only six immediate (adjacent) neighbouring pixels are considered. Hence, it does not provide sufficient information for a more accurate indication of the presence of an edge or

noise. An improved application of probabilistic relaxation to edge labeling [Hancock and Kittler 1990] uses a representation of the edge-processes. By specifying the probabilistic framework used to represent the world model, internal consistency is ensured. For each object, prior knowledge of the structure is represented by a dictionary of labeling possibilities for the entire context-conveying neighbourhood. Restrictions on the representational capacity of the scheme are avoided. This dictionary-based approach is capable of enhancing edge structures in the presence of noise without filters, but the dictionary can become very large as the application task is made more complex. Another type of probabilistic relaxation is based on an automaton approach [Mandayam, Thathachar and Sastry 1986]. The probability updating is accomplished through learning automata. For different types of situations, the learning algorithm can be chosen depending on the problem at hand [Thathachar and Sastry 1985]. The learning automata algorithms are very simple to perform and hence they can be hardware implemented. However, their ability to accurately represent the different labelings is very limited. A stochastic relaxation scheme [Geman and Geman 1984] is aimed at incorporating observational information by regarding the labeling task as maximum *a posteriori* probability estimation. This scheme adopts a Bayesian approach to a 'hierarchical' stochastic model based on the Gibbs distribution. A new restoration algorithm for computing the maximum *a posteriori* (MAP) estimation of the image is based on stochastic relaxation and annealing. The scheme generates a sequence of images that converges in an appropriate sense to the MAP estimation. The algorithm is highly parallel and exploits the equivalence between Gibbs distributions and Markov random fields. If convergence is

slow, the relaxation scheme is computationally expensive because many images have to be generated.

2.3.2 Context Dependent Edge Detection

Based on local edge coherence, context information of the whole image is used in the edge detection process [Haralick and Lee 1990]. The edge evaluation is formulated as a Bayesian decision problem. The monotonically increasing paths begin at any pixels located at boundaries in the image above the selected pixel, pass through the selected pixel, and end at some pixels located at boundaries below the selected pixel. A pixel is assigned to an 'edge' state if the edge probability of the best 'edge' path is higher than the average probability of the best 'no-edge' paths. This technique is computationally less expensive than other edge detection techniques using relaxation labeling. However, the path with the highest probability might not correctly depict an 'edge' path as the pixels in the path could be the locations of very strong noise rather than edges. This technique considers only the pixel values on the path, without considering if those are 'edge' pixels or 'noise' pixels. Many paths can be generated but not all of these paths represent valid 'edge' paths.

2.3.3 Detecting Edges by using Multiple Scales

An algorithm for finding a single good path through the set of edge points detected using the gradient of the Gaussian operator was proposed by [Williams and Shah 1990]. The algorithm uses one scale for finding contours and then extends to multiple scales to produce

improved detection of weak edges. A weight assigned at each edge point is based on four factors : a measure of noisiness, a measure of curvature, contour length, and the gradient magnitude. The edge point with the largest average weight is chosen as the edge point on the contour. In the multiple scale algorithm, the search for a contour proceeds as for the single scale, using the largest scale to locate the best partial contour, then followed by the next smaller scale to locate the next best partial contour. The algorithm is able to improve detection of edges that are close together and interacts at scales which are large enough to remove noise, as well as also improving the detection of weak edges. However, the algorithm is not able to detect edge elements that are apart. Hence, these edges will be lost as the algorithms are not able to interpolate well. Since the algorithm is able to detect weak edges only if they are well-defined, the not well-defined valid weak edges will be lost.

2.3.4 Contour Tracing

Contour tracing from a set of edge points is tackled by a combination of edge linking, polygonal approximation, thinning, and neighbourhoods to a contour segment [Bell and Pau 1990]. A contour is stored by encoding the direction from one edge element to its neighbour, using an eight-valued Freeman chain code scheme. To generate edge contours, an edge image is recursively processed until the contour is closed or until all pixels on the edge have been processed. A false isolated edge contour could be mistaken for a valid edge contour as each edge element is individually processed and several neighbouring edge elements generated by strong noise may have the same orientation by chance. A hierarchical approach for fast par-

allel processing of chain-codable contours [Meer, Sher and Rosenfeld 1990] is based on the chain pyramid for extracting and analyzing contours. The chain pyramid employs the chain code representation concept [Freeman 1974] and the contours are represented as linked lists. A local connectivity algorithm is used to generate the linked lists representing the contours. A probabilistic allocation algorithm is then used to label the string of contour pixels. A gap bridging algorithm is then used to fill the gaps in the contours. The technique allows fast parallel processing of the contours, however, it cannot deal with noise or contours that are more than three pixels wide. Therefore, in order to use the chain pyramid technique, preprocessing must be done to eliminate noise and the contours must be thinned.

2.4 Neural Networks

There has been increasing interest in artificial neural networks (neural nets), due to new net topologies, learning algorithms and massive parallelism [Lippmann 1987]. Neural net models explore many computing hypotheses simultaneously using massively parallel nets composed of many computational elements (nodes) connected by links with weights. A neural net is specified by the net topology, node characteristics, and the training or learning rule. The potential benefits of neural nets extend beyond the high computation speed provided by massive parallelism. Neural nets also provide a great degree of robustness or fault tolerance because there are many processing nodes. Damage to a few nodes or links may not affect the overall performance of the net significantly.

2.4.1 Multi-Layer Neural Nets

Currently, multi-layer nets are one of the most popular neural net models. These nets employ hidden nodes connected to both the input and output nodes [Rumelhart, Hinton and McClelland 1986]. The output from the nodes in each layer is fed to the nodes in the next layer through weighted feedforward interconnections. The capabilities of multi-layer nets stem from their abilities to form complex decision regions and to be trained [Rumelhart, Hinton and Williams 1986]. Two-layer nets can be trained to form both convex and disjoint decision regions, and three-layer nets can be trained to form arbitrary complex decision regions [Huang and Lippman 1988]. A convex decision region is a region whereby the boundary between the distributions of populations is smooth and simple. Conversely, a complex decision region has a complex boundary. disjoint decision regions are decision regions which are partitioned and do not overlap. However, there are problems with a multi-layer net. Firstly, when complex decision regions are required, convergence time can be excessively long. Hence, learning in a multi-layer net is slow. Secondly, the number of nodes must be large enough to form a decision region. However, it must not be so large that the weights cannot be reliably estimated from the available training data. Therefore, each net is only capable of performing a specific task and any expansion of the original task requires an extensive modification to the structure of the net. Another problem is the difficulty in providing the hidden nodes with a training signal. There are a number of common applications for multi-layer neural nets such as classification [Widrow, Winter and Baxter 1988; Gupta, Sayeh and Tammana 1990; Khotanzad and Lu 1990], modeling biological compensatory eye

motion [Fanelli, Raphan and Schnabolk 1990], feature extraction [Linsker 1988; Dupaguntla and Venuri 1989], and pattern/character recognition [Kammerer and Kupper 1988; Waibel, Hanazawa, et al. 1989; LeCun, Boser, et al. 1989].

2.4.2 High-Order Neural Nets

There are various ways of incorporating high order effects into a neural net before input to the node : 'sigma-pi' units [Rumelhart, Hinton and McClelland 1986], 'meta-connections' [Pomerleau 1987], and nonlinear combinations of pattern elements [Klassen, Pao and Chen 1988]. High order involves the simultaneous utilization of the inputs for processing by the neural net. The major drawback of the high-order approach is the combinatorial explosion of high-order terms [Minsky and Papert 1988]. However, there are various methods for dealing with this problem : prior restrictions of high-order terms, reduced interconnections, and using prior knowledge of the problem domain to select only those terms which are useful. High-order neural nets have impressive computing, storing, and learning capabilities [Giles and Maxwell 1987], while maintaining simple architectures [Pao and Beer 1988]. High-order neural net incorporating nonlinear links has shown fast learning capability and also enhanced computational capability due to the generation of enhanced combinations of features [Sobajic 1988]. The memory capacity of a high-order net is also improved through the introduction of higher order memory functions which enhance the pattern discriminating capability of the neural net [Lee, Doolen, et al. 1986]. Despite the lack of more comprehensive simulations, the empirical results show that higher dimensionalities and higher-ordered correlations in the

architecture of the neural net do indeed improve the memory capacity of the net [Simpson 1990].

2.4.3 Cooperative-Competitive Neural Nets

A cooperative-competitive neural net consists of laterally interconnected nodes. The interactions in a cooperative-competitive neural net include positive interaction (cooperative) from a node to itself and negative interaction (competitive) from a node to its neighbours. A cooperative-competitive net can be used as a content-addressable memory (associative memory) [Lapedes and Farber 1986; Xu and Tsai 1990]. The Hopfield net [Hopfield 1984; Hopfield and Tank 1986] is a cooperative-competitive net designed specifically as a content-addressable memory. When cooperative-competitive neural nets are used as content-addressable memory, the number of patterns that can be stored and accurately recalled is severely limited by the large number of nodes required for the recognition of a relatively small number of patterns. Cooperative-competitive neural nets have found applications in recognizing multiple groupings of data [Cohen and Grossberg 1987], weak patterns in noisy inputs [Roth 1989], and recognizing categories [Carpenter and Grossberg 1987,1990].

2.4.4 Structured Neural Nets

Structured neural networks are a relatively new approach to neural net model design [Feldman, Fanty and Goddard 1988]. A structured neural net model can be viewed as a synthesis of two traditionally opposed approaches to artificial intelligence. Some early AI in-

vestigators focused on the parallelism and robustness of biological brains and explored ways of generating this high performance in non-biological networks. The other group concentrated on the detailed structure of tasks and algorithms and expressed them in conventional computer notation. Structured neural net models attempt to capture the best of both paradigms.

2.4.5 Complex Neural Nets

There are other neural net models with more complex architectures [Fukushima 1988; Carpenter and Grossberg 1988,1990]. These models are composed of highly parallel building blocks that are interconnected to construct highly complex systems. These systems are usually massively parallel and engaged in a multilevel or hierarchically fashion. Each of these nets consists of many layers of nodes. The net has forward, backward and/or lateral connections between nodes. Some of the connections are variable while others are fixed. The variable connections are trained to enable the nodes to acquire the ability to learn to perform correctly. Some of the connections are excitatory while other connections are inhibitory. Complex neural net models are specific in their tasks and can perform very well for their assigned tasks, but these nets lack modularity. Enhancement to the neural net usually involves extensive modifications to the structure.

Chapter 3

Generating Local Edge Pattern for Neural Net Input

3.1 Introduction

In this chapter, the generation of the edge image from the original gray-level image is described. An edge element is a pixel in a small area where the local gray-level values are changing rapidly in a monotonic way. These edge elements collectively construct edge contours. Therefore, an edge image is an edge map of the original gray-level image. An edge operator is able to detect the presence of a local edge element by computing its gradient magnitude and determining its orientation. The gradient magnitude and the orientation of the edge element can in turn be used to improve the detected edges, interpolate missing edges, or remove noise and false edges.

3.2 Computing the Edge Measurements

The edge image is generated from an original input image by using the Sobel operator [Duda and Hart 1973]. The Sobel edge operator computes the magnitude and the direction of maximal gray-level change. The Sobel operator is designed to approximate the discrete gradient function by computation of the appropriate horizontal and vertical components.

It can be regarded as a combination of two gradient masks (Figs. 3.2a, 3.2b), one for the horizontal direction and the other for the vertical direction.

a	b	c
d	e	f
g	h	i

Figure 3.1 Elements in a 3x3 window

-1	0	1
-2	0	2
-1	0	1

a) The horizontal component

-1	-2	-1
0	0	0
1	2	1

b) The vertical component

Figure 3.2 Sobel masks

The gradient magnitude is obtained from the two orthogonal mask outputs. The horizontal and the vertical components are denoted by 'S_x' and 'S_y', respectively and are defined by (Fig. 3.1) :

$$S_x = (c+2f+i)-(a+2d+g), \qquad (3.1)$$

$$S_y = (g+2h+i)-(a+2b+c). \qquad (3.2)$$

The gradient magnitude at the central point, 'g_e', is defined by :

$$g_e = \sqrt{S_x^2+S_y^2}. \qquad (3.3)$$

The direction at the central point (denoted by 'θ_e') is determined by :

$$\theta_e = \arctan\left(\frac{S_y}{S_x}\right). \qquad (3.4)$$

The direction at point 'e' is coded into eight principal orientations according to Fig. 3.3, where 'n' denotes north, 'nw' denotes north-west, 'se' denotes south-east, etc.

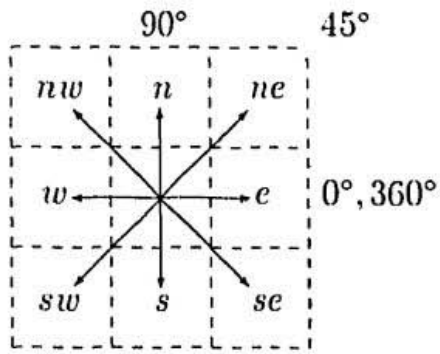


Figure 3.3 Eight principal orientations

The orientation of each edge element is represented by a set of direction values :

$$\{d^{(n)}, d^{(s)}, d^{(e)}, d^{(w)}, d^{(ne)}, d^{(sw)}, d^{(nw)}, d^{(se)}\}, \tag{3.5}$$

where the superscripts denote the principal orientations.

Each element in the set of direction values has binary value '0' or '1', where the value assigned is dependent on the orientation of the edge element. For example, if an edge element has a north orientation, the direction value '$d^{(n)}$' is set to '1', and the other direction values, namely, '$d^{(s)}, \ldots, d^{(se)}$' are set to '0'. Hence, the set of direction values for this edge element is :

$$\{d^{(n)}, d^{(s)}, d^{(e)}, d^{(w)}, d^{(ne)}, d^{(sw)}, d^{(nw)}, d^{(se)}\} = \{1,0,0,0,0,0,0,0\}. \tag{3.6}$$

For a non-edge element, since the element does not have an orientation, the set of direction values for the element consists of '0's. That is :

$$\{d^{(n)}, \ldots, d^{(se)}\} = \{0, \ldots, 0\}. \tag{3.7}$$

3.3 Thresholding the Edge Image

Thresholding is used to simplify an image while retaining information about shapes and geometric structures. Thresholding is performed on the gradient magnitude in order to determine non-edge elements.

There are two kinds of thresholding : bilevel and multilevel. In the bilevel thresholding [Ostu 1978; Kittler and Illingworth 1986; Abutaleb 1989], the histogram of the image is bimodal and the threshold is chosen as a value between the peaks of the two distributions. In the multilevel thresholding [Wang and Haralick 1984; Boukharouba, Rebordao and Wendel 1985; Hertz and Schafer 1988], the histogram has several peaks, and the values of the thresholds are set to separate these peaks.

The thresholding technique used in this thesis is a global threshold algorithm. This algorithm searches for the valley between two peaks in the histogram of gradient magnitude. The valley between the first and second peaks determines the threshold value 'T' for distinguishing non-edge elements from edge elements (Fig. 3.4). An element with a gradient magnitude 'g_{e_i}' greater than or equal to the threshold value 'T' is initialized to an edge element. Otherwise, the element is initialized to a non-edge element. That is, if $f(e_i)$ is the threshold image then

$$f(e_i) = \begin{cases} g_{e_i} & \text{if } g_{e_i} \geq T \\ g_0 & \text{otherwise,} \end{cases} \qquad (3.8)$$

where 'e_i' denotes an element in the thresholded edge image, 'g_{e_i}' denotes the gradient magnitude of the element, 'g_0' denotes the gradient magnitude for the non-edge element.

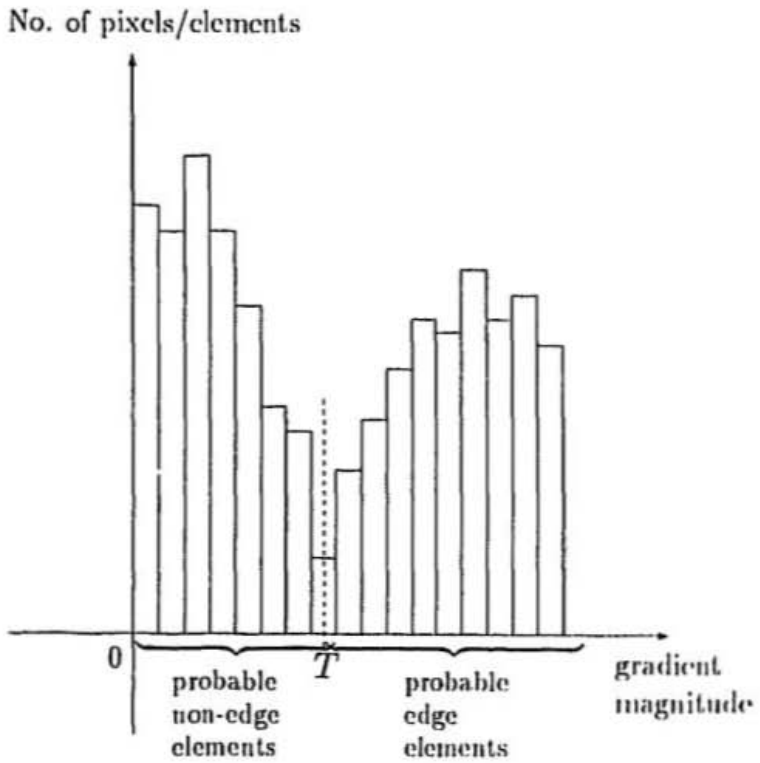


Figure 3.4 Threshold determination from gradient magnitude histogram

For many complex real-world images, a global threshold or set of thresholds will yield unsatisfactory results due to noise, gradual variations in gray-level [Ballard and Brown 1982b] or non-uniform lighting [Hertz and Schafer 1988]. However, with the application of the hierarchical neural network system to process the edge image, an improved edge image with very satisfactory results can still be obtained.

3.4 Selecting Window Size

The local edge pattern to be input to the neural network system is a 5×5 window in the edge image (Fig. 3.5). The central pixel (denoted by 'X') in the window is the pixel under consideration and both its orientation and gradient magnitude can be adjusted by the

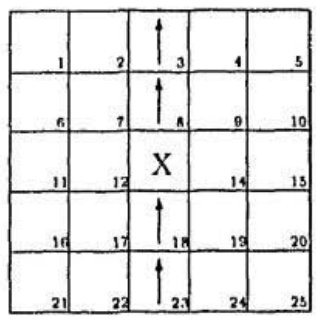


Figure 3.5. A window for the input of local edge pattern

hierarchical neural network system. An appropriate window size has to be selected. If the window is too small, the length of the edge contour in the local edge pattern will be too short to be of any significance, which makes the neural network system less effective. On the other hand, a large window allows better interpolation of missing edge elements but results in more complicated local edge patterns and a very complex neural net design.

3.5 Generating Input Vectors

A neural net processes the information from the input pattern in a vector form. There are three kinds of input vectors to the neural network. Each component in a vector is associated with a corresponding element/pixel in a window. Input vector 'G' contains the normalized

gradient magnitude of the elements in the window, ($G = \{g_j\}$, $j = 25$). The eight sets of direction vectors, '$D^{(i)}$' ($D^{(i)} = \{d_k^{(i)}\}$, $k = 25$, $i \in \{n, \ldots, sc\}$), correspond to the eight edge orientations. An example is used to illustrate how to derive the input direction vectors. In Fig. 3.6, elements 'e_1', 'e_6' have north orientation and elements 'e_7', 'e_8' have west orientation while the other elements are non-edge elements.

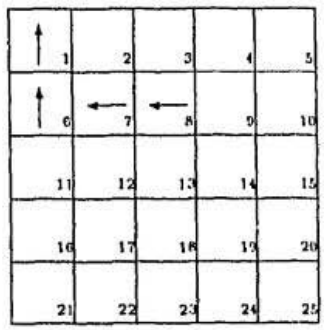


Figure 3.6 Edge elements in a window

The set of direction values for 'e_1' is :

$$\{d_1^{(n)}, d_1^{(s)}, d_1^{(e)}, d_1^{(w)}, d_1^{(ne)}, d_1^{(sw)}, d_1^{(nw)}, d_1^{(se)}\} = \{1,0,0,0,0,0,0,0\}. \tag{3.9}$$

The set of direction values for 'e_7' is :

$$\{d_7^{(n)}, d_7^{(s)}, d_7^{(e)}, d_7^{(w)}, d_7^{(ne)}, d_7^{(sw)}, d_7^{(nw)}, d_7^{(se)}\} = \{0,0,0,1,0,0,0,0\}. \tag{3.10}$$

The set of direction values for 'e_2' is :

$$\{d_2^{(n)}, d_2^{(s)}, d_2^{(e)}, d_2^{(w)}, d_2^{(ne)}, d_2^{(sw)}, d_2^{(nw)}, d_2^{(se)}\} = \{0,0,0,0,0,0,0,0\}. \tag{3.11}$$

Hence, the input vector '$D^{(n)}$' associated with the north orientation is described by :

$$D^{(n)} = \{d_1^{(n)}, d_2^{(n)}, d_3^{(n)}, d_4^{(n)}, d_5^{(n)}, d_6^{(n)}, d_7^{(n)}, d_8^{(n)}, d_9^{(n)}, \ldots, d_{25}^{(n)}\} \quad (3.12)$$

$$= \{1, 0, 0, 0, 0, 1, 0, 0, 0, \ldots, 0\}.$$

Since elements 'e_1' and 'e_6' have a north orientation, their direction values associated with the north orientation are '1's. For elements 'e_7' and 'e_8', since they have an orientation which is not north, their direction values associated with the north orientation are '0's. The other elements have direction values of '0's because they do not have any orientation. Similarly, the input vector '$D^{(w)}$' associated with the west orientation is described by :

$$D^{(w)} = \{d_1^{(w)}, d_2^{(w)}, d_3^{(w)}, d_4^{(w)}, d_5^{(w)}, d_6^{(w)}, d_7^{(w)}, d_8^{(w)}, d_9^{(w)}, \ldots, d_{25}^{(w)}\} \quad (3.13)$$

$$= \{0, 0, 0, 0, 0, 0, 1, 1, 0, \ldots, 0\}.$$

For the input vector, '$D^{(i)'}$' ($D^{(i)'} = \{d_l^{(i)'}\}, l = 25, i \in \{n, \ldots, se\}$), the components in vector '$D^{(i)'}$' contain the complement values of the components in vector '$D^{(i)}$'. For example,

$$D^{(n)} = \{1, 0, 0, 0, 0, 1, 0, 0, 0, \ldots, 0\}. \quad (3.14)$$

$$D^{(n)'} = \{0, 1, 1, 1, 1, 0, 1, 1, 1, \ldots, 1\}. \quad (3.15)$$

The vectors 'G', '$D^{(i)}$', and '$D^{(i)'}$' where $i \in \{n, \ldots, se\}$ are derived and used as input pattern vectors for the neural net.

Chapter 4

Edge Contour Detection Subnet

4.1 Introduction

In Chapter three, the computation of the initial edge measurement was described. The gradient magnitude and orientation for a pixel in the input image is obtained by using the Sobel Operator. A global threshold algorithm is then used to segregate edge elements from non-edge elements.

In this chapter, the subnet in the first level of the hierarchical neural network system, called the Edge Contour Detection Subnet, is described. This subnet receives input data generated by the edge operator and detects the presence of edge contours in various local edge patterns. The Edge Contour Detection Subnet can accurately detect edge contours by simultaneously utilizing all available information.

4.2 Information Contributing to the Detection of Edge Contours

Three sources of information contribute to the detection of an edge contour and hence affect the adjustment to the edge measurement of the central edge element. They are :

1. The gradient magnitude of the neighbouring edge elements in a local edge pattern. A collective consideration of the gradient magnitude of neighbouring edge elements can provide a good indication of the possibility of the presence of an edge contour.

2. The orientation of neighbouring edge elements. If these edge elements have the same orientation, this is a good indication that they would construct an edge contour.

3. The relative positions of appropriate neighbouring pixels/edge elements in the local edge pattern. That is, the locations of neighbouring edge elements which are the constituents of an edge contour.

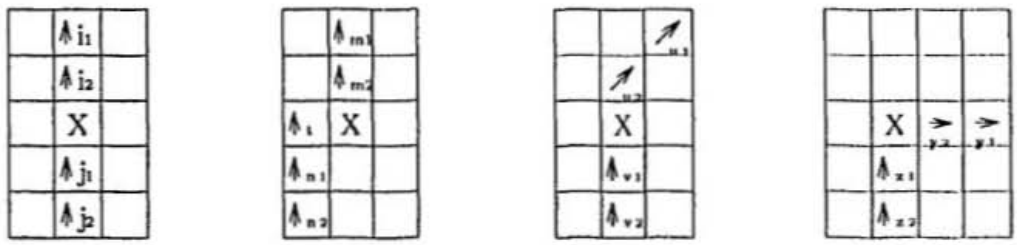
The information provided by the gradient magnitude and the orientation of appropriate edge elements in the local edge pattern must be utilized simultaneously to achieve the correct detection of an edge contour. Therefore, neighbouring edge elements which have the same orientation and have large gradient magnitude will give a very good indication that the central edge element lies on the detected edge contour.

4.3 Selective Functional Expansion Model

In the functional expansion model, a set of functions $(f_1, f_2, \ldots, f_n)$ maps an input pattern into a larger pattern space. Such nonlinear combinations of pattern elements are introduced as a functional link. Each input pattern is extended by some nonlinear transformation before it is presented to the node in the net. Hence, the functional processes generated by the functional link enable the enhancement of the input pattern to higher-order terms.

The general functional expansion model generates all the possible functional processes. A major drawback of such nets is the combinatorial explosion of high-order terms [Minsky and Papert 1988]. In order to avoid this, the functional-link nets for the Edge Contour Detection Subnet are specially designed and are refered to as the Selective Functional Expansion

Model. Usually, functional processes beyond the second order only have a small possibility of being useful and do not significantly contribute to the net output [Pao 1988; Pao 1989$_a$]. Furthermore, not all second order functional processes need to be considered to characterize valid edge patterns. In order to adjust the edge measurement, seven types of functional processes are selected for the functional link. The 'p', 'q', 'q'', 'q''', 'r' and 's' type processes are used for detecting various valid edge contours in local edge patterns and hence are used for reinforcing the edge measurement. The 'h' type process on the other hand is used for detecting the absence of valid local edge patterns and hence is used for suppressing the edge measurement.



a) Rectilinear b) Non-symmetrical c) Curvilinear d) Corner

Figure 4.1 Edge contours

4.3.1 Processes for Detecting Rectilinear Edge Contour

A rectilinear edge contour (Fig. 4.1a) is characterized by the 'p' type functional process. The 'p' type functional process has four different processes, namely, $p^{(k)}_{i_1,j_1}$, $p^{(k)}_{i_1,j_2}$, $p^{(k)}_{i_2,j_1}$, and $p^{(k)}_{i_2,j_2}$, where 'k' denotes the orientation, $k \in \{n,\ldots,se\}$. A 'p' type process is associated with two edge elements on the rectilinear edge contour in the local edge pattern. Each of

these edge elements is located on opposite sides of the edge contour which is divided by the central element 'X'. For example, for a rectilinear edge contour (Fig. 4.1a), the two edge elements associated with a 'p' type process, $p^{(k)}_{i_1,j_1}$ are the edge elements at locations indexed by 'i_1' and 'j_1' in the local pattern, the central element 'X' is situated between the edge elements at locations 'i_1' and 'j_1'. The 'p' type processes are :

$$p^{(k)}_{i_1,j_1} = d^{(k)}_{i_1} * d^{(k)}_{j_1} * g_{i_1} * g_{j_1} \tag{4.2}$$

$$p^{(k)}_{i_1,j_2} = d^{(k)}_{i_1} * d^{(k)}_{j_2} * g_{i_1} * g_{j_2} \tag{4.3}$$

$$p^{(k)}_{i_2,j_1} = d^{(k)}_{i_2} * d^{(k)}_{j_1} * g_{i_2} * g_{j_1} \tag{4.4}$$

$$p^{(k)}_{i_2,j_2} = d^{(k)}_{i_2} * d^{(k)}_{j_2} * g_{i_2} * g_{j_2} \tag{4.5}$$

where $d^{(k)}_{i_1}$, $d^{(k)}_{j_1}$ denote the direction values and g_{i_1}, g_{j_1} denote the gradient magnitudes of the two edge elements indexed by 'i_1' and 'j_1'. In order to detect a rectilinear edge contour, all four 'p' type processes, $p^{(k)}_{i_1,j_1}$, $p^{(k)}_{i_1,j_2}$, $p^{(k)}_{i_2,j_1}$, and $p^{(k)}_{i_2,j_2}$ are considered. However, the activation of each process is dependent on the direction values.

a) Pattern - I b) Pattern - II

Figure 4.2 Local edge patterns

For example, a rectilinear edge contour with a north-east orientation in a local edge pattern (Fig. 4.2a) is characterized by the following 'p' type processes associated with the north-east

orientation.

$$p_{5,16}^{(ne)} = d_5^{(ne)} * d_{16}^{(ne)} * g_5 * g_{16} \quad (4.6)$$

$$p_{5,20}^{(ne)} = d_5^{(ne)} * d_{20}^{(ne)} * g_5 * g_{20} \quad (4.7)$$

$$p_{9,16}^{(ne)} = d_9^{(ne)} * d_{16}^{(ne)} * g_9 * g_{16} \quad (4.8)$$

$$p_{9,20}^{(ne)} = d_9^{(ne)} * d_{20}^{(ne)} * g_9 * g_{20}. \quad (4.9)$$

The subscripts denote the respective edge elements in the local edge pattern. Since the edge elements associated with the processes $p_{5,16}^{(ne)}$, $p_{5,20}^{(ne)}$, $p_{9,16}^{(ne)}$, $p_{9,20}^{(ne)}$ have the north-east orientation, the direction values for these edge elements ($d_5^{(ne)}, d_9^{(ne)}, d_{16}^{(ne)}, d_{20}^{(ne)}$) are '1's. Therefore,

$$p_{5,16}^{(ne)} = 1 * 1 * g_5 * g_{16} \quad (4.10)$$

$$p_{5,20}^{(ne)} = 1 * 1 * g_5 * g_{20} \quad (4.11)$$

$$p_{9,16}^{(ne)} = 1 * 1 * g_9 * g_{16} \quad (4.12)$$

$$p_{9,20}^{(ne)} = 1 * 1 * g_9 * g_{20}. \quad (4.13)$$

All the functional processes associated with a north-east orientation for the rectilinear edge contour are activated. However, if the orientation of an edge element is different from the associated orientation of the process, then the corresponding functional process will be de-activated. In Fig. 4.2b, edge elements at locations '9' and '20' in the local edge pattern do not have a north-east orientation and as such their direction values are '0's. Therefore,

$$p_{5,16}^{(ne)} = 1 * 1 * g_5 * g_{16} = g_5 * g_{16} \quad (4.14)$$

$$p_{5,20}^{(ne)} = 1 * 0 * g_5 * g_{20} = 0 \quad (4.15)$$

$$p_{9,16}^{(ne)} = 0 * 1 * g_9 * g_{16} = 0 \qquad (4.16)$$

$$p_{9,20}^{(ne)} = 0 * 0 * g_9 * g_{20} = 0. \qquad (4.17)$$

In this situation, only process $p_{5,16}^{(ne)}$ is activated while processes $p_{5,20}^{(ne)}$, $p_{9,16}^{(ne)}$, $p_{9,20}^{(ne)}$ are deactivated. Therefore, if there are more neighbouring edge elements with the same orientation, more functional processes are activated. On the other hand, if the local edge pattern does not form an edge contour, the functional processes will not be activated.

4.3.2 Processes for Detecting Non-Symmetrical Linear Edge Contour

A non-symmetrical linear edge contour (Fig. 4.1b) is characterized by the 'q', 'q'', and 'q''' type functional processes. For each of these three types of processes, there are four different functional processes, namely, $q_{m_1,n_1}^{(k)}$, $q_{m_1,n_2}^{(k)}$, $q_{m_2,n_1}^{(k)}$, $q_{m_2,n_2}^{(k)}$, $q_{m_1,n_1}^{(k)'}$, $q_{m_1,n_2}^{(k)'}$, $q_{m_2,n_1}^{(k)'}$, $q_{m_2,n_2}^{(k)'}$, $q_{m_1,n_1}^{(k)''}$, $q_{m_1,n_2}^{(k)''}$, $q_{m_2,n_1}^{(k)''}$ and $q_{m_2,n_2}^{(k)''}$. The 'q', 'q'' and 'q''' type processes are associated with two edge elements on the non-symmetrical linear edge contour. Each of these edge elements is located on opposite sides of the edge contour which is divided by the central element 'X'. For example, for a non-symmetrical linear edge contour (Fig. 4.1b), the two edge elements associated with either a 'q', 'q'' or 'q''' type process, namely, $q_{m_1,n_1}^{(k)}$, $q_{m_1,n_1}^{(k)'}$ or $q_{m_1,n_1}^{(k)''}$ are the edge elements at locations indexed by 'm_1' and 'n_1' in the local pattern. The central element 'X' is situated between the edge elements at locations 'm_1' and 'n_1'. The 'q' type process is of the second order as two edge elements are considered. The 'q'' and 'q''' type processes are of the third order as each of these functional processes incorporates three edge elements.

$$q_{m_1,n_1}^{(k)} = d_{m_1}^{(k)} * d_{n_1}^{(k)} * g_{m_1} * g_{n_1} \qquad (4.18)$$

$$\vdots$$

$$q^{(k)}_{m_2,n_2} = d^{(k)}_{m_2} * d^{(k)}_{n_2} * g_{m_2} * g_{n_2} \quad (4.21)$$

$$q^{(k)'}_{m_1,n_1} = d^{(k)}_{m_1} * d^{(k)}_{n_1} * d^{(k)}_{t} * g_{m_1} * g_{n_1} \quad (4.22)$$

$$\vdots$$

$$q^{(k)'}_{m_2,n_2} = d^{(k)}_{m_2} * d^{(k)}_{n_2} * d^{(k)}_{t} * g_{m_2} * g_{n_2} \quad (4.25)$$

$$q^{(k)''}_{m_1,n_1} = d^{(k)}_{m_1} * d^{(k)}_{n_1} * d^{(k)}_{X} * g_{m_1} * g_{n_1} \quad (4.26)$$

$$\vdots$$

$$q^{(k)''}_{m_2,n_2} = d^{(k)}_{m_2} * d^{(k)}_{n_2} * d^{(k)}_{X} * g_{m_2} * g_{n_2}. \quad (4.29)$$

The 'q'' and 'q''' type processes are similar to the 'q' type process except that the 'q'' type process also takes the orientation of the element (d_t) adjacent to the central edge element into account. Whereas for the 'q''' type process, the orientation of the central edge element (d_X) is considered (Fig. 4.1b). For each 'q'' type process, a corresponding 'q' type process is used to counteract the effects of each other. The measurement of the central edge element 'X' should not be strengthened, because the element 't' is an edge element and it completes the edge contour, increasing the edge measurement of element 'X' will thicken the edge contour, or create false edge elements.

a) A 'complete' edge contour b) An 'incomplete' edge contour

Figure 4.3 Non-symmetrical edge patterns

For example, in the functional-link net associated with the east orientation (Fig. 4.3a), the direction values '$d_8^{(e)}$', ..., '$d_{12}^{(e)}$' have the value '1'. Therefore,

$$q_{9,11}^{(e)} = 1 * 1 * g_9 * g_{11} \qquad (4.30)$$

$$\vdots$$

$$q_{10,12}^{(e)} = 1 * 1 * g_{10} * g_{12}. \qquad (4.33)$$

Hence, the functional processes described in expressions (4.30 - 4.33) are activated to reinforce the central edge element 'X'. However, since edge element '8' exists and completes the edge contour, edge element 'X' should not be reinforced because the edge contour is already 'complete'. Therefore, the corresponding 'q'' type processes must be activated to counter the responses of the activated 'q' type processes. That is,

$$q_{9,11}^{(e)'} = 1 * 1 * 1 * g_9 * g_{11} \qquad (4.34)$$

$$\vdots$$

$$q_{10,12}^{(e)'} = 1 * 1 * 1 * g_{10} * g_{12}. \qquad (4.37)$$

In order to offset the effects of the activated 'q' type processes by the 'q'' type processes, a positive weight is assigned to the 'q' type process, while a negative weight is assigned to the corresponding 'q'' type process. The absolute values of these weights are equal and these weighted processes are connected to the same neural node. Therefore, if a 'q' type process is activated and the edge element '8' (Fig. 4.3a) completes the edge contour, the corresponding 'q'' type process is also activated to nullify the effects of the 'q' type processes. However, if element '8' does not complete the edge contour (Fig. 4.3b), the 'q'' type processes will not

be activated because the direction value for element '8' ($d_8^{(c)}$) is '0'. Therefore, only the 'q' type processes will be activated and the resultant effect is a positive output from the 'q' type processes.

Neighbouring elements with orientations different from the orientation handled by the functional-link net are used to suppress the central element in the local edge pattern (details concerning suppression are discussed in section 4.3.1.5). Due to the offsetting effects of the 'q' type and 'q'' type processes, a central edge element in the valid edge will ultimately be eliminated by the suppression. In order to preserve the central valid edge element but not the false edge element, 'q''' type processes take into account the orientation of the central edge element. For example, in Fig. 4.3a, if edge element 'X' has an east orientation, its direction value will be '1' and therefore the following processes will be activated :

$$q_{9,11}^{(c)''} = 1 * 1 * 1 * g_9 * g_{11} \qquad (4.38)$$

$$\vdots$$

$$q_{10,12}^{(c)''} = 1 * 1 * 1 * g_{10} * g_{12}. \qquad (4.41)$$

By considering the orientation of the central element, if the central element is a non-edge element, none of the 'q''' type processes will be activated and this will prevent the creation of false edge elements.

4.3.3 Processes for Detecting Curvilinear Edge Contour

A curvilinear edge contour (Fig. 4.1c) is characterized by the 'r' type functional process. The 'r' type functional process has four different processes, namely, $r_{u_1,u_1}^{(k,l)}$, $r_{u_1,u_2}^{(k,l)}$, $r_{u_2,u_1}^{(k,l)}$, and

$r^{(k,l)}_{u_2,v_2}$, where : (1) 'k' and 'l' denote the orientations, and (2) 'u_1', 'u_2', 'v_1' and 'v_2' index the locations of the edge elements in the local pattern. An 'r' type process is also of the second order and is associated with two edge elements on the curvilinear edge contour in the local edge pattern. Each of these edge elements is located on opposite sides of the edge contour which is divided by the central element 'X'. In order to detect a curvilinear edge contour, all four 'r' type processes are considered :

$$r^{(k,l)}_{u_1,v_1} = d^{(l)}_{u_1} * d^{(k)}_{v_1} * g_{u_1} * g_{v_1} \qquad (4.42)$$

$$r^{(k,l)}_{u_1,v_2} = d^{(l)}_{u_1} * d^{(k)}_{v_2} * g_{u_1} * g_{v_2} \qquad (4.43)$$

$$r^{(k,l)}_{u_2,v_1} = d^{(l)}_{u_2} * d^{(k)}_{v_1} * g_{u_2} * g_{v_1} \qquad (4.44)$$

$$r^{(k,l)}_{u_2,v_2} = d^{(l)}_{u_2} * d^{(k)}_{v_2} * g_{u_2} * g_{v_2}. \qquad (4.45)$$

where : (i) if $k =$ 'n' then $l \in \{ne, nw\}$; (v) if $k =$ 'ne' then $l \in \{n, e\}$;

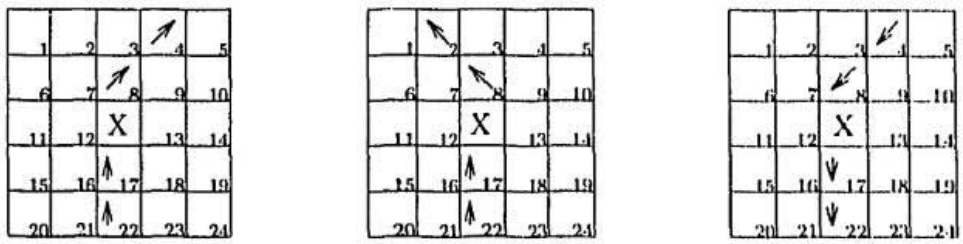
(ii) if $k =$'s' then $l \in \{se, sw\}$; (vi) if $k =$ 'nw' then $l \in \{n, w\}$;

(iii) if $k =$ 'e' then $l \in \{ne, se\}$; (vii) if $k =$ 'sw' then $l \in \{s, w\}$;

(iv) if $k =$ 'w' then $l \in \{nw, sw\}$; (viii) if $k =$ 'se' then $l \in \{s, e\}$.

The activation of an 'r' type process is dependent on the direction values which are associated with different orientations. Hence, this activation requirement for an 'r' type process is different from that for either a 'p', 'q', 'q'', or 'q''' type process. A 'p', 'q', 'q'', or 'q''' type process is activated only if all the edge elements associated with the functional process have the same orientation as the orientation handled by the functional process. However, in order to activate an 'r' type process in a functional-link net, an orientation different from

the orientation associated with the functional-link net is also considered. For example, if a curvilinear edge contour has a north orientation, north-east (Fig. 4.4a) or north-west (Fig. 4.4b) orientations are considered to activate the corresponding functional processes. Similarly, for a curvilinear edge contour with a south orientation, the corresponding orientations are south-west (Fig. 4.4c) or south-east, and so on.



(a) With a north orientation - I (b) With a north orientation - I I (c) With a south orientation

Figure 4.4 Curvilinear edge contours

In Fig. 4.4a, the curvilinear edge contour with a north orientation is characterized by the following 'r' type processes in the functional-link net associated with the north orientation :

$$r_{4,17}^{(n,ne)} = d_4^{(ne)} * d_{17}^{(n)} * g_4 * g_{17} \qquad (4.46)$$

$$r_{4,22}^{(n,ne)} = d_4^{(ne)} * d_{22}^{(n)} * g_4 * g_{22} \qquad (4.47)$$

$$r_{8,17}^{(n,ne)} = d_8^{(ne)} * d_{17}^{(n)} * g_8 * g_{17} \qquad (4.48)$$

$$r_{8,22}^{(n,ne)} = d_8^{(ne)} * d_{22}^{(n)} * g_8 * g_{22}. \qquad (4.49)$$

The edge elements at locations '4' and '8' in the local edge pattern must have a north-east orientation in order to activate the 'r' type processes in the functional-link net associated

with the north orientation. The edge elements at locations '17' and '22' on the other hand must have a north orientation. Therefore, the processes $r_{4,17}^{(n,nc)}$, $r_{4,22}^{(n,nc)}$, $r_{8,17}^{(n,nc)}$, $r_{8,22}^{(n,nc)}$ are activated :

$$r_{4,17}^{(n,nc)} = 1 * 1 * g_4 * g_{17} \qquad (4.50)$$

$$r_{4,22}^{(n,nc)} = 1 * 1 * g_4 * g_{22} \qquad (4.51)$$

$$r_{8,17}^{(n,nc)} = 1 * 1 * g_8 * g_{17} \qquad (4.52)$$

$$r_{8,22}^{(n,nc)} = 1 * 1 * g_8 * g_{22}. \qquad (4.53)$$

Similarly, in Fig. 4.4c, processes $r_{4,17}^{(s,sw)}$, $r_{4,22}^{(s,sw)}$, $r_{8,17}^{(s,sw)}$ and $r_{8,22}^{(s,sw)}$ (which characterize a curvilinear edge contour with a south orientation) in the functional-link net associated with the south orientation are activated if edge elements at locations '4' and '8' have a south-west orientation and edge elements at locations '17' and '22' have a south orientation.

4.3.4 Processes for Detecting Edge Contours at a Corner

Edge contours at a corner (Fig. 4.1d) are characterized by the 's' type functional process. The 's' type functional process has four different processes, namely, $s_{y_1,z_1}^{(k,b,c)}$, $s_{y_1,z_2}^{(k,b,c)}$, $s_{y_2,z_1}^{(k,b,c)}$, and $s_{y_2,z_2}^{(k,b,c)}$, where : (1) 'k', 'b', and 'c' denote the orientations, and (2) 'y_1', 'y_2','z_1' and 'z_2' index the locations of the edge elements in the local pattern. An 's' type process is associated with two edge elements on the edge contours, with each on opposite sides of the 'corner', which is located at the central element 'X'. An 's' type process is a second order functional process :

$$s_{y_1,z_1}^{(k,b,c)} = d_{y_1}^{(b)} * d_{z_1}^{(c)} * g_{y_1} * g_{z_1} \qquad (4.54)$$

$$s_{y_1,z_2}^{(k,b,c)} = d_{y_1}^{(b)} * d_{z_2}^{(c)} * g_{y_1} * g_{z_2} \qquad (4.55)$$

$$s_{y_2,z_1}^{(k,b,c)} = d_{y_2}^{(b)} * d_{z_1}^{(c)} * g_{y_2} * g_{z_1} \qquad (4.56)$$

$$s_{y_2,z_2}^{(k,b,c)} = d_{y_2}^{(b)} * d_{z_2}^{(c)} * g_{y_2} * g_{z_2}, \qquad (4.57)$$

where : (i) if $k =$ 'n' then $b, c \in \{ne, nw\}$; (v) if $k =$ 'ne' then $b, c \in \{n, e\}$;

(ii) if $k =$'s' then $b, c \in \{se, sw\}$; (vi) if $k =$ 'nw' then $b, c \in \{n, w\}$;

(iii) if $k =$ 'e' then $b, c \in \{ne, se\}$; (vii) if $k =$ 'sw' then $b, c \in \{s, w\}$;

(iv) if $k =$ 'w' then $b, c \in \{nw, sw\}$; (viii) if $k =$ 'se' then $b, c \in \{s, e\}$;

and $b \neq c$.

The difference between an 's' type functional process and the other functional process types is that the two edge elements associated with an 's' type functional process must have an orientation different from the orientation handled by the functional process in order to activate the 's' type process. For example, if the central edge element at a 'corner' has a north orientation, then the corresponding orientations characterizing the edge contours at the 'corner' are north-east *and* north-west (Fig. 4.5a). On the other hand, if the central edge element at a 'corner' has a north-east orientation, then the corresponding orientations are north *and* east (Fig. 4.5b).

a) Corner with a north orientation b) Corner with a north-east orientation

Figure 4.5 Corner edge patterns

The following 's' type processes in the functional-link net are associated with the north-east orientation :

$$s_{14,17}^{(ne,e,n)} = d_{14}^{(e)} * d_{17}^{(n)} * g_{14} * g_{17} \qquad (4.57)$$

$$s_{14,22}^{(ne,e,n)} = d_{14}^{(e)} * d_{22}^{(n)} * g_{14} * g_{22} \qquad (4.58)$$

$$s_{13,17}^{(ne,e,n)} = d_{13}^{(e)} * d_{17}^{(n)} * g_{13} * g_{17} \qquad (4.59)$$

$$s_{13,22}^{(ne,e,n)} = d_{13}^{(e)} * d_{22}^{(n)} * g_{13} * g_{22}. \qquad (4.60)$$

The edge elements at locations '13' and '14' in the local edge pattern must have an east orientation and edge elements at locations '17' and '22' must have a north orientation in order to activate the 's' type processes associated with the north-east orientation. Therefore, the processes $s_{14,17}^{(ne,e,n)}$, $s_{14,22}^{(ne,e,n)}$, $s_{13,17}^{(ne,e,n)}$, $s_{13,22}^{(ne,e,n)}$ are activated :

$$s_{14,17}^{(ne,e,n)} = 1 * 1 * g_{14} * g_{17} \qquad (4.61)$$

$$s_{14,22}^{(ne,e,n)} = 1 * 1 * g_{14} * g_{22} \qquad (4.62)$$

$$s_{13,17}^{(ne,e,n)} = 1 * 1 * g_{13} * g_{17} \qquad (4.63)$$

$$s_{13,22}^{(ne,e,n)} = 1 * 1 * g_{13} * g_{22}. \qquad (4.64)$$

4.3.5 Suppressing Functional Processes

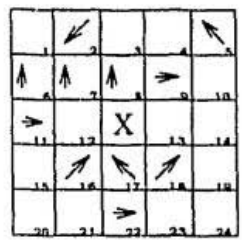


Figure 4.6 Pattern for suppressing
central edge element

The 'h' type process has twenty-four different processes, where each process is associated with a pixel in the window, excluding the central pixel (Fig. 4.6) :

$$h_1^{(i)} = d_1^{(i)'} g_1 \qquad (4.65)$$

$$\vdots$$

$$h_{24}^{(i)} = d_{24}^{(i)'} g_{24}, \qquad (4.88)$$

where $i \in \{n, \ldots, se\}$. The first order 'h' type process is used to suppress the false edge element in the local edge pattern. For example (in Fig. 4.6), a functional-link net is used to detect the edge contour with a north orientation at the central element. If more neighbouring elements do not have a north orientation, then it is more likely the central element with a north orientation is a false edge element. If a neighbouring element does not have a north orientation, the 'h' type process associated with this element will be activated. By considering all the neighbouring elements in the local area, the ability to detect a false edge element by the 'h' type processes is dependent on two factors :

1. The number of elements having different orientations from that associated with the functional-link net.

2. The number of non-edge elements in the local area.

Therefore, when the local pattern does not characterize any valid edge contour, the effect of the 'h' type processes is greatest.

4.3.6 Functional Link with Selective Functional Expansion Model

Each functional-link net receives the input data from five input vectors. The input vector 'G' contains the gradient magnitudes of the elements in the local edge pattern. The other four input vectors contain the direction values of the elements. For example, a functional-link net associated with the north orientation receives the data from the following input vectors containing the direction values :

1. $D^{(n)}$ - vector associated with the north orientation.

2. $D^{(ne)}$ - vector associated with the north-east orientation.

3. $D^{(nw)}$ - vector associated with the north-west orientation.

4. $D^{(n)'}$ - vector containing the complement values of the components in vector '$D^{(n)}$'.

The vectors '$D^{(n)}$', '$D^{(ne)}$' and '$D^{(nw)}$' are concerned with reinforcement and hence these vectors provide the direction values to the 'p', 'q', 'q'', 'q''', 'r' and 's' type functional processes. The vector '$D^{(n)'}$', on the other hand, is concerned with suppression and hence it

provides the direction values to the 'h' type functional processes. For each functional-link net, the model generates selected processes, namely, the 'p', 'q', 'q'', 'q''', 'r', 's' and 'h' type functional processes (Fig. 4.7).

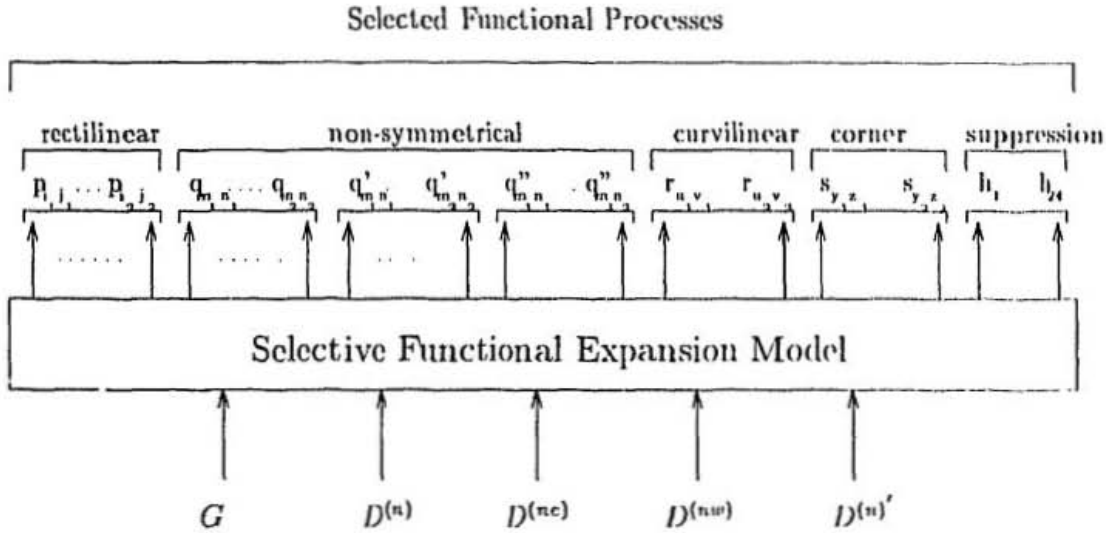


Figure 4.7 Schematic illustration of a selective functional expansion model for the north orientation

4.4 Hypothesized Edge Patterns

In the proposed neural network system, the selection of appropriate functional processes enables the Edge Contour Detection Subnet to perform correctly. The task of selecting the appropriate functional processes is achieved by using *a priori* knowledge of various expected edge patterns in an image. Incorporation of this knowledge about the structure of various

expected edge patterns into the functional-link nets is achieved through the use of hypothesized edge patterns. A hypothesized edge pattern (HEP) is a hypothetical construction of a local edge pattern in an $n \times n$ window.

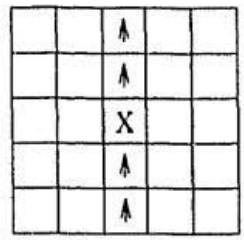
In this thesis, a 5×5 window is used for the HEP. The edge contour in each HEP consists of five edge elements and is assumed to be 'one-pixel' wide. For example, a window depicting a HEP for an edge contour with a north orientation is shown in Fig. 4.8(a). The arrowheads indicate the orientations of edge elements. Collectively, these edge elements construct the edge contour. The HEPs are grouped and categorized according to the orientations of the central element in the window. There are eleven HEPs for each orientation. Figure 4.8 shows the HEPs for the north orientation. Figure 4.9 shows the HEPs for the northeast orientation. The proposed HEPs are able to characterize the following types of edge contours :

1. Rectilinear edge contours.

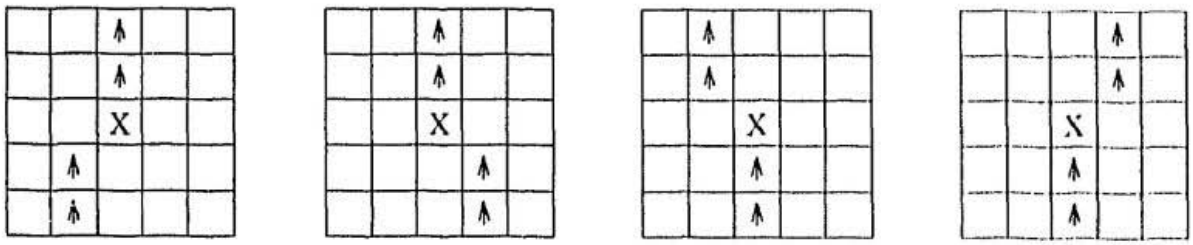
2. Non-symmetrical linear edge contours whereby an edge contour is not a continuous straight line.

3. Curvilinear edge contours.

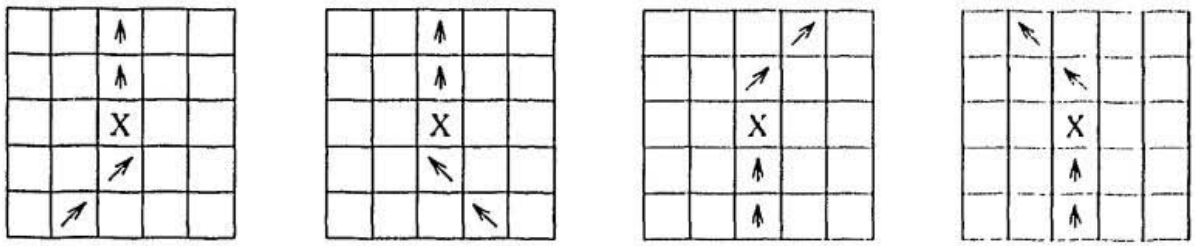
4. Edge contours at a corner.



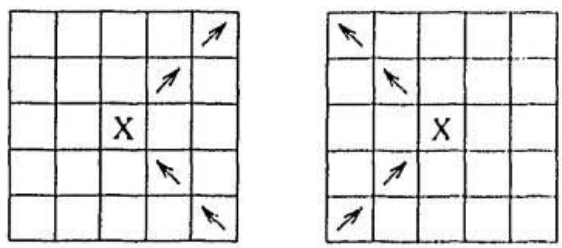
(a) Rectilinear edge contour



(b) Non-symmetrical linear edge contours

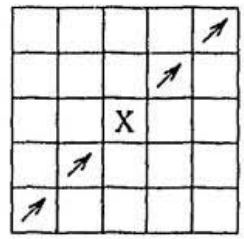


(c) Curvilinear edge contours

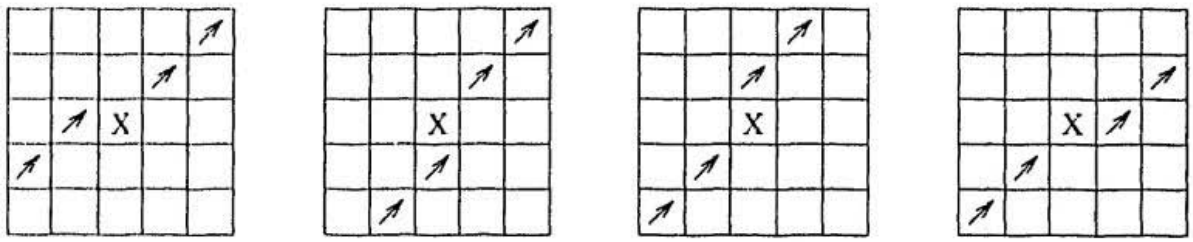


(d) Edge contours at a corner

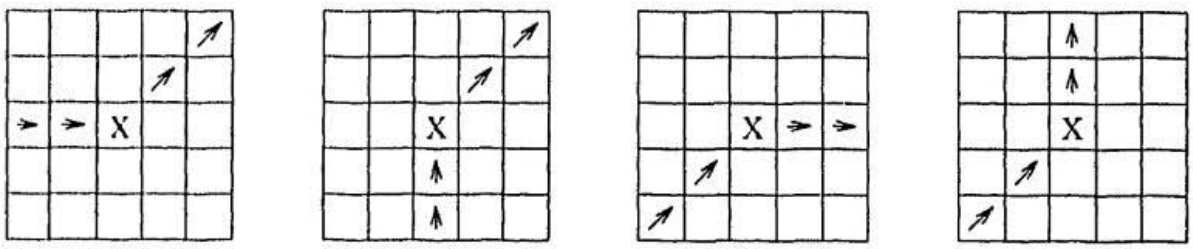
Figure 4.8 HEPs for the north orientation



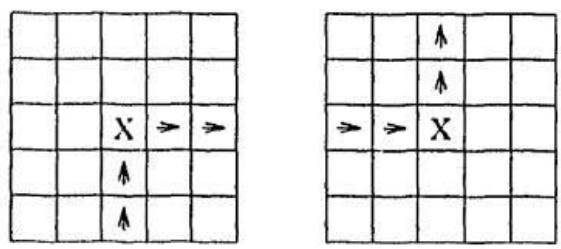
(a) Rectilinear edge contour



(b) Non-symmetrical linear edge contours



(c) Curvilinear edge contours



(d) Edge contours at a corner

Figure 4.9 HEPs for the north-east orientation

4.5 Architecture of Edge Contour Detection Subnet

The architecture of the Edge Contour Detection Subnet (ECDS) consists of eight selective functional-link nets. Each of these nets work independently and in parallel with each other. Each selective functional-link net is associated with a particular orientation. That is, each functional-link net detects only edge contours with that particular orientation and ignores all other edge contours with any other orientations. In Fig. 4.10, each functional-link net consists of a node and a set of functional processes generated by the Selective Functional Expansion Model. The forward connections from the functional processes provide inputs to the node.

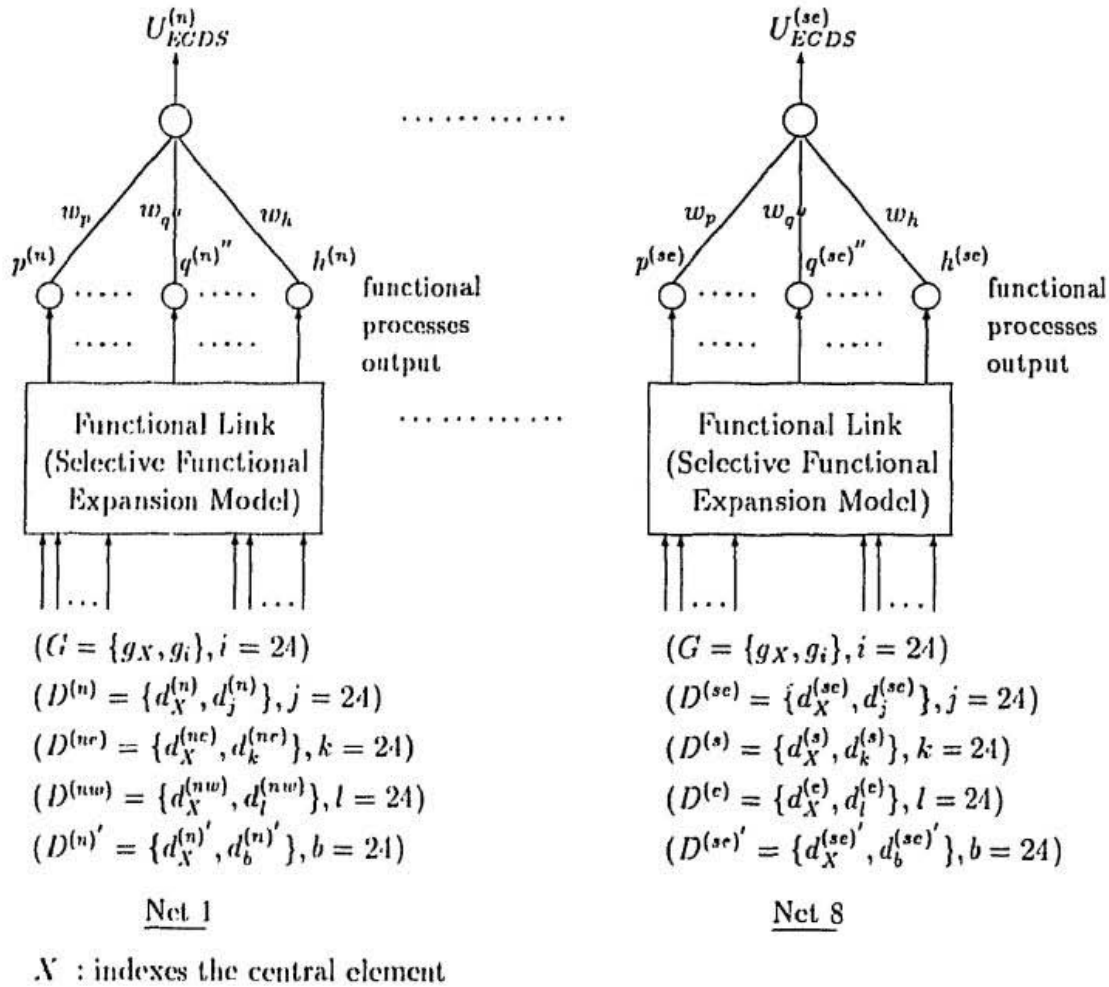


Figure 4.10 Architecture of selective functional-link nets in Edge Contour Detection Subnet

The node is activated in accordance with the input to the node, the sigmoidal activation function of the node, the bias of the node, and a threshold parameter. Consider the functional-link net associated with the north orientation, the net input to the node in the

functional-link net is the sum of output from all types of processes :

$$
\begin{aligned}
net \;=\; & \sum_{x\in\{3,8\}}\sum_{y\in\{17,22\}} w_{p_{x,y}^{(n)}} p_{x,y}^{(n)} + \\
& \sum_{x\in\{3,8\}}\sum_{y\in\{16,18,21,23\}} \left(w_{q_{x,y}^{(n)}} q_{x,y}^{(n)} + w_{q_{x,y}^{(n)'}} q_{x,y}^{(n)'} + w_{q_{x,y}^{(n)''}} q_{x,y}^{(n)''} \right) + \\
& \sum_{x\in\{17,21\}}\sum_{y\in\{3,8\}} w_{r_{x,y}^{(n,ne)}} r_{x,y}^{(n,ne)} + \sum_{x\in\{17,23\}}\sum_{y\in\{3,8\}} w_{r_{x,y}^{(n,nw)}} r_{x,y}^{(n,nw)} + \\
& \sum_{x\in\{17,22\}}\sum_{y\in\{2,4,7,9\}} \left(w_{q_{x,y}^{(n)}} q_{x,y}^{(n)} + w_{q_{x,y}^{(n)'}} q_{x,y}^{(n)'} + w_{q_{x,y}^{(n)''}} q_{x,y}^{(n)''} \right) + \\
& \sum_{x\in\{4,8\}}\sum_{y\in\{17,22\}} w_{r_{x,y}^{(n,ne)}} r_{x,y}^{(n,ne)} + \sum_{x\in\{2,8\}}\sum_{y\in\{17,22\}} w_{r_{x,y}^{(n,nw)}} r_{x,y}^{(n,nw)} + \\
& \sum_{x\in\{5,9\}}\sum_{y\in\{18,24\}} w_{s_{x,y}^{(n,ne,nw)}} s_{x,y}^{(n,ne,nw)} + \sum_{x\in\{16,20\}}\sum_{y\in\{1,7\}} w_{s_{x,y}^{(n,ne,nw)}} s_{x,y}^{(n,ne,nw)} + \\
& \sum_{x=1}^{24} w_{h_x^{(n)}} h_x^{(n)}
\end{aligned}
\tag{4.89}
$$

where :

$p_{x,y}^{(n)}$: output of a 'p' type process associated with the north orientation,

$q_{x,y}^{(n)}$, $q_{x,y}^{(n)'}$ and $q_{x,y}^{(n)''}$: outputs of the 'q', 'q'' and 'q''' type processes associated with the north orientation respectively,

$r_{x,y}^{(n,ne)}$: output of an 'r' type process (associated with the north orientation) where the edge elements indexed by subscript 'x' are concerned with the north-east orientation,

$r_{x,y}^{(n,nw)}$ same as $r_{x,y}^{(n,ne)}$ except that the edge elements indexed by subscript 'x' are concerned with the north-west orientation,

$s_{x,y}^{(n,ne,nw)}$: output of an 's' type process (associated with the north orientation) where the edge elements indexed by subscripts 'x' and 'y' are concerned with the north-east and north-west orientations respectively,

$h_x^{(n)}$: output of an 'h' type process associated with the north orientation,

$w_{p_{x,y}^{(n)}}$: weight associated with a 'p' type process,

$w_{q_{x,y}^{(n)}}$, $w_{q_{x,y}^{(n)'}}$ and $w_{q_{x,y}^{(n)''}}$: weights associated with the 'q', 'q'' and 'q''' type processes respectively,

$w_{r_{x,y}^{(n,ne)}}$: weight associated with an 'r' type process where the edge elements indexed by subscript 'x' are concerned with the north-east orientation,

$w_{r_{x,y}^{(n,nw)}}$ same as $w_{r_{x,y}^{(n,ne)}}$ except that the edge elements are concerned with the north-west orientation,

$w_{s_{x,y}^{(n,ne,nw)}}$: weight associated with an 's' type process where the edge elements indexed by subscripts 'x' and 'y' are concerned with the north-east and north-west orientations respectively,

$w_{h_x^{(n)}}$: weight associated with an 'h' type process,

x and y indexes the edge elements in the local edge pattern (HEP).

The outputs of the functional processes are determined by high order terms with multiplicative connections amongst the components of these terms. For example the output of the second order 'p' type functional process '$p_{x,y}^{(i)}$', $i \in \{n, \ldots, sc\}$ is described by :

$$p_{x,y}^{(i)} = g_x . d_x^{(i)} . g_y . d_y^{(i)} \qquad (4.90)$$

where :

g : *gradient magnitude*

$d^{(i)}$: *direction value*

x, y : *indexes the edge elements in the local edge pattern (HEP).*

Similarly, the outputs of the other functional process types, namely, 'q', 'q'', 'q''', 'r', 's' and 'h' type processes are likewise described as in (4.90). Let the components 'g_x', '$d_x^{(i)}$', 'g_y' and '$d_y^{(i)}$' of a 'p' type functional process (4.90) be represented by '$a^{(1)}_{p^{(i)}_{x,y}}$', '$a^{(2)}_{p^{(i)}_{x,y}}$', '$a^{(3)}_{p^{(i)}_{x,y}}$' and '$a^{(4)}_{p^{(i)}_{x,y}}$' respectively, where 'i' denotes the orientation associated with the process.

Similarly, let the components of 'q', 'r' and 's' type functional processes be represented by '$a^{(1)}_{q^{(i)}_{x,y}}$', '$a^{(2)}_{q^{(i)}_{x,y}}$', '$a^{(3)}_{q^{(i)}_{x,y}}$', '$a^{(4)}_{q^{(i)}_{x,y}}$'; '$a^{(1)}_{r^{(i,j)}_{x,y}}$', '$a^{(2)}_{r^{(i,j)}_{x,y}}$', '$a^{(3)}_{r^{(i,j)}_{x,y}}$', '$a^{(4)}_{r^{(i,j)}_{x,y}}$'; and '$a^{(1)}_{s^{(i,j,k)}_{x,y}}$', '$a^{(2)}_{s^{(i,j,k)}_{x,y}}$', '$a^{(3)}_{s^{(i,j,k)}_{x,y}}$', '$a^{(4)}_{s^{(i,j,k)}_{x,y}}$' respectively, where '$i$' denotes the orientation associated with the process, 'j' denotes the orientation associated with the element indexed by 'x', and 'k' denotes the orientation associated with the element indexed by 'y'.

Components in 'q'' and 'q''' type functional processes are represented by '$a^{(1)}_{q^{(i)'}_{x,y}}$', '$a^{(2)}_{q^{(i)'}_{x,y}}$', '$a^{(3)}_{q^{(i)'}_{x,y}}$', '$a^{(4)}_{q^{(i)'}_{x,y}}$', '$a^{(5)}_{q^{(i)'}_{x,y}}$' and '$a^{(1)}_{q^{(i)''}_{x,y}}$', '$a^{(2)}_{q^{(i)''}_{x,y}}$', '$a^{(3)}_{q^{(i)''}_{x,y}}$', '$a^{(4)}_{q^{(i)''}_{x,y}}$', '$a^{(5)}_{q^{(i)''}_{x,y}}$' respectively, where '$i$' denotes the orientation associated with the process.

The components in an 'h' type process are in turn represented by '$a^{(1)}_{h^{(i)}_x}$' and '$a^{(2)}_{h^{(i)}_x}$'.

Therefore the net input to the node can be described as :

$$
\begin{aligned}
net \;=\; & \sum_{x\in\{3,8\}}\sum_{y\in\{17,22\}} w_{p^{(n)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{p^{(n)}_{x,y}} + \qquad (4.91)\\
& \sum_{x\in\{3,8\}}\sum_{y\in\{16,18,21,23\}} \left(w_{q^{(n)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{q^{(n)}_{x,y}} + w_{q^{(n)'}_{x,y}} \cdot \prod_{z=1}^{5} a^{(z)}_{q^{(n)'}_{x,y}} + w_{q^{(n)''}_{x,y}} \cdot \prod_{z=1}^{5} a^{(z)}_{q^{(n)''}_{x,y}} \right) + \\
& \sum_{x\in\{17,21\}}\sum_{y\in\{3,8\}} w_{r^{(n,ne)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{r^{(n,ne)}_{x,y}} + \sum_{x\in\{17,23\}}\sum_{y\in\{3,8\}} w_{r^{(n,nw)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{r^{(n,nw)}_{x,y}} + \\
& \sum_{x\in\{17,22\}}\sum_{y\in\{2,4,7,9\}} \left(w_{q^{(n)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{q^{(n)}_{x,y}} + w_{q^{(n)'}_{x,y}} \cdot \prod_{z=1}^{5} a^{(z)}_{q^{(n)'}_{x,y}} + w_{q^{(n)''}_{x,y}} \cdot \prod_{z=1}^{5} a^{(z)}_{q^{(n)''}_{x,y}} \right) + \\
& \sum_{x\in\{4,8\}}\sum_{y\in\{17,22\}} w_{r^{(n,ne)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{r^{(n,ne)}_{x,y}} + \sum_{x\in\{2,8\}}\sum_{y\in\{17,22\}} w_{r^{(n,nw)}_{x,y}} \cdot \prod_{z=1}^{4} a^{(z)}_{r^{(n,nw)}_{x,y}} +
\end{aligned}
$$

$$\sum_{x\in\{5,9\}}\sum_{y\in\{18,24\}} w_{s_{x,y}^{(n,ne,nw)}}\cdot\prod_{z=1}^{4} a_{s_{x,y}^{(n,ne,nw)}}^{(z)} + \sum_{x\in\{16,20\}}\sum_{y\in\{1,7\}} w_{s_{x,y}^{(n,ne,nw)}}\cdot\prod_{z=1}^{4} a_{s_{x,y}^{(n,ne,nw)}}^{(z)} + \sum_{x=1}^{24} w_{h_x^{(n)}}.a_{h_x^{(n)}}^{(1)}.a_{h_x^{(n)}}^{(2)}$$

where :

$a_{p_{x,y}^{(n)}}^{(z)}$: components in a 'p' type functional process,

$a_{q_{x,y}^{(n)}}^{(z)}$: components in a 'q' type functional process,

$a_{q_{x,y}^{(n)'}}^{(z)}$: components in a 'q'' type functional process,

$a_{q_{x,y}^{(n)''}}^{(z)}$: components in a 'q''' type functional process,

$a_{r_{x,y}^{(n,ne)}}^{(z)}$: components in an 'r' type functional process where the edge elements indexed by 'x' are concerned with the north-east orientation,

$a_{r_{x,y}^{(n,nw)}}^{(z)}$: components in an 'r' type functional process where the edge elements indexed by 'x' are concerned with the north-west orientation,

$a_{s_{x,y}^{(n,ne,nw)}}^{(z)}$: components in an 's' type functional process where the edge elements indexed by 'x' and 'y' are concerned with the north-east and north-west orientations respectively,

$a_{h_x^{(n)}}^{(1)}$ and $a_{h_x^{(n)}}^{(2)}$: components in an 'h' type functional process,

$w_{p_{x,y}^{(n)}}$, $w_{q_{x,y}^{(n)}}$, $w_{q_{x,y}^{(n)'}}$, $w_{q_{x,y}^{(n)''}}$, $w_{r_{x,y}^{(n,ne)}}$, $w_{r_{x,y}^{(n,nw)}}$, $w_{s_{x,y}^{(n,ne,nw)}}$ and $w_{h_x^{(n)}}$: weights associated with the functional processes,

x, y : indexes the edge elements in the local edge pattern (HEP),

z : indexes the respective components in the functional processes.

The input to the neural node of the functional-link net (4.89) is a sigma-pi connection. A sigma-pi interconnection type has the ability to gate the inputs to the node [Rumelhart,

Hinton and McClelland 1986]. In this case, the set of orientations of the edge elements in the interconnection is used to gate the information (gradient magnitudes) before they are input to the node. This enables the simultaneous incorporation of all available information for processing.

The output of the node is determined by a sigmoidal activation function :

$$U_{ECDS}^{(n)} = \frac{1}{1+\exp(-(net+\alpha)/\beta)} - T_1 \qquad (4.92)$$

The parameter 'α' serves as a bias. The effect of 'α' is to shift the activation function along the horizontal axis. The effect of 'β' is to modify the shape of the sigmoid. A low value of 'β' tends to make the sigmoid take on the characteristics of a threshold-logic unit (TLU), whereas a high value of 'β' results in a more gently varying function. As shown in Fig. 4.11, the output of a sigmoidal function is never negative and only reaches zero when the total input to the node is an infinitely negative value. Similarly, the output never reaches one unless the node receives in total an infinitely positive value. In order to adjust the output of the neural node to cover the range from negative to positive values, a threshold parameter 'T_1' is added to the activation function. Therefore, if the net input is '0', the output of the node is '0'. If the computed output is greater than the threshold 'T_1', the output of the node is positive. Otherwise, the output of the node is negative.

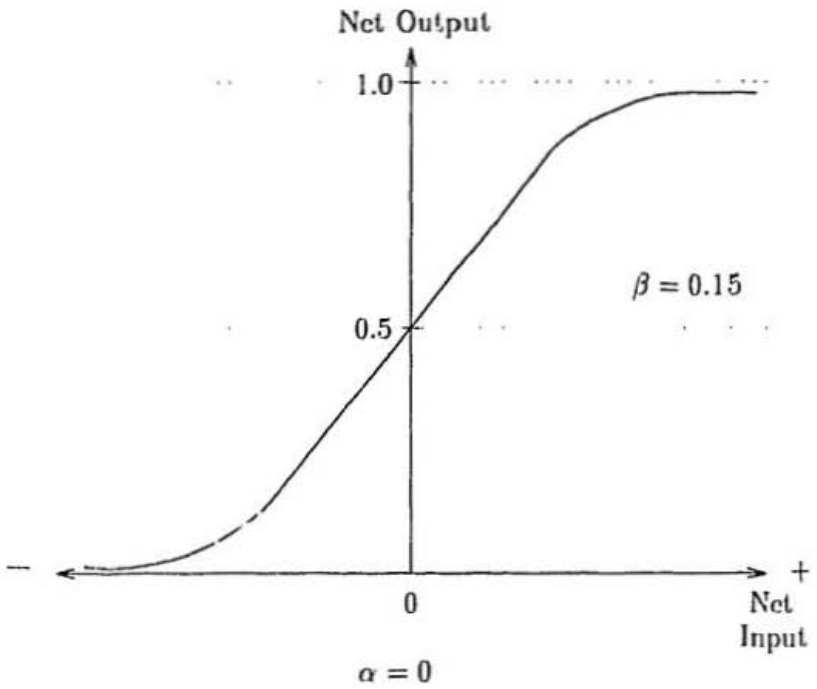


Figure 4.11 Characteristics of sigmoidal activation function

Choice of Parameter Values and Weights

The parameter 'α' is set at '0' so that the output of the node is '0' when the net input is '0'. The parameter 'β' is set at 0.15. This value results in a gently varying function which enables the range of values generated by the activation function to vary with the range of values of the input to the node. The parameter 'T_i' is set at 0.5. This value results in:

1. a zero output from the node when the input is zero;

2. a positive output from the node when the input is positive;

3. a negative output from the node when the input is negative.

Suitable weights for the functional processes are derived through a learning process described in chapter eight.

Chapter 5

Maximum Detection Subnet

5.1 Introduction

In Chapter four, the detection of edge contour(s) in a local edge pattern by the Edge Contour Detection Subnet was described. For each functional-link net in the Edge Contour Detection Subnet, the output denotes the strength of an edge contour with a particular orientation in the local edge pattern. Hence, the output shows the possibility of the presence of an edge contour with a particular orientation. The Maximum Detection Subnet in the second level of the hierarchical neural network system is introduced to determine the orientation of the most probable edge contour (if one exists) in the local edge pattern. If this orientation is the same as the orientation of the central element, then there is a very good possibility that the central element is a valid edge element through which the edge contour passes.

5.2 Maximum Detection Subnet Design

Most neural networks in the brain, especially those in the cerebral neocortex, are essentially layers of processing cells or neurons densely interconnected through lateral feedback [Kohonen 1984; Kandel and Schwartz 1985]. Fig. 5.1 is a schematic illustration of a layer of neurons. Each neuron receives the excitatory primary input 'P_i', that is, the initial input to the neuron and a number of excitatory and inhibitory lateral connections from the out-

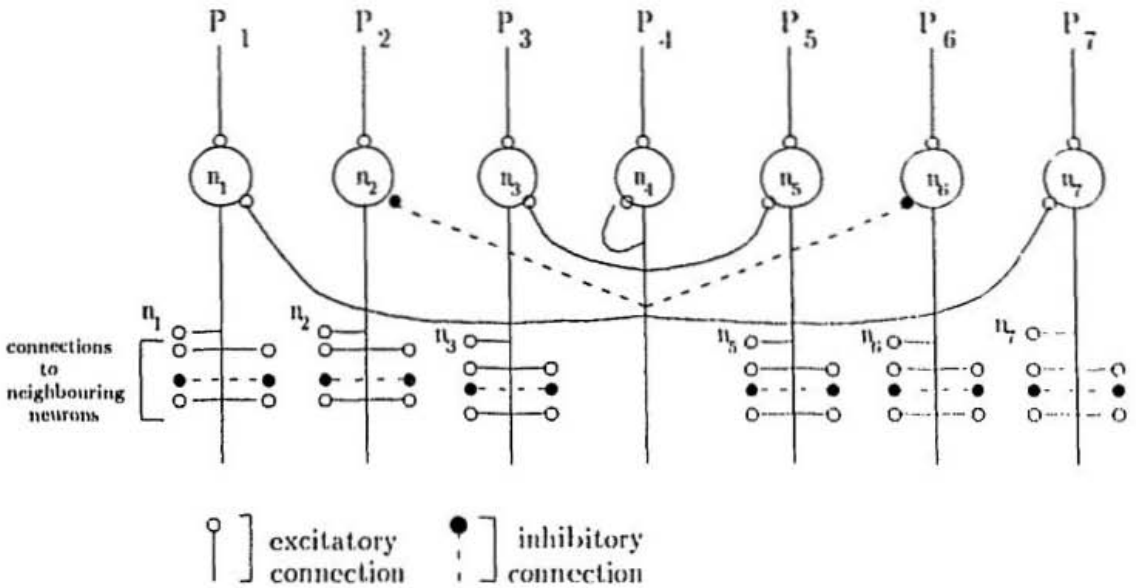


Figure 5.1 Laterally interconnected neurons

puts of other neighbouring neurons. Based on both anatomical and physiological evidences from the mammalian brains [Kohonen 1984; Kandel and Schwartz 1985], the following types of lateral interaction exist between neurons :

I : Short-range central lateral excitation : the excitatory area reaching up to a lateral radius of 50 to 100 μm (in primates).

II : Inhibitory action : the inhibitory area reaching up to a radius of 200 to 500 μm surrounds the central excitatory area.

III : Weaker excitatory action : an area reaching up to a radius of several centimeters in turn surrounding the inhibitory area.

The form of lateral interactions amongst neurons is a 'Mexican hat' (Fig. 5.2).

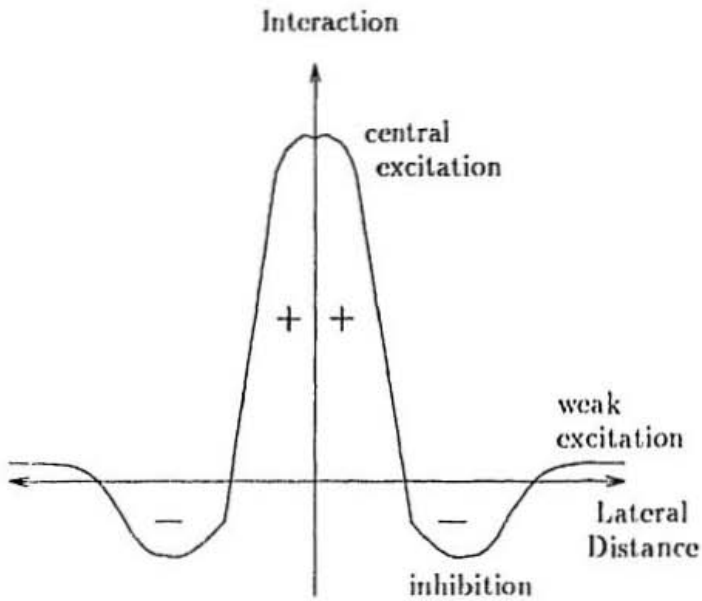


Figure 5.2 Schematic representation of lateral interaction

The design and mechanism of the Maximum Detection Subnet are based on these characteristics of biological systems in the mammalian brain. The lateral interconnections amongst the nodes in the Maximum Detection Subnet serve two purposes, namely :

1. A connection from a node 'n_i' to itself serves as an excitatory input for the node.

2. Connections from other neighbouring nodes 'n_j' to node 'n_i' where $j \neq i$ serve to inhibit the activity of the node 'n_i'.

The excitatory connection from a node to itself is analogous to the short-range central lateral excitation found in biological systems (type I). The inhibitory connections from neighbouring nodes are analogous to the area of inhibitory action surrounding the central

excitatory area (type II). For the Maximum Detection Subnet, there are some modifications to the biological concepts of lateral interactions amongst neurons. The area of inhibitory action and the amount of inhibition do not correspond to the distance from the excitatory area. All neighbouring nodes will impose an inhibiting action on the primary node (the node under consideration) and none of these nodes will excite the primary node.

The architecture of the Maximum Detection Subnet (MDS) consisting of a single layer of eight laterally interconnected nodes is shown schematically in Fig. 5.3.

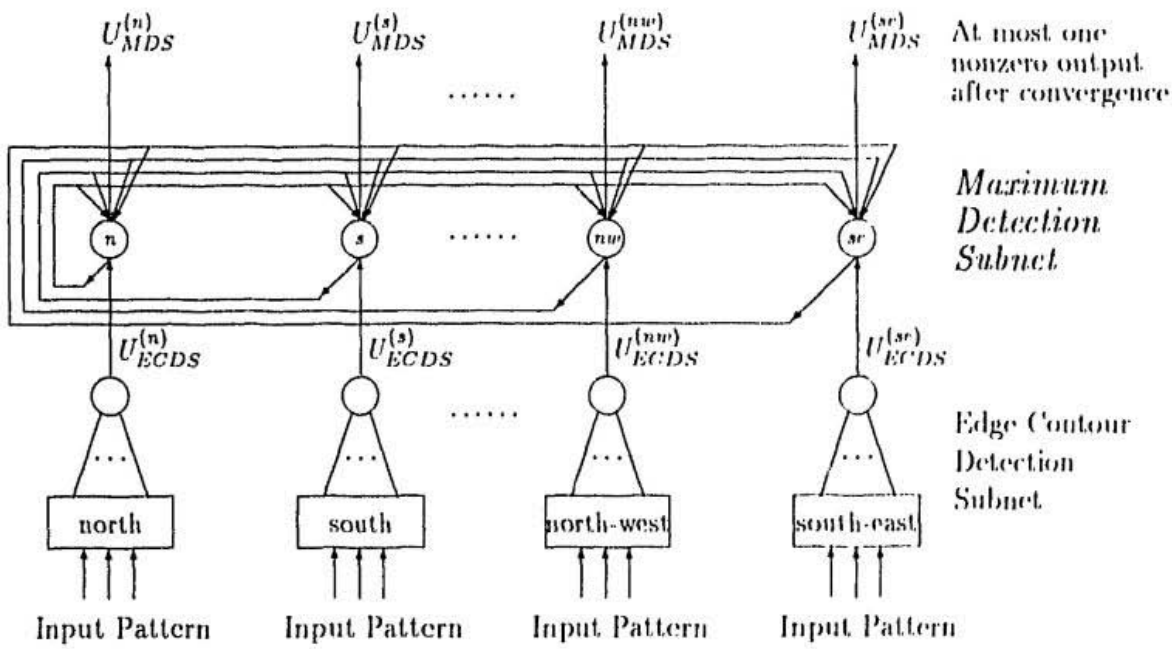


Figure 5.3 Architecture of Maximum Detection Subnet

The outputs from the functional-link nets in the Edge Contour Detection Subnet (ECDS) are presented as a set of initial (primary excitatory) inputs to the nodes in the Maximum

Detection Subnet. Each node in the Maximum Detection Subnet is associated with a particular orientation.

5.3 Mechanism of Maximum Detection Subnet

Each node in the Maximum Detection Subnet reinforces its own activity level by its output and suppresses the activity levels of neighbouring nodes by lateral inhibition. Eventually, only the node which received the largest initial input value from the Edge Contour Detection Subnet has a positive output.

In the Maximum Detection Subnet, the weight, w_{jk}, from node 'j' to node 'k' is defined as :

$$w_{jk} = \begin{cases} 1 & j = k \\ -\varepsilon & j \neq k,\ \varepsilon < \frac{1}{N},\ j,k \in \{n,\ldots,sc\}, \end{cases} \tag{5.1}$$

where 'N' denotes the number of nodes in the Maximum Detection Subnet.
The weights from each node to itself have a value of '1'. The weights to other nodes have a value of '$-\varepsilon$' where $\varepsilon < \frac{1}{N}$.

The computation of the output value of node 'j' at 'time' $t+1$ utilizes the output values computed at 'time' t. That is :

$$U^{(j)}_{(MDS)}(t+1) = \Phi(w_{jj}.U^{(j)}_{(MDS)}(t) + \sum_{k \neq j} w_{jk}.U^{(k)}_{(MDS)}(t)), \tag{5.2}$$

where $j,k \in \{n,\ldots,sc\}$. When the Maximum Detection Subnet starts processing ($t = 0$), the output value of each node is initialized to the initial input from the corresponding functional-link net if the input is positive, otherwise the output value is initialized to '0'.

That is,

$$U^{(i)}_{(MDS)}(0) = \begin{cases} U^{(i)}_{(ECDS)} & \text{if } U^{(i)}_{(ECDS)} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad (5.3)$$

$$where \quad : \quad i \in \{n, \ldots, sc\}.$$

The output $U^{(j)}_{(MDS)}(t+1)$ will tend to be 'laterally inhibited' by all the outputs from the neighbouring nodes computed in the previous iteration. On the other hand, the output $U^{(j)}_{(MDS)}(t+1)$ of a node 'j' will tend to be 'laterally excited' by the node's own output computed in the previous iteration. The amount of inhibition on each node is therefore dependent on the number of neighbouring activated nodes and the strength of the output of these nodes. The output of each node is affected by its output strength in the previous iteration and the output strength of the neighbouring nodes.

The activation function of each node, $\Phi(Y)$ is described by :

$$\Phi(Y) = \begin{cases} 1.0 & \text{if } Y \geq 1.0 \\ Y & \text{if } T_2 < Y < 1.0 \\ 0 & \text{if } Y \leq T_2, \end{cases} \qquad (5.4)$$

where 'Y' denotes the input to the function and 'T_2' denotes the threshold value for activation.

If the argument 'Y' of the function '$\Phi(Y)$' is greater than the threshold value 'T_2', then the output $U^{(j)}_{(MDS)}(t+1)$ will be positive, otherwise it will be driven to '0'. When the output of a node is driven to '0', the node is de-activated.

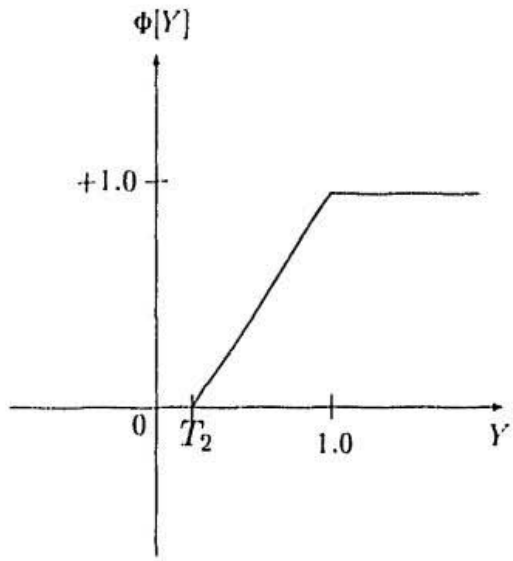


Figure 5.4 Characteristics of function 'Φ'

The activation function 'Φ' is often refered to as the subthreshold summation activation function [Jordan 1986] (Fig. 5.4). The input to each node never reaches the saturation level '1'. Since the excitatory weight is '1', the amount of excitation is never greater than the activity level in the previous iteration. Also, since the output of each functional-link net is never equal to or greater than '1', the input to each node in the Maximum Detection Subnet is always below the saturation level '1'.

In the experimental tests carried out by the hierarchical neural network system, when convergence is attained, the outputs of the nodes in the Maximum Detection Subnet are driven to '0' except for the node with the largest initial input. The remaining positive output of the node is then set to '1' and fed to the following subnets :

1. The Gradient Adjustment Subnet in level three.

2. The Orientation Determination Subnet in level four.

The output from the Maximum Detection Subnet (as an excitatory signal) provides the information concerning the most probable orientation for the central edge element.

Two difficult situations could be encountered by the Maximum Detection Subnet. Firstly, if at least two input values are equal to the largest input value, the output of the nodes with the same largest values are driven to zero simultaneously as a result of lateral feedback whereby these nodes receive the same amount of inhibition from each other simultaneously. However, a local edge pattern very unlikely has two or more edge contours with different orientations but having equal strength. Even if this situation occurs for some edge elements, the edge measurements of their neighbouring edge elements are also concurrently adjusted during the iteration. In the next iteration, the strength of the edge contours in the edge pattern are changed based on the information of the neighbouring edge elements. Hence, such a situation will not happen in following iterations and the edge elements will ultimately be determined. Secondly, convergence may not be achieved quickly because the input values to the nodes are normalized and these values could be driven to very s...all positive values before reaching negative. More iterations are required in order to drive the values to zero. In order to shorten the processing time without the expense of incorrect or inaccurate output results, the threshold value 'T_2' for the activation function is set to a small positive value. As a result, convergence could be achieved faster and the node with the largest initial input value is also correctly determined by the Maximum Detection Subnet.

Chapter 6

Gradient Adjustment Subnet

6.1 Introduction

The task of the Maximum Detection Subnet is to determine the orientation of the most probable edge contour in a local edge pattern. The Subnet uses the information provided by the Edge Contour Detection Subnet in level one which concerns the strength of edge contours with different orientations. The orientation of the most probable edge contour in the local edge pattern is then used to affect the adjustment of the edge measurement.

In this chapter, the subnet in the third level of the hierarchical neural network system, called the Gradient Adjustment Subnet, is introduced to perform four primary tasks, namely;

Task 1 : To determine the occurrence of appropriate conditions for adjusting the gradient magnitude of the central element.

Task 2 : To determine the appropriate amount of adjustment to the gradient magnitude.

Task 3 : To compute the new adjusted gradient magnitude.

Task 4 : To determine if the central element is an edge element or a non-edge element.

The subnet in layer one of the Gradient Adjustment Subnet is called the Condition Ascertainment Subnet. The subnet performs tasks one and two by using the information provided

by the Maximum Detection Subnet and set $D'_X = \{d_X^{(n)'}, d_X^{(s)'}, \ldots, d_X^{(sw)'}, d_X^{(se)'}\}$ which contains the complement of the direction values of the central element. These information are utilized simultaneously in order to determine correctly the appropriate conditions for the adjustment of the gradient magnitude as well as the appropriate amount of adjustment. The structure of the Condition Ascertainment Subnet is based on the concept of the functional link with the tensor model [Klassen, Pao and Chen 1988]. Tasks three and four are performed by the Gradient Computation Subnet in layer two of the Gradient Adjustment Subnet. The new adjusted gradient magnitude of the central element is computed based on the current gradient magnitude of the central element and the amount of adjustment provided by the Condition Ascertainment Subnet.

6.2 Conditions for Edge Measurement Adjustment

There are two cases for reinforcing and two cases for suppressing the gradient magnitude of the central element in a local edge pattern. These are detailed in the following sections.

6.2.1 Cases for Reinforcement

Reinforcement is used to enhance detected edges, strengthen weak edges, and recover missing edges.

Case 1 : If the orientation of the central element is the same as the orientation of the edge contour which has the greatest possibility of occurrence in the local edge pattern.

This condition is effective in reinforcing a valid or true edge element as the edge contour

passes through the central element.

Case 2 : If the central element is considered as a non-edge element, but the edge contour which passes through the central element has been determined to be the most probable edge contour in the local edge pattern.

In this situation, the central element should be a valid edge element which constitutes part of the edge contour. Therefore, the central element should be considered as an edge element. That is, the gradient magnitude of the central element should be initialized and an appropriate orientation should be assigned to the central element. This condition is effective in recovering missing edge elements (i.e. interpolation).

6.2.2 Cases for Suppression

Suppression is used to eliminate noise as well as false and spurious edges.

Case 1 : If the orientations of the central element and the edge contour which is most likely to be present in the local edge pattern are different.

This condition is effective for removing noise and false edge elements.

Case 2 : If there is no edge contour in the local pattern.

In this situation, the central element in the local pattern is most probably a false edge element or isolated noise. Hence, the gradient magnitude of the central element should be suppressed.

6.3 Condition Ascertainment Subnet

The subnet in layer one of the Gradient Adjustment Subnet, called the Condition Ascertainment Subnet (CAS), performs the task of ascertaining the occurrence of the appropriate conditions for adjusting the gradient magnitude of the central element. The Condition Ascertainment Subnet also determines the amount of adjustment to the gradient magnitude of the central element.

6.3.1 Selective Tensor Model

In the general tensor model, each component in the input vector multiplies the entire input vector to generate a vector of enhanced components. The model yields an entire vector from each of the individual components. However, such functional transforms greatly increase the number of processes which are generated as a result of the enhancement to the components in the input vector. Most of the processes do not provide the necessary information to determine the occurrence of the conditions for adjusting the gradient magnitude of the central element. To select only those relevant processes which provide the necessary information, the functional-link net for the Condition Ascertainment Subnet is especially designed and is refered to as the Selective Tensor Model.

The 'a' type functional process selected for the functional link is used to ascertain the occurrence of the condition(s) for reinforcing the gradient magnitude of the central element. The activation of an 'a' type process reinforces the gradient magnitude of the central element. If none of the 'a' type processes is activated, the gradient magnitude of the central

element is suppressed.

6.3.1.1 Processes for Reinforcing Gradient Magnitude

An 'a' type functional process, '$a^{(i)}$', associated with the 'i^{th}' orientation where $i \in \{n,\ldots,sc\}$ is described by :

$$a^{(i)} = \prod_{j\in\{n,\ldots,sc\}}^{j\neq i} d_X^{(j)'} * U_{MDS}^{(i)} \qquad (6.1)$$

where '$d_X^{(j)'}$' denotes the complement of the direction value associated with the 'j^{th}' orientation, $j \in \{n,\ldots,sc\}$ and '$U_{MDS}^{(i)}$' denotes the output from the Maximum Detection Subnet (MDS) for the 'i' orientation, $i \in \{n,\ldots,sc\}$. There are eight 'a' type functional processes in the functional-link net. Each of the 'a' type functional processes is associated with a particular orientation.

One condition for reinforcing the gradient magnitude of the central element is that the orientations of the central element and the most probable edge contour in the local edge pattern be the same. For example, if the orientation of the central element is north and the most probable edge contour in the local edge pattern also has a north orientation, then the '$a^{(n)}$' functional process associated with the north orientation is described by :

$$a^{(n)} = d_X^{(s)'} * d_X^{(e)'} * d_X^{(w)'} * d_X^{(ne)'} * d_X^{(nw)'} * d_X^{(sw)'} * d_X^{(se)'} * U_{MDS}^{(n)}. \qquad (6.2)$$

Since the orientation of the central element is north, the direction values associated with any other orientations besides north ($d_X^{(s)},\ldots,d_X^{(se)}$) are '0's. Therefore, the complements of these direction values ($d_X^{(s)'},\ldots,d_X^{(se)'}$) are '1's. The output from the Maximum Detection

Subnet ($U^{(n)}_{MDS}$) is '1' since the most probable edge contour in the local edge pattern has a north orientation. Therefore,

$$a^{(n)} = 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 = 1. \qquad (6.3)$$

Hence, the activation of process '$a^{(n)}$' will reinforce the gradient magnitude.

Another condition for reinforcing is that the central element is a non edge element but the edge contour which passes through the central element is the most probable edge contour in the local edge pattern. For example, if a central element is a non-edge element and the most probable edge contour in the local edge pattern has a north orientation, then the direction values associated with all the orientations ($d^{(n)}_X, \ldots, d^{(se)}_X$) are '0's because the central element has no orientation. Therefore, the complements of these direction values ($d^{(n)'}_X, \ldots, d^{(se)'}_X$) are '1's. The signal value from the Maximum Detection Subnet ($U^{(n)}_{MDS}$) is '1' since the most probable edge contour in the local edge pattern has a north orientation. Therefore,

$$a^{(n)} = 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 = 1. \qquad (6.4)$$

Hence, the activation of process '$a^{(n)}$' will result in a reinforcement to the gradient magnitude.

6.3.1.2 Suppressing Gradient Magnitude

If none of the 'a' type functional processes is activated, the gradient magnitude of the central element is suppressed. There are two cases whereby none of the 'a' type functional processes is activated.

The first case is when the orientations of the central element and the most probable edge contour in the local edge pattern are different. For example, if the orientation of the

central element is north and the orientation of the most probable edge contour in the local edge pattern is south, then, $d_X^{(n)} = 1$; $d_X^{(n)'} = 0$; $d_X^{(s)}, \ldots, d_X^{(se)} = 0$; $d_X^{(s)'}, \ldots, d_X^{(se)'} = 1$; $U_{MDS}^{(s)} = 1$; $U_{MDS}^{(n)}, U_{MDS}^{(e)}, \ldots, U_{MDS}^{(se)} = 0$. The '$a^{(n)}$' functional process is not activated because $U_{MDS}^{(n)} = 0$.

$$a^{(n)} = d_X^{(s)'} * d_X^{(e)'} * d_X^{(w)'} * d_X^{(ne)'} * d_X^{(nw)'} * d_X^{(sw)'} * d_X^{(se)'} * U_{MDS}^{(n)} \qquad (6.5)$$

$$= 1 * 1 * 1 * 1 * 1 * 1 * 1 * 0 = 0.$$

The other 'a' type functional processes are also not activated because $d_X^{(n)'} = 0$. For example,

$$a^{(s)} = d_X^{(n)'} * d_X^{(e)'} * d_X^{(w)'} * d_X^{(ne)'} * d_X^{(nw)'} * d_X^{(sw)'} * d_X^{(se)'} * U_{MDS}^{(s)} \qquad (6.6)$$

$$= 0 * 1 * 1 * 1 * 1 * 1 * 1 * 1 = 0.$$

Since none of the 'a' type functional processes is activated, there is no signal for reinforcement. Hence, the gradient magnitude of the central element will be suppressed.

In the second case, suppression to the gradient magnitude of the central element occurs if there is no edge contour in the local pattern ($U_{MDS}^{(n)}, \ldots, U_{MDS}^{(se)} = 0$). Hence, $a^{(n)}, \ldots, a^{(se)} = 0$. As a result, there is no signal for reinforcement and the gradient magnitude of the central element will be suppressed.

6.3.1.3 Functional Link with the Selective Tensor Model

In the selective tensor model (Fig. 6.1), selected processes are the 'a' type functional processes. The input vectors with sets of components $\{u_i\}$ and $\{v_i\}$ can be enhanced to the sets of components $\{u_i * v_i\}$, $\{u_i * v_i * \overset{j>i}{v_j}\}$, $\{u_i * v_i * \overset{j>i}{v_j} * \overset{k>j}{v_k}\}$, ... where '$i$', '$j$', '$k$' indexes

the components in the input vectors. The enhanced components are explicitly available to the net. It enables the appropriate processes to be activated only if the input information indicates that all the necessary conditions for activation are satisfied. The joint activations or simultaneous utilization of the information in the input vectors allows the information to gate each other. Hence, the functional-link net is able to determine the conditions for modifying the gradient magnitude of the central element. There are two input vectors to the functional-link net :

1. A vector containing the complement of the direction values of the central element.

2. A vector containing the output values from the Maximum Detection Subnet.

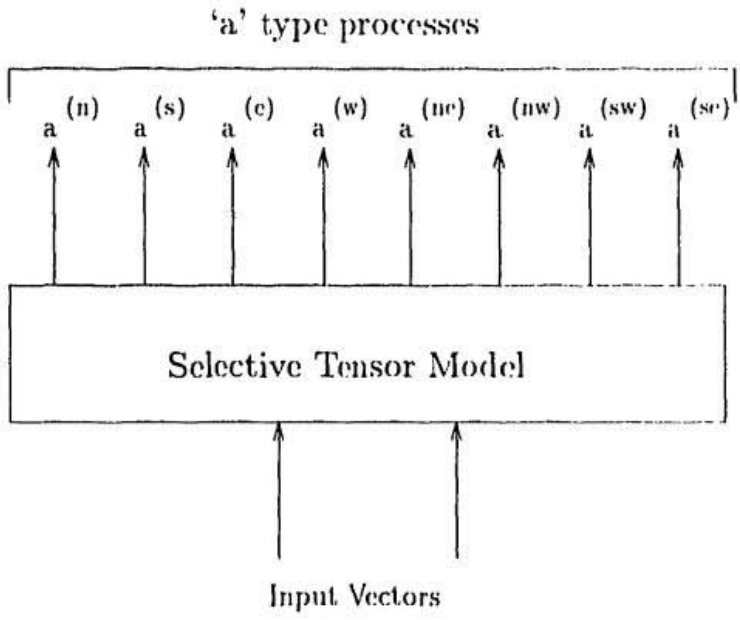


Figure 6.1 Schematic illustration of a selective tensor model

In Fig. 6.2, the Condition Ascertainment Subnet consists of a node and a set of selected functional processes generated by the Selective Tensor Model. The forward connections from the functional processes provide inputs to the node.

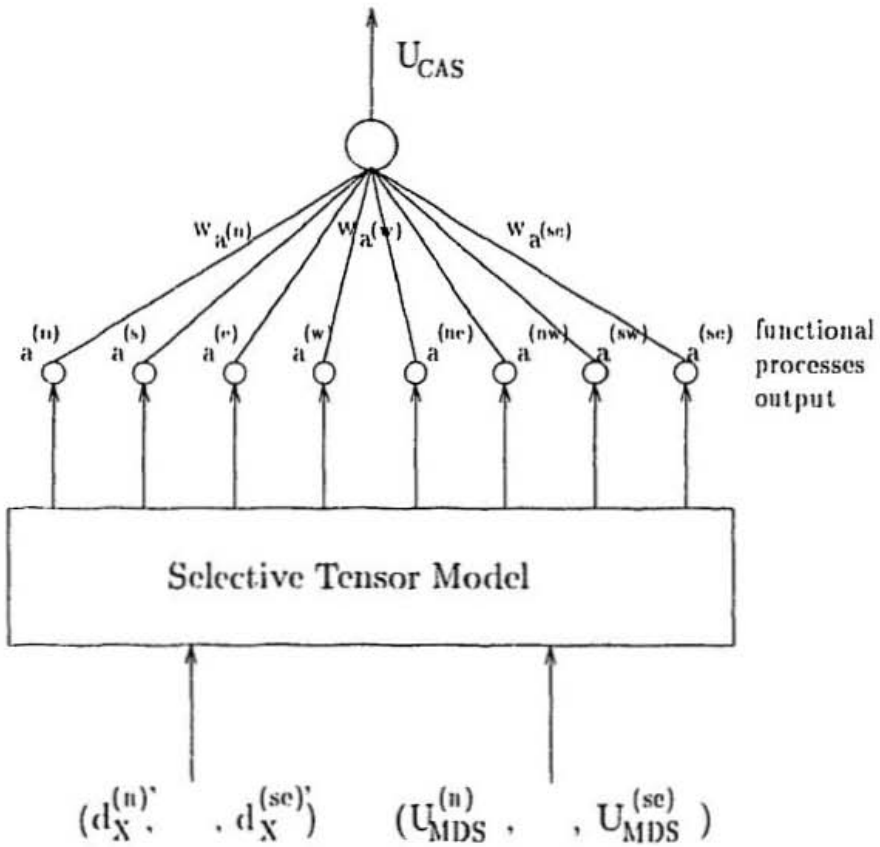


Figure 6.2 Architecture of Condition Ascertainment Subnet

6.3.2 Mechanism of Condition Ascertainment Subnet

The neural node of the Condition Ascertainment Subnet (CAS) is characterized by a hard-limiter nonlinearity. The node is activated in accordance with the net input to the node, the hard-limiter activation function, and a threshold 'T_3'. The net input to the node

is the sum of the weighted outputs of the functional processes and is given by :

$$net_{CAS} = \sum_{i \in \{n,\ldots,se\}} a^{(i)} * w_{a(i)} \qquad (6.7)$$

where '$a^{(i)}$' denotes the output of the 'a' type functional process associated with the 'i' orientation, and '$w_{a(i)}$' denotes the weight associated with the 'a' type functional process for the 'i' orientation.

The output of the node of the Condition Ascertainment Subnet is given by :

$$U_{CAS} = \mu[net_{CAS}], \qquad (6.8)$$

where 'μ' is a hard-limiter function which is described by :

$$\mu[net_{CAS}] = \begin{cases} 0.05 & \text{if } net_{CAS} \geq T_3 \\ -0.08 & \text{otherwise.} \end{cases} \qquad (6.9)$$

The function 'μ' yields either a positive or negative value depending on the strength of the net input to the node. If the net input exceeds the threshold 'T_3' (which is set to '1'), the positive output of the node is the amount of reinforcement to the gradient magnitude of the central element. Otherwise, the negative output of the node is the amount of suppression to the gradient magnitude. A value of 0.05 enables the gradient magnitude of an edge element to be gradually reinforced, without excessive increases in the edge strength in each iteration. On the other hand, a value of -0.08 enables sufficient suppression to the gradient magnitude of an element in each iteration without eliminating an element entirely in the first few iterations. This would prevent the elimination of weak edges which might initially be assumed to be false edges or noise but subsequently these edges are determined to be

true edges as a result of the propagation of global information. The characteristics of the hardlimiter function 'μ' are shown in Fig. 6.3.

Choice of Weights

The weight associated with an 'a' type functional process should be properly assigned to ensure that the net input to the node exceeds the threshold 'T_3'. If any of the 'a' type functional processes is activated, the output of the node is 0.05.

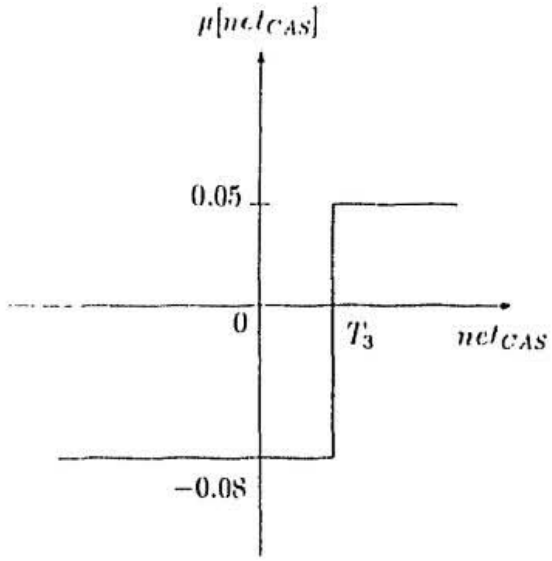


Figure 6.3 Characteristics of function 'μ'

6.4 Gradient Computation Subnet

The subnet in layer two of the Gradient Adjustment Subnet, called the Gradient Computation Subnet (GCS), performs two functions : firstly, to compute the newly adjusted gradient magnitude of the central element; and secondly, to determine if the central element

is considered an edge element or a non-edge element. The Gradient Computation Subnet is a semilinear feedforward net with no hidden layers (Fig. 6.4). In the layer 'j', there are two inputs from two sources :

1. The output of the Condition Ascertainment Subnet.

2. The current gradient magnitude of the central element.

The outputs from layer 'j' are then fed to the nodes in layer 'k' through feedforward connections. There are two nodes in the output layer 'k', namely, '$nd_k^{(1)}$' and '$nd_k^{(2)}$'.

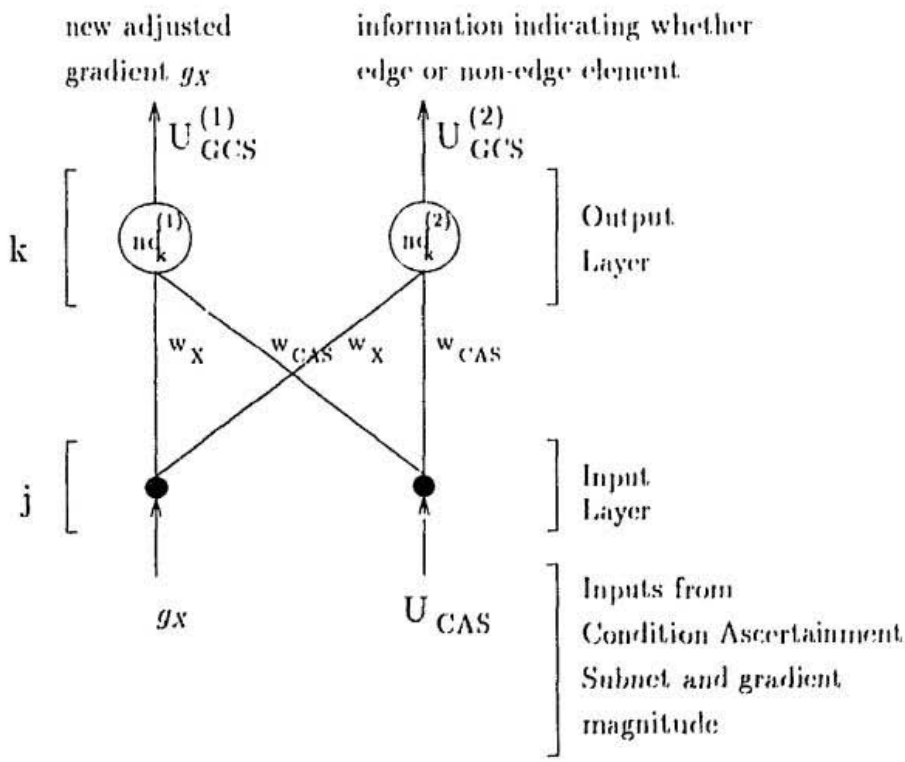


Figure 6.4 Architecture of Gradient Computation Subnet

The net input to each node in layer 'k' is the sum of the weighted outputs from layer 'j' and is given by :

$$net_k = g_X * w_X + U_{CAS} * w_{CAS} \qquad (6.10)$$

where 'U_{CAS}' denotes the output from the Condition Ascertainment Subnet; 'g_X' is the gradient magnitude of the central element prior to adjustment; 'w_X' and 'w_{CAS}' are the associated weights with value of '1'.

6.4.1 Newly Adjusted Gradient Magnitude for Central Element

The newly modified gradient magnitude for the central element is given by the output value of node '$nd_k^{(1)}$' in the output layer 'k' of the GCS :

$$U_{GCS}^{(1)} = \varphi[net_k] \qquad (6.11)$$

where 'φ' is the activation function of node '$nd_k^{(1)}$'.

The activation function 'φ' is characterized by a nonlinearity termed subthreshold summation [Jordan 1986], and is described by :

$$\varphi[net_k] = \begin{cases} 1.0 & \text{if } net_k > 1.0 \\ net_k & \text{if } T_3' \le net_k \le 1.0 \\ g_0(0.015) & \text{otherwise} \end{cases} \qquad (6.12)$$

where 'T_3'' is a threshold and 'g_0' is set to a value of 0.015.

The activation function 'φ' will output a minimum positive, non-zero output value 'g_0' if the net input to the node is less than a pre-set threshold value 'T_3''. Therefore, if an

element after adjustment has a gradient magnitude less than the threshold 'T_3'', the gradient magnitude of the element will be set to a value 'g_0'. This element consequently will be considered as a non-edge element. In the next iteration, the minimum gradient magnitude 'g_0' will provide suppression in the Edge Contour Detection Subnet. The characteristics of the function 'φ' are shown in Fig. 6.5.

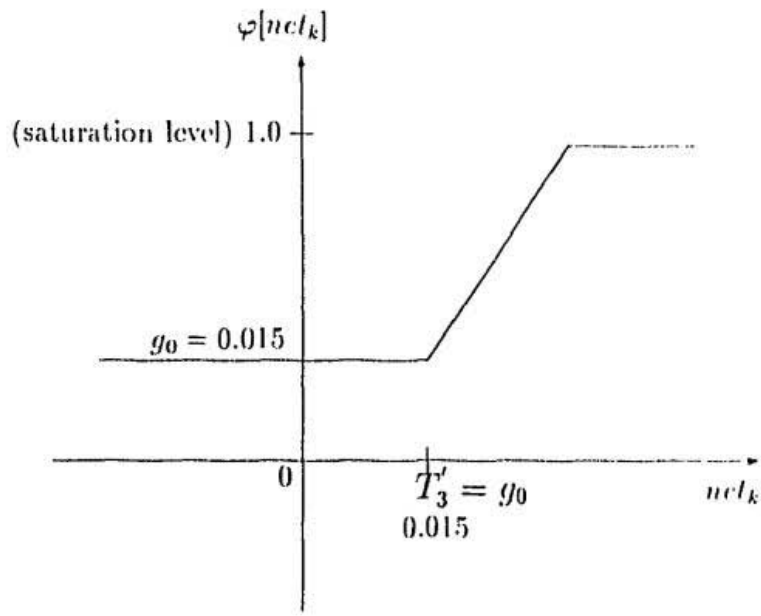


Figure 6.5 Characteristics of function 'φ'

The output of the node is linear if the net input to the node exceeds threshold value 'T_3''. The activation function 'φ' is also constructed taking into account that all physical systems have a limited dynamic range, that is, the response of a system cannot exceed a certain maximum response.

As shown in Fig. 6.5, if the output reaches the saturation level, no further increase in the output is allowed and the output of the function is set to '1'.

6.4.2 Determining Non-Edge Element

Node '$nd_k^{(2)}$' determines if the central element is considered as an edge element or as a non-edge element. Activation of the node '$nd_k^{(2)}$' indicates the central element as a non-edge element. Conversely, de-activation of the node indicates the central element as an edge element. Output of node '$nd_k^{(2)}$' provides information to the Orientation Determination Subnet in level four of the hierarchical neural network system to modify the orientation of the central element. The output of node '$nd_k^{(2)}$' is given by :

$$O_{GC's}^{(2)} = \psi[net_k] \tag{6.13}$$

where 'ψ' is the activation function of node '$nd_k^{(2)}$'.

The activation function 'ψ' is described by (see Fig. 6.6) :

$$\psi[net_k] = \begin{cases} 1 & \text{if } net_k < T_3''' \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

where 'T_3'''' is a threshold and is set to a value of 0.015.

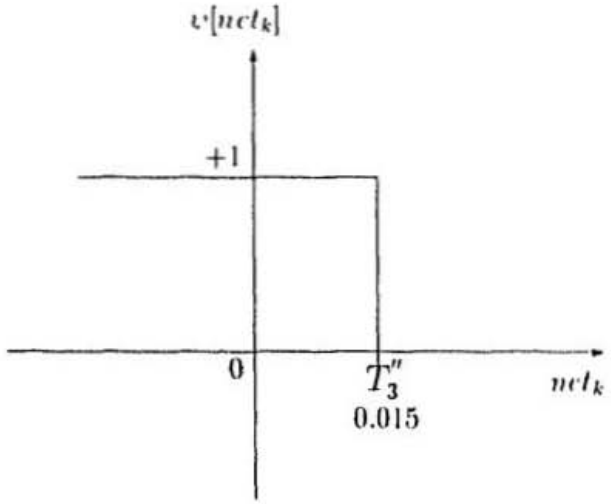


Figure 6.6 Characteristics of function 'ψ'

The output of the function 'ψ' is dependent on the net input to the node. Since the threshold values for both activation functions in nodes '$nd_k^{(1)}$' and '$nd_k^{(2)}$' are the same (T_3', T_3'' - 0.015), the output values of these nodes are correlated. That is, the outcome of the activation of node '$nd_k^{(1)}$' is reflected in the output of node '$nd_k^{(2)}$' and vice versa. For example, if the output of node '$nd_k^{(1)}$' is less than threshold value 'T_3'' (0.015), the central element is considered as a non-edge element and hence its gradient magnitude is assigned value 0.015. This is also reflected by the activation of node '$nd_k^{(2)}$'. An output value of '1' from node '$nd_k^{(2)}$' indicates the central element as a non-edge element. On the other hand, if the output of node '$nd_k^{(1)}$' is greater than the threshold value 'T_3''', the central element is considered as an edge element and correspondingly, the output value of node '$nd_k^{(2)}$' is '0'. The output value from node '$nd_k^{(2)}$' is provided as information to the Orientation Determination Subnet (discussed in the next chapter) in level four to determine the appropriate orientation for the central element.

Chapter 7

Orientation Determination Subnet

7.1 Introduction

In chapter six, the computation of the new gradient magnitude for the central element by the Gradient Adjustment Subnet was described. The Gradient Adjustment Subnet also determines whether the central element is to be considered as an edge element or a non-edge element.

In this chapter, the Orientation Determination Subnet in level four of the hierarchical neural network system is introduced. Its function is to generate the new set of direction values for the central element whose orientation is modified only when the status (edge or non-edge) of the central element is changed. Otherwise, the orientation of the central element remains unchanged. The Orientation Determination Subnet ascertains the occurrence of these conditions to determine the appropriate orientation for the central element. The orientation of the central element can be modified in agreement with the orientations of the surrounding edge elements in the structure of an edge contour.

7.2 Architecture of Orientation Determination Subnet

The Orientation Determination Subnet is a two-layer semilinear feedforward net (Fig. 7.1). In the input layer 'I', there are seventeen inputs. These inputs originate from three

sources. The first source is from the output of the Maximum Detection Subnet in level two which provides information about the orientation of the most probable edge contour in the local edge pattern. The second source is from the output of the Gradient Adjustment Subnet in level three which determines whether the central element is an edge element or a non-edge element. The third source is from the set of direction values for the central element. In the output layer 'm', there are eight nodes, with each node associated with a particular orientation, namely, north, ..., south-east. The outputs from layer 'l' are connected to the nodes in layer 'm' through feedforward connections.

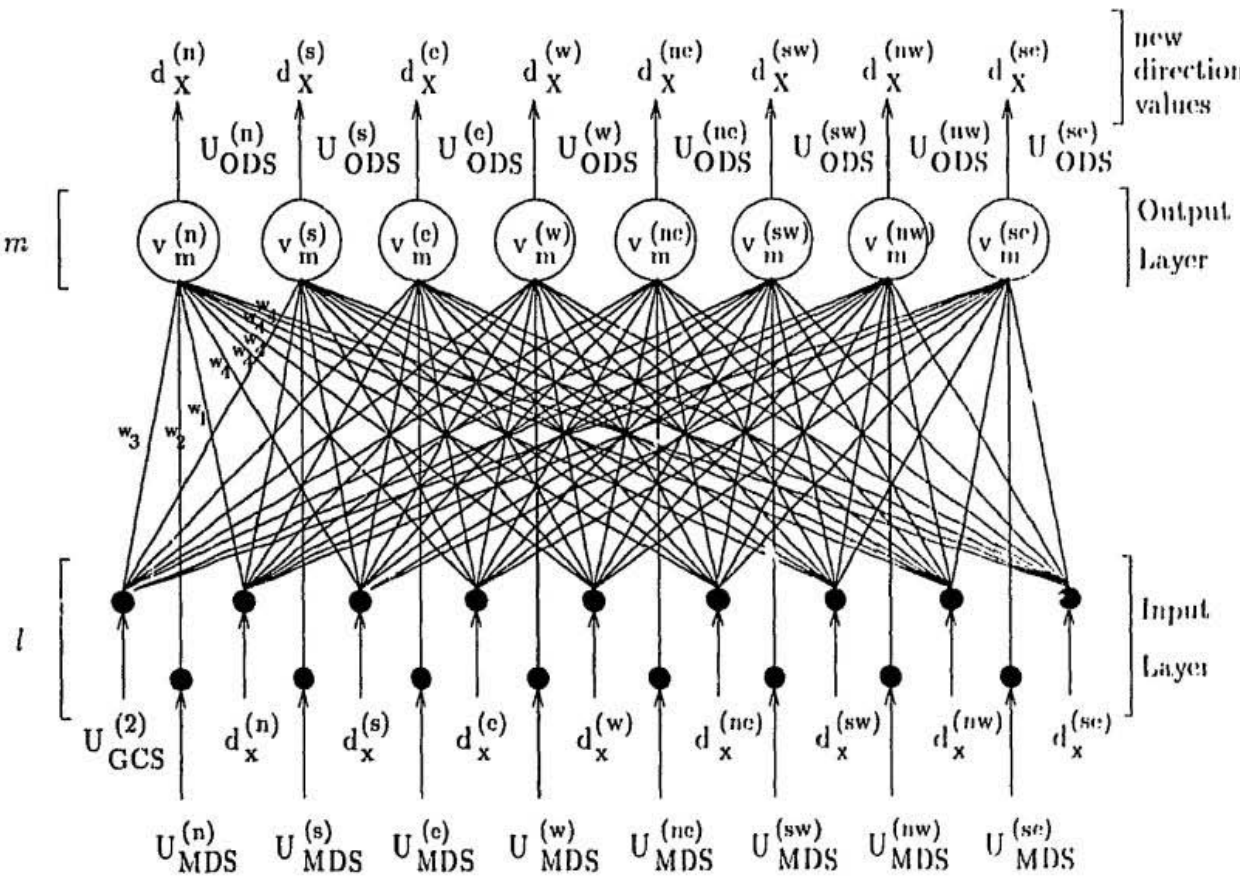


Figure 7.1 Architecture of Orientation Determination Subnet

7.3 Mechanism of Orientation Determination Subnet

Each output from layer 'l' sends the signal value to the nodes denoted by '$v_m^{(i)}$' ($i \in \{n, \ldots, sc\}$) in layer 'm'. The net input to each node, '$v_m^{(i)}$', is the sum of the weighted outputs from layer 'l'. The inputs to a node '$v_m^{(i)}$' consist of :

1. The set of eight direction values for the central element, '$d_X^{(j)}$', $j \in \{n, \ldots, sc\}$.

2. The signal from the Gradient Adjustment Subnet (Gradient Computation Subnet), '$U_{GCS}^{(2)}$'.

3. The signal from the corresponding node in the Maximum Detection Subnet, '$U_{MDS}^{(i)}$'.

For example, node '$v_m^{(n)}$' associated with the north orientation receives its input from the output, '$U_{MDS}^{(n)}$', of the node in the Maximum Detection Subnet associated with the north orientation. Therefore, the net input to a node '$v_m^{(i)}$' in layer 'm' is given by :

$$net_m^{(i)} = (d_X^{(i)} * w_1) + (U_{MDS}^{(i)} * w_2) + (U_{GCS}^{(2)} * w_3) + \sum_{k \in \{n,\ldots,sc\}}^{k \neq i} d_X^{(k)} * w_4 \qquad (7.1)$$

where :

$d_X^{(i)}$, $d_X^{(k)}$: The direction values for the central element;

$U_{MDS}^{(i)}$: The output signal associated with the 'i^{th}' orientation from the Maximum Detection Subnet;

$U_{GCS}^{(2)}$: The output signal from the Gradient Adjustment Subnet (Gradient Computation Subnet);

$w_1, \ldots, w_4$: the associated connection weights.

The inputs to the node serve either to excite or inhibit the activity of the node. The excitatory effects on the node '$v_m^{(i)}$' are provided by :

1. The direction value, '$d_X^{(i)}$'.

2. The signal from the Maximum Detection Subnet, '$U_{MDS}^{(i)}$'.

A positive weight '2' is assigned to the '*excitatory*' connections (w_1, w_2) to accentuate the excitatory signals and to provide sufficient strength to exceed the threshold for activating the node. The inhibitory effects on the node '$v_m^{(i)}$' are provided by :

1. The direction values, '$d_X^{(k)}$', $k \neq i$.

2. The signal from the Gradient Adjustment Subnet, '$U_{GCS}^{(2)}$'.

These input signals serve to de-activate the node. A negative value '-4' is assigned to each of the '*inhibitory*' connection weights (w_3, w_4) to accentuate the inhibitory signals and to provide sufficient strength to de-activate the node. The output of a node '$v_m^{(i)}$' in layer 'm' is given by :

$$U_{ODS}^{(i)} = \Psi[net_m^{(i)}], \quad i \in \{n, \ldots, sc\} \tag{7.2}$$

where 'Ψ' is a nonlinear activation function and takes on the characteristics of a threshold-logic unit (TLU) [Jordan 1986] :

$$\Psi[net_m^{(i)}] = \begin{cases} 1 & \text{if } net_m^{(i)} > T_4 \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

The node, '$v_m^{(i)}$', is activated only if the net input to the node exceeds the threshold 'T_4' which is set to '1', otherwise, the node is de-activated. The function 'Ψ' is shown in Fig. 7.2.

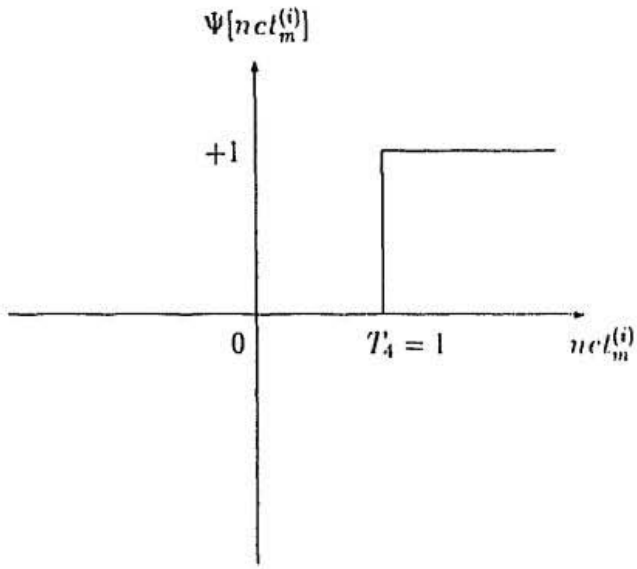


Figure 7.2 Characteristics of function 'Ψ'

7.4 Case Analysis for Operation of Orientation Determination Subnet

A node is activated if the input to the node is greater than '1', i.e. $net_m^{(i)} > 1$, otherwise, the node is de-activated.

7.4.1 Cases for Activation

A node '$v_m^{(i)}$', $i \in \{n, \ldots, se\}$ is activated only if both the following two conditions are

satisfied :

1. The node does not receive any de-activating input signal, i.e. from '$U_{GCS}^{(2)}$' or '$d_X^{(j)}$', where $j \in \{n, \ldots, se\}$ and $j \neq i$.

2. The node receives an activating input signal, i.e. from '$U_{MDS}^{(i)}$' or '$d_X^{(i)}$'.

The activation cases for node '$v_m^{(i)}$' are :

Case 1 : (i) The central element is considered as an edge element ($U_{GCS}^{(2)} = 0$), and (ii) the orientations associated with the node and the central element are the same ($d_X^{(i)} = 1$).

For example, the central element of the input edge pattern is an edge element and has a north orientation, i.e. '$d_X^{(n)} = 1$', '$d_X^{(s)}, \ldots, d_X^{(se)} = 0$', and '$U_{GCS}^{(2)} = 0$'. Therefore, the net input, '$net_m^{(n)}$', to node '$v_m^{(n)}$' is given by :

$$\begin{aligned} net_m^{(n)} &= (d_X^{(n)} * w_1) + (U_{MDS}^{(n)} * w_2) + (U_{GCS}^{(2)} * w_3) + \sum_{j \in \{s,\ldots,se\}} d_X^{(j)} * w_4 \qquad (7.4) \\ &= (1 * 2) + (U_{MDS}^{(n)} * 2) + (0 * -4) + (0 * -4) + \cdots + (0 * -4) \qquad > 1. \end{aligned}$$

Regardless of the input from '$U_{MDS}^{(n)}$', the net input to the node is greater than '1' since the de-activating signals are absent and there is at least one activating input signal. If an activating signal from '$U_{MDS}^{(n)}$' is present, the net input would only get larger. On the other hand, if the activating signal is absent, the input signal from '$d_X^{(n)}$' still exceeds '1'. Since there is an excitatory and no inhibitory effect on the node, '$v_m^{(n)}$', the node will be activated (set to '1').

Case 2 : (i) The central element was a non-edge element ($d_X^{(i)} = 0$) in previous iteration. But in the current iteration, the central element is considered as an edge element

($U_{GCS}^{(2)} = 0$). (ii) The most probable orientation of the edge contour in the pattern is same as the associated orientation for the node ($U_{MDS}^{(i)} = 1$).

For example, in the previous iteration the central element was a non-edge element, '$d_X^{(n)}, \ldots, d_X^{(se)} = 0$', but currently, it is considered as an edge element, '$U_{GCS}^{(2)} = 0$' and the most probable edge contour has a north orientation, '$U_{MDS}^{(n)} = 1$'. Therefore, the net input '$net_m^{(n)}$' to node '$v_m^{(n)}$' associated with the north orientation is given by :

$$\begin{aligned} net_m^{(n)} &= (d_X^{(n)} * w_1) + (U_{MDS}^{(n)} * w_2) + (U_{GCS}^{(2)} * w_3) + \sum_{j \in \{s,\ldots,se\}} d_X^{(j)} * w_4 \qquad (7.5) \\ &= (0 * 2) + (1 * 2) + (0 * -4) + \cdots + (0 * -4) \quad > 1. \end{aligned}$$

Since there is an excitatory and no inhibitory effect, the node '$v_m^{(n)}$' will be activated.

7.4.2 Cases for De-Activation

A node '$v_m^{(i)}$' is de-activated if it receives a de-activating input signal. The de-activation cases for node '$v_m^{(i)}$' are :

Case 1 : The orientations associated with the node and the central element are different ($d_X^{(j)} = 1, j \neq i$).

For example, the central element has an east orientation, '$d_X^{(c)} = 1$'. Therefore, the net input to node '$v_m^{(n)}$' associated with the north orientation is given by :

$$\begin{aligned} net_m^{(n)} &= (d_X^{(n)} * w_1) + (U_{MDS}^{(n)} * w_2) + (U_{GCS}^{(2)} * w_3) + \sum_{j \in \{s,\ldots,se\}} d_X^{(j)} * w_4 \qquad (7.6) \\ &= (0 * 2) + (U_{MDS}^{(n)} * 2) + (U_{GCS}^{(2)} * -4) + (0 * -4) + (1 * -4) + \\ &\quad (0 * -4) + \cdots + (0 * -4) \quad < 1. \end{aligned}$$

Regardless of the inputs from '$U_{MDS}^{(n)}$' and '$U_{GCS}^{(2)}$', the net input to the node is less than '1'. If '$U_{MDS}^{(n)} = 1$', the de-activating effect is still greater and hence will annul the activating effect from '$U_{MDS}^{(n)}$'. If '$U_{GCS}^{(2)} = 1$', the de-activating input signals would only get stronger. Since there is an inhibitory effect on the node, '$v_m^{(n)}$', the node will not be activated (set to '0').

Case 2 : The central element is determined by GCS as a non-edge element ($U_{GCS}^{(2)} = 1$).

The net input to the node '$v_m^{(n)}$' is given by :

$$\begin{aligned} net_m^{(n)} &= (d_X^{(n)} * w_1) + (U_{MDS}^{(n)} * w_2) + (U_{GCS}^{(2)} * w_3) + \sum_{j \in \{s,\ldots,se\}} d_X^{(j)} * w_4 \qquad (7.7) \\ &= (d_X^{(n)} * 2) + (U_{MDS}^{(n)} * 2) + (1 * -4) + \sum_{j \in \{s,\ldots,se\}} d_X^{(j)} * w_4 \quad < 1. \end{aligned}$$

Similar to the previous case analysis, the node '$v_m^{(n)}$' will not be activated since there is an inhibitory effect on the node, regardless to $d_X^{(n)} = 1$ and/or $U_{MDS}^{(n)} = 1$.

The outputs of the nodes in the Orientation Determination Subnet constitute the new set of direction values for the central element. An activated node indicates that the orientation associated with the node is the most appropriate orientation for the central element. On the other hand, a de-activated node indicates that the orientation associated with the node is inappropriate for the central element. If all the nodes in the Orientation Determination Subnet are not activated, the new direction values (associated with all the eight orientations) for the central element are '0's (i.e. the central element has no orientation) and the central element is determined to be a non-edge element.

Chapter 8

Adapting Weights Through Supervised Learning

8.1 Introduction

Chapters four to seven have described the architecture and mechanism of the hierarchical neural network system. In order for the neural network system to perform correctly and accurately, each of the neural subnets in the system must be given suitable weights. Pre-determined weights can be assigned to a subnet if either of the following two conditions are satisfied : firstly, the task encountered by the subnet is simple, and secondly, each of the nodes in the subnet has a local representation. However, assigning pre-determined weights becomes impossible as the complexity of the task increases. In a complex environment, there are numerous edge patterns which consist of edge contours of varying strength, noise, and false edges. Therefore, a learning algorithm is required to acquire this knowledge for the subnets. Furthermore, learning must be fast for any cases of practical significance.

It is highly desirable and important to have a mechanism which is able to adjust or modify the weights according to input patterns and generate output values as close as possible to the desired values. Adjusting the weights is often refered to as learning by the neural net. The adjusted weights should enable generalization for the neural net such that accurate

output values can be generated for input patterns which the net has not encountered during training. Because there are so many possible edge patterns that could be encountered by the system, it is not possible to train the neural network system with all the possible edge patterns in all probable situations. For a 5×5 window, even without considering the different edge strength, each edge element is allowed to take eight different orientations and one non-orientation. Hence, there are a total of 9^{25} possible patterns. It would take centuries to learn this enormous number of possible patterns (which amounts to more than *trillions* of patterns), assuming that it takes one second to process one million patterns.

There are three basic classes of learning procedures, namely, reinforcement learning, unsupervised learning and supervised learning. Reinforcement learning involves assigning credit to a local decision by measuring how it correlates with the global reinforcement signal. The network performs gradient ascent in the expected reinforcement by altering the probability distribution of the value of each weight in the direction that increases the expected reinforcement [Barto, Sutton and Brouwer 1981; Barto, Sutton and Anderson 1983; Hinton 1989]. One disadvantage with reinforcement learning is the inefficiency for large systems with many weights because many trials are required to assign credit correctly [Hinton 1989]. Unsupervised learning performs learning without receiving any additional information or training signal [Carpenter and Grossberg 1987; Fukushima 1988; Carpenter and Grossberg 1988; Carpenter, Grossberg and Mehanian 1989]. Unsupervised learning clusters the data into similarity groups under certain assumptions on the nature of the data [Gallant 1990]. However, an unsupervised learning algorithm cannot learn arbitrary functions. Another

drawback is the inability to determine if the output generated by the neural net is correct or useful [Giles and Maxwell 1987]. The third class of learning is the supervised learning [Hinton 1989]. When the system is in training or learning, an external teacher provides the desired responses for the corresponding training patterns.

In this thesis, supervised learning is selected as a means of training for the hierarchical neural network system. The advantages of using supervised learning are : firstly, the activation function can be learned by modeling the function with the input training patterns and corresponding desired responses [Gallant 1990]. Secondly, by providing the desired responses to the net, there is a direct and accurate means of control over the learning process. Currently, an error back-propagation method (the generalized delta rule) [Rumelhart, Hinton and Williams 1986] is one of the most widely-used supervised learning algorithms for adapting connection weights in multi-layered neural nets. However, there are some limitations, one of which is the extremely slow rate of convergence [Pomerleau 1987; Jacobs 1988]. Learning rules for adapting multi-layered networks require thousands of iterations to converge and sometimes do not converge at all, due to the local minimum problem [Giles and Maxwell 1987]. Another limitation is the difficulty in obtaining desired responses for the nodes in the hidden layers [Widrow and Winter 1988]. There is no simple way to provide the nodes in the hidden layers with a training signal.

In order to overcome the drawbacks of the generalized delta rule and benefit from the advantages of supervised learning, the *modified delta rule* is proposed as the learning procedure. The modified delta rule, a supervised learning algorithm, is used to derive a set of weights

for each of the functional-link nets in layer one (Edge Contour Detection Subnet). These sets of weights enable the functional-link nets to generate correct output values for different edge patterns. The major emphasis during the training of the functional-link nets is the synthesis of mappings between pairs of input edge pattern descriptions and corresponding output values generated by these nets. The modified delta rule, with the incorporation of a momentum term, is different from the delta rule [Stone 1986; Rumelhart, Hinton and Williams 1986]. The learning rule developed in this thesis is superior to other error back-propagation algorithms. Firstly, the high-order functional-link nets in the Edge Contour Detection Subnet can be adapted very quickly. With the absence of hidden layers, adaptation of each functional-link net is accomplished with simplicity. Secondly, the incorporation of a momentum term prevents wild oscillations during learning and enables faster convergence to the most suitable set of weights. Thirdly, together with the architecture of the functional-link nets, the learning rule enables the nets to have good generalization capabilities.

8.2 Modified Delta Rule Learning Algorithm

The modified delta rule like other various error back-propagation algorithms requires differentiability of the neural net's output signals [Widrow and Winter 1988]. The node in the functional-link net activated by a sigmoidal activation function satisfies this differentiability criterion. The modified delta rule, being a back-propagation procedure, does not mimic a biological model. As a biological model, back-propagation is implausible. There is no evidence that synapses can be used in the reverse direction, or neurons can propagate error

derivatives backwards [Hinton 1989; Gallant 1990]. Hence, the primary objective in this chapter is functionality rather than biologically accurate modeling.

In the learning process, a training edge pattern and a corresponding desired target output value form a training data pair. Training data are presented to each of the eight functional-link nets in the Edge Contour Detection Subnet. The input pattern, the functional processes and the sigmoidal activation function of the node in the functional-link net produce an output value. If the computed value is equal to the desired output value, then no weight change takes place. Otherwise, the weights are modified to reduce the difference between the target output and the computed values. The rule for determining the weight adjustment is given by :

$$\Delta_k W_i^{(j)} = \eta * \delta_k^{(j)} * f_{ki}^{(j)} \qquad (8.1)$$

where '$\Delta_k W_i^{(j)}$' is the change in the weight of the connection from the 'i^{th}' functional process in the 'j^{th}' functional-link net ($j \in \{n, \ldots, se\}$) following presentation of the 'k^{th}' pattern; 'η' is a learning rate constant which controls the speed of learning; '$\delta_k^{(j)}$' is the difference between the actual output produced by the net 'j' and the desired output level; '$f_{ki}^{(j)}$' is the value of the output of the functional process.

Therefore, '$\delta_k^{(j)}$' is the amount of error at the node of net 'j' and is given by :

$$\delta_k^{(j)} = t_k^{(j)} - o_k^{(j)} \qquad (8.2)$$

where '$t_k^{(j)}$' is the target output; '$o_k^{(j)}$' is the actual output of the node.

'$f_{ki}^{(j)}$', the output of the 'i^{th}' functional process is given by :

$$f_{ki}^{(j)} = d_{m_1} * g_{m_1} * \cdots * d_{m_N} * g_{m_N} \qquad (8.3)$$

where 'd' is the direction value and 'g' is the gradient magnitude of an edge element; the subscripts of 'd' and 'g' indexes the edge elements in the local edge pattern; 'N' denotes the order of the functional process.

In the update rule (expression 8.1), the amount of adjustment to the weight is proportional to two factors, namely, the amount of error at the node and the output of the functional process. By incorporating the output of the functional process into the learning rule, the weight associated with the functional process is modified only if the process is activated. Such incorporation has two advantages : firstly, it is very difficult for the net to be simultaneously trained for all the training patterns. By including the output from the functional process, the functional-link net responds only to the newest training pattern, and incurs minimal disturbance to the responses for some of the previous training patterns. Secondly, because the output of the functional process affects the amount of adjustment, faster convergence can be achieved as the process output contributes to the error.

The learning process involves two phases. The first phase involves the presentation of the training pattern to the input layer. Specific functional processes which characterize the edge contour in the training pattern are activated and the output of an activated functional process is propagated forward to the node. The activation function in the node then generates an activation level for the node. In the second phase, the error determined at the node is propagated backward and the amount of adjustment to each weight in the net for each

pattern is computed using the following procedure.

For each training pattern 'k';

begin

compute the error $\delta_k^{(j)} = t_k^{(j)} - o_k^{(j)}$;

For each functional process 'i' in the functional-link net 'j';

begin

determine the output from the functional process, $f_{ki}^{(j)}$;

compute the amount of adjustment '$\Delta_k W_i^{(j)}$' to the weight associated with the functional process for pattern 'k';

end

end

For each pattern, the error function is given by :

$$E_k = \frac{1}{2} \sum_{j \in \{n,\ldots,sc\}} (t_k^{(j)} - o_k^{(j)})^2 \qquad (8.4)$$

where 'E_k' is the measure of the error for the 'k^{th}' pattern; '$t_k^{(j)}$' is the target output value; '$o_k^{(j)}$' is the actual output value; 'j' indexes the orientations associated with the functional-link nets. Therefore, for each pattern, the error is measured by the sum of the squares of the errors for all the functional-link nets.

The learning rule performs a gradient descent in error space to minimize the sum of the pattern errors over the training set. In order to perform a true gradient search in total error

space, the following average total system error function is introduced :

$$E_T = \frac{1}{P}\sum_{k=1}^{P} E_k \qquad (8.5)$$

where 'E_T' is the average total system error; 'k' indexes the patterns in the set of training patterns; 'P' denotes the total number of training patterns.

If the weights are modified for each pattern at a time, this is known as "on-line" learning. Such modifications can increase the errors for patterns in the training set, which is clearly undesirable. To ensure that modifications to the weights are performed only after all training patterns have been presented (analogous to the Least Mean Square algorithm [Widrow and Stearns 1985] or the Widrow-Hoff Delta Rule [Rumelhart, Hinton and Williams 1986]), the learning process changes the weights to minimize the average total system error. Therefore, the gradient search is performed on the direction of gradient descent in total error space.

A large learning rate 'η' corresponds to large changes to the weights when they are adjusted. However, since large learning rate might lead to wild weight oscillations, a small learning rate is preferable to ensure small adjustments to the weights during learning. In order to maintain a relatively high speed of finding the solution weight set while avoiding weight oscillations, a momentum term is incorporated into the learning rule. The new learning rule is given by :

$$\Delta_k W_i^{(j)}(l+1) = (\eta * \delta_k^{(j)} * f_{ki}^{(j)}) + (\alpha * \Delta_k W_i^{(j)}(l) * d_x * d_y) \qquad (8.6)$$

where 'k', '$\Delta_k W_i^{(j)}$', 'η', '$\delta_k^{(j)}$', and '$f_{ki}^{(j)}$' are same as in (8.1). In the second term (momentum term), 'α' is the constant which determines the effect of past weight changes on the current

change to be made; 'd' denotes the direction value of an edge element; 'x' and 'y' indexes the two edge elements in the local edge pattern. To compute the weight adjustment at 'time' $t+1$, the weight adjustment computed at 'time' t (in the previous iteration) is used for determining the value of the momentum term (8.6). Before the learning process begins ($t = 0$), the amount of weight adjustment is initialized to zero, i.e., $\Delta_k W_i^{(j)}(0) = 0.0$. For a first order functional process (only one edge element is involved), expression (8.6) will be modified to

$$\Delta_k W_i^{(j)}(t+1) \;=\; (\eta * \delta_k^{(j)} * f_{ki}^{(j)}) + (\alpha * \Delta_k W_i^{(j)}(t) * d_x). \tag{8.7}$$

The learning rule for a third order functional process is :

$$\Delta_k W_i^{(j)}(t+1) \;=\; (\eta * \delta_k^{(j)} * f_{ki}^{(j)}) + (\alpha * \Delta_k W_i^{(j)}(t) * d_x * d_y * d_z) \tag{8.8}$$

where 'x', 'y', 'z' indexes the three edge elements in the local edge pattern.

The direction value in the momentum term ensures that the momentum term will affect the computation of the weight adjustment for the functional process only if the process is activated. This prevents any adjustment to the weight when the functional process does not respond to the training pattern. The momentum term in the learning rule is used to avoid wild oscillations and to find the solution much more quickly.

The modified weight '$W_i^{(j)'}$' for the 'i^{th}' functional process in net 'j' is given by :

$$W_i^{(j)'} \;=\; W_i^{(j)} + \sum_k \Delta_k W_i^{(j)} \tag{8.9}$$

where '$W_i^{(j)}$' is the value of the weight before adjustment.

The learning process is described in the following pseudo-code :

```
while (Average Total System Error > Threshold Value) do
    begin
        do (for each training pattern)
          begin
              do (for each functional-link net)
                begin
                    COMPUTE NET OUTPUT
                    COMPUTE ERROR AT THE NET
                    do (for each weight in the net)
                       COMPUTE WEIGHT ADJUSTMENT
                end
              COMPUTE ERROR FOR THE TRAINING PATTERN
          end
        COMPUTE AVERAGE TOTAL SYSTEM ERROR
        UPDATE WEIGHTS
    end
```

8.3 Effectiveness of the Learning Process

There are three ways to improve the effectiveness of a learning process in neural networks.

1. Only necessary information is provided to the neural network [Pao 1989$_b$].

2. No hidden (internal) layers are allowed, hence no training signals for these layers are required [Giles and Maxwell 1987].

3. The amounts of the adjustments to the weights in each iteration are controlled to prevent wild oscillations and hence enable convergence [Rumelhart, Hinton and Williams 1986].

In the following sections, it is shown that the modified delta rule together with the functional-link nets satisfy all three above conditions to enable fast and accurate learning.

8.3.1 Improving Rate of Learning

It is a waste of time for a network to learn information already known in advance. By providing the network with the known information instead of generating these information again, the training is performed only to learn aspects of the task which the network does not already know [Giles and Maxwell 1987; Hinton 1989]. In the Edge Contour Detection Subnet, the known information is embodied into the architecture of the functional-link nets. For each functional-link net, a functional process represents the structural information concerning a segment of an edge contour. When a functional process is activated, the net is provided with the information that a segment (characterized by the process) of an edge contour is present in the local edge pattern. By making this information available to the net, there is no need for the net to learn the structure of the segment of the edge contour. Hence, a large portion of the learning process is avoided. Furthermore, there are no hidden or internal layers in the functional-link nets. Hence, the difficult task of providing the training signal to the 'hidden'

nodes is avoided. Finally, the incorporation of a momentum term in the learning rule enables fast convergence. The momentum term in the modified delta rule is used to specify that the current adjustment to the weight is affected by the previous weight adjustment. In this way, some inertia is built in and momentum will prevent any large changes to the weights at any one time and hence avoids wild oscillations.

The performance of the learning process for the functional-link nets is very impressive. The adaptation of the Edge Contour Detection Subnet is fast and accurate. The performance of the learning process is shown in Fig. 8.1. It shows the rate of decrease of the system error with the number of presentations of the set of training patterns. The output generated by each functional-link net is normalized to a range (0,1). As the activation function for the node of each functional-link net cannot have output values of '1' or '0' without infinitely large positive or negative input respectively, a strong desired response is taken to be 0.9, a medium desired response is 0.5 and a weak desired response is 0.1. After training, the Edge Contour Detection Subnet generates correct output values for edge patterns which were not encountered during training.

In the experimental tests, in order to achieve high learning speed and also to avoid wild oscillations, the following values for the learning rate 'η' and the proportion 'α' of contribution of weight change computed in the previous iteration to the momentum term were used: (i) $\eta = 0.9$ and (ii) $\alpha = 0.7$. Other values for η and α were tested but those values required more iterations for convergence. The average total system error (E_T) for convergence was set at 0.0000045 (threshold value). This small system error for convergence ensures that

all the 'learned' weights are suitable for the functional-link nets to perform accurately. The learning process required only 275 iterations to achieve convergence. The values selected for η, α and E_T enable fast learning of accurate weights for the functional-link nets to perform correctly.

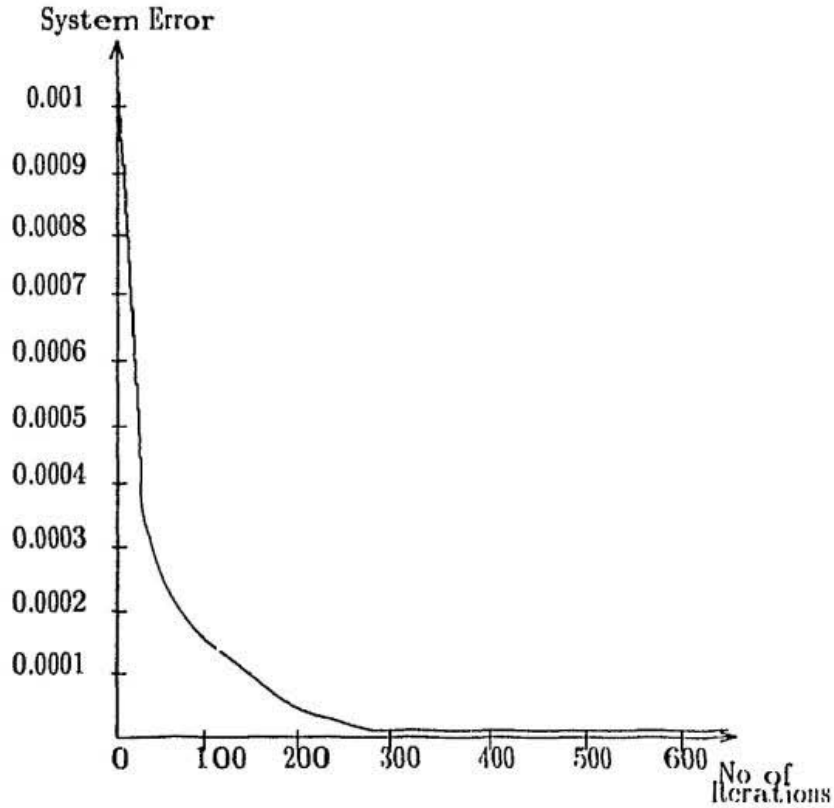


Figure 8.1 Rate of learning for the Edge Contour Detection Subnet

8.3.2 Enhancing Generalization Capability

In addition to good learning capability, a learning rule should also provide good generalization capability. A major goal of connectionist learning is to produce a network that

generalizes correctly to new situations after training on a sufficient number of typical cases. Generalization enables the network to be applied to typical real world tasks [Hinton 1989]. Merely learning the training patterns can be accomplished by storing these patterns and their associated desired responses in a look-up table [Widrow and Winter 1988]. This approach is definitely not feasible for the hierarchical neural network system as there are too many patterns and their corresponding desired responses to be stored. Good generalization on complex tasks can be obtained by designing a network architecture that contains a certain amount of a priori knowledge about the task [LeCun 1989].

This is precisely what the architectures of the selected functional-link nets have achieved. The functional processes in the functional-link nets characterize various edge contour patterns. Therefore these priori knowledge are provided to the selective functional-link nets by encoding them (the knowledge of edge patterns) into the architectures of the nets. Since each pattern is assumed to be within a 5×5 window, all edge patterns (i.e. rectilinear, curvilinear, etc.) are constructed by the same number of edge elements. The central element which lies on the edge contour is the edge element of concern. Hence, in each edge pattern, there are four other edge elements to construct the edge contour. Different edge patterns would involve edge elements from different pixel locations in the window.

For each type of edge pattern (i.e. rectilinear, non-symmetrical, curvilinear, etc.), regardless of the orientation associated with the pattern, the technique for characterizing the edge contours is the same. For example, a rectilinear edge contour with a north orientation involves the same number of edge elements as a rectilinear edge contour with a north-east

orientation. Similarly for a non-symmetrical edge contour, a curvilinear edge contour, etc. For each particular type of edge pattern of any orientation, the relative positions of the edge elements involved for each segment with respect to the entire edge contour are the same. For example, in Fig. 8.2a, the pairs of edge elements involved for characterizing the segments

a) North orientation b) North-east orientation

Figure 8.2 Edge patterns

of a rectilinear edge contour with a north orientation are: (e_3, e_{18}), (e_3, e_{23}), (e_8, e_{18}), (e_8, e_{23}) where 'e' denotes an edge element and the subscripts denote the relative positions in the window. For a rectilinear edge contour with a north-east orientation (Fig. 8.2b), the pairs of edge elements involved are: (e_5, e_{17}), (e_5, e_{21}), (e_9, e_{17}), (e_9, e_{21}). Therefore the relative positions of the edge elements (e_3, e_{18}) and (e_5, e_{17}) in the window are the same with respect to the positions of the other edge elements involved for the two edge contours. Similarly for (e_3, e_{23}) and (e_5, e_{21}), (e_8, e_{18}) and (e_9, e_{17}), (e_8, e_{23}) and (e_9, e_{21}). Therefore training could involve only edge patterns for one orientation, for example, north orientation, and the training could easily be generalized to the edge patterns for the other seven orientations, namely, south, east, ..., south-east. This capability has enabled the Edge Contour Detection Subnet (functional-link nets) to have good generalization capability.

8.4 Performing Necessary Training to the Subnets

The advantage of the hierarchical neural network system lies in the modularity of the system architecture. One way of avoiding a complex system architecture is to introduce a modular hierarchical structure in which different modules are only loosely coupled [Simon 1969]. Modularity is important for fast learning and good generalization [Hinton 1989]. If a complex task can be decomposed into a set of easier sub-tasks, the sub-tasks can be learned independently as a result of loosely coupled modularity. Hence, fast and correct learning can be achieved.

The hierarchical neural network system can be broken down to four modules or subnets, each achieving a particular sub-task. The four sub-tasks are the constituents of the complex task to be achieved by the hierarchical neural network system. Therefore, each subnet can be trained independently.

In any neural net, a concept is represented by the node(s) in the net. There are basically two types of representation, namely, local representation and distributed representation [Hinton 1989]. In a local representation, each concept is represented by a single node [Feldman 1986], whereas in a distributed representation, a concept is distributed over several nodes. Each node represents a constituent of a concept and is also involved in representing the constituents of several other different concepts [Hinton, McClelland and Rumelhart 1986]. In a network that uses local representation, no training is required. It is feasible to set all the weights by hand because each weight corresponds to a specific relationship between two nodes. That is, there is a unique relationship between the concepts represented by the two

nodes. However, if a network uses distributed representation, it may be very difficult to set suitable weights by hand and so a learning procedure is required [Hinton 1989].

In the hierarchical neural network system, the Edge Contour Detection Subnet requires training because the functional-link nets in the Subnet adopt the distributed representation concept. The adopted concept for detecting an edge contour is based on detecting various segments of the edge contour. Each functional process characterizes a segment of an edge contour. Hence, several functional processes are required to characterize that edge contour. Therefore, the representation of the concept for characterizing an edge contour is distributed over several functional processes.

Edge contours can have various strengths. This means that there are also many different concepts for characterizing edge contours with the same pattern and same orientation but with different strengths. Hence, each functional process is associated with many different concepts for characterizing an edge segment. It is not possible to pre-determine suitable weights for each functional process because there are too many concepts to consider. Therefore, training the functional-link nets is necessary in order to derive sets of suitable weights for the functional processes in the nets.

The nets in levels two to four of the hierarchical neural network system, namely, the Maximum Detection Subnet, the Gradient Adjustment Subnet and the Orientation Determination Subnet, do not require training. These subnets are based on the local representation concept.

In the Maximum Detection Subnet, the inhibitory weights are set to '$-\varepsilon$', where $-\varepsilon < \frac{1}{N}$

and 'N' is the number of nodes in the Maximum Detection Subnet and the excitatory weights are set to '1' (see section 5.3). Each node represents a concept : a probable orientation for the edge pattern. The relationship between a node and each of its neighbours is inhibitory, while a node has an excitatory relationship with itself. Since the specific relationships exist amongst the nodes representing the concepts, the weights in the Maximum Detection Subnet need not be learned as each weight corresponds to a specific relationship between two nodes.

In the Gradient Adjustment Subnet, there are two subnets. The Condition Ascertainment Subnet consists of a functional-link net of the Selective Tensor Model. Each of the functional processes in the net represents a specific concept for determining the occurrence of the appropriate conditions (cases) for the adjustment of the gradient magnitude. The cases for reinforcing the gradient magnitude of the central element are :

1. If both the central edge element and the strongest edge contour in the local edge pattern have the same orientation.

2. If the central element is considered as a non-edge element but the strongest edge contour in the local edge pattern passes through it.

The concept for representing these two cases (for each particular orientation) in the functional-link net is achieved through the 'a' type functional process. Each of the '$a^{(i)}$' functional processes, where $i \in \{n, \ldots, se\}$, is concerned with representing the concept for determining the occurrence of the conditions for reinforcing the gradient magnitude of the central element for a particular orientation (i.e. north, south, ..., south-east). If the conditions for reinforcement are absent, suppression to the gradient magnitude occurs. Hence, each

of the functional processes represents a specific concept for reinforcement relative to a particular orientation. These concepts also have specific relationships with each other in the sense that each functional process can be activated only if the other functional processes are de-activated. Therefore, since each functional process represents a specific concept and the relationships amongst the functional processes are specific, the weights associated with the functional processes need not be learned. These weights can be pre-determined and assigned by hand.

Layer two (Gradient Computation Subnet) of the Gradient Adjustment Subnet is a semilinear feedforward subnet. This subnet has no internal layers and there are two nodes in the output layer. One node represents the concept for adjusting the gradient magnitude, the other represents the concept for ascertaining a non-edge element. Each output node receives its input signals from : (1) the Condition Ascertainment Subnet and (2) the current gradient magnitude of the central element under consideration. Therefore, since each node represents a specific concept and the relationship between the node and each of its inputs is specific, the Gradient Computation Subnet need not be trained. Hence, the weights for the subnet can be preset.

The Orientation Determination Subnet in level four of the hierarchical neural network system is a semilinear feedforward subnet with the absence of internal layers. Each node in the output layer represents the concept for determining the appropriate orientation for the central element under consideration. There are eight output nodes and each node is associated with a particular orientation. The activation of a node indicates that the orien-

tation associated with the node is the orientation for the edge element. Each node receives its input signals from three sources : (1) the orientation of the edge element; (2) the signal from the Maximum Detection Subnet; (3) the signal from the Gradient Adjustment Subnet. The concept represented by each node can be described as :

1. The node is activated if :

   (a) The orientation of the edge element is the same as the orientation associated with the node; and/or

   (b) The signal value from the Maximum Detection Subnet is '1'.

2. The node is de-activated if :

   (a) The orientation of the edge element is different from the orientation associated with the node; and/or

   (b) The signal value from the Gradient Adjustment Subnet is '1'.

Therefore, the activation and de-activation of a node is dependent on the particular inputs the node receives. Hence, there are specific relationships between the node and each of its inputs. Therefore, the Orientation Determination Subnet need not be trained. Suitable weights can be pre-determined and assigned to the Subnet.

The modularity in the structure of the hierarchical neural network system has enabled fast derivation of suitable weights for the network system to perform correctly. Training is provided only to those subnets requiring training. Hence, training the entire system is avoided. Therefore, the amount of information which the nets need to learn is drastically

reduced. The architecture of the nets has also enabled these nets to have good generalization capability. Therefore, only a small number of training patterns is required to train the nets and still derive suitable weights for the nets to perform accurately and efficiently.

Chapter 9

Conclusions and Future Research

9.1 Summary of Contributions

In this thesis, a hierarchical neural network system has been developed for improving edge measurements in an edge image. The system consists of four levels of neural nets. The first level is the Edge Contour Detection Subnet which consists of eight functional-link nets working in parallel. The functional-link nets are high-order nets of the Selective Functional Expansion Model. Each functional-link net is associated with a particular compass orientation and contains selected functional processes which characterize the structures of different edge contours. Each functional-link net receives its input data from the edge measurements of the edge elements. Through the high-order terms in the functional processes, each functional-link net is provided with our *a priori* knowledge of the structures of different edge contours. By utilizing the gradient magnitude and the orientation of the appropriate edge elements simultaneously, the functional-link nets are able to accurately detect the edge contours in the local edge patterns. The modularity in the architectural design enables the functional-link net to be easily modified to handle new and more complex edge patterns.

In the second level of the hierarchical neural network system is a cooperative-competitive

neural net model, the Maximum Detection Subnet. Its function is to determine the orientation of the strongest edge contour in the local edge pattern. The Maximum Detection Subnet receives its input from the Edge Contour Detection Subnet. There are eight nodes in the Maximum Detection Subnet and each node is associated with a particular orientation. Each node uses lateral inhibition to inhibit the activities of its neighbouring nodes while providing excitation to its own activity. After converging, only one node is activated while the other nodes are de-activated. The activated node indicates the orientation of the strongest edge contour in the local edge pattern.

The Gradient Adjustment Subnet in the third level consists of two layers. The first layer is the Condition Ascertainment Subnet, which is a functional-link net of the Selective Tensor Model. The functions of the Condition Ascertainment Subnet are to ascertain the occurrence of appropriate conditions for adjusting the gradient magnitude and also determine the appropriate amount of adjustment to the gradient magnitude. By using high-order processes, all the input information can be utilized simultaneously and made explicitly available to the Condition Ascertainment Subnet for the latter to correctly determine the appropriate conditions for adjustment. The second layer is the Gradient Computation Subnet, a semilinear feedforward net with no hidden layers. The Gradient Computation Subnet computes the new gradient magnitude of the element of concern and also determines if the element is an edge element or a non-edge element. The Gradient Computation Subnet is able to concurrently compute the new gradient magnitude and also send a signal indicating the status of the element (edge or non-edge).

The Orientation Determination Subnet in the fourth level, determines the new orientation for the element of concern. The Subnet generates the new set of direction values for the element of concern. The Orientation Determination Subnet is a semilinear feedforward net with no hidden layers that has eight nodes in the output layer. The Orientation Determination Subnet is able to ascertain the occurrence of different conditions to determine the most appropriate orientation for the element of concern. This enables the element of concern to have an orientation which is in agreement with the orientations of surrounding edge elements in the structure of an edge contour.

In this thesis, an improved learning algorithm based on the delta rule has also been developed. This algorithm enables fast convergence to suitable sets of weights for the functional-link nets in the Edge Contour Detection Subnet to perform correctly. The nets learn fast and have good generalization capabilities. Learning by the functional-link nets requires only 275 iterations to attain an average total system error of 0.000045. Only twenty seven typical training edge patterns for each orientation are used for training as compared to the more than trillions of possible training patterns (9^{25} possible patterns). Correct results are then obtained for edge patterns of any orientation that were not encountered during training.

In the experimental tests, each original gray-level test image is degraded and corrupted by additive random noise and non-uniform illumination (Fig. 9.1). The test images are corrupted by 15% - 35% additive random noise. The edge image obtained is generally very poor with missing valid edge elements; presence of spurious, false edge elements; very weak edge elements; and presence of noise (Fig. 9.2). After 15 iterations of processing by the

hierarchical neural network system, an improved edge image is obtained (Fig. 9.3):

1. True edge elements in different types of edge contours, namely, rectilinear edge contours, non-symmetrical linear edge contours, curvilinear edge contours and edge contours at a corner are reinforced or enhanced.

2. Missing edge elements are interpolated and recovered.

3. Spurious and false edge elements are effectively suppressed.

4. Noise is effectively eliminated.

The benefits of highly parallel processing and fast computing time can be realized in a hardware implementation.

Test results are shown in Appendix A (Figs. A1, A2, A3). Degraded and noise corrupted (by non-uniform illumination and 15% - 35% additive random noise) gray-level images (Figs. A1) were converted to edge images (Figs. A2) and then improved edge images (Figs. A3) were obtained after 15 iterations of processing by the hierarchical neural network system. The simulation package is written in C, running on a MIPS-M120S under the UNIX operating system.

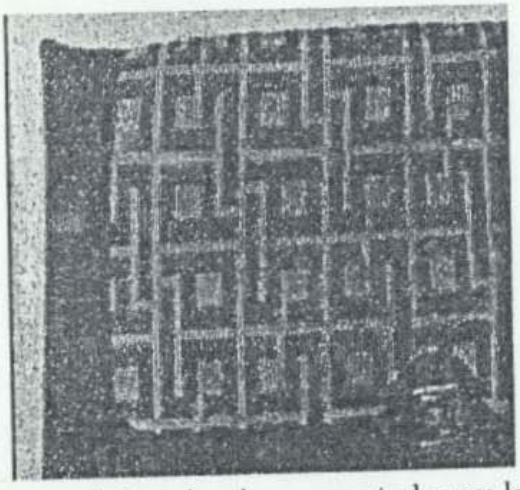


Figure 9.1 Degraded and noise corrupted gray-level image

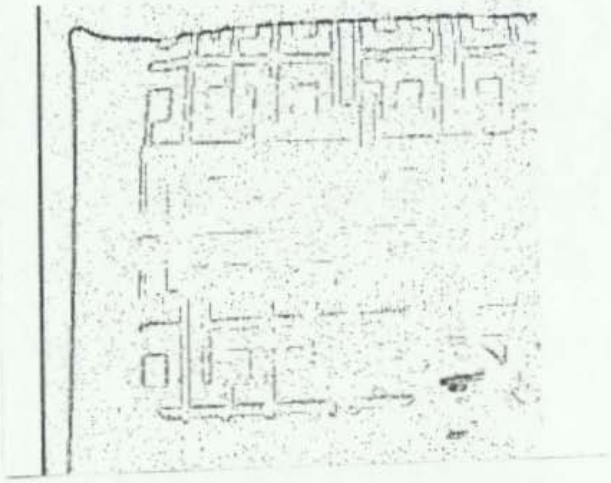


Figure 9.2 Edge image - before processing by neural network system

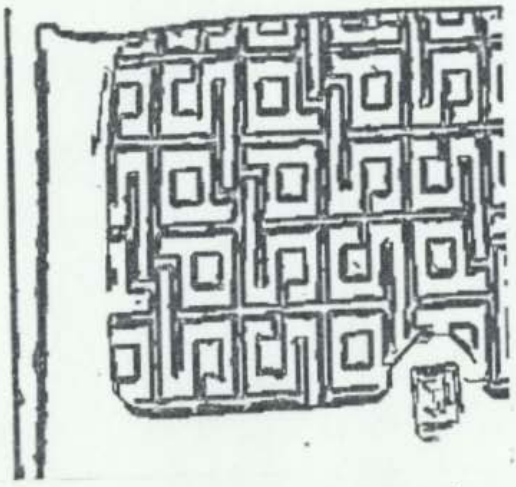


Figure 9.3 Improved edge image - after processing by neural network system

9.2 Directions for Further Research

Although the hierarchical neural network system performs very well, more research is needed to further improve the performance of the system.

9.2.1 Thinning of Edges

A built-in thinning operation could be incorporated into the neural network system. One possible method of thinning can be achieved by adding functional processes for the functional-link in the Edge Contour Detection Subnet. These functional processes would be associated with neighbouring edge elements located at positions adjacent to the central element of concern along the width of the edge contour. These processes will provide a suppressing effect on the central element of concern, thereby performing a thinning operation by suppressing weaker edge elements along the outer borders of the edge contour.

9.2.2 Recovering Consecutive Missing Edge Elements

Some missing edge elements are not interpolated well, especially when they occur in four or more consecutive spatial positions. A possible technique to overcome this is to introduce multiple over-lapping windows. This would require more functional processes for characterizing the edge contours along more pixel locations for the edge contours.

9.2.3 Detecting More Complex Edge Patterns

More complex edge patterns could be considered and more functional processes for characterizing these edge contours could be generated to enhance the capability of the hierarchical neural network system.

9.2.4 Improving on the Speed of Learning

Even though the nets can learn very fast with the improved learning algorithm developed in this thesis, the speed of learning can still be improved upon. One possible means of increasing the learning speed is to modify the momentum term to be adaptive during the learning process. Another possible way of increasing the learning speed is to modify the learning rate to be adaptive during the learning process. An adapted momentum term and an adapted learning rate could bring faster convergence.

9.2.5 Improving on the Ability to Further Eliminate Spurious and Noisy Edges

Some spurious and noisy edges are still present after processing by the neural net, especially when neighbouring false edge elements have the same orientation. A possible technique to remove these spurious and noisy edges is to incorporate local information from neighbouring adjacent windows, by utilizing more local information from surrounding windows and simultaneously with global information, spurious and noisy edges can be eliminated.

REFERENCES

Abutaleb A.S. (1989). "Automatic Thresholding of Gray-Level Pictures Using 2-D Entropy", *Computer Vision, Graphics and Image Processing*, Vol. 47, Number 1, pp. 22 - 32.

Ballard D.H. and Brown C.M. (1982)$_a$. *Computer Vision*, Prentice-Hall Inc, New Jersey, pp. 76 - 85.

Ballard D.H. and Brown C.M. (1982)$_b$. *Computer Vision*, Prentice-Hall Inc, New Jersey, pp. 152 - 153.

Barto A.G., Sutton R.S. and Anderson C.W. (1983). "Neuronlike Elements that can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 13, Number 5, pp. 834 - 846.

Barto A.G., Sutton R.S. and Brouwer P.S. (1981). "Associative Search Network : A Reinforcement Learning Associative Memory", *Biological Cybernetics*, Vol. 40, pp. 201 - 211.

Bell B. and Pau L.F. (1990). "Contour Tracking and Corner Detection in a Logic Programming Environment", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, Number 9, pp. 913 - 917.

Berzins V. (1984). "Accuracy of Laplacian Edge Detectors", *Computer Vision, Graphics and Image Processing*, Vol. 27, pp. 195 - 210.

Boukharouba S., Rebordao J.M. and Wendel P.L. (1985). "An Amplitude Segmentation Method Based on the Distribution Function of an Image", *Computer Vision, Graphics and Image Processing*, Vol. 29, Number 1, pp. 47 - 59.

Canny J. (1986). "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, Number 6, pp. 679 - 697.

Carpenter G.A. and Grossberg S. (1987). "ART2 : Self-Organization of Stable Category Recognition Codes for Analog Input Patterns", *Applied Optics*, Vol. 26, pp. 4919 - 4930.

Carpenter G.A. and Grossberg S. (1988). "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *Computer*, Vol. 21, Number 3, pp. 77 - 88.

Carpenter G.A. and Grossberg S. (1990). "ART3 : Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures", *Neural Networks*, Vol. 3, Number 2, pp. 129 - 152.

Carpenter G.A., Grossberg S. and Mehanian C. (1989). "Invariant Recognition of Cluttered Scenes by a Self-Organizing ART Architecture : CORT-X Boundary Segmentation", *Neural Networks*, Vol. 2, pp. 169 - 181.

Cohen M.A. and Grossberg S. (1987). "Masking Fields : A Massively Parallel Neural Architecture for Learning, Recognizing and Predicting Multiple Groupings of Patterned Data", *Applied Optics*, Vol. 26, Number 10, pp. 1866 - 1891.

Duda R. and Hart P. (1973). *Pattern Classification and Scene Analysis*, Wiley, New York, pp. 271 - 272.

Dupaguntla N.R. and Vemuri V. (1989). "A Neural Network Architecture for Texture Segmentation and Labelling", *International Joint Conference on Neural Networks*, Vol. 1, June 18-22, pp. I-127 - I-133.

Fanelli R., Raphan T. and Schnabolk C. (1990). "Neural Networks Modelling of Eye Compensation During Off-Vertical-Axis Rotation", *Neural Networks*, Vol. 3, Number 3, pp. 265 - 276.

Feldman J.A. (1986). "Neural Representation of Conceptual Knowledge", *Technical Report TR189*, Dept. of Computer Science, University of Rochester, Rochester, NY.

Feldman J.A., Fanty M.A. and Goddard N.H. (1988). "Computing with Structured Neural Networks", *Computer*, Vol. 21, Number 3, pp. 91 - 103.

Freeman H. (1974). "Computer Processing of Line-Drawing Images", *Computer Surveys*, Vol. 6, pp. 57 - 97.

Fukushima K. (1988). "Neocognitron : A Hierarchical Neural Network Capable of Visual Pattern Recognition", *Neural Networks*, Vol. 1, pp. 119 - 130.

Gallant S.I. (1990). "Perceptron-Based Learning Algorithms", *IEEE Transactions on Neural Networks*, Vol. 1, Number 2, pp. 179 - 191.

Geman S. and Geman D. (1984). "Stochastic Relaxation, Gibbs Distribution and the Bayesian Restoration of Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, Number 6, pp. 721 - 741.

Giles C. L. and Maxwell T. (1987). "Learning, Invariance, and Generalization in High-Order Neural Networks", *Applied Optics*, Vol. 26, Number 23, pp.4972 - 4978.

Gupta L., Sayeh M.R. and Tammana R. (1990). "A Neural Network Approach to Robust Shape Classification", *Pattern Recognition*, Vol. 23, Number 6, pp. 563 - 568.

Hancock E.R. and Kittler J. (1990). "Edge-Labelling Using Dictionary-Based Relaxation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, Number 2, pp. 165 - 181.

Haralick R.M. and Lee J.S.J. (1990). "Context Dependent Edge Detection and Evaluation", *Pattern Recognition*, Vol. 23, Number 1-2, pp. 1 - 19.

Hertz L. and Schafer R.W. (1988). "Multilevel Thresholding Using Edge Matching", *Computer Vision, Graphics and Image Processing*, Vol. 44, Number 3, pp. 279 - 295.

Hinton G.E. (1989). "Connectionist Learning Procedures", *Artificial Intelligence*, Vol. 40, Numbers 1-3, pp. 185 - 234.

Hinton G.E., McClelland J.L. and Rumelhart D.E. (1986). "Distributed Representations", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Vol. 1 : Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA. MIT Press, pp. 77 - 109.

Hopfield J.J. (1984). "Neurons with Graded Response have Collective Computational Properties like those of Two-State Neurons", *Proceedings of National Academy of Science USA*, Vol. 81, pp. 3088 - 3092.

Hopfield J.J. and Tank D.W. (1986). "Computing with Neural Circuits : A Model", *Science*, Vol. 233, pp. 625 - 633.

Huang W.M. and Lippmann R.P. (1988). "Neural Net and Traditional Classifiers", *Neural Information Processing Systems*, D. Anderson, Eds. New York : American Institute of Physics, pp. 387 - 396.

Huang J.S. and Tseng D.H. (1988). "Statistical Theory of Edge Detection", *Computer Vision, Graphics and Image Processing*, Vol. 43, Number 3, pp. 337 - 346.

Hummel R.A. and Zucker S.W. (1983). "On the Foundations of Relaxation Labelling Processes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, pp. 267 - 287.

Jacobs R.A. (1988). "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, Vol. 1, Number 4, pp. 295 - 307.

Jordan M.I. (1986). "An Introduction to Linear Algebra in Parallel Distributed Processing", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Vol. 1 : Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA. MIT Press, pp. 418 - 421.

Kammerer B. and Kupper W. (1988). "Experiments for Isolated-Word Recognition with Single- and Multi-Layer Perceptrons", *Neural Networks*, Vol. 1, Supplement 1, Abstracts of the First Annual INNS Meeting, Boston, pp.302.

Kandel E.R. and Schwartz J.H. (1985). *Principles of Neural Science*, Elsevier, New York, pp. 320 - 330.

Khotanzad A. and Lu J.H. (1990). "Classification of Invariant Image Representations Using a Neural Network", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 38, Number 6, pp. 1028 - 1038.

Kittler J. and Illingworth J. (1986). "Minimum Error Thresholding", *Pattern Recognition*, Vol. 19, pp. 41 - 47.

Klassen M., Pao Y.H. and Chen V. (1988). "Characteristics of the Functional Link Net : A Higher Order Delta Rule Net", *IEEE International Conference on Neural Networks*, San Diego, California, July 24-27, pp. I-507 - I-513.

Kohonen T. (1984). *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, pp. 128 - 133.

Kundu A. (1990). "Robust Edge Detection", *Pattern Recognition*, Vol. 23, Number 5, pp. 423 - 439.

Lapedes A. and Farber R. (1986). "A Self-Optimizing, Nonsymmetrical Neural Net for Content Addressable Memory and Pattern Recognition", *Physica*, Vol. 22D, pp. 247 - 259.

LeCun Y. (1989). "Generalization and Network Design Strategies", *Technical Report CRG-TR-89-4*, University of Toronto.

LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W. and Jackel L.D. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, Vol. 1, pp. 541 - 551.

Lee Y.C., Doolen G., Chen H.H., Sun G.Z., Maxwell T. and Lee H. (1986). "Machine Learning Using a Higher Order Correlation Network", *Physica*, Vol. 22D, Number 1-3, pp. 276 - 306.

Linsker R. (1988). "Self-Organization in a Perceptual Network", *Computer*, Vol. 21, Number 3, pp. 105 - 117.

Lippmann R.P. (1987). "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, Vol. 4, Number 2, pp. 4 - 22.

Mandayam A., Thathachar L. and Sastry P.S. (1986). "Relaxation Labelling with Learning Automata", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, Number 2, pp. 256 - 268.

Marr D. and Hildreth E. (1980). "Theory of Edge Detection", *Proceedings of Royal Society of London*, Series B, Vol. 207-208, pp. 187 - 212.

Meer P., Sher C.A. and Rosenfeld A. (1990). "The Chain Pyramid : Hierarchical Contour Processing", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, Number 4, pp. 363 - 376.

Minsky M.L. and Papert S. (1988). *Perceptrons*, MIT Press, Cambridge, MA, pp. 247 - 280.

Modestino J.W. and Fries R.W. (1977). "Edge Detection in Noisy Images using Recursive Digital Filtering", *Computer Graphics and Image Processing*, Vol. 6, pp. 409 - 433.

Nalwa V.S. and Binford T.O. (1986). "On Detecting Edges", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, Number 6, pp. 699 - 714.

Neuvo Y., Heinonen P. and Defee I. (1987). "Linear-Median Hybrid Edge Detectors", *IEEE Transactions on Circuits and Systems*, Vol. CAS-34, Number 11, pp. 1337 - 1343.

Neuvo Y., Nieminen A. and Heinonen P. (1987). "A New Class of Detail-Preserving Filters for Image Processing", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, pp. 74 - 90.

Nevatia R. and Babu K.R. (1980). "Linear Feature Extraction and Description", *Computer Graphics and Image Processing*, Vol. 13, pp. 257 - 269.

Ostu N. (1978). "A Threshold Selection Method from Gray-Level Histogram", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-8, pp. 62 - 66.

Pao Y.H. (1988). "Characteristics of the Functional Link Net : A Higher Order Delta Rule Net", *IEEE International Conference on Neural Networks*, pp. I-507 - I-513.

Pao Y.H. $(1989)_a$. *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, pp. 197 - 206.

Pao Y.H. $(1989)_b$. *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, pp. 238 - 245.

Pao Y.H. and Beer R.D. (1988). "The Functional Link Net : A Unifying Network Architecture Incorporating Higher Order Effects", International Neural Network Society, First Annual Meeting, Sept. 6-10, Boston, Mass., pp. 40.

Park R.H. and Choi W.Y. (1989). "A New Interpretation of the Compass Gradient Edge Operators", *Computer Vision, Graphics and Image Processing*, Vol. 42, Number 2, pp. 259 - 265.

Peli T. and Malah D. (1982). "A Study of Edge Detection Algorithms", *Computer Graphics and Image Processing*, Vol. 20, pp. 1 - 21.

Pitas I. and Venetsanopoulos A.N. (1986). "Edge Detectors Based on Nonlinear Filters", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, Number 4, pp. 538 - 550.

Pomerleau D.A. (1987). "The Meta-Generalized Delta Rule : A New Algorithm for Learning in Connectionist Networks", *Computer Science Dept. Report, CMU-CS-87-185*, Carnegie-Mellon University, Pittsburg, PA.

Prager J.M. (1980). "Extracting and Labelling Boundary Segments in Natural Scenes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, Number 1, pp. 16 - 27.

Robinson G.S. (1977). "Edge Detection by Compass Gradient Masks", *Computer Graphics and Image Processing*, Vol. 6, pp. 492 - 501.

Roth M.W. (1989). "Neural Networks for Extraction of Weak Targets in High Clutter Environments", *International Joint Conference on Neural Networks*, Vol. 1, June 18-22, pp. I-275 - I-282.

Rumelhart D.E., Hinton G.E. and McClelland J.L. (1986). "A General Framework for Parallel Distributed Processing", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Vol. 1 : Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA. MIT Press, pp. 45 - 76.

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986). "Learning Internal Representation by Error Propagation", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Vol. 1 : Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA. MIT Press, pp. 318 - 362.

Saito N. and Cunningham M.A. (1990). "Generalized E-Filter and Its Application to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, Number 8, pp. 814 - 817.

Simon H.A. (1969). *The Sciences of the Artificial*, MIT Press, Cambridge, MA, pp. 84 - 118.

Simpson P.K. (1990). "Higher-Ordered and Intraconnected Bidirectional Associative Memories", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, Number 3, pp. 637 - 653.

Sobajic D. (1988). "Artificial Neural Networks for Transient Stability Assessment of Electric Power Systems", *Ph. D. Thesis*, Computer Science Dept., Case Western Reserve University, Cleveland, OH.

Stone G.O. (1986). "An Analysis of the Delta Rule and the Learning of Statistical Associations", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Vol. 1 : Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA. MIT Press, pp. 444 - 459.

Strickland R.N., Draelos T. and Mao Z. (1990). "Edge Detection in Machine Vision using a Simple L_1 Norm Template Matching Algorithm", *Pattern Recognition*, Vol. 23, Number 5, pp. 411 - 421.

Thathachar M.A.L. and Sastry P.S. (1985). "A New Approach to the Design of Reinforcement Schemes for Learning Automata", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-15, pp. 168 - 175.

Waibel A., Hanazawa T., Hinton G., Shikano K. and Lang K.J. (1989). "Phoneme Recognition Using Time-Delay Neural Networks", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, Number 3, pp. 328 - 339.

Wang S. and Haralick R.M. (1984). "Automatic Multi-Threshold Selection", *Computer Vision, Graphics and Image Processing*, Vol. 25, Number 1, pp. 46 - 67.

Widrow B. and Stearns S.D. (1985). *Adaptive Signal Processing*, Prentice Hall, Englewood, Cliffs, N.J., pp. 99 - 114.

Widrow B. and Winter R. (1988). "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition", *Computer*, Vol. 21, Number 3, pp. 25 - 39.

Widrow B., Winter R.G. and Baxter R.A. (1988). "Layered Neural Nets for Pattern Recognition", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 36, Number 7, pp. 1109 - 1118.

Williams D.J. and Shah M. (1990). "Edge Contours Using Multiple Scales", *Computer Vision, Graphics and Image Processing*, Vol. 51, Number 3, pp. 256 - 274.

Xu X. and Tsai W.T. (1990). "Constructing Associative Memories Using Neural Networks", *Neural Networks*, Vol. 3, Number 3, pp. 301 - 309.

Zucker S.W., Hummel R.A. and Rosenfeld A. (1977). "An Application of Relaxation Labelling to Line and Curve Enhancement", *IEEE Transactions of Computers*, Vol. C-26, Number 4, pp. 394 - 403.

APPENDIX - A

Test Results

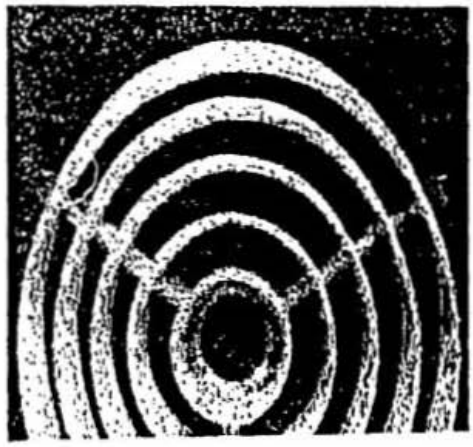


Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

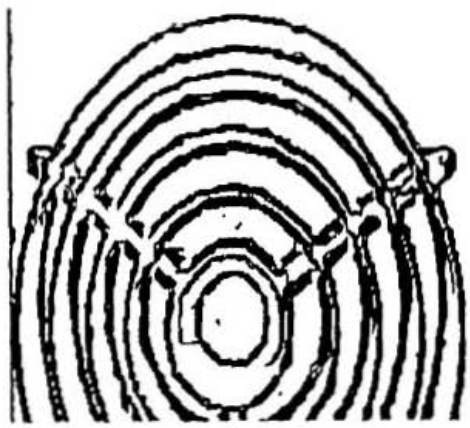


Figure. A3 Improved edge image - after processing by neural network system

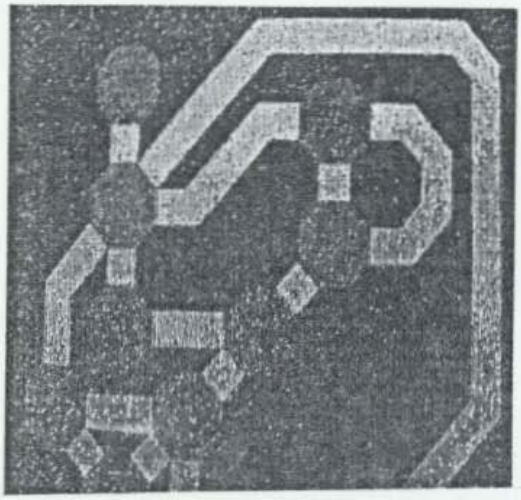


Figure. A1 Degraded and noise corrupted original image

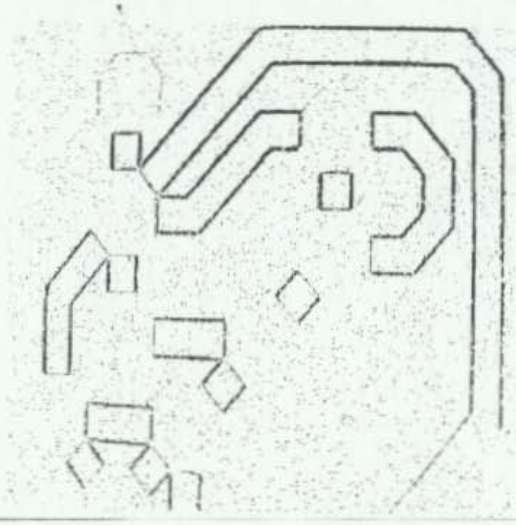


Figure. A2 Edge image - before processing by neural network system

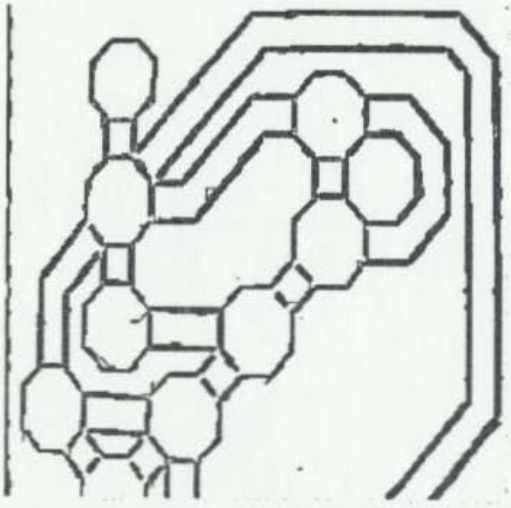


Figure. A3 Improved edge image - after processing by neural network system

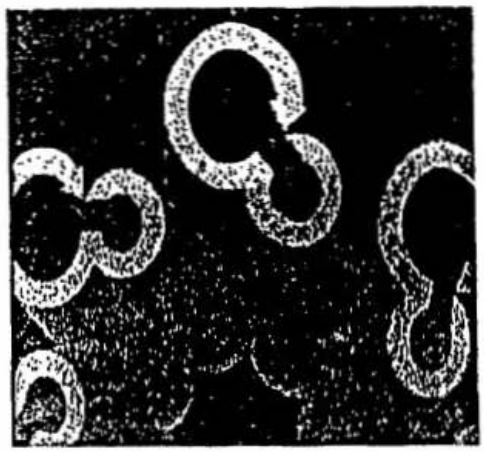


Figure. A1 Degraded and noise corrupted original image

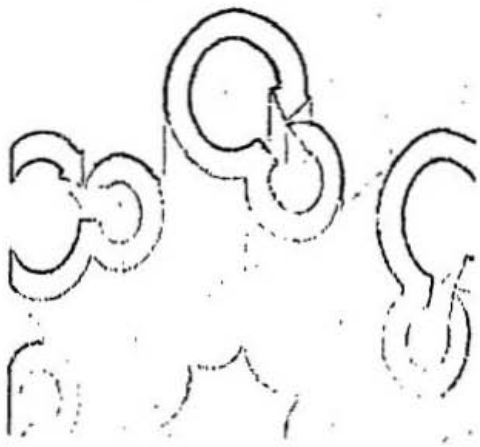


Figure. A2 Edge image - before processing by neural network system

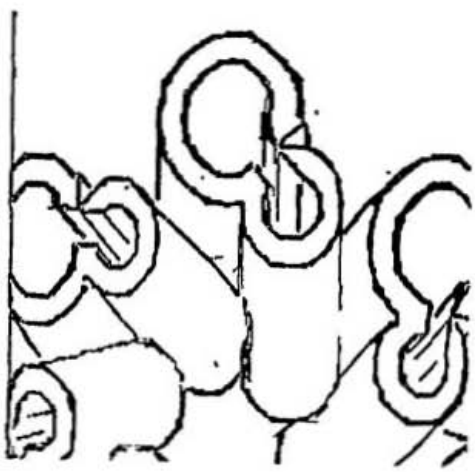


Figure. A3 Improved edge image - after processing by neural network system

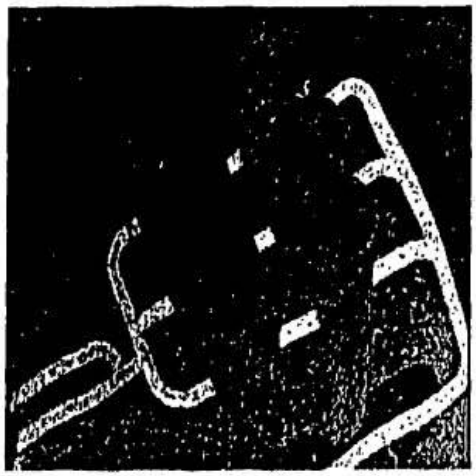


Figure. A1 Degraded and noise corrupted original image

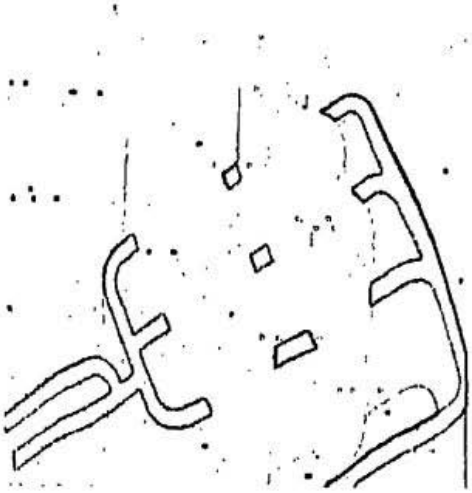


Figure. A2 Edge image - before processing by neural network system

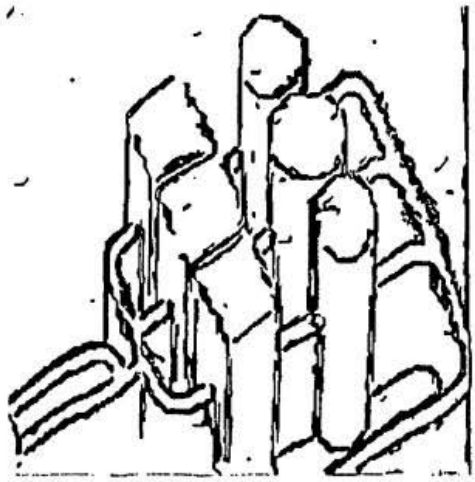


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

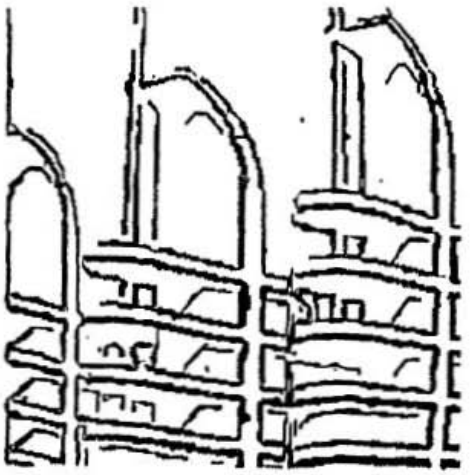


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

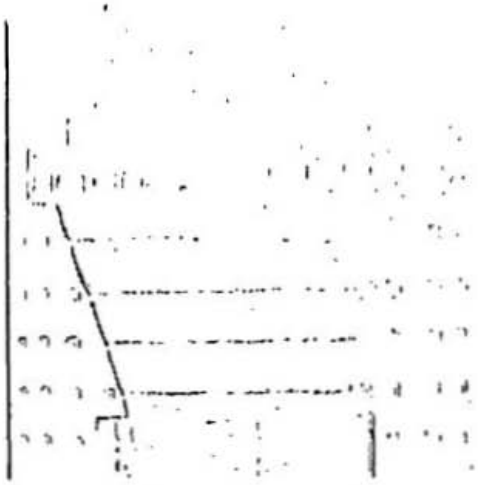


Figure. A2 Edge image - before processing by neural network system

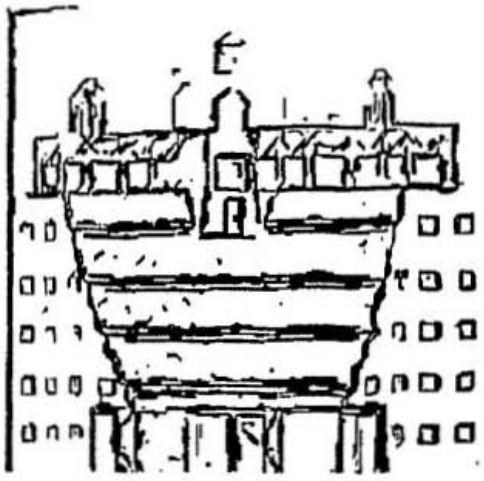


Figure. A3 Improved edge image - after processing by neural network system

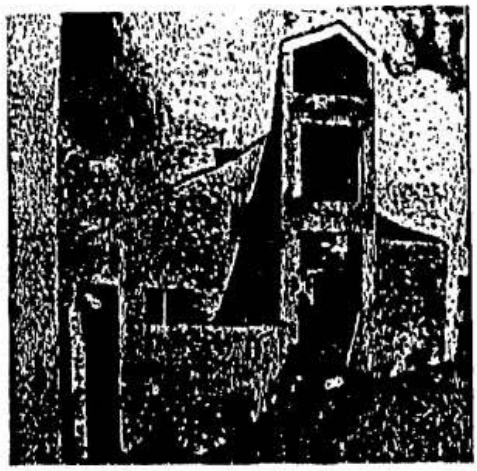


Figure. A1 Degraded and noise corrupted original image

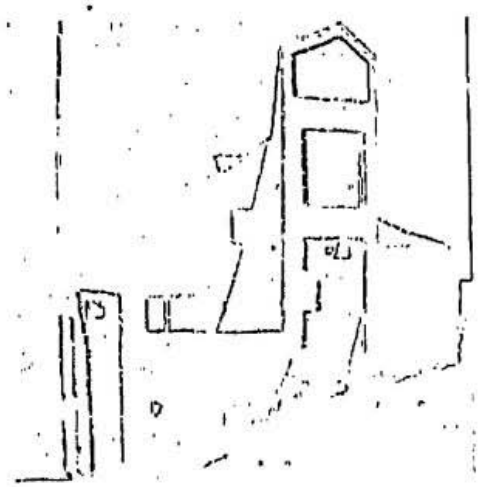


Figure. A2 Edge image - before processing by neural network system

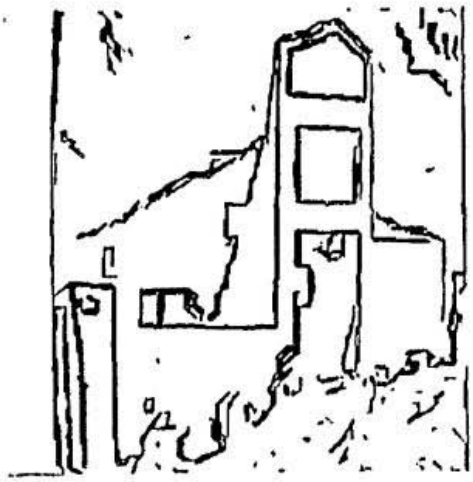


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

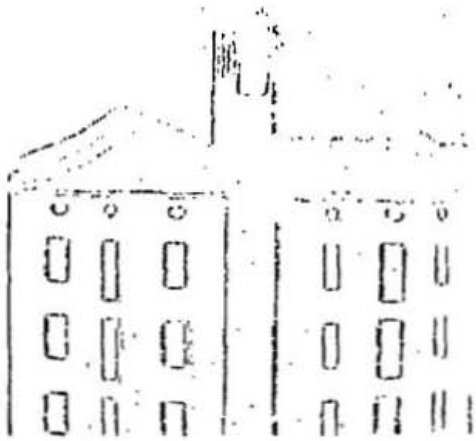


Figure. A2 Edge image - before processing by neural network system

Figure. A3 Improved edge image - after processing by neural network system

Figure. Ai Degraded and noise corrupted original image

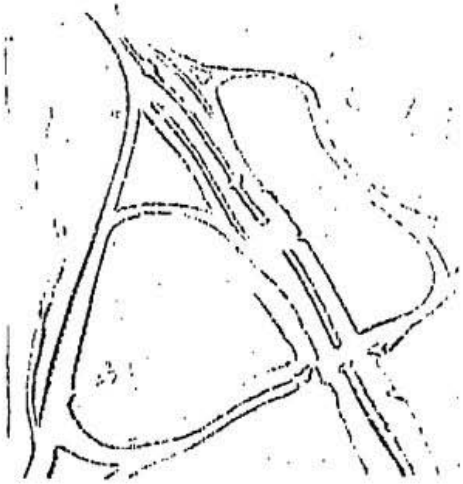


Figure. A2 Edge image - before processing by neural network system

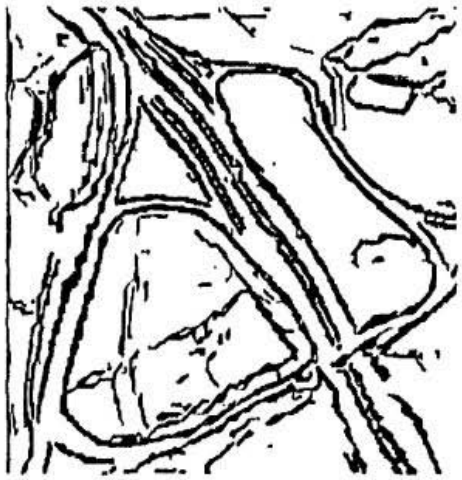


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

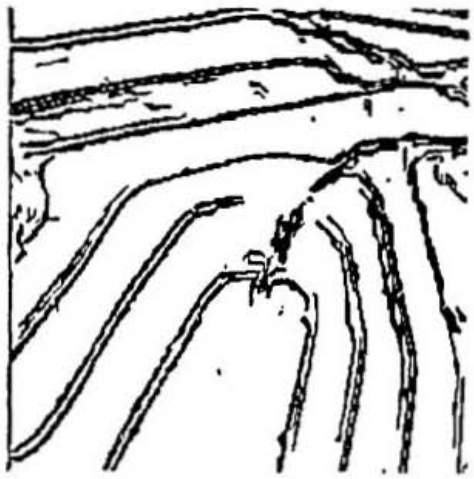


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

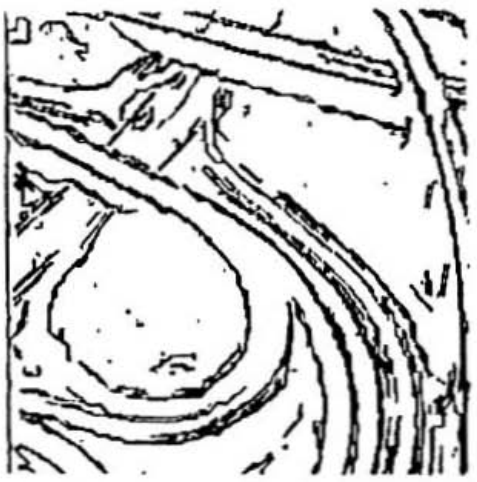


Figure. A3 Improved edge image - after processing by neural network system

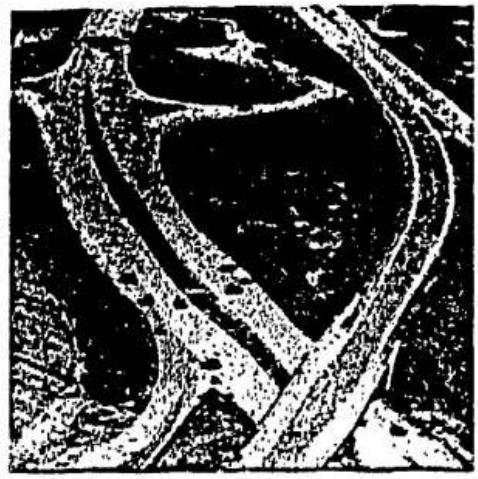


Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

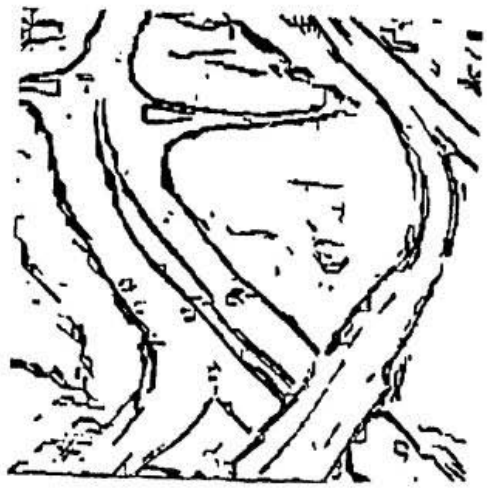


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

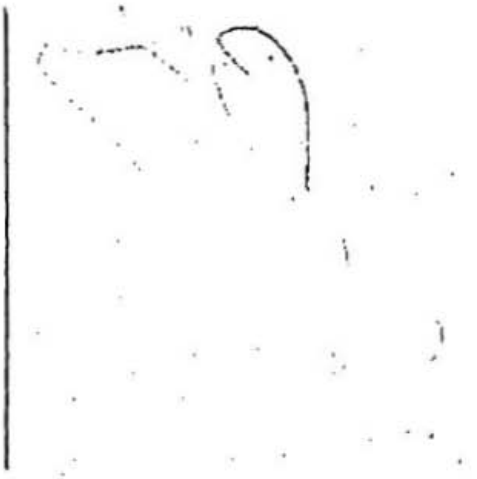


Figure. A2 Edge image - before processing by neural network system

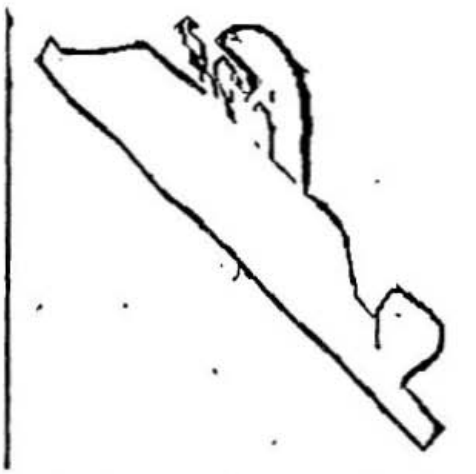


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

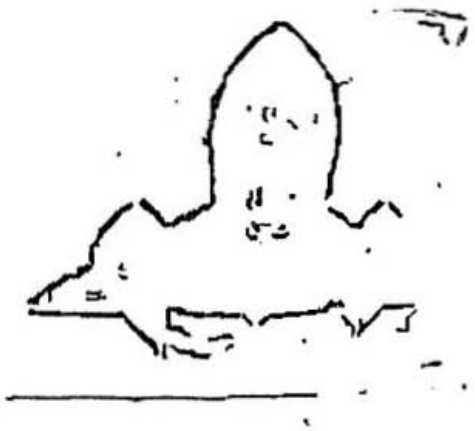


Figure. A3 Improved edge image - after processing by neural network system

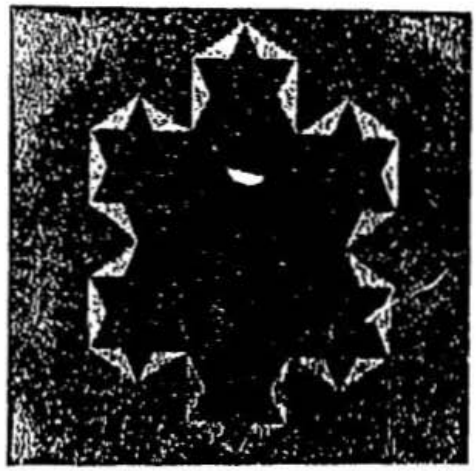


Figure. A1 Degraded and noise corrupted original image

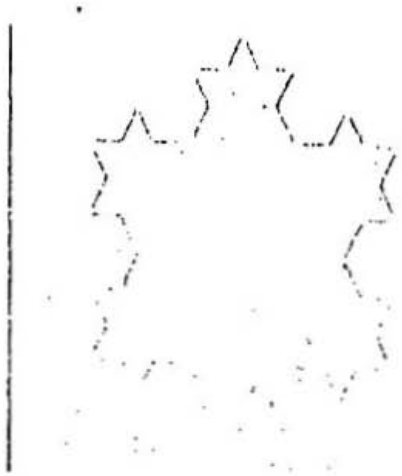


Figure. A2 Edge image - before processing by neural network system

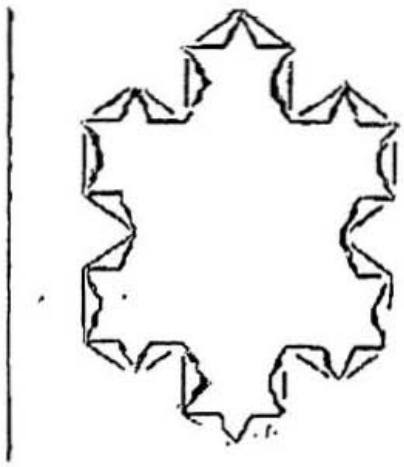


Figure. A3 Improved edge image - after processing by neural network system

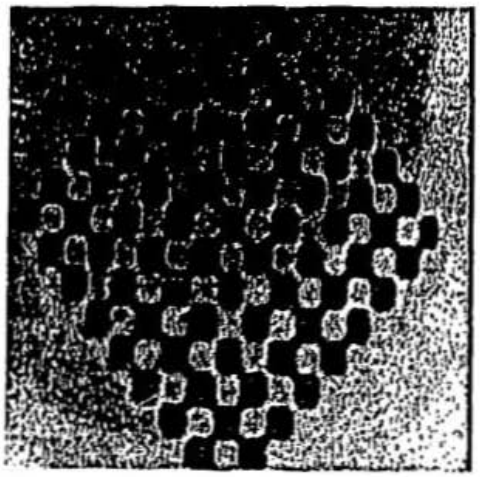


Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

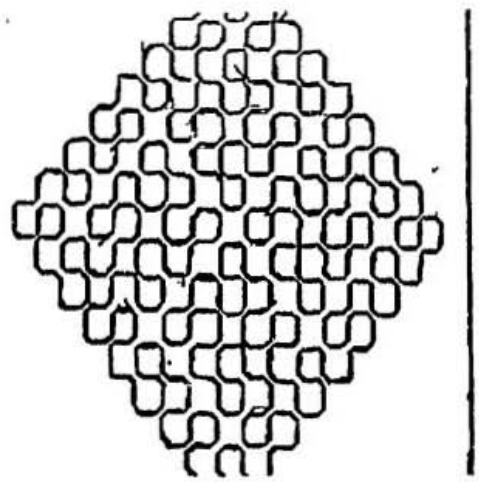


Figure. A3 Improved edge image - after processing by neural network system

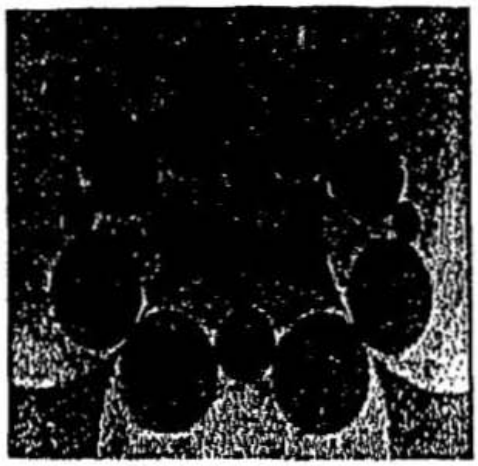


Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

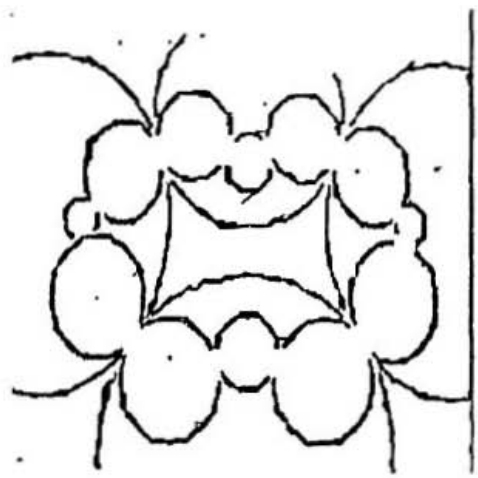


Figure. A3 Improved edge image - after processing by neural network system

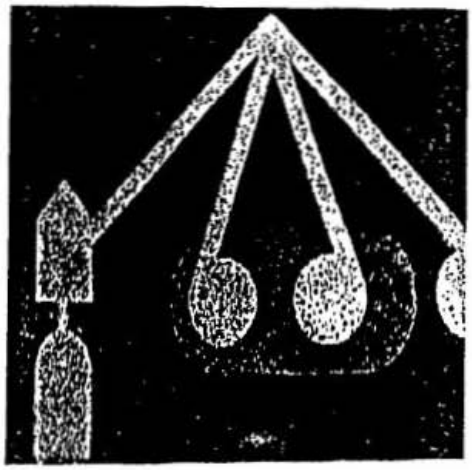


Figure. A1 Degraded and noise corrupted original image

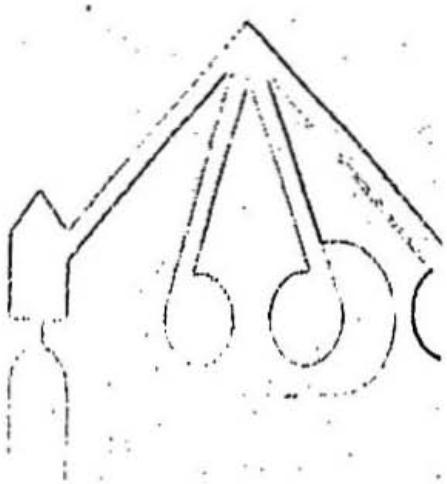


Figure. A2 Edge image - before processing by neural network system

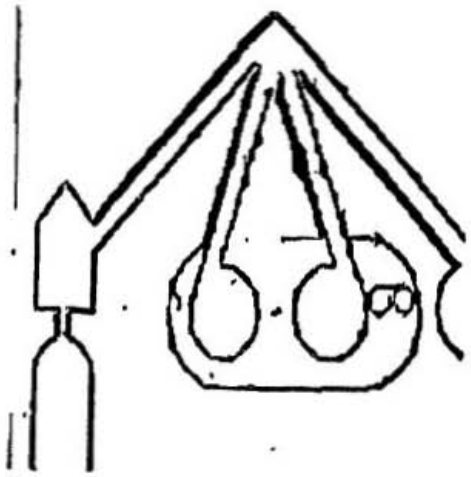


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

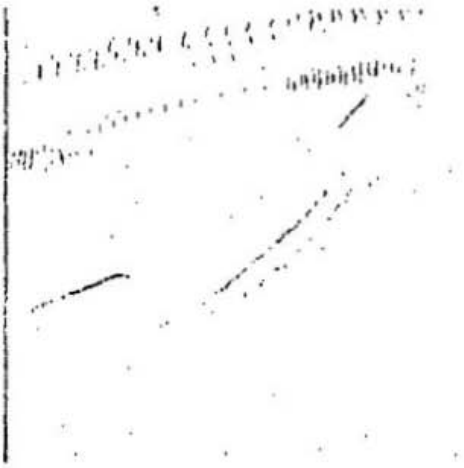


Figure. A2 Edge image - before processing by neural network system

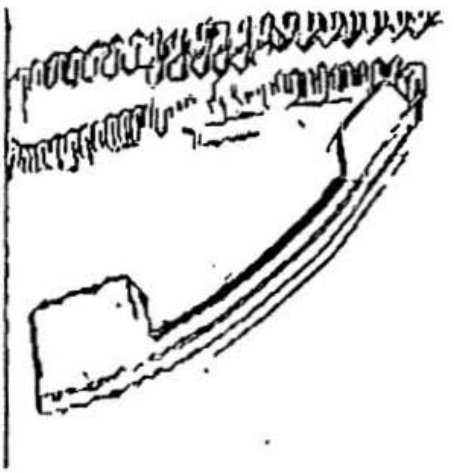


Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

Figure. A2 Edge image - before processing by neural network system

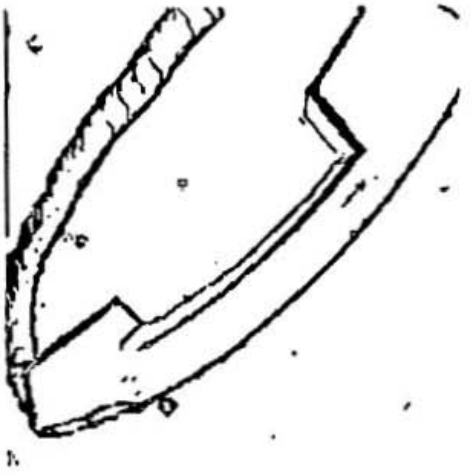


Figure. A3 Improved edge image - after processing by neural network system

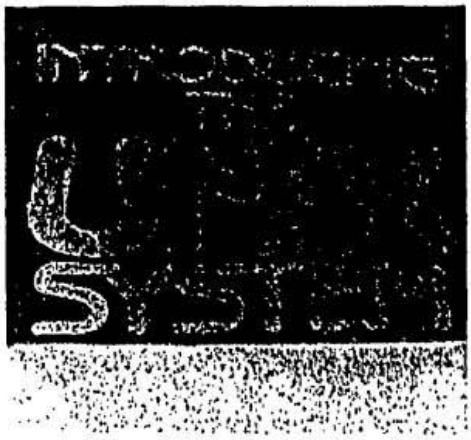


Figure. A1 Degraded and noise corrupted original image

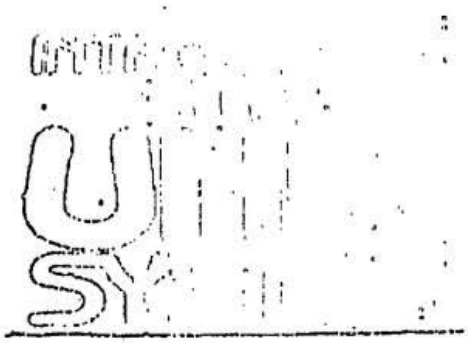


Figure. A2 Edge image - before processing by neural network system

Figure. A3 Improved edge image - after processing by neural network system

Figure. A1 Degraded and noise corrupted original image

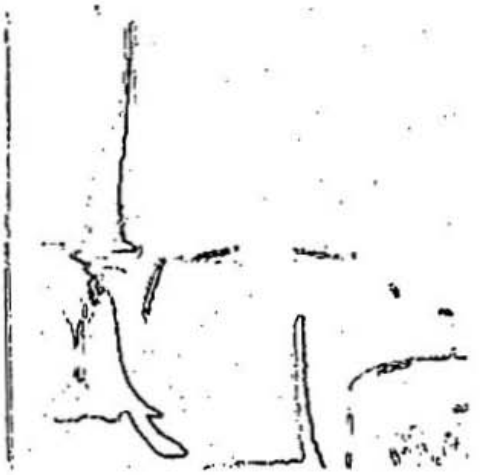
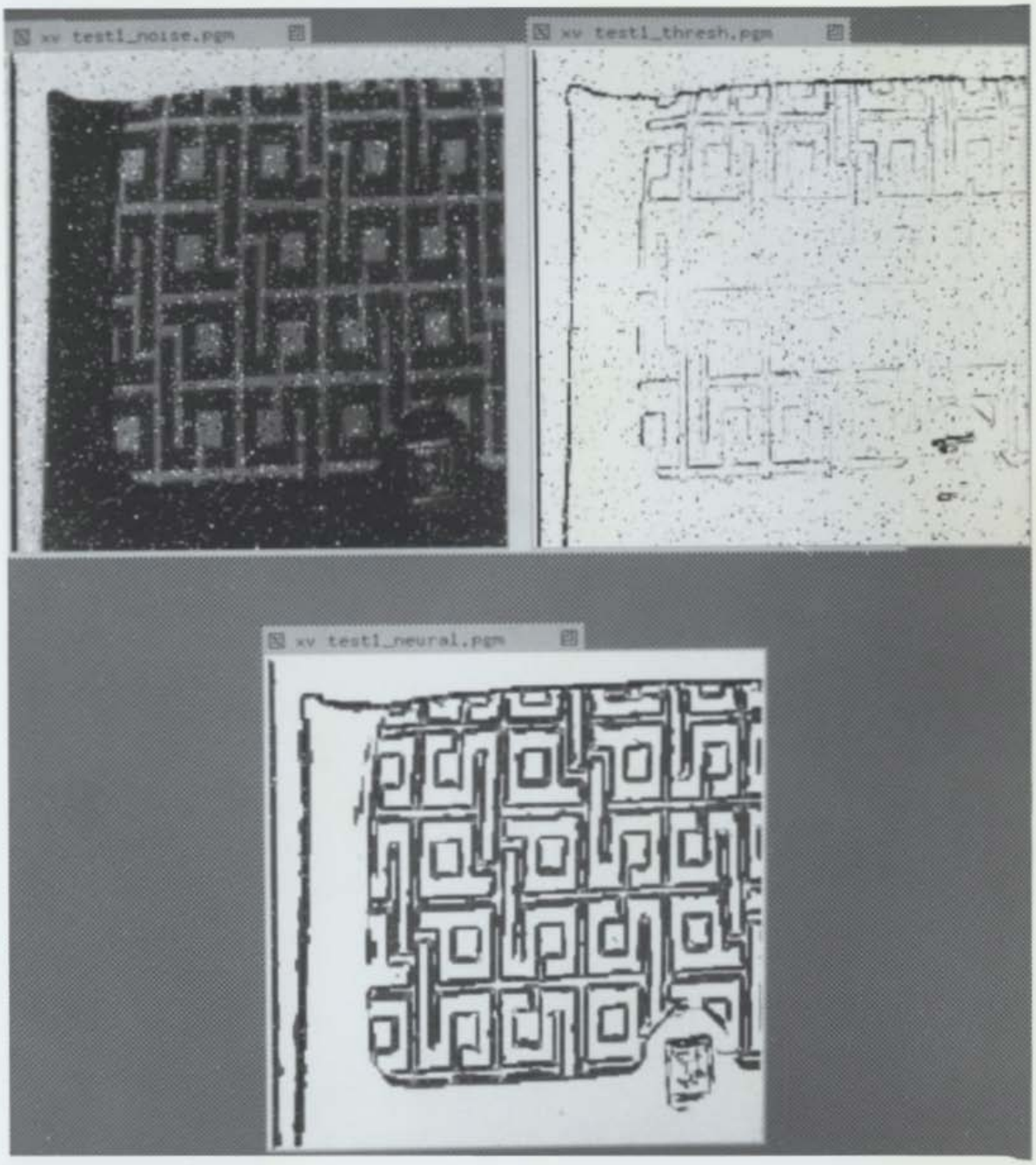
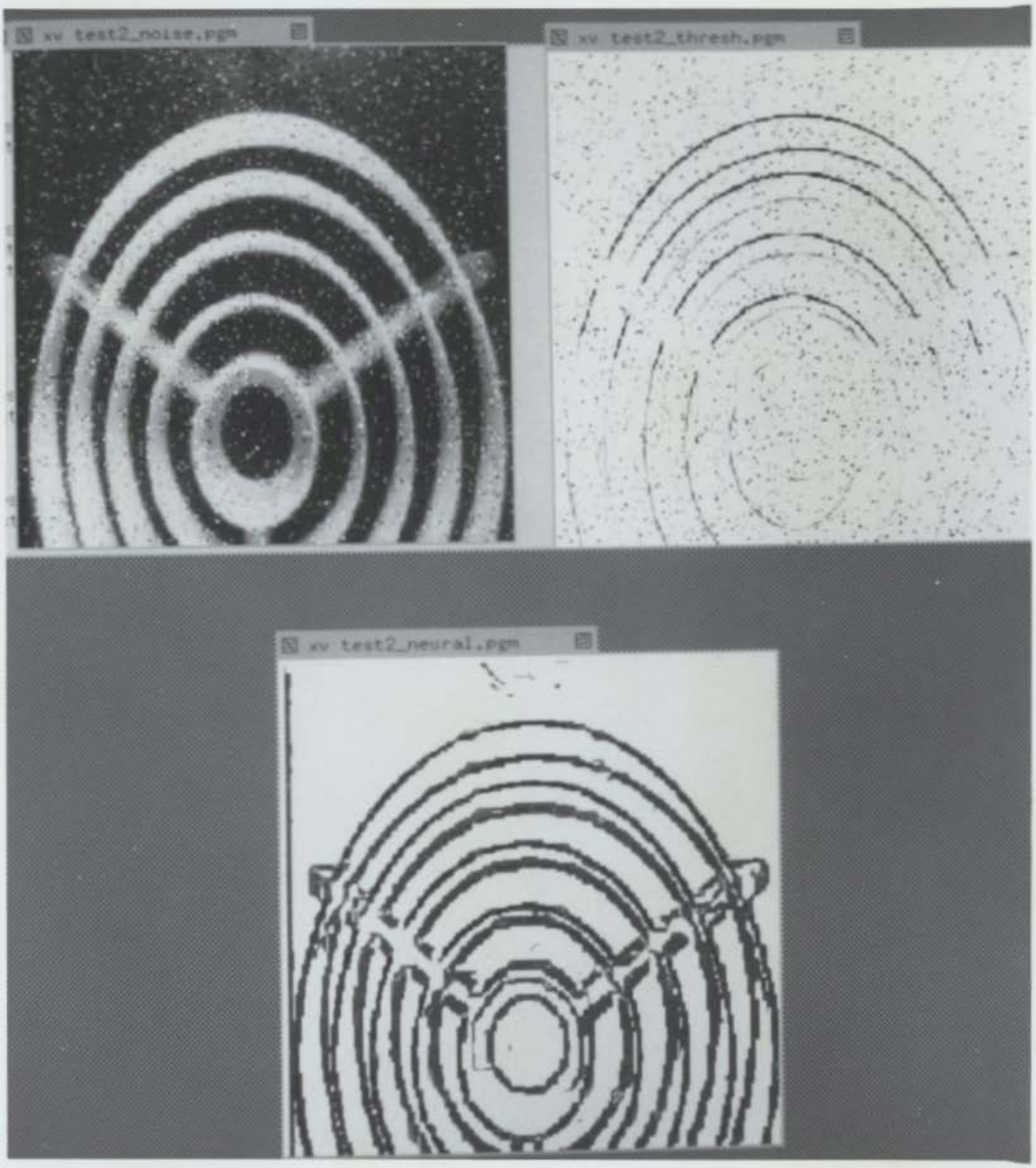
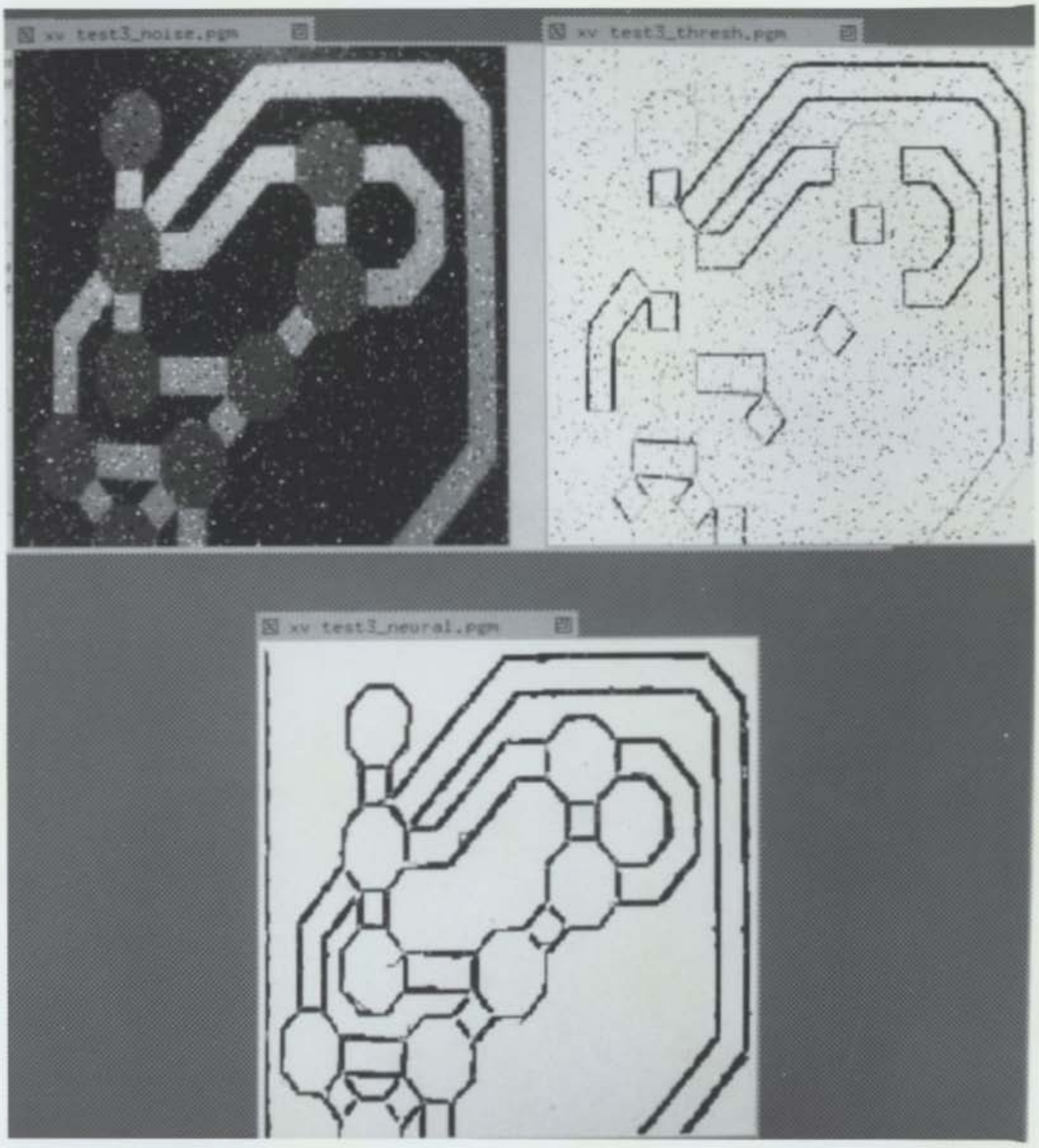


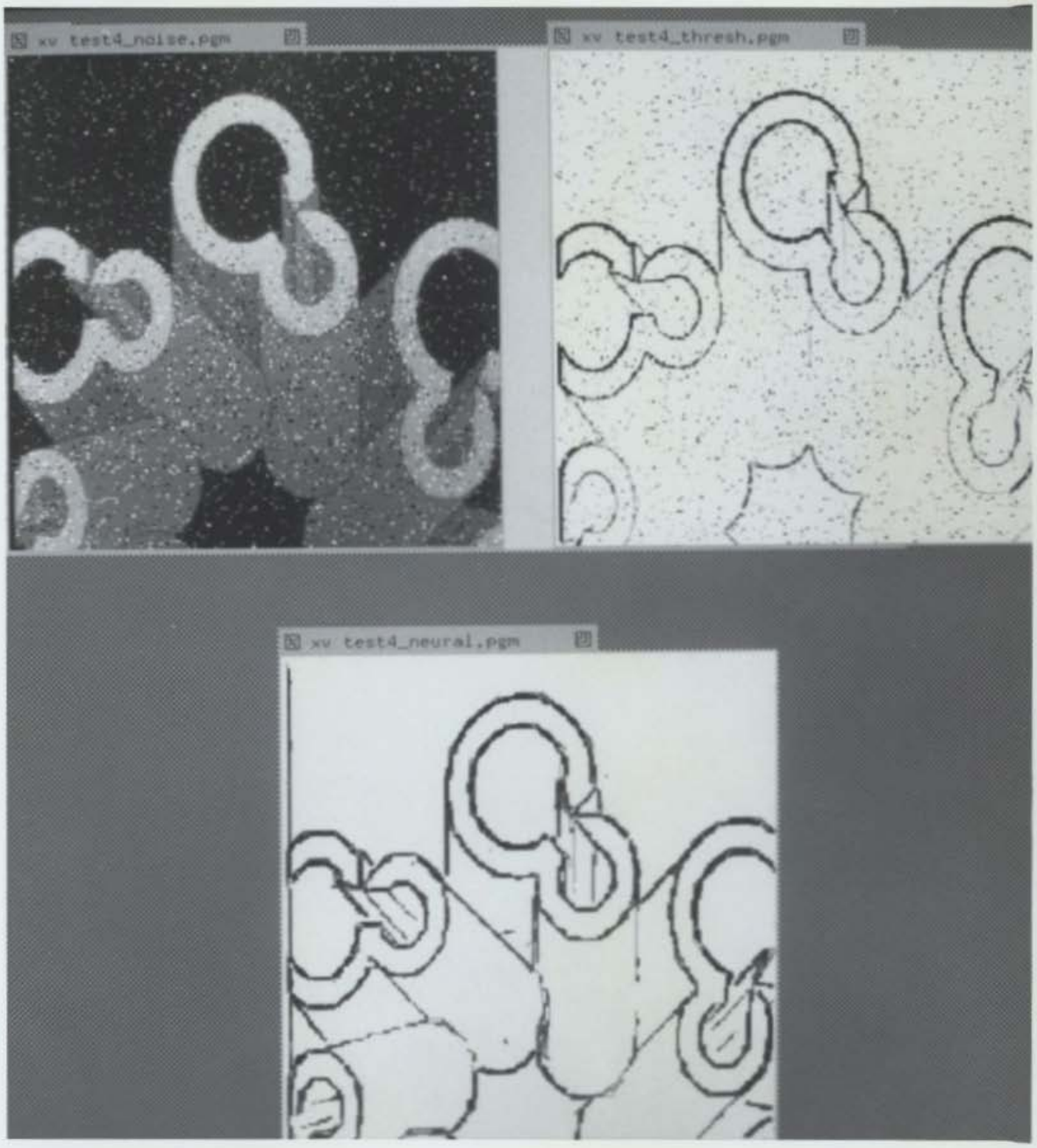
Figure. A2 Edge image - before processing by neural network system

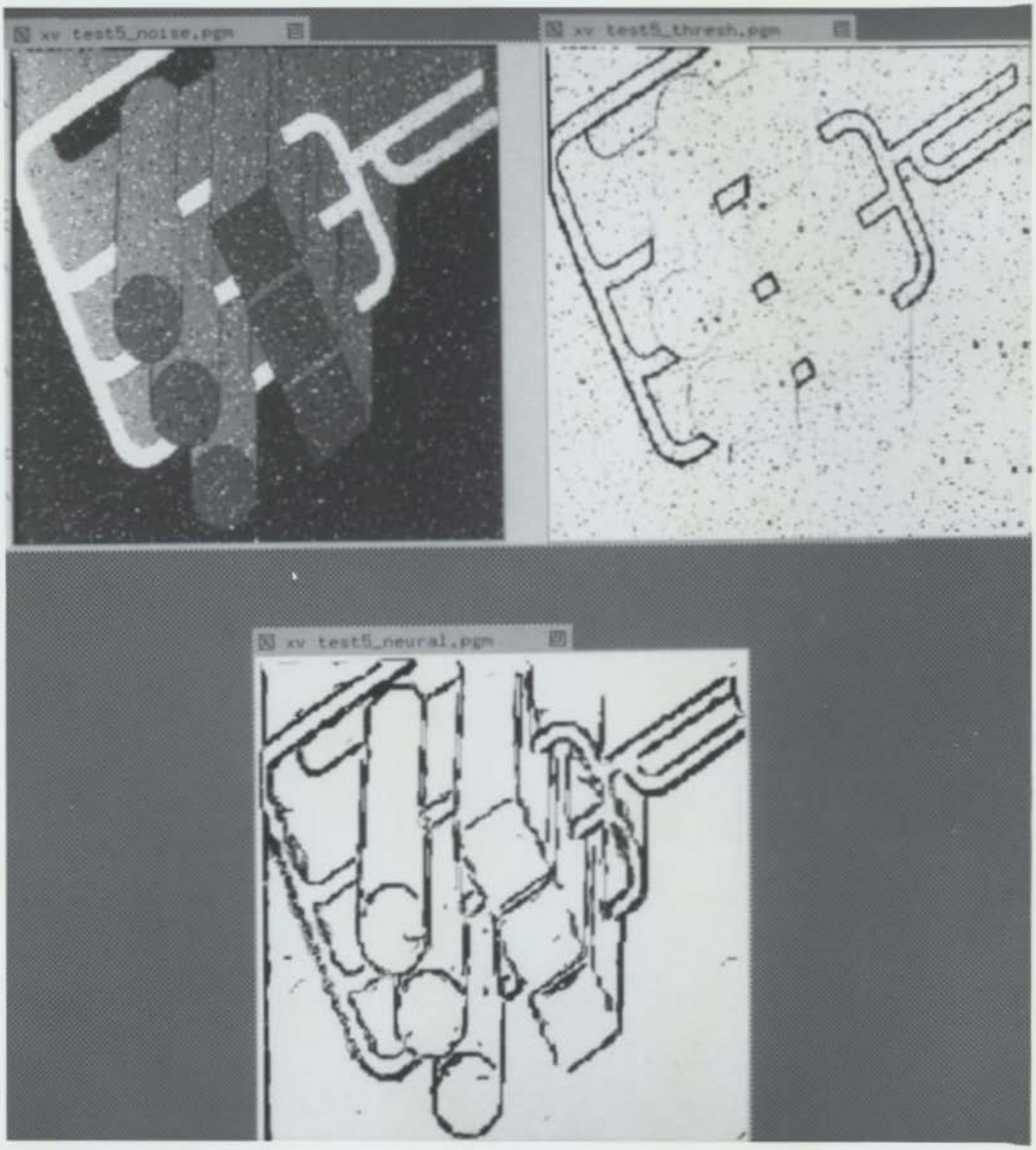
Figure. A3 Improved edge image - after processing by neural network system

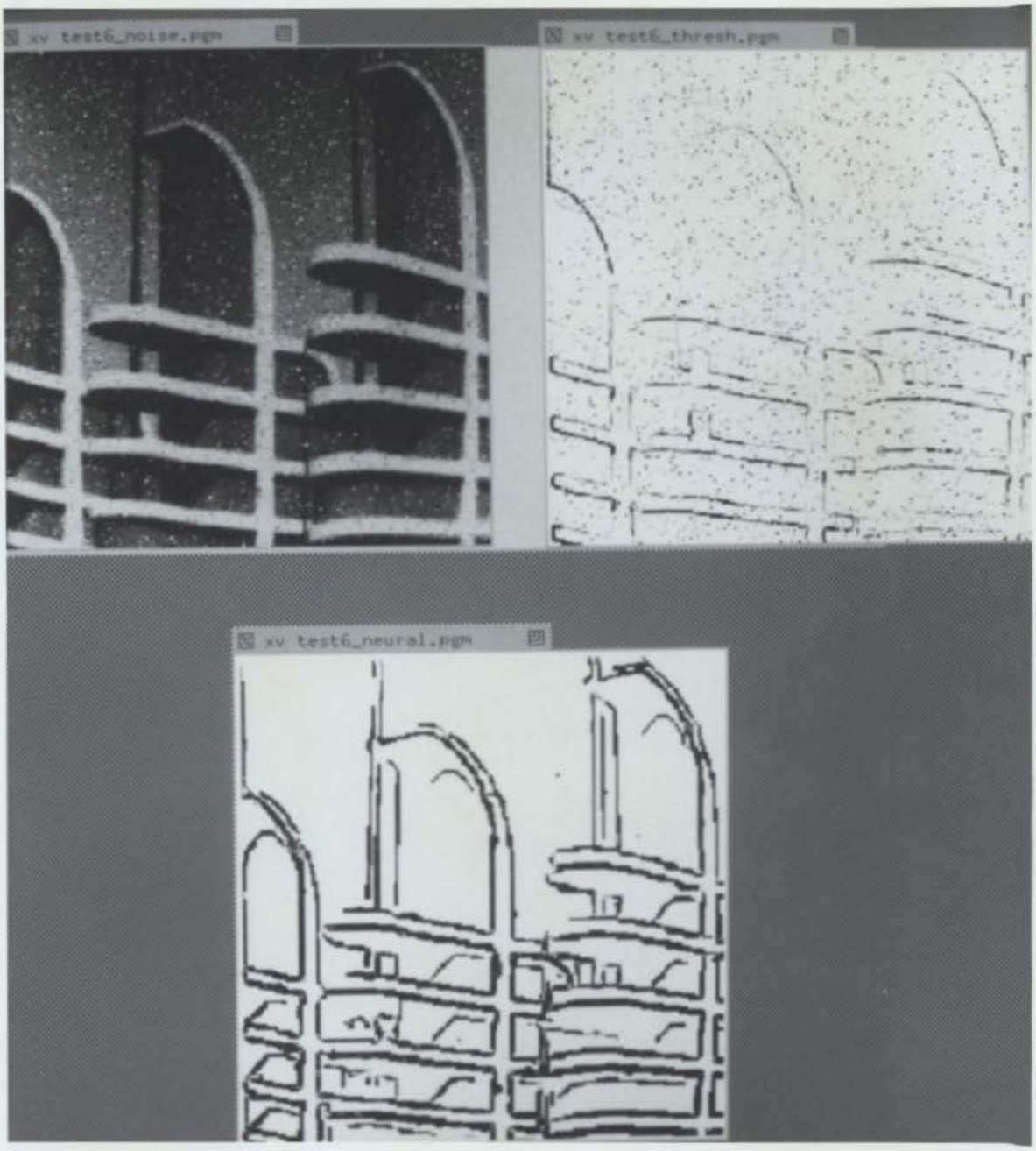

xv test1_noise.pgm
xv test1_thresh.pgm
xv test1_neural.pgm

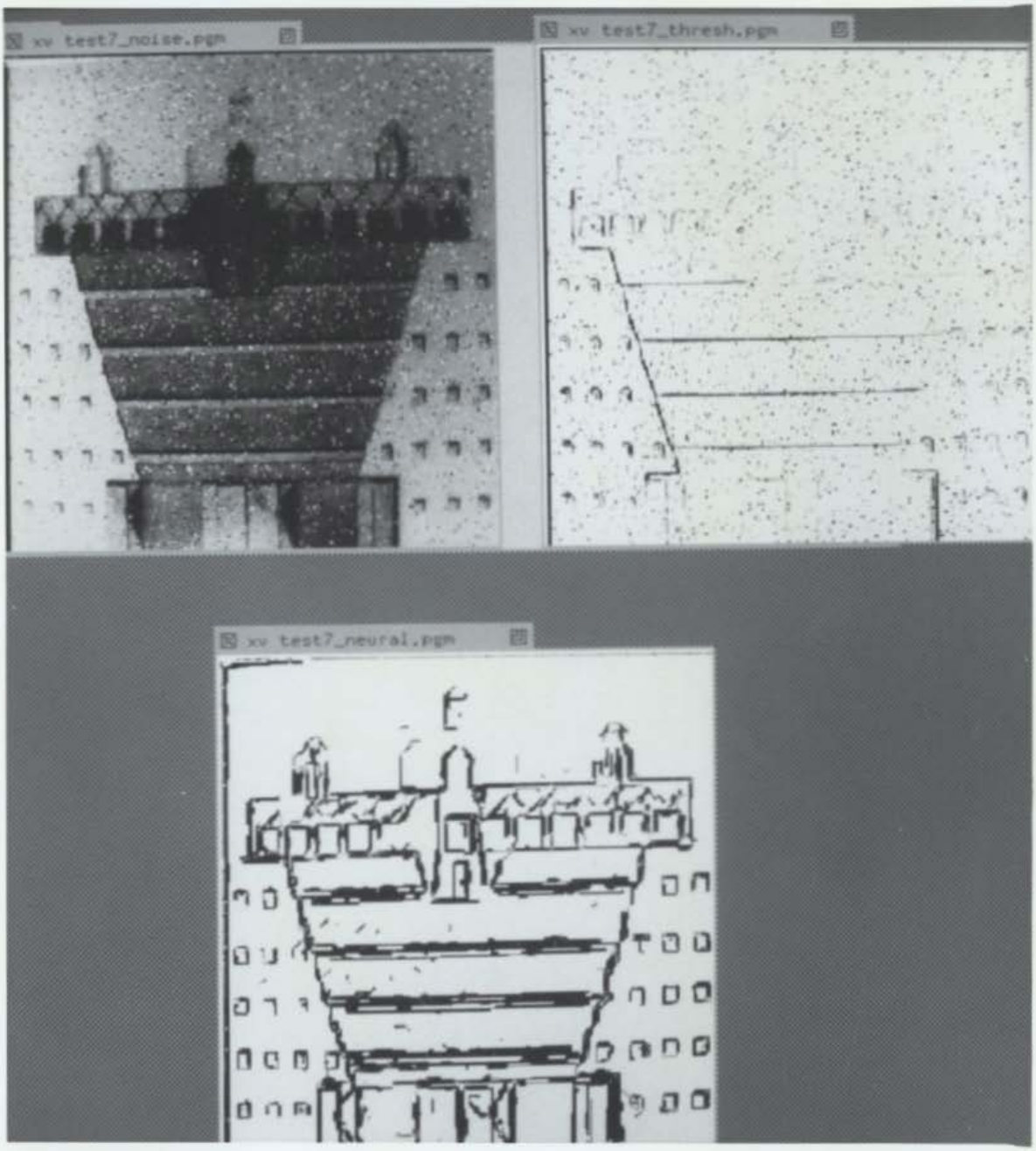

xv test2_noise.pgm
xv test2_thresh.pgm
xv test2_neural.pgm

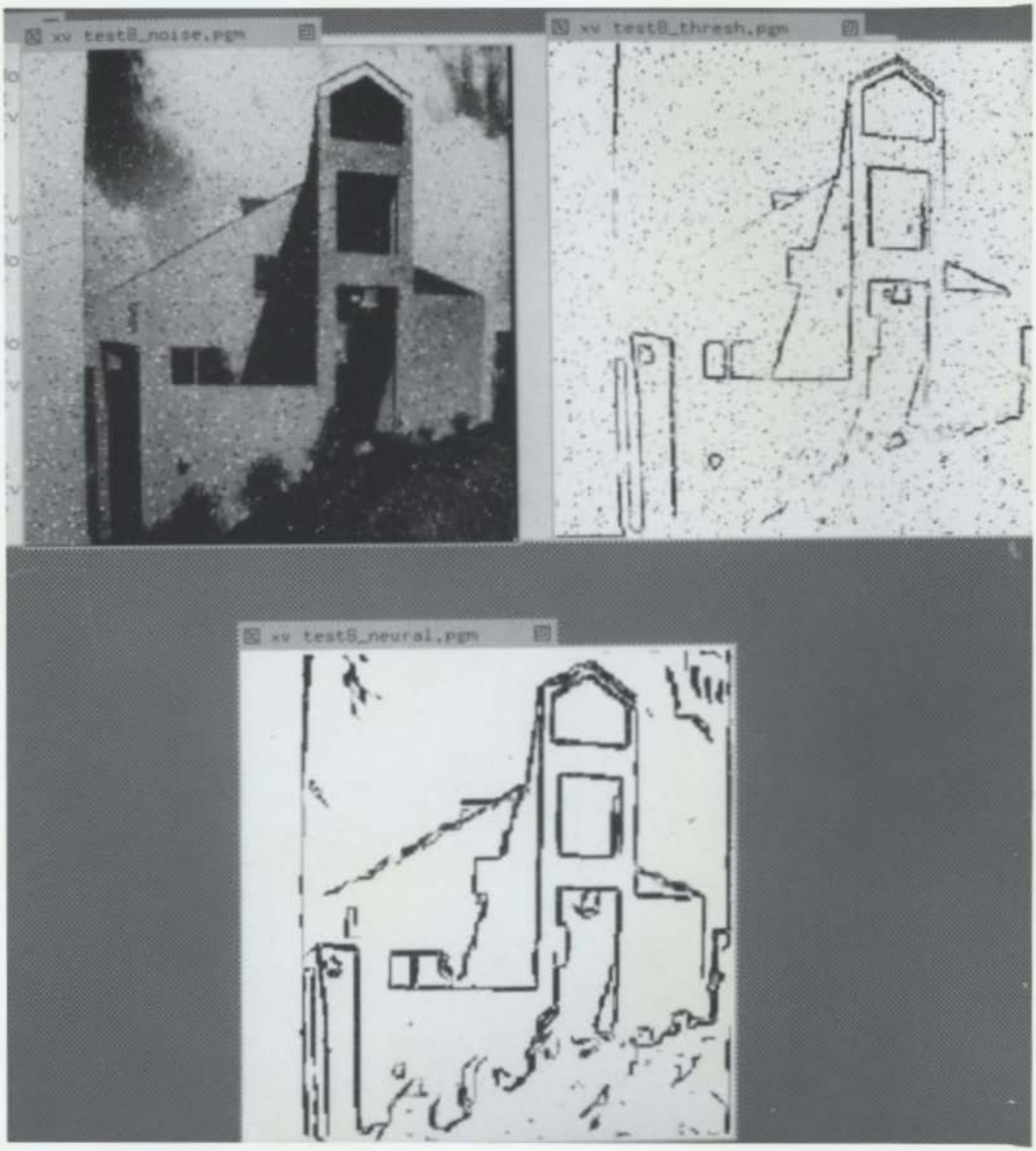

xv test3_noise.pgm
xv test3_thresh.pgm
xv test3_neural.pgm

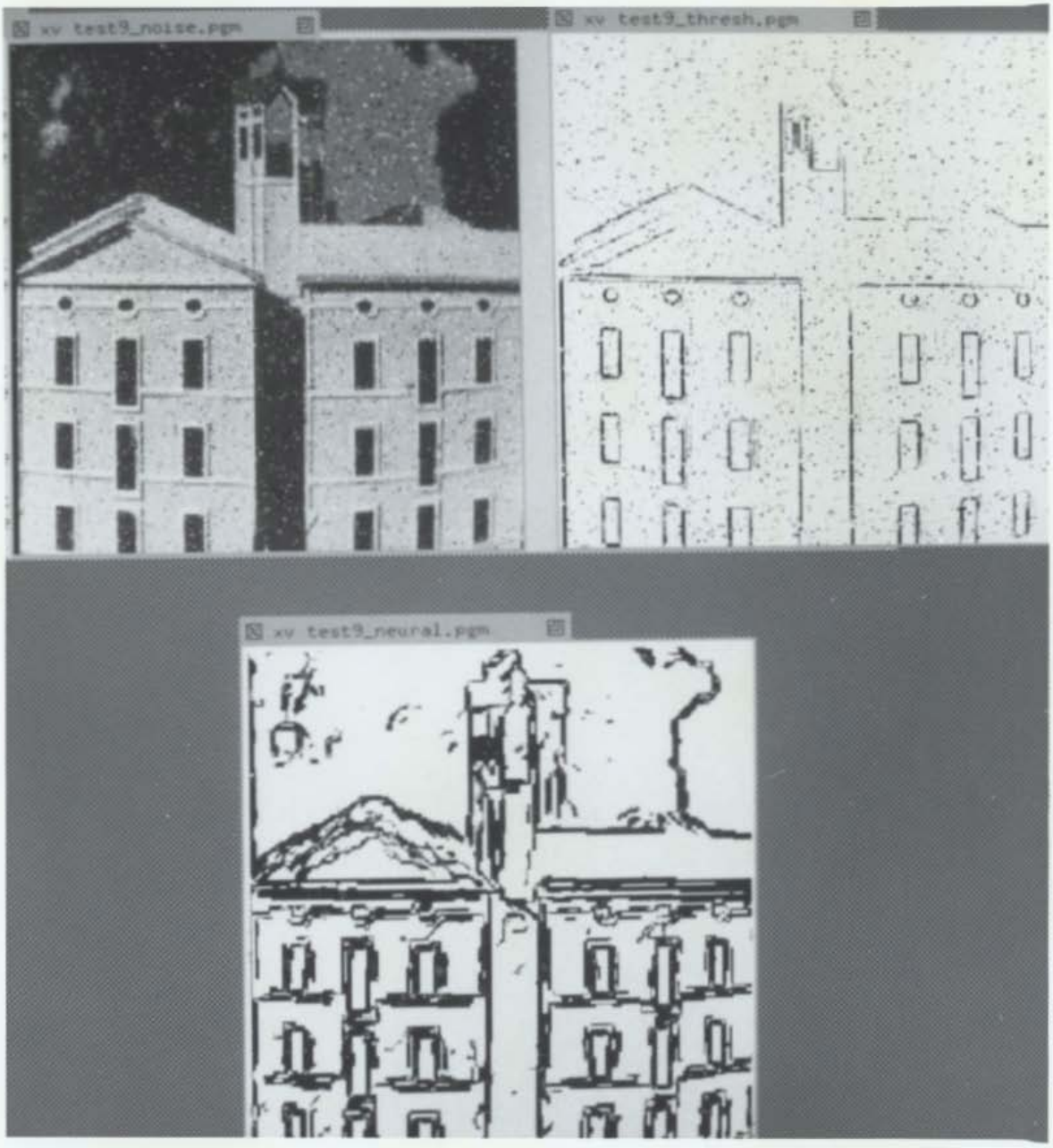

xv test4_noise.pgm
xv test4_thresh.pgm
xv test4_neural.pgm

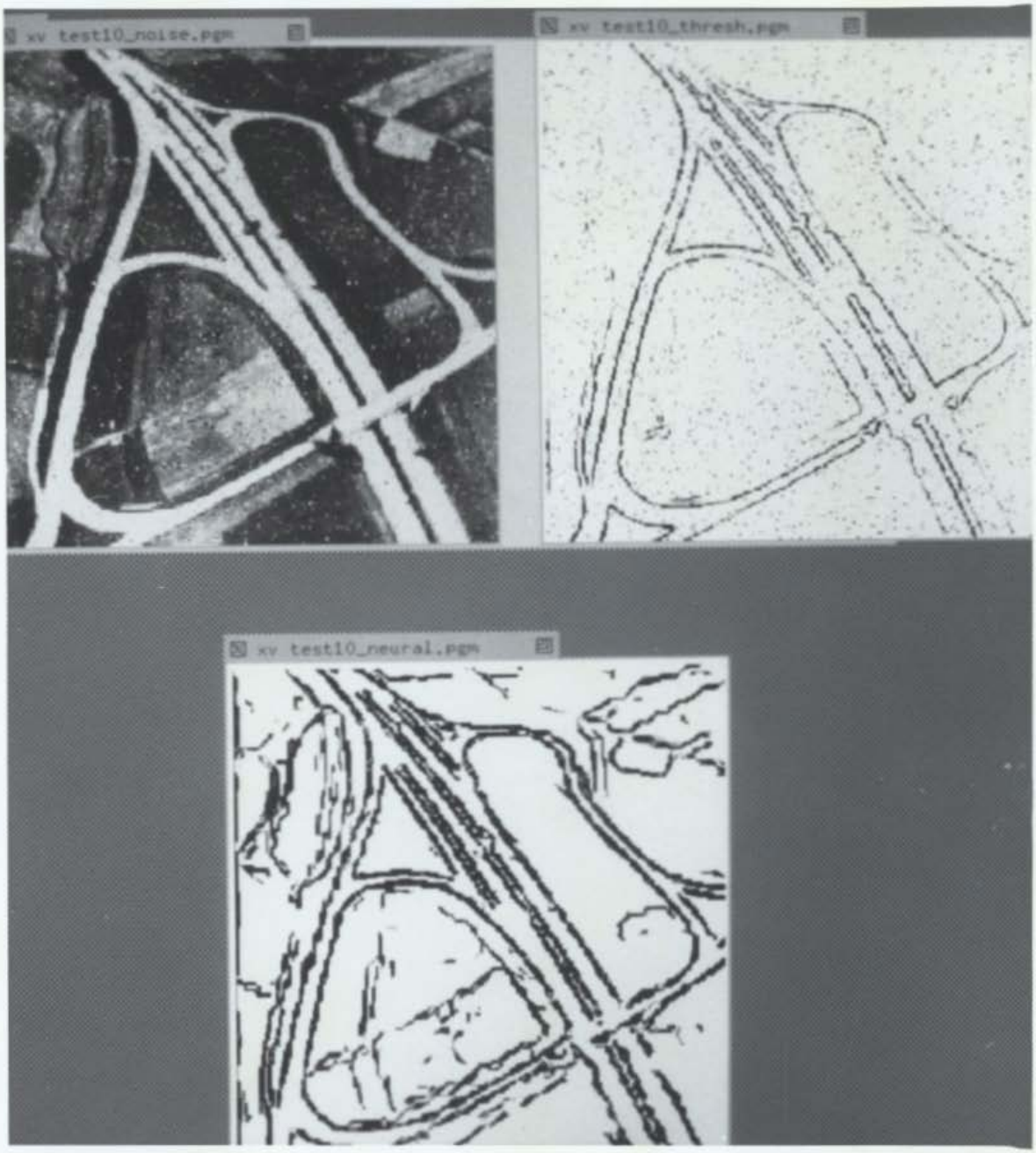

xv test5_noise.pgm
xv test5_thresh.pgm
xv test5_neural.pgm


xv test6_noise.pgm
xv test6_thresh.pgm
xv test6_neural.pgm

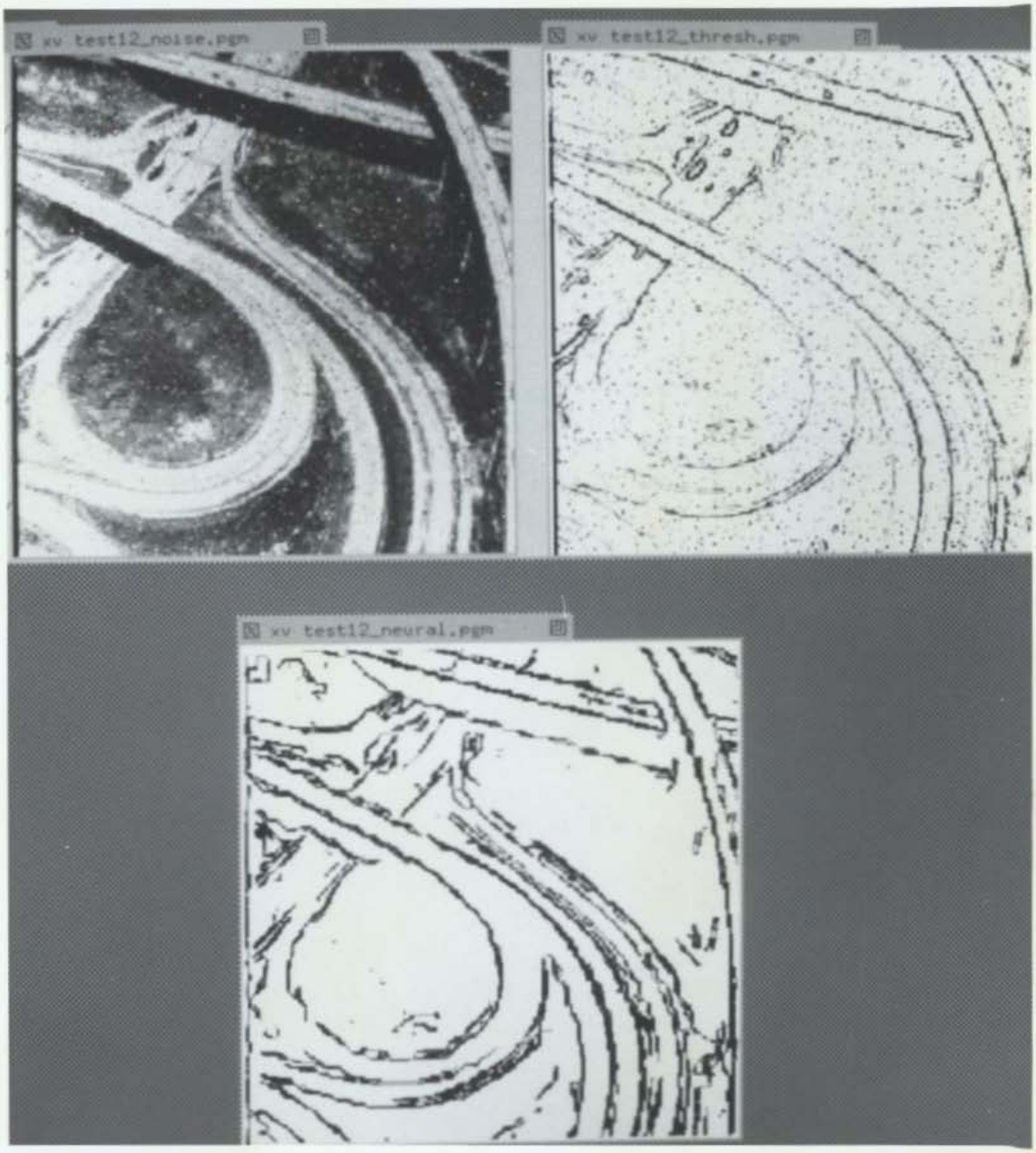

xv test7_noise.pgm
xv test7_thresh.pgm
xv test7_neural.pgm

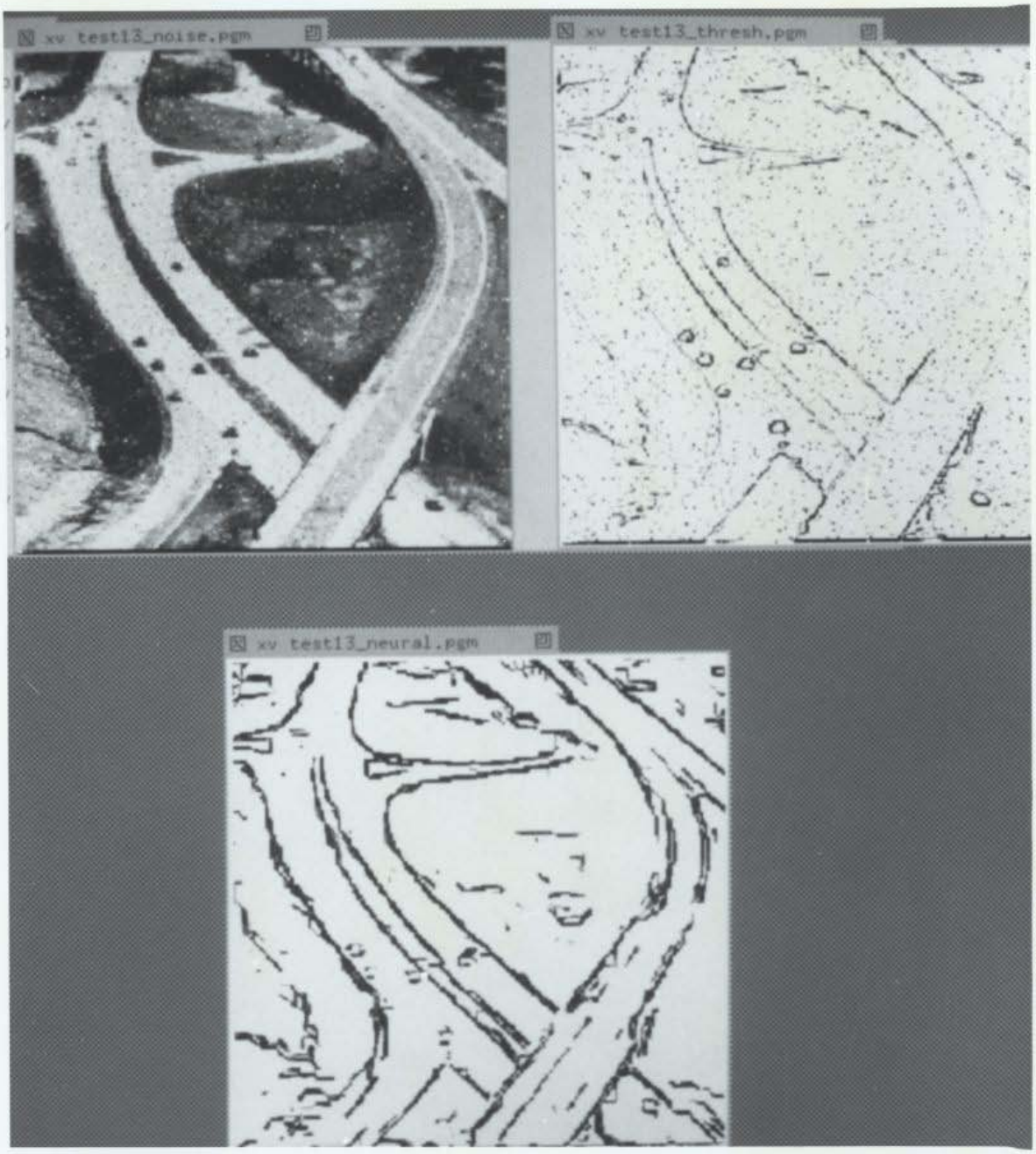

xv test8_noise.pgm
xv test8_thresh.pgm
xv test8_neural.pgm

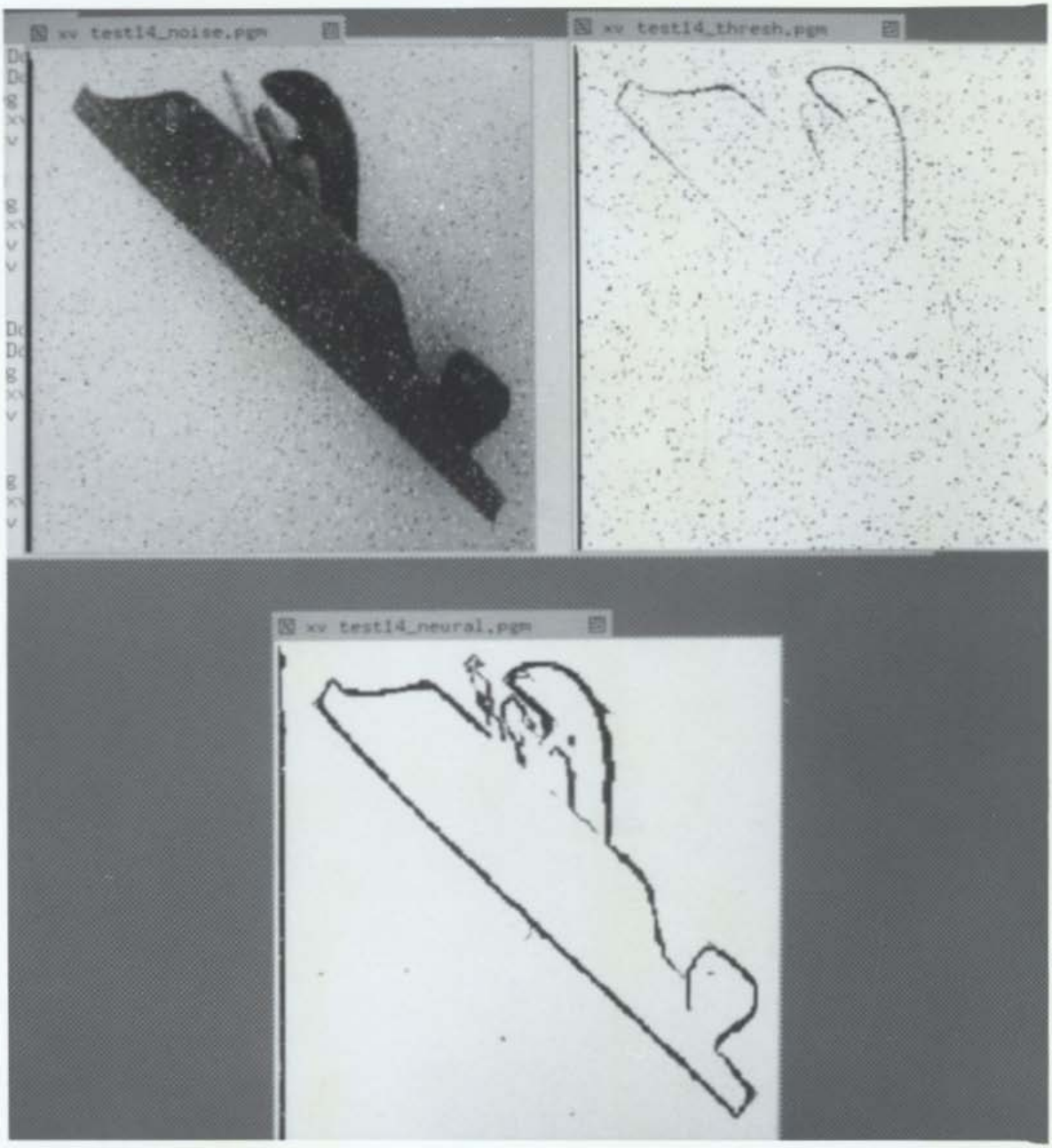

xv test9_noise.pgm
xv test9_thresh.pgm
xv test9_neural.pgm

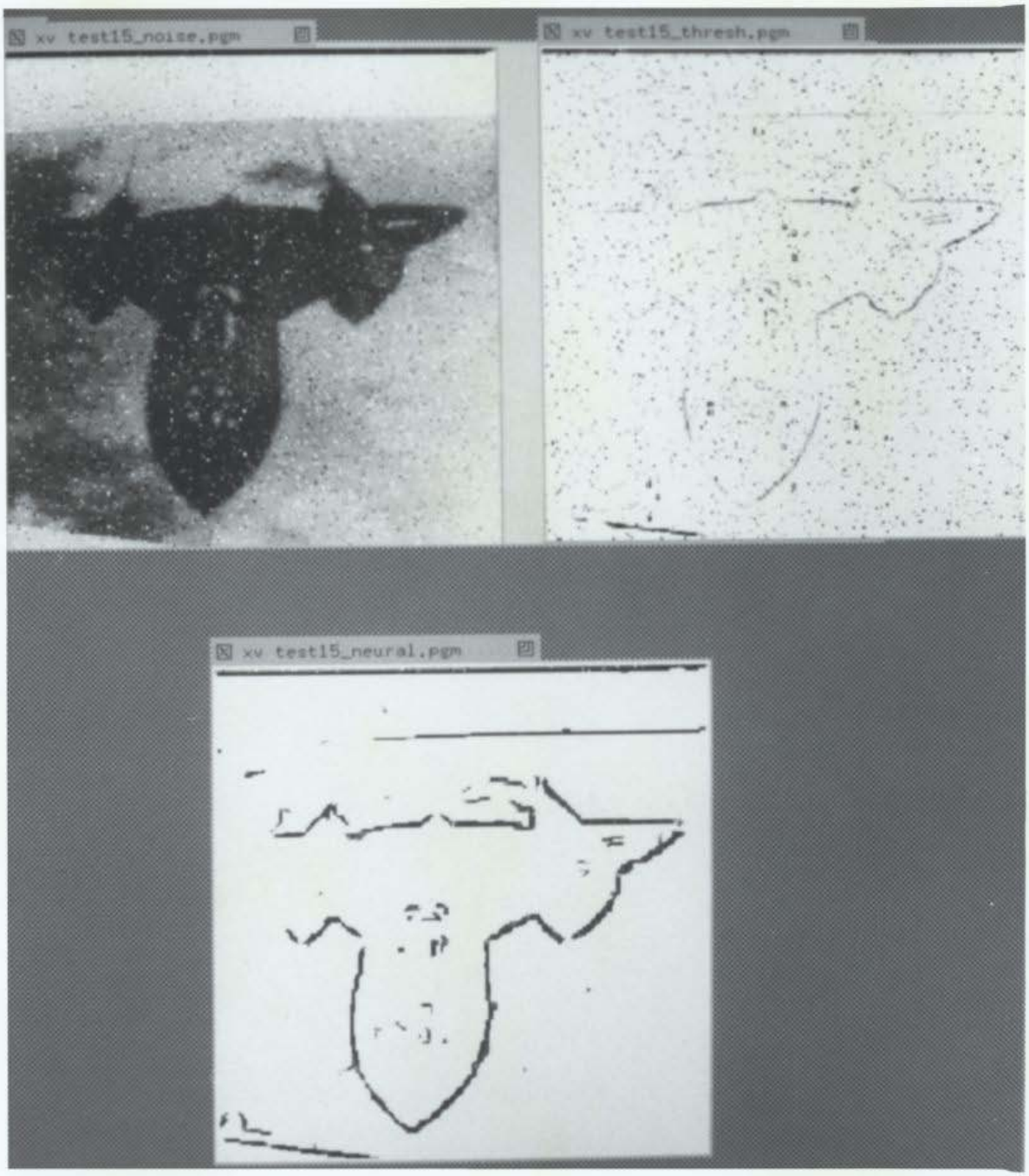

xv test10_noise.pgm
xv test10_thresh.pgm
xv test10_neural.pgm

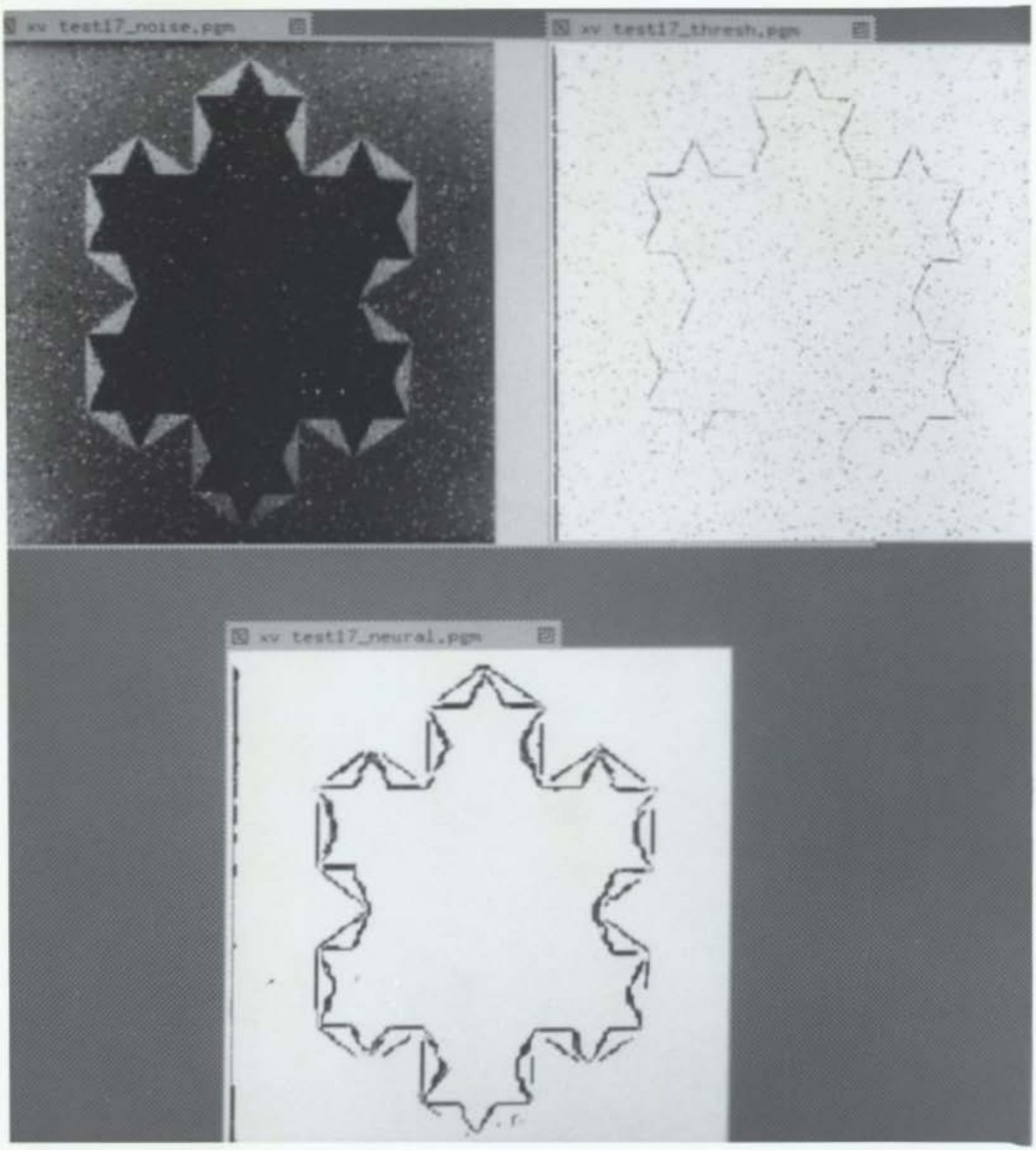

xv test11_noise.pgm
xv test11_thresh.pgm
xv test11_neural.pgm

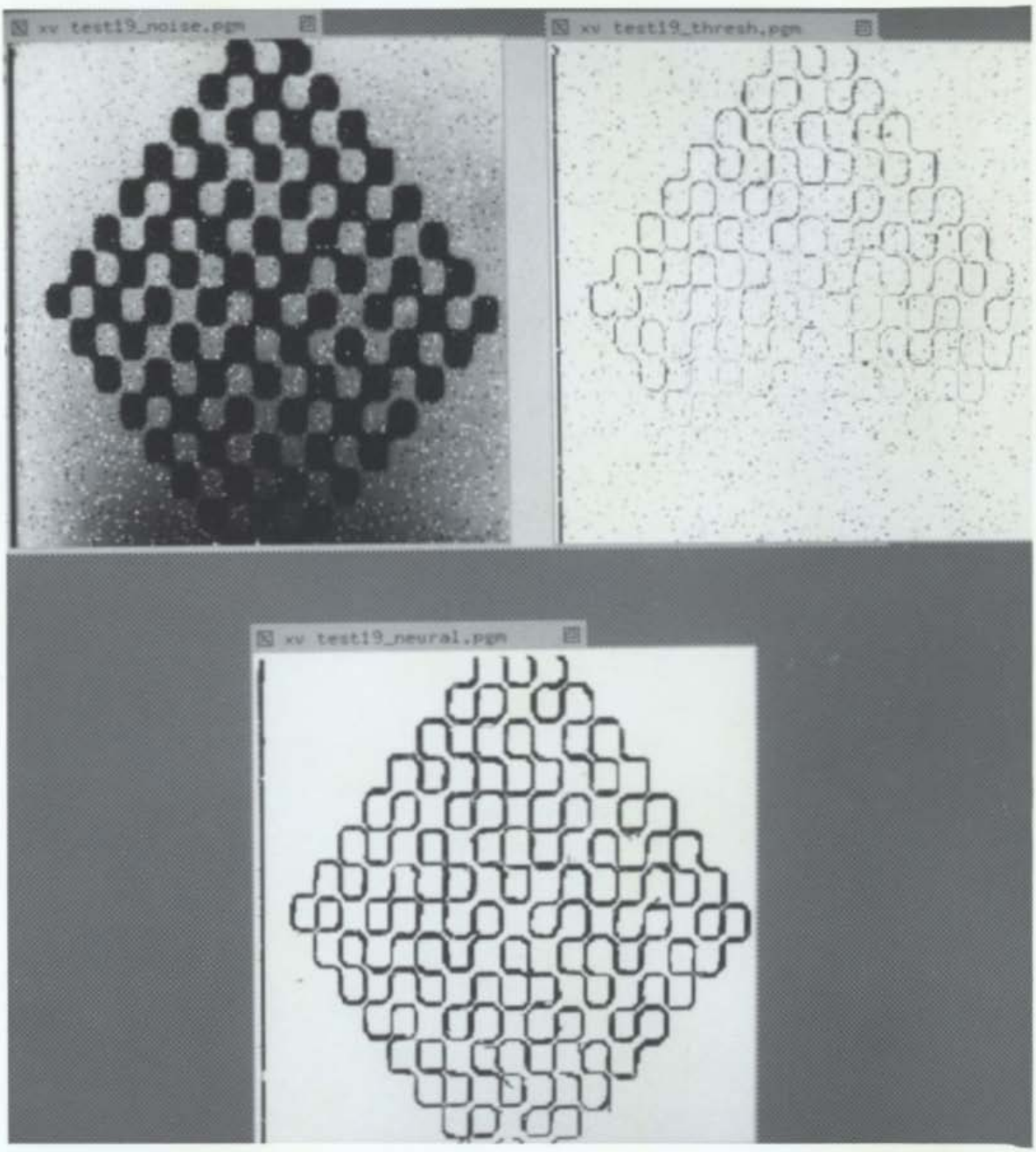

xv test12_noise.pgm
xv test12_thresh.pgm
xv test12_neural.pgm

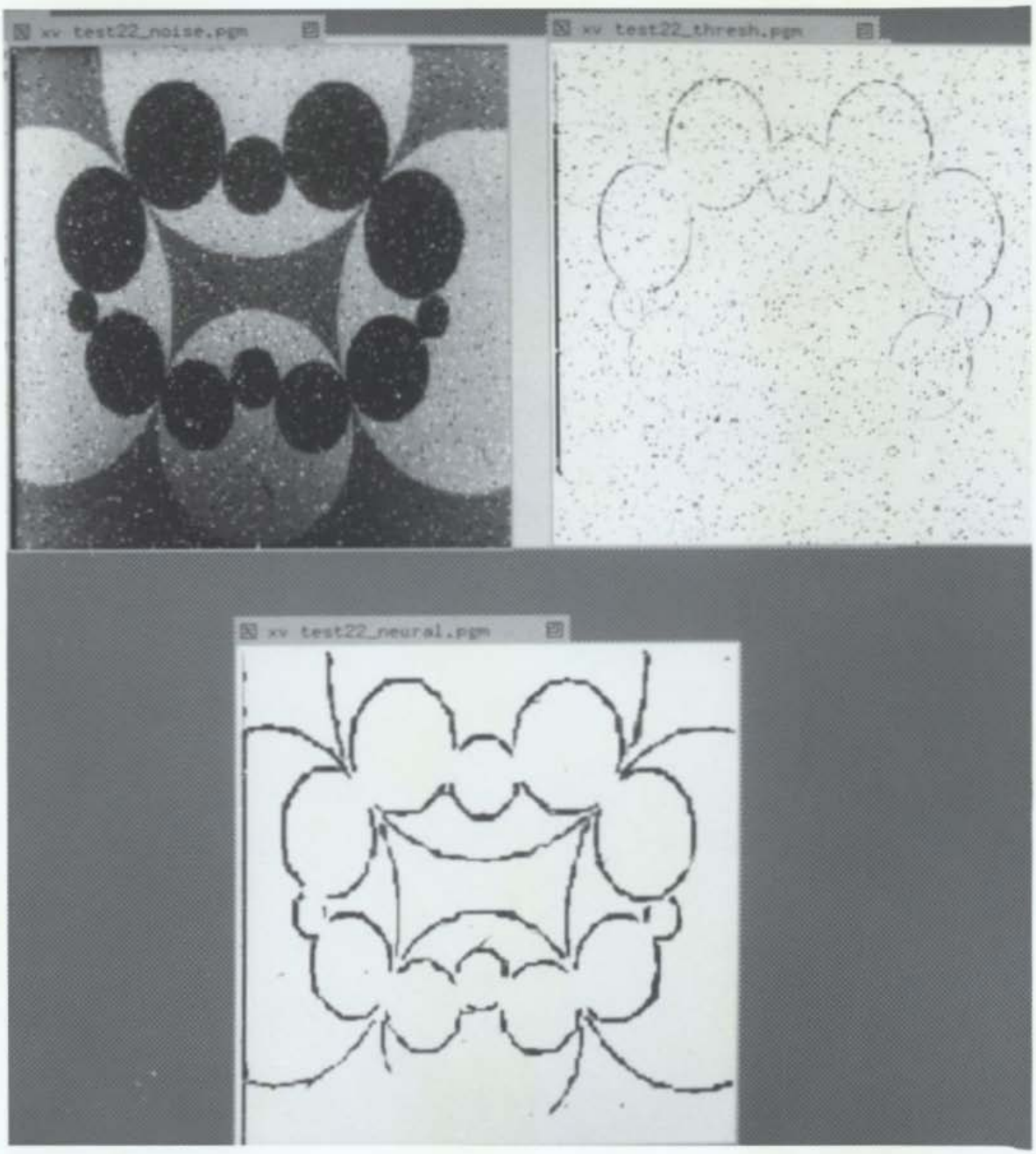

xv test13_noise.pgm
xv test13_thresh.pgm
xv test13_neural.pgm

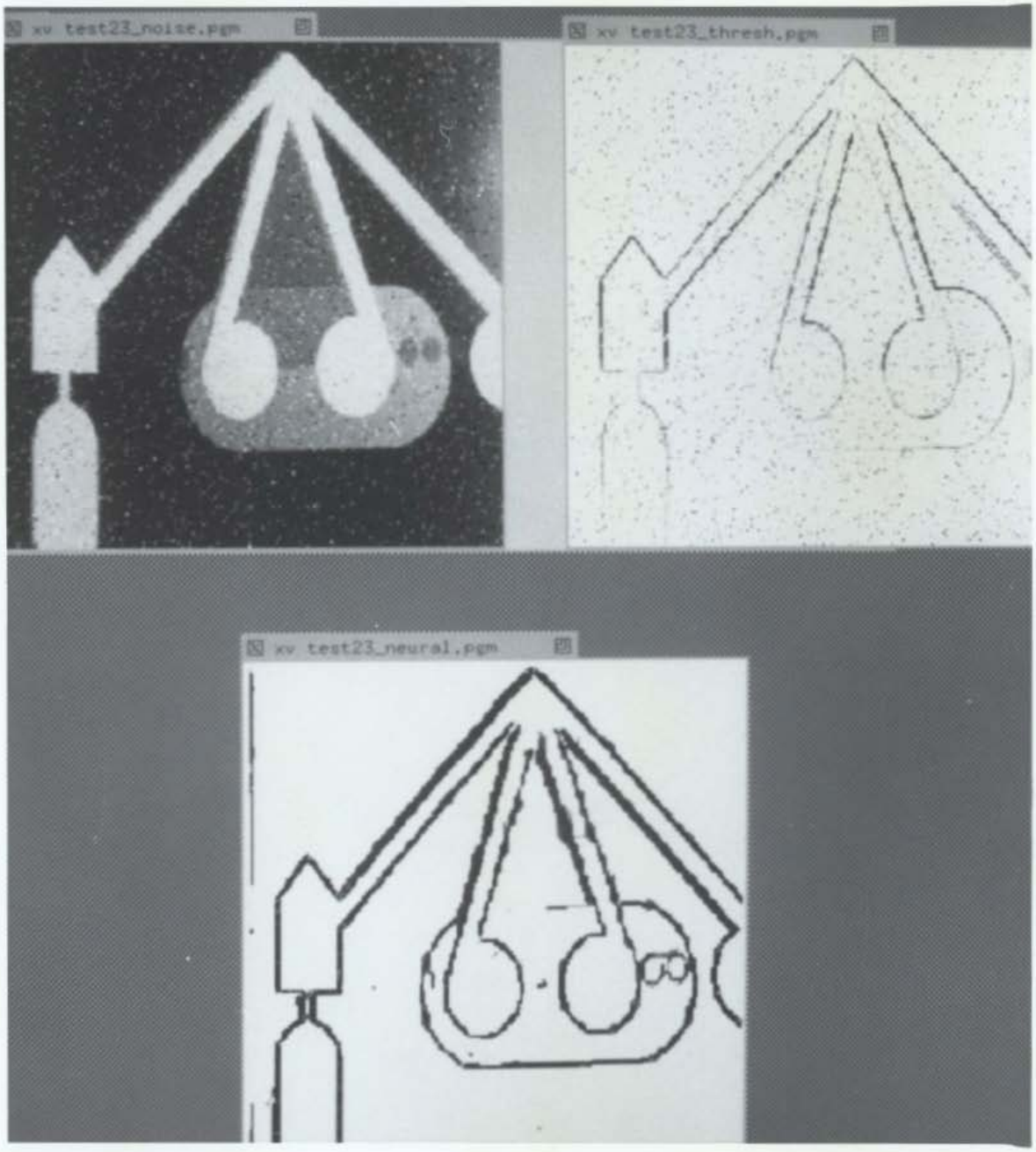

xv test14_noise.pgm
xv test14_thresh.pgm
xv test14_neural.pgm

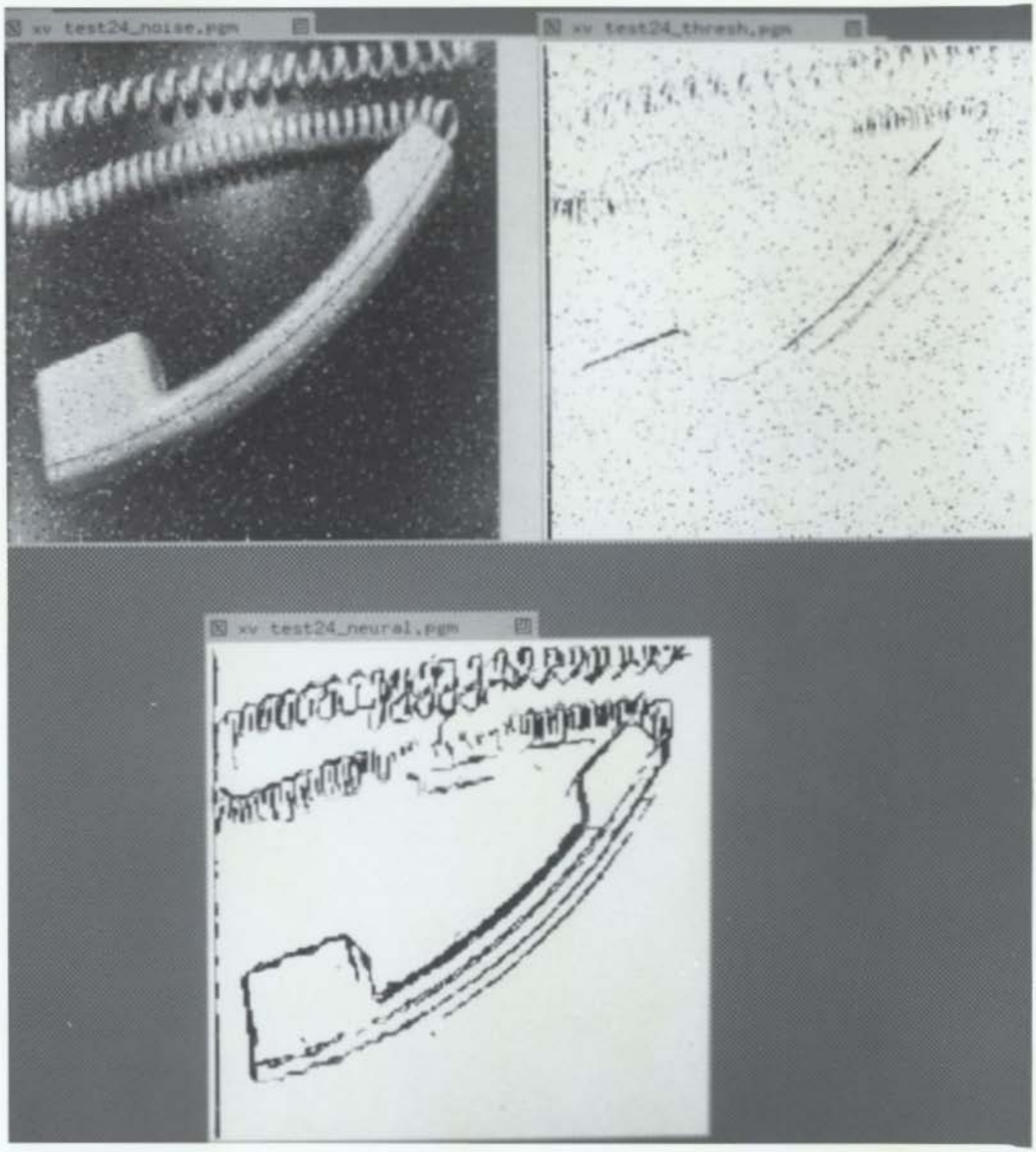

xv test15_noise.pgm
xv test15_thresh.pgm
xv test15_neural.pgm

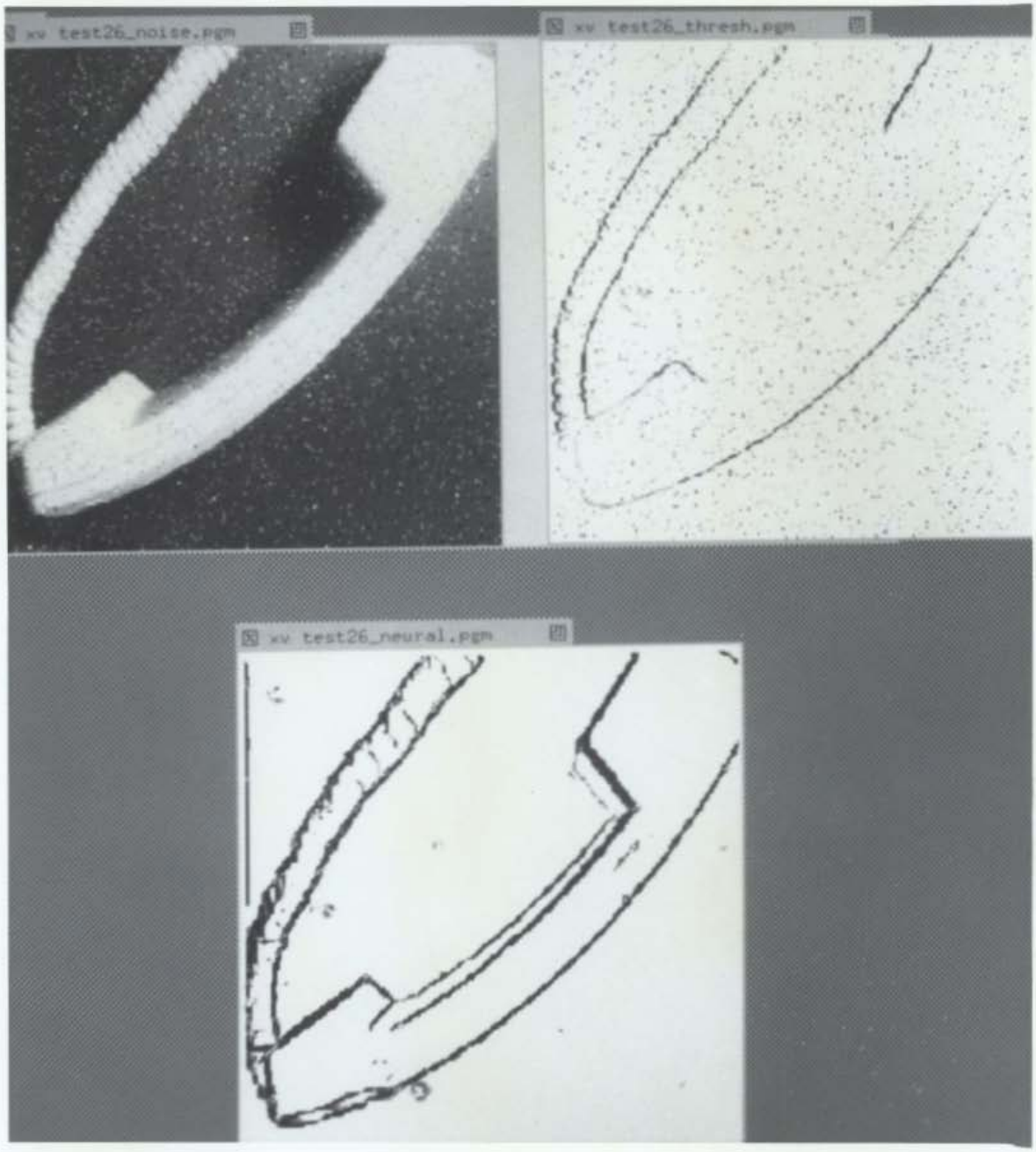

xv test17_noise.pgm
xv test17_thresh.pgm
xv test17_neural.pgm

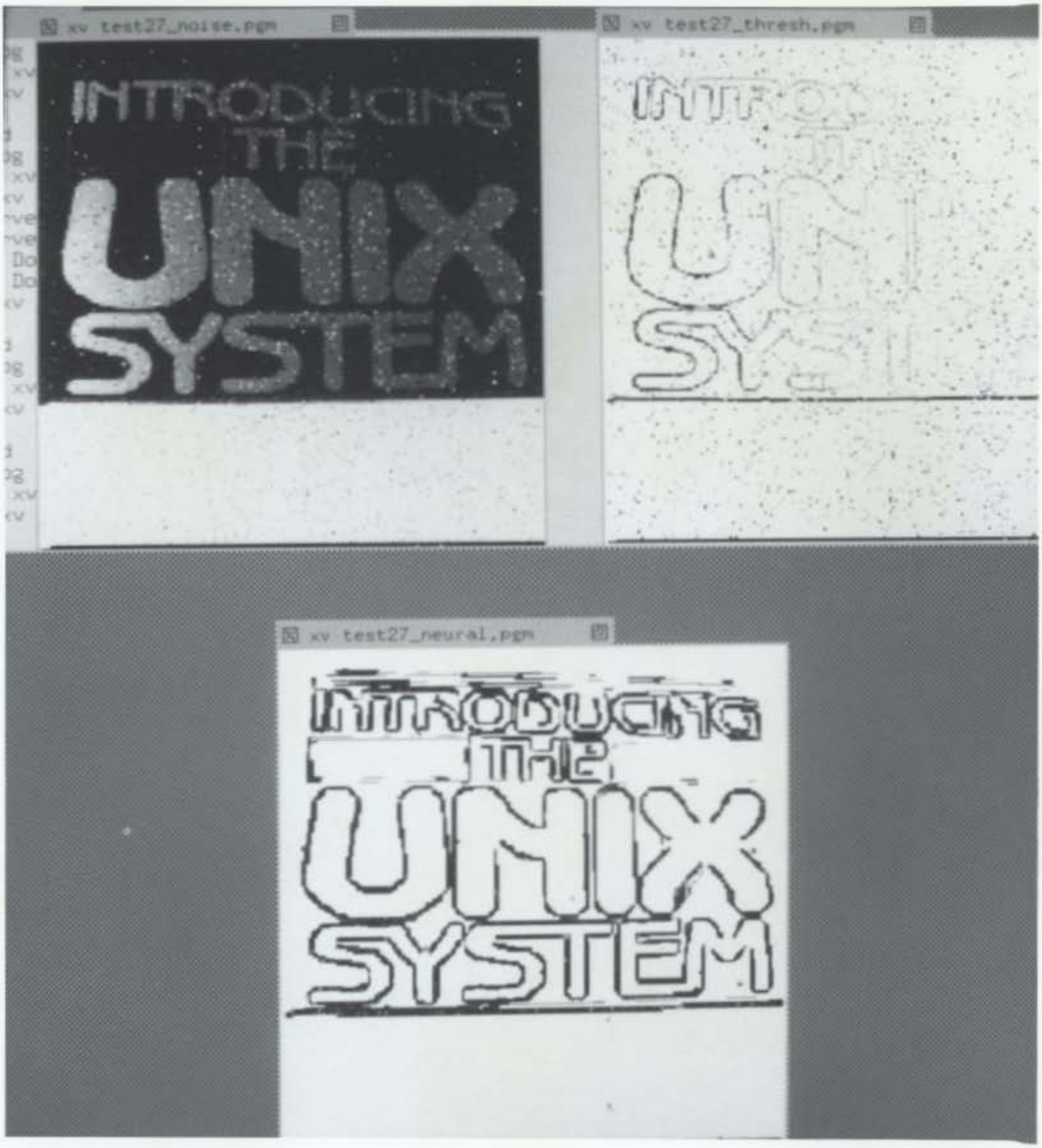

xv test19_noise.pgm
xv test19_thresh.pgm
xv test19_neural.pgm

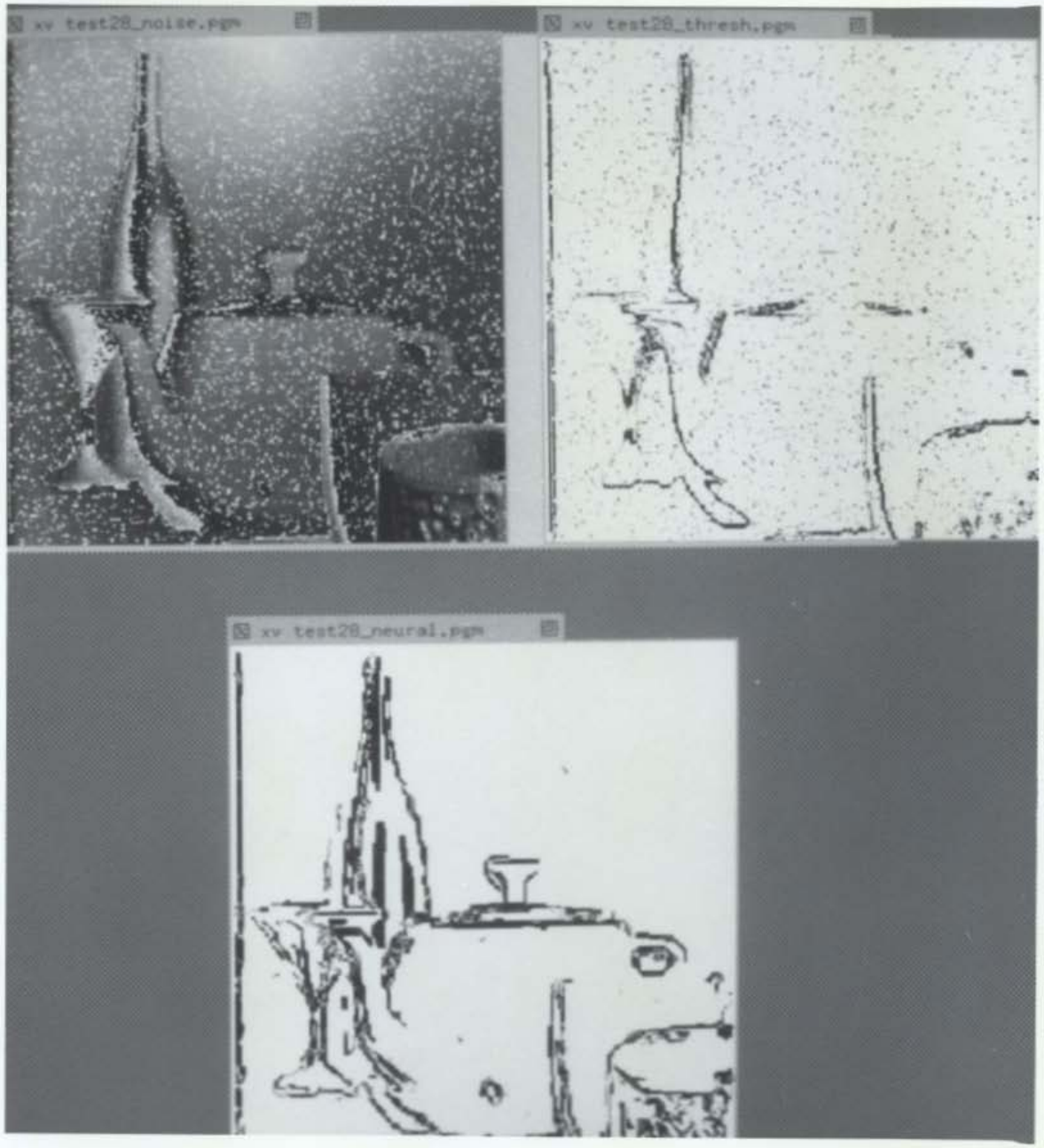

xv test22_noise.pgm
xv test22_thresh.pgm
xv test22_neural.pgm


xv test23_noise.pgm
xv test23_thresh.pgm
xv test23_neural.pgm


xv test24_noise.pgm
xv test24_thresh.pgm
xv test24_neural.pgm


xv test26_noise.pgm
xv test26_thresh.pgm
xv test26_neural.pgm


xv test27_noise.pgm
INTRODUCING
THE
UNIX
SYSTEM
xv test27_thresh.pgm
xv test27_neural.pgm
INTRODUCING
THE
UNIX
SYSTEM


xv test28_noise.pgm
xv test28_thresh.pgm
xv test28_neural.pgm