NEAR EAST UNIVERSITY

GRADUATE SCHOOL OF APPLIED AND SOCIAL SCIENCES

CHARACTER RECOGNITION USING NEURAL NETWORKS

Jehad I. S. Jabarin

MASTER THESIS

Department of Electrical & Electronic Engineering

Nicosia - 2005

Approval of the Graduate School of Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Electrical & Electronic Engineering

Examining Committee in charge:

Prof. Dr. Fakhraddin Mamedov, Committee Chairman, Dean of Engineering Faculty, NEU

Assist. Prof. Dr. Kadri Bürüncük, Committee Member, Vice Dean of Engineering Faculty, NEU

Assoc. Prof. Dr. Sameer M. Ikhdair, Committee Member, Electrical & Electronic Engineering Department, NEU

Assoc. Prof. Dr. Adnan Khashman, Supervisor, Chairman of Electrical & Electronic Engineering Department, NEU

Dedicated To My Mother

ACKNOWLEDGEMENTS

I could not have prepared this project without the generous help of my colleagues, professors, friends, and family.

My deepest thanks to my supervisor Assoc. Prof. Dr. Adnan Khashman for his help and answering questions.

I would like to express my gratitude to Prof. Dr. Fakhraddin Mamedov, Prof. Dr. Şenol Bektaş and Assoc. Prof. Dr. Sameer Ikhdair.

In addition, a special thanks to Mr. Tayseer Alshanableh and his family. Finally, I could never have prepared this thesis without the encouragement and support of my parents and dearest wife Isra Jabarin.

ABSTRACT

Attempting to endow a computer with the ability to recognize characters requires the deployment of an artificial neural network, a system modeled on the function and behavior of the human brain. If successful, the computer will 'think" for itself, meaning that it acquired some level of artificial intelligence throughout learning. Thus, when presented with a distorted version of a pattern, the network will correctly classify it.

The work within this thesis presents research into developing a neural network that can recognize alphabetical characters regardless of various degrees of pattern corruption (noise).

The objective of this work is to endow the reader with a stronger concept of the processes in character recognition by giving insight into its predecessor, image processing. In addition, to project the simplicity by which neural networks may be used for basic character recognition and to demonstrate how a simple pattern recognition can be designed implementing the back-propagation algorithm.

This thesis forms a base for further research of character recognition such as optical character recognition applied to scanners and faxes. These applications of character recognition must learn to deal with noise of imperfect data due to encountered problems with transmitting data.

CONTENTS

INTRODUCTION

Neural networks are becoming more popular as a technique to perform image processing and character recognition due to reported high recognition accuracy. They are also capable of providing good recognition with the presence of noise in which other methods normally fail. Neural networks with various architectures and training algorithms have been successfully applied for image processing and character recognition.

Chapter 1 introduces the idea that an intelligent, human-like machine dawned over a century ago. This marked the conception of the present-day artificial neural network. A neural network contains an arrangement of neurons arranged into layers, and weights in which the learning process stores acquired knowledge. Many learning processes evolved, including back-propagation, memory-based learning, Hebbian learning, competitive learning, and Boltzmann learning.

The second chapter presents image analysis as a process of discovering, identifying, and understanding patterns relevant to the performance of an image-based task. One of the principal goals of image analysis by computer involves endowing a machine with the capability to approximate the same ability in human beings. An automated image analysis system should attain the capability of exhibiting various degrees of intelligence. The concept of intelligence appears vague with reference to a machine because it cannot acquire the extent present in humans. However, the machine may learn basic characteristics of intelligent behavior. Even with these characteristics, image analysis systems may only perform for limited operational environments. Currently, endowing these systems with a level of intelligence close to that found in humans appears far-fetched; however, research in biological and computational systems continually uncovers new and promising theories.

Chapter 3 shows the widespread use of neural networks in image processing. The mid-eighties introduced the back-propagation learning algorithm for neural networks, making it feasible for the first time to train a non-linear network equipped with layers of hidden nodes. Since then, neural networks with one or more hidden layers can, in theory, train to perform virtually any task. In their 1993 review article on image segmentation, Pal and Pal predicted that neural networks would become widely applied in image processing [5]. This prediction proved correct.

In chapter 4, character recognition itself is also applied to neural networks. For character recognition, created matrices representing the letters of the alphabet form the input presented together with the target vector into a multi-layer, feed-forward, back-propagation neural network. The network trains with and without noise, and learns to produce a correct output by making error-based adjustments to the weights which includes a sigmoid transfer function. Large matrices may be manually compressed in order to decrease the size of total data, and idea of similar import employed in image processing.

Lastly, chapter 5 practically applies neural networks to the character recognition problem by using MatLab. A 2-layer, log-sigmoid neural network with an architecture of 400 input, 52 hidden, and 26 output nodes initialize the MatLab program. Two copies of the network are trained; the first on just ideal, and the second on both ideal and noisy vectors. Afterwards, both networks are tested on variant levels of noise and their degrees of accuracy when recalling characters is compared.

This thesis presents work aimed at:

- Exploring the applications of neural networks and image processing for character recognition
- Developing a character recognition system based on using neural networks
- Simulating the above neural network system using MatLab

1. ARTIFICIAL NEURAL NETWORKS

1.1 Overview

This chapter presents a general introduction to neural networks. History, definitions, common algorithms with emphasis on back-propagation, learning tasks, activation functions, and the neural network analogy to the brain will be discussed.

1.2 Neural Network Definition

More properly defined as an "artificial neural network" (ANN), a neural network represents an artificial prototype of the biological neural network known as the human brain. Biological neural networks function a great deal more complex than the mathematical model; however, referring to them simply as neural networks nowadays appears customary.

A neural network indicates an information-processing paradigm inspired by the way biological nervous systems, such as the brain, process information. The novel structure of the information processing system constitutes the key element of this paradigm. The structures consist of a large number of highly-interconnected processing elements, called neurons, working in unison to solve specific problems. Neural networks, like people, learn by example. A learning process configures the network for a specific application, such as pattern recognition or data classification. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons; true of the artificial neural networks as well.

- *Definition:*

  A machine designed to model the way in which the brain preferences a particular taste or function. The neural network usually implements using electronic components or simulates as software.

- *Simulated:*

  A mathematical model of the human brain that consists of many simple, highly interconnected processing elements organized into layers and operating in parallel devoid of central control. Processing units attain neural propensity for storing experiential knowledge and making it available for use through connections between the units, which store numeric weights in which the learning element modifies.

  It resembles the brain in two respects:

  1. The network acquires knowledge from its environment through a learning process.
  2. Acquired knowledge stores as the inter-neuron connection strength, known as synaptic weights.

Neural networks go by many aliases as shown in figure 1.1 below:

- Parallel Distributed Processing Models
- Connectives / Connectionism Models
- Adaptive Systems
- Self-Organizing Systems
- Neurocomputing
- Neuromorphic Systems

Figure 1.1 Neural Network Aliases

All the above names refer to this new form of information processing. The general and most commonly used term, "neural network", shall define the broad classes of artificial neural systems.

1.3 History of Neural Networks

Neural networks boast a broad history spanning from the present day all the way back to the 18th century. Out of the desire for human beings to create a machine which can mimic certain intelligible abilities of human beings, artificial neural networks originated. Table 1.1 below shows the neural network development history:

Present	↑	Present	interest explodes with conferences, articles, simulation, new companies, and government funded research
Late Infancy		1982	vast development in the training and learning of neural networks
Stunted Growth		1969 - 1981	funding cuts to neural network research as a result of excessive hype
Excessive Hype		Mid 1960's	potential of neural networks publicly exaggerated
Early Infancy		50's – 60's	new neural networks created and applied to real-life problems
Birth		1956	group project of the best in the field held in order to attempt to create intelligent machines
Gestation		1950's	Age of Computer Simulation, Neural Network terminology defined
Conception		1890 - 1949	simple neural network model created, researchers began to look to anatomy and physiology for clues about creating intelligent machines

Table 1.1 Neural Network Development History

1.4 Analogy to the Brain

The human nervous system may receive notice as a three stage system, as depicted in figure 1.2.

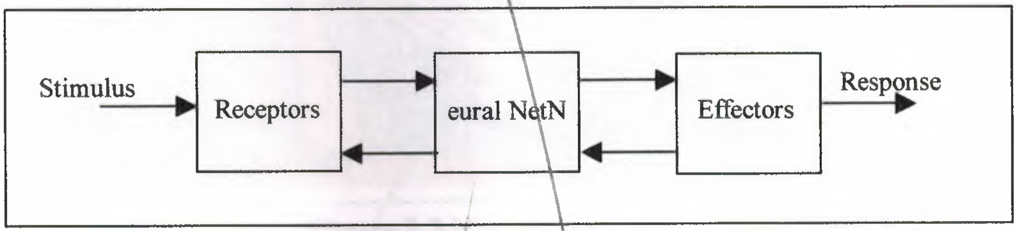


Figure 1.2 Block Diagram of the Nervous System

The neural network denotes the central element to the system, the brain, which continually receives information, perceives, and makes appropriate decisions. The above block diagram shows two sets of arrows. Those pointing from left to right indicate the forward transmission of information-bearing signals through the system. The receptors convert stimuli from the human body or the external environment into electrical impulses which convey information to the biological neural network, the brain. The effectors convert electrical impulses by the neural network into discernible responses as system outputs.

1.4.1 Natural Neuron

A neuron imitates a nerve cell with all of its processes. These cells make distinction between animal and plant matter, as plants lack nerve cells. Furthermore, the various classes of neurons in humans range up to one hundred. The wide variation relates to how restrictively a class receives definition. Neurons generally seem microscopic, but some neurons in the legs span as long as three meters. Figure 1.3 shows the type of neuron found in the retina.

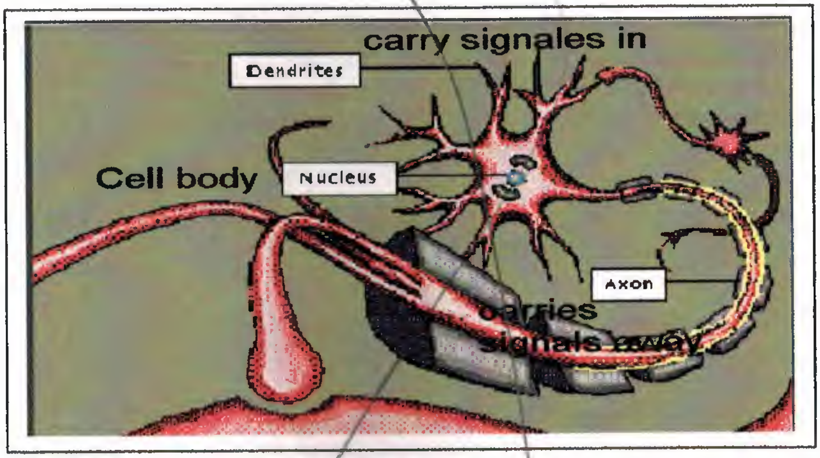


Figure 1.3 Natural Neuron

A bi-polar neuron forms one example. Its name implies two processes. The cell body contains the nucleus and one or more dendrites leading into the nucleus. These branching, tapering processes of the nerve cell, as a rule, conduct impulses toward the cell body. The axon corresponds to the nerve cell process that conducts the impulse type of neurons. This gives humans the functionality and vocabulary needed to make analogies.

1.4.2 Artificial Neuron

Figure 1.4 starts by copying the simplest element: the neuron. It is also referred to as an artificial neuron, , a processing element, or PE for short. Additionally, the word “node” also represents this simple building block, which a circle in figure 1.4 signifies.

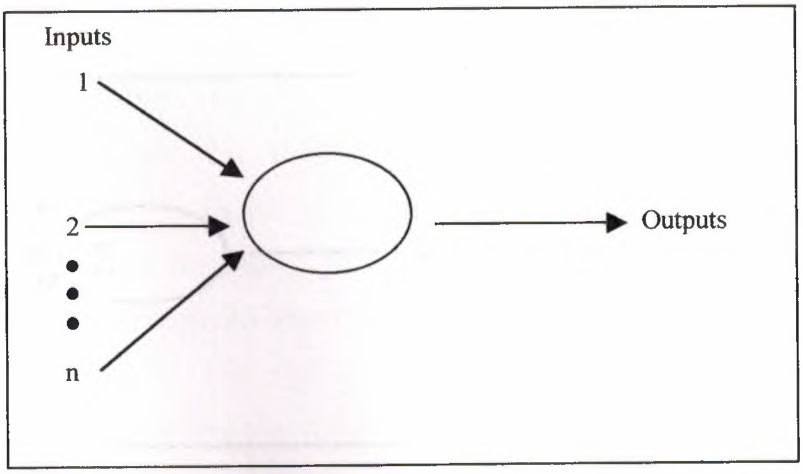


Figure 1.4 Artificial Neuron

The artificial neuron handles several basic functions: (1) evaluating the input signals and determining the strength of each one, (2) calculating the total for the combined input signals and comparing that total to some threshold level, and (3) determining the output.

Input and Output

Just as many inputs to a neuron exist, so many input signals should as well. All of them should meet simultaneously at the neuron. In response, a neuron either "fires", or "does not fire" depending on some threshold level. The artificial neuron only allows a single output signal, just as present in a biological neuron. The network possesses many inputs, yet only one output.

Weighting Factors

Each input receives a relative weight which affects its impact. Figure 1.5 presents an artificial neuron containing a single-node with weighted inputs.

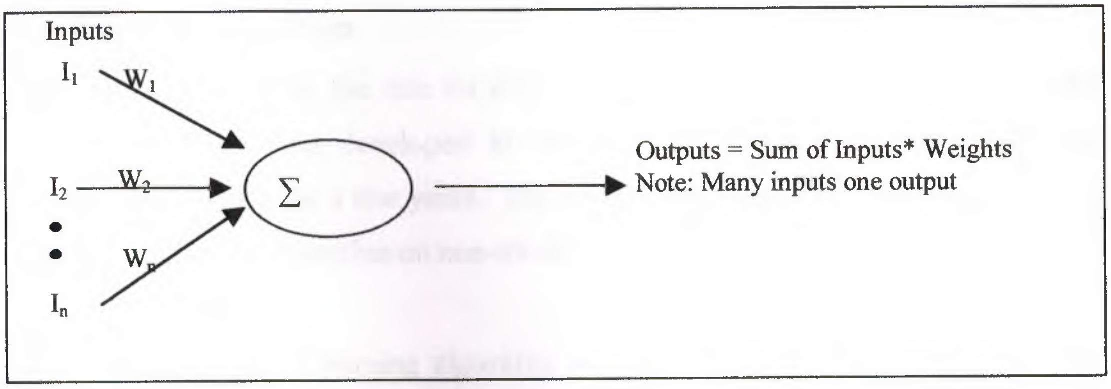

Inputs
I_1
W_1
I_2
W_2
W_n
I_n
Σ
Outputs = Sum of Inputs* Weights
Note: Many inputs one output

Figure 1.5 A Single-Node Artificial Neuron

This compares to the varying synaptic strengths of the biological neurons. Some inputs become more important than others in the way that they combine to produce an impulse.

1.5 Model of a Neuron

The neuron represents the basic processor in neural networks. Each neuron gives one output, which generally relates to the state of the neuron, meaning its activation, which may fan out to several other neurons. Each neuron receives several inputs over these connections, called synapses. The inputs signify the activations of the neuron. This computes by applying a threshold function to this product. Figure 1.6 shows an abstract model of a neuron.

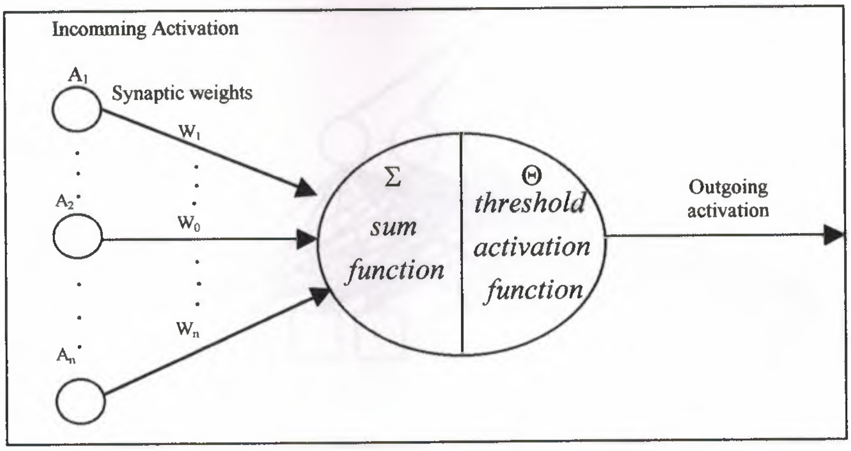


Figure 1.6 Diagram of Abstract Neuron Model

1.6 Back-Propagation

Back-propagation holds the title for most popular method for learning of the multi-layer network. First developed in 1886 by Rumelhart Hinton and Williams, received little notice for a few years. The reason may correlate to the computational requirements of the algorithm on non-trivial problems.

The back-propagation learning algorithm works well on multi-layer, feed-forward networks, using gradient descent in weight space to minimize the output error. It converges to a locally optimal solution, and proves successful in a variety of applications. As with all hill-climbing techniques, however, no guarantee assures that it will find a global solution. Furthermore, its convergence often proves slow.

1.6.1 Back-Propagation Learning

Suppose a problem requires the constructing of a network, a two-layer network will constitute as the starting point. Ten attributes describe each example, thus requiring ten input units. Figure 1.7 shows a network with four hidden units, which proves useful for the particular problem.

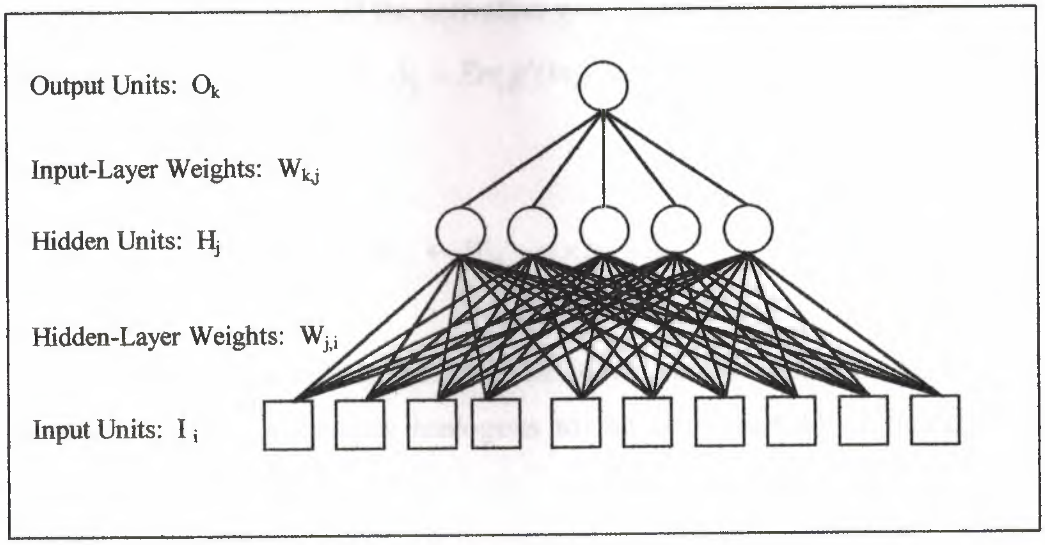

Output Units: O_k
Input-Layer Weights: $W_{k,j}$
Hidden Units: H_j
Hidden-Layer Weights: $W_{j,i}$
Input Units: I_i

Figure 1.7 A Two-Layer Feed-Forward Network

Example inputs present into the network, and if the network computes an output vector that matches the target, no additional steps take place. If an error appears, indicating a difference between the output and target, then weights adjust to reduce this error. The trick of back-propagation consists of assessing the blame for an error and dividing it among the contributing weights. In multi-layer networks, many weights connect each input to an output and each of these weights contributes to more than one output.

The network attempts to minimize the error between each target output and the output actually computed. At the output layer, the weight update rule compares to the rule for the perceptron, however, with the exception of two differences: the activation of the hidden unit a_j replaces the input value and the rule contains a term for the gradient of the activation function. If Err_i represents the error (T_i-O_i) at the output node, then the weight update rule for the link from unit j to unit i calculates by:

$$W_{ji} \leftarrow W_{ji} + \alpha \times Err_i \times g'(in_i) \tag{1.1}$$

where g' = the derivative of the activation g, it then becomes convenient to define a new error term Δ_i for which $\Delta_i = Err_i g'(in_i)$ defines the output node. The update rule then becomes:

$$W_{ji} \leftarrow W_{ji} + \alpha \times a_j \times \Delta_i \tag{1.2}$$

For updating the connections between the input and the hidden units, it becomes necessary to define a quantity analogous to the error term for output node. The propagation rule:

$$\Delta_j = g'(in_j)\sum_i W_{ji}\Delta_i \tag{1.3}$$

Now the weight update rule for the weights between the inputs and the hidden layer almost matches to the update rule for the output layer:

$$W_{kj} \leftarrow W_{kj} + \alpha \times I_k \times \Delta_j \tag{1.4}$$

```
**Function** Back-Prop-UPDATE (network, examples, $\alpha$) returns a network with modified weights.
**Inputs:** network, a multi-layer network
Examples, asset of input/output pairs $\alpha$, the learning rate.
**Repeat**
For each e in example do
$O \leftarrow TUN - NETWORK(network, I^e)$
$Err^e \leftarrow T^e - O$
$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i^e \times g'(in_i)$
For each subsequent layer in network do
$\Delta_j \leftarrow g'(in_j)\sum_i W_{j,i}\Delta_j$
$W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$
**end**
**end**
until network converges
**return** network
```

Figure 1.8 Back-Propagation Algorithm for Updating Weights

Back-propagation provides a way of dividing the calculation of the gradient among the unit to calculate the change in each weight by the unit to the attached weight using only local information.

The sum of squared errors over the output values used:

$$E = \frac{1}{2}\sum_i (T_i - O_i)^2 \tag{1.5}$$

where O_i, is a function of the weights for general two-layer network, written as follows:

$$E(W) = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} a_j))^2 \qquad (1.6)$$

$$E(W) = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} g(\sum W_{k,j} I_k)))^2 \qquad (1.2)$$

1.7 Learning Processes

Learning presents a process by which the free parameters of a neural network adapt through a process of stimulation by the environment embedded in the network. Determining the type of learning depends on the manner in which the parameter change takes place.

This definition of the learning process implies the following sequence of events:

- An environment simulates a neural network.
- The neural network undergoes changes in its parameters as a result of this stimulation.
- The neural network responds in a new way to the environment because of the occurred changes in its internal structure.

A learning algorithm defines a prescribed set of well-defined rules for the solution of a learning problem.

Basically, learning algorithms differ from each other in the way in which the adjustment to a synaptic weight of neurons formulates. Another factor to consider includes the manner in which a neural network comprises a set of interconnected neurons. A learning paradigm refers to a model of the environment in which the neural network operates.

1.7.1 Memory-Based Learning

In memory-based learning, a large memory of correctly classified input-output examples explicitly stores all or most of the past experiences. The formula results as follows:

$$\{(x_i, d_i)\}_i^N = 1 \tag{1.8}$$

where x_i denotes an input vector and d_i denotes the corresponding desired response.

1.7.2 Hebbian Learning

When an axon of cell A approaches close enough to excite a cell B, it repeatedly or persistently takes part in firing it. Some growth processes, or metabolic changes take place in one or both cells:

1. If two neurons on either side of a synapse selectively activate simultaneously, then the strength of that synapse increases.
2. If two neurons on either side of a synapse activate asynchronously, then that synapse either weakens or becomes eliminated.

The following constitute four key mechanisms that characterize a Hebbian synapse:

1. Time-dependent mechanism. This mechanism refers to the fact that the modification in a Hebbian synapse depends on the exact time occurrence of the presynaptic and postsynaptic signals.
2. Local mechanism. By its nature a synapse acts as the transmission site where information-bearing signals, which represent ongoing activity in the presynaptic and postsynaptic units, exist in spatiotemporal contiguity.

3. Interactive mechanism. The occurrence of a change in the Hebbian synapse depends on signals on both sides of the synapse.
4. Conjunctional or co-relational mechanism. One interpretation of Hebb's postulate of learning comments that the condition for a change in synaptic efficiency presents the conjunction of presynaptic and posynaptic signals.

1.7.2.1 Synaptic Enhancement and Depression

The conception of a Hebbian modification recognizes that positively correlated activity produces synaptic weakening; synaptic for depression may also exist as a non-interactive type. The classification of modifications such as Hebbian, anti-Hebbian, and non-Hebbian, according to this scheme, increases its strength when these signals either uncorrelate or negatively correlate.

1.7.2.2 Mathematical Models of Hebbian Modifications

To formulate Hebbian learning in mathematical terms, consider a synaptic weight W_{kj} of neuron k with presynaptic and postsynaptic signals denoted by x_j and y_k respectively. The adjustment applied to the synaptic weight W_{kj}, at time step n, expressed in the general form:

$$\Delta w_{kj}(n) = f(y\ \ (n), x_j(n)) \tag{1.9}$$

where the signals $x_j(n)$ and $y_k(n)$ often treated as dimensionless.

1.7.2.3 Hebbian Hypothesis

The simplest form of Hebbian learning:

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n) \tag{1.10}$$

where η represents a positive constant that determine the rate of learning, it clearly emphasizes the co-relational nature of a Hebbian synapse, sometimes referred to as the activity product rule (see figure 1.9).

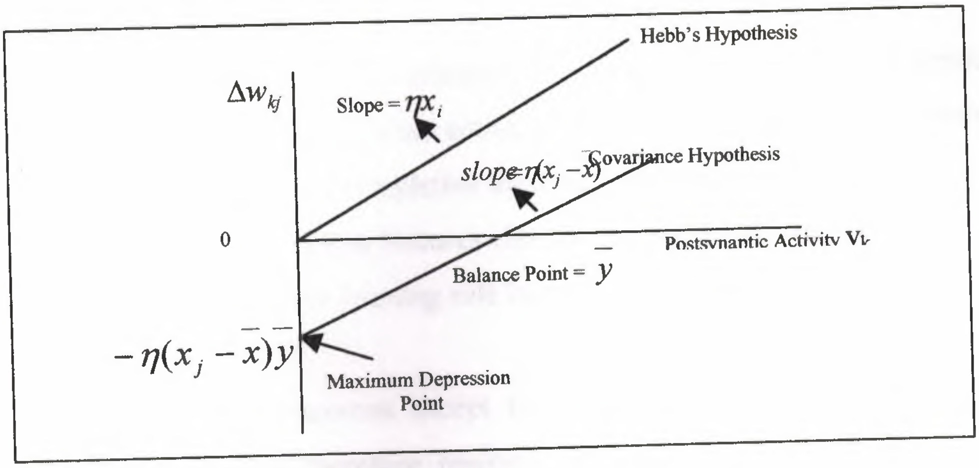


Figure 1.9 Illustration of Hebb's Hypothesis and the Covariance Hypothesis.

With the change Δw_{kj} plotted versus the output signal y_k, exponential growth finally drives the synaptic connection into saturation. At that point no information will store in the synapse and becomes selectivity lost.

Covariance hypothesis: One way of overcoming the limitation of Hebb's hypothesis includes using covariance hypothesis introduced by Sejnowski. In this hypothesis, the departure of presynaptic and postsynaptic signals from their respective values over a certain time interval replaces the presynaptic and postsynaptic. Let $\bar{x}$ and $\bar{y}$ denote the time average values of the presynaptic signal x_j, and postsynaptic signal y_k respectively according to the covariance hypothesis. The adjustment applied to the synaptic weight w_{kj}:

$$\Delta w_{kj} = \eta(x_j - \bar{x})(y_k - \bar{y}) \qquad (1.11)$$

where η represents the learning rate parameter and the average values x and y constitute presynaptic and postsynaptic thresholds. This determines the sign of synaptic modification.

1.7.3 Competitive Learning

In competitive learning, as the name implies, the output neurons of a neural network compete among themselves to become active and fire. The several output neurons may activate simultaneously in completive learning; yet only a signal output neuron remains active at any time. These features classify a set of input patterns. The three basic elements to a competitive learning rule include:

- A set of identical neurons except for some randomly distributed synaptic weight and which therefore respond differently to a given set of input patterns
- A limit imposed on the strength of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of input.

In the simplest form of competitive learning, the neural network contains a single layer of output neurons, each of which fully connects to the input nodes. The network may include feed-back connection among the neurons as indicated in figure 1.10.

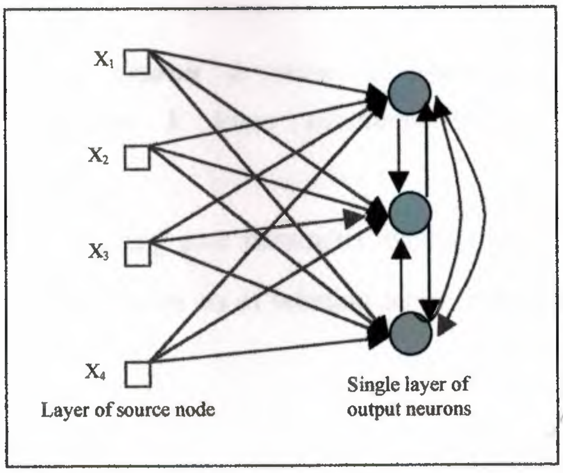


Figure 1.10 Feed-back Connections among the Neurons

For a neuron *k* to act as the winning neuron, it induces local field v_k for a specified input pattern. *X* must denote the largest among all the neurons in the network. The output signal y_k of winning neurons *k* sets equal to one. The output signals of all the neurons that lose the competition set equal to zero. Thus:

$$y_k = \begin{cases} 1\, if v_k > v_j forallj, j \neq k \\ o \qquad otherwise \end{cases} \tag{1.12}$$

The induced local field v_k represents the combined action of all the forward and feedback inputs to neuron *k*.

Let w_{kj} denote the synaptic weight connecting input node *j* to neuron *k*. Suppose that each neuron allots a fixed amount of synaptic weight, which distributes among its input node as the following:

$$\sum_j w_{kj} = 1 \text{ For all k} \tag{1.13}$$

The change Δw_{kj} applied to synaptic weight w_{kj} :

$$w_{kj} = \begin{cases} \eta(x_j - w_{kj}) if\ neuron \quad k \quad wins \quad the competition \\ 0 \qquad if\ neuron \quad k \quad loses \quad the competition \end{cases} \tag{1.14}$$

where η represents the learning rate parameter which creates the overall effect of moving the synaptic weight vector w_k of winning neurons *k* toward the input pattern *x*.

1.7.4 Boltzmann Learning

The Boltzmann learning rule a stochastic learning algorithm derived from ideas rooted in statistical mechanics, received its name in honor of Ludwig Boltzmann. In a Boltzmann machine, the neurons constitute a recurrent structure and operate in a binary manner since they either hold an on-state denoted by +1, or an off state determined by the particular states occupied by the individual neurons of the machine as shown by:

$$E = -\frac{1}{2}\sum_{j}\sum_{k} w_{kj} x_k x_j \qquad (1.15)$$

where x_j constitutes the state of neuron j and w_{kj} signifies the synaptic weight connecting neuron j to neuron k, the fact that $j \neq k$ means simply that none of the neurons in the machine possess self feed-back. The machine operates by choosing a neuron at random, for example neuron k at some step of the learning process then flipping the state of neuron k from state x_k at some temperature T with probability:

$$p(x_k \rightarrow -x_k) = \frac{1}{1+\exp(-\Delta E_k / T)} \qquad (1.16)$$

where ΔE_k indicates the energy change resulting from such a flip, notice that T designates not physical temperature but rather a pseudo temperature.

The neurons of a Boltzmann machine partition into two functional groups: visible and hidden. The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely. Two modes of operation to consider:

- Clamped condition in which the visible neurons all clamp onto specific states determined by the environment.

- Free running condition in which all the neurons, visible and hidden, operate freely.

According to the Boltzmann learning rule, the change Δw_{kj} applied to the synaptic weight w_{kj} from neuron j to neuron k by:

$$\Delta w_{kj} = \eta(p_{\underset{kj}{+}} - p_{\underset{kj}{-}}),\ j \neq k \tag{1.17}$$

where η identifies a learning rate parameter, note that both $p_{\underset{kj}{+}}$ and $p_{\underset{kj}{-}}$ range in value from -1 to $+1$.

1.8 Learning Tasks

Identification of six learning tasks that apply to the use of the neural network in different forms shall take place.

a. Pattern Association

An associative memory presents a brain-like, distributed memory that learns by association. Acknowledged as a prominent feature of human memory, even Aristotle used association for basic operations. The operation of an associative memory involves two phases:

- Storage phase, which refers to the training of the network in accordance with $x_k \rightarrow y_{k,}\quad k = 1,2,3....q$
- Recall phase, which involves the retrieval of a memorized pattern in response to the presentation of a noisy or distorted version of a key pattern to the network.

b. **Pattern Recognition**

Humans own the capability for pattern recognition. They receive data from the world around via senses and possess the ability to recognize the source of the data. Pattern recognition gained a formal definition as the process whereby a received pattern/signal receives designation to one of a prescribed number of classes or categories.

c. **Function Approximation**

The third learning task of interest—function approximation.

d. **Control**

Another learning task, the control of a plant, may process within a neural network. A process or critical art of a system requiring maintenance in a controlled condition defines a plant.

e. **Filtering**

The term filter often refers to an algorithm device used to extract information about a prescribed quantity of interest from a set of noisy data.

f. **Beam-forming**

The term Beam-forming characterizes a spatial form of filtering and employs to distinguish between the spatial properties of a target signal and background noise. A beam-former identifies a device used to do beam-forming.

1.9 Activation Functions

Generally, some form of non-linear function correlates to the threshold function. One simple non-linear function, the step function, proves one of the most suitable for discrete neural networks. One variant of the step function:

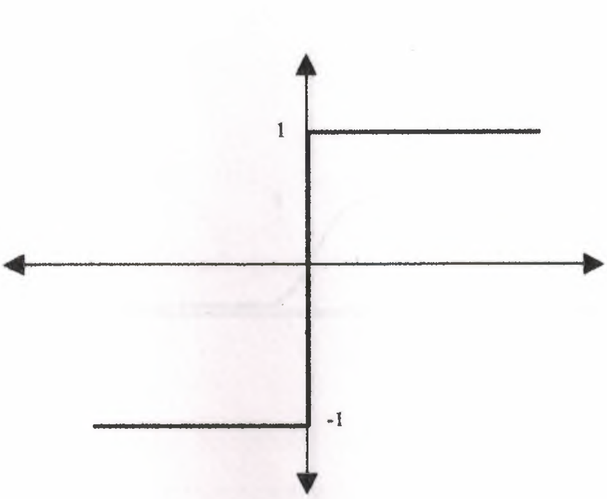


Figure 1.11 Step Function

$$f(x)=\begin{cases}1 & x>0\\ f^{\prime}(x) & x=0\\ -1 & x<0\end{cases} \tag{1.18}$$

where *f' (x)* refers to the previous value of *f(x)* in which the activation of the neuron will not change and *x* specifies the summation over all the incoming neuron) of the product of the incoming neuron's activation, and the connection:

$$X=\sum_{i=0}^{n} A_i w_i \tag{1.19}$$

for which *A* indicates the vector of incoming neurons and *w* the vector of synaptic weights connecting the incoming neurons to the examined neurons. One more appropriate to analog includes the sigmoid, or squashing, function; illustrated in figure 1.12.

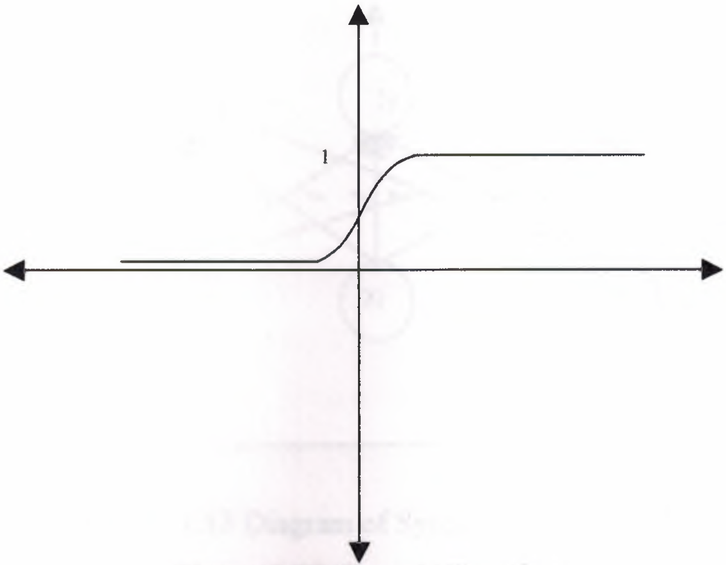


Figure 1.12 Sigmoid Functions

$$f(x)=\frac{1}{1+e^{-x}} \tag{1.20}$$

Another popular alternative:

$$f(x)=\tanh(x) \tag{1.21}$$

A non-linear activation function deems very important, especially when employing a multi-layer network because non-linear activation functions applied to multi-layer networks compute identical to single-layer networks.

1.9.1 Artificial Neural Network

Synapses store all of the knowledge that a neural network possesses. Figure 1.13 shows the weights of the connections between the neurons.

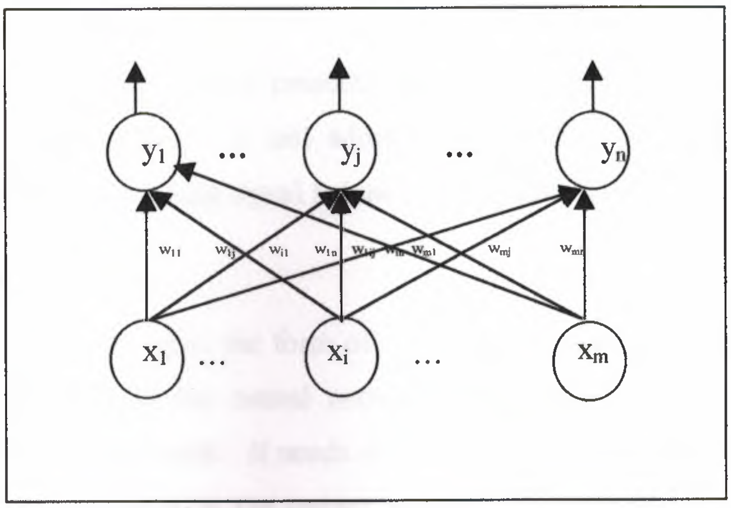


Figure 1.13 Diagram of Synapse Layer Model

However, the network acquires that knowledge during training during which pattern associations presents to the network in sequence, and the weights adjust to capture this knowledge. This weight adjustment scheme designates as the learning law. Hebbian learning became one of the first learning methods formulated.

Donald Hebb, in his organization of behavior formulated the concept of correlation learning. This formed the idea that the weight of a connection adjusts based on the values of the neurons it connects:

$$\Delta w_{ij} = \alpha a_i a_j \tag{1.22}$$

were α represents the learning rate, a_i indicates the activation of the "*i*"th neuron in one neuron layer, a_j denotes the activation of the "*i*"th neuron in another layer, and w_{ij} symbolizes the connection strength between the two neurons. The signal Hebbian Law identifies a variant of this learning rule:

$$\Delta w_{ij} = -w_{ij} + s(a_i)s(a_j) \tag{1.23}$$

where *s* indicates a sigmoid function

1.9.2 Unsupervised Learning

The unsupervised learning method presents one method of learning. In general, unsupervised learning methods do not adjust weights based on comparison with some target output, or a teaching signal fed into the weight adjustments.

1.9.3 Supervised Learning

In many models, learning takes the form of supervised training. Input pattern one after the other present to the neural network, and the recalled output pattern compares with the desired result. It needs some way of adjusting the weights which takes into account any error in the output pattern. An example of a supervised learning law-the Error Correction Law:

$$\Delta w_{ij} = \alpha a_i \lfloor c_j - b_j \rfloor \tag{1.24}$$

where α indicates the learning rate, a_i the activation of the “*i*”th neuron, b_j the activation of the “*i*”th neuron in the recalled pattern, and c_j the deired activation of the “*i*”th neuron.

1.9.4 Reinforcement Learning

Another learning method, known as reinforcemnet learing, fits into the general category of supervised learning. However, its formula differs from the error correction formula just presented. This type of learning corresponds to supervised learning except that each ouput neuron gets an error value. Only one error value computes for each ouput neuron. Thus, the weight adjustment formula:

$$\Delta w_{ij} = \alpha [v - \Theta j] e_{ij} \tag{1.25}$$

where α represents the learning rate, v the single value indicting the total error of the output pattern, and Θ the threshold value for the “*i*”th output neuron. It becomes necessary to spread out this generalized error for the “*i*”th output neuron to each of the incoming *i* neurons, a value representing the eligibility of the weight for updating. This computes as:

$$e_{ij} = \frac{d \ln g_i}{dw_{ij}} \tag{1.26}$$

where g_i denotes the probability of the correct output given the input from the "*i*"th incoming neuron. The probability function results from a heuristic estimate and manifests itself differently from specific model to specific model.

1.10 Back-Propagation Model

The back-propagation model applies to a wide class of problems and now holds the title of the most pre-dominant supervised training algorithm. Supervised learning requires the availability of a set of good pattern associations to train with. Figure 1.14 presents the back-propagation model.

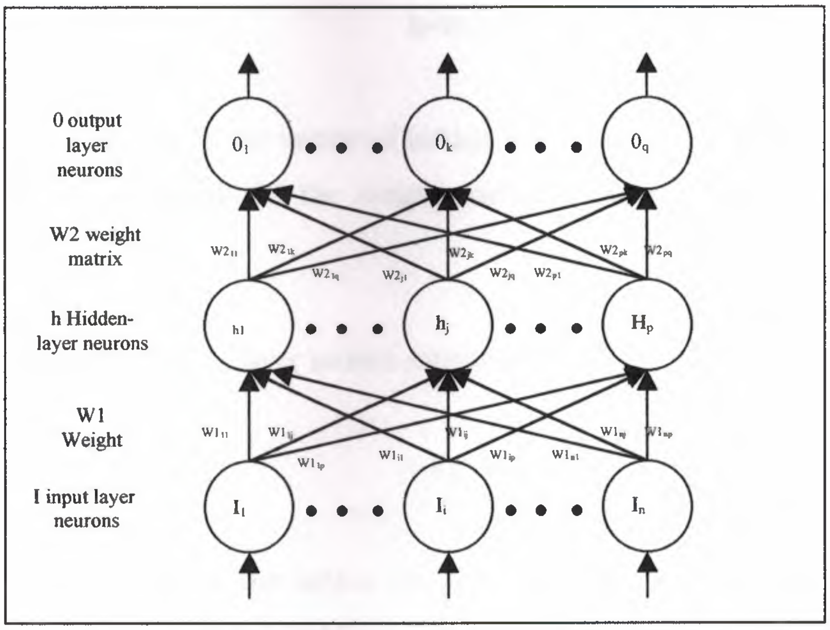


Figure 1.14 Diagram of Back-Propagation Topology

It comprises two layers of neurons: an input layer considered only as interface as it requires no caculation, a hidden layer, and an output layer. In addition, there are two layers of synaptic weights. There includes a learning rate term, α, in the subsequent formulas indicating how much of the weight changed to effect on each pass—typically a number between 0 and 1. Also, a momentum term, Θ, indicating how much a previous weight change should influence the current weight change. Finally, a term indicating the amount of tolerable error.

1.10.1 Back-Propagation Algorithm

Random values between –1 and +1 assign to the weghts between the input and hidden layers, the weights between the hidden and output layers, and the threshold for the hidden layer and output layer neurons train the network by preforming the following procedure for all pattern pairs:

Forward Pass

1. Compute the hidden layer neuron activations:

$$h=F(iW1) \tag{1.27}$$

   where *h* represents the vector of hidden layer neurons, *i* the vector of input layer neurons, and *W1* the weight matrix between the input and hidden layers.

2. Compute the output layer neuron activation:

$$O=F(hW2) \tag{1.28}$$

   where *o* represents the output layer, *h* the hidden layer, *W2* the matrix of synapses connecting the hidden and output layers, and *F* a sigmoid activation function—the logistic function is given by equation 1.20.

Backward Pass

3. Compute the output layer error (the difference between the target and the observed output):

$$d = O(1-O)(O-t) \tag{1.29}$$

where d corresponds to the vector of errors for each output neuron, o the output layer, and t the target correct activation of the output layer.

4. Compute the hidden layer error:

$$e = h(1-h)W2d \tag{1.30}$$

where e symbolizes the vector of errors for each hidden layer neuron.

5. Adjust the weights for the second layer of synapses:

$$W2 = W2 + \Delta W2 \tag{1.31}$$

where $\Delta W2$ indicates a matrix representing the change in matrix $W2$, computed as follows:

$$\Delta W2_t = \alpha hd + \Theta \Delta W2_{t-1} \tag{1.32}$$

where α indicates the learning rate and Θ the momentum factor used to allow the previous weight change to influence the weight change in this time period. This does not mean that time incorporates into the mode. It only indicates the adjustment of weights.

6. Adjust the weights for the first layer of synapses:

$$W1 = W1 + W1_t \tag{1.33}$$

$$W1_t = \alpha ie + \Theta \Delta W1_{t-1} \tag{1.34}$$

Repeat steps 1 through 6 on all pattern pairs until the output layer error (vector d) contains a value within the specified tolerance for each pattern and for each neuron.

Recall

Present this input to the input layer of neurons of the back-propagation net:

- Compute the hidden layer activation:

$$h = F(W1i) \tag{1.35}$$

- Compute the output layer:

$$O = F(W2h) \tag{1.36}$$

where vector o denotes the recalled pattern

1.10.2 Strengths and Weaknesses

The back-propagation network boasts the ability to learn any arbitrarily complex, and non-linear mapping due to the introduction of the hidden layer. It also possesses a capacity much grater than the dimensionality of its input and output layers, however, not always true of all neural network models.

However, back-propagation may involve extremely long and potentially infinite training time. If there a strong relationship exists between input and outputs, and results within a relatively acceptable time, then this algorithm proves ideal.

1.11 Summary

This chapter provides the reader with a brief introduction to neural networks. Various learning algorithms, learning tasks, activation functions, models, and definitions were presented.

2. IMAGE PROCESSING

2.1 Overview

This chapter presents insight into basic image processing outlined by image analysis, pattern classes, error matrices, classifying image data, discrete wavelet transform of an image, and quantization. In addition object recognition and optical recognition are introduced.

2.2 Elements of Image Analysis

The spectrum of techniques in image analysis divides into three basic areas: (1) low-level processing, (2) intermediate-level processing, and (3) high-level processing. Although these sub-divisions enclose no real or definite boundaries, they do provide a useful framework for categorizing the various processes of an autonomous image analysis system. Figure 2.1 illustrates these concepts, with the overlapping dashed lines indicating that clear-cut boundaries between processes do not really exist.

Low-level processing deals with functions viewed as automatic reactions that require no intelligence on the part of the image analysis system, including image acquisition and preprocessing. This classification encompasses activities from the image formation process itself to compensations, such as noise reduction or image de-blurring. Low-level functions compare to the sensing and adaptation processes that a person goes through when trying to find a seat immediately after entering a dark theater from bright sunlight. The intelligent process of finding an unoccupied seat cannot begin until the availability of a suitable image. The process followed by the brain in adapting the visual system to produce an image indicates an automatic, unconscious reaction.

Intermediate-level processing deals with the task of extracting and characterizing components in an image resulting from a low-level process. As figure 2.1 indicates, intermediate-level processes encompass segmentation and description, using techniques. Flexible segmentation procedures must build some capabilities for intelligent behavior. For example, bridging small gaps in a segmented boundary involves more sophisticated elements of problem solving than mere low-level automatic reactions.

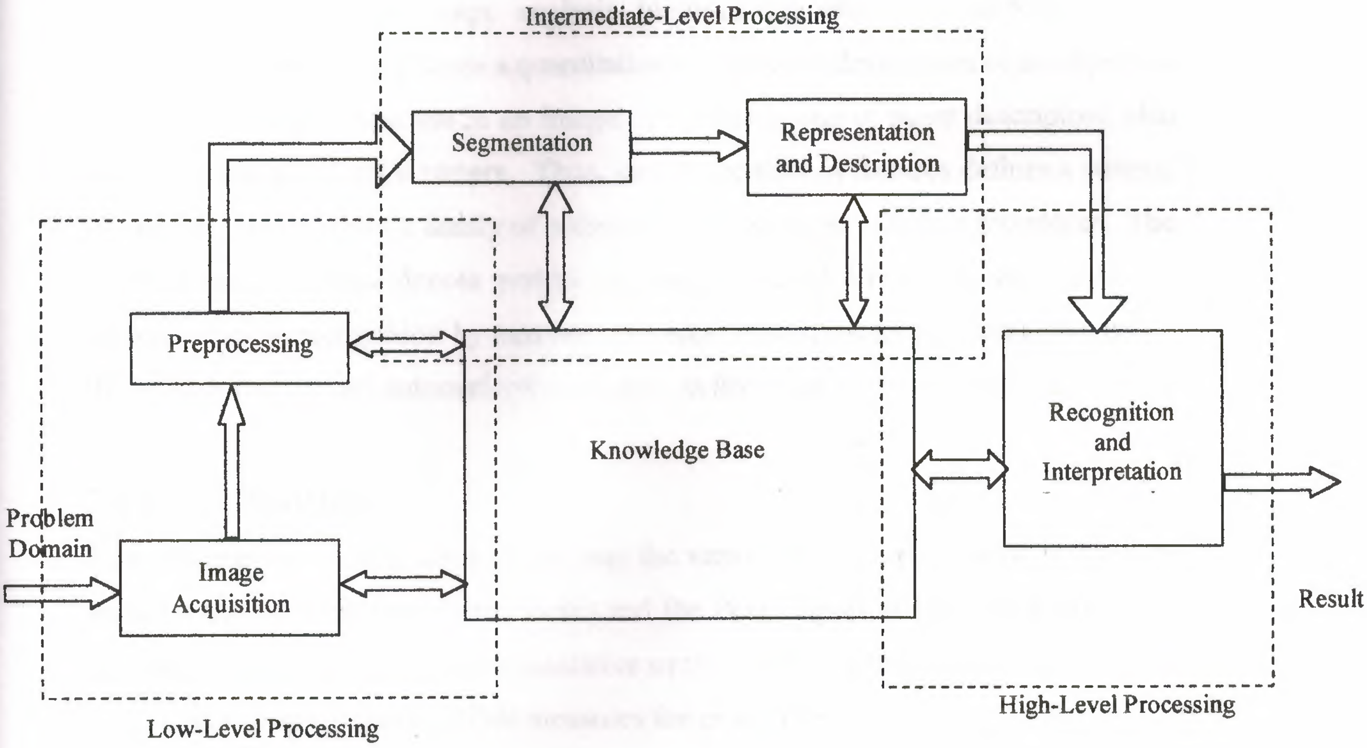


Figure 2.1 Elements of Image Analysis

Finally, high-level processing involves recognition and interpretation. These two processes grasp a stronger resemblance to the term intelligent cognition. The majority of techniques used for low and intermediate-level processing encompass a reasonably well-defined set of theoretic formulations. However, venturing into recognition and interpretation requires that knowledge and understanding of fundamental principles become more speculative. This relative lack of understanding ultimately results in a formulation of constraints and idealizations intended to reduce task complexity to a manageable level. The final product consists of a system with highly specialized operational capabilities.

2.3 Pattern Classes

A fundamental step in image analysis includes the ability to perform pattern recognition. A pattern indicates a quantitative or structural description of an object, or some other entity of interest in an image. In general, one or more descriptors, also known as features, form a pattern. Thus, an arrangement of features defines a pattern. A pattern class indicates a family of patterns that share some common properties. The symbols ω_1, ω_2, $\ldots\omega_M$ denote pattern classes, where M represents the number of classes. Pattern recognition by machine involves techniques for assigning patterns to the irrespective classes automatically and with as little human intervention as possible.

2.4 Error Matrices

Two of the error matrices used to compare the various image compression techniques includes the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The mean squared error uses the cumulative squared error between the compressed and the original image, whereas PSNR measures the peak error.

$$MSE = \frac{1}{MN}\sum_{y=1}^{M}\sum_{x=1}^{N}\left[I(x,y) - I'(x,y)\right]^2 \qquad (2.1)$$

PSNR = 20 * log10 (255 / sqrt(MSE))

where I(x,y) represents the original image, I'(x,y) the approximated version (in actually the decompressed image), and M and N the dimensions of the images.

A lower value for MSE indicates less error, and a higher value of PSNR indicates a higher ratio of signal to noise, making it preferred. Here, the signal denotes the original image, and the noise the error in reconstruction. So, a lower MSE and a high PSNR indicates a decent compression scheme.

2.5 The Outline

Take a close look at compressing grey scale images. The algorithms explained can easily extend to color images, either by processing each of the color planes separately, or by transforming the image from RGB representation to other convenient representations.

The usual steps involved in compressing an image include:

1. Specifying the rate, bits available, and distortion, tolerable error, parameters for the target image.
2. Dividing the image data into various classes based on their importance.
3. Dividing the available bit budget among these classes with minimum distortion.
4. Quantize each class separately using the bit allocation information derived in step three.
5. Encode each class separately using an entropy coder and write to the file.

Reconstructing the image from the compressed data usually takes less time than compression. The steps consist of:

1. Reading in the quantized data from the file using an entropy decoder. (Reverse of Step 5)

2. De-quantize the data. (Reverse of step 4).
3. Rebuild the image. (Reverse of step 2).

2.5.1 Classifying Image Data

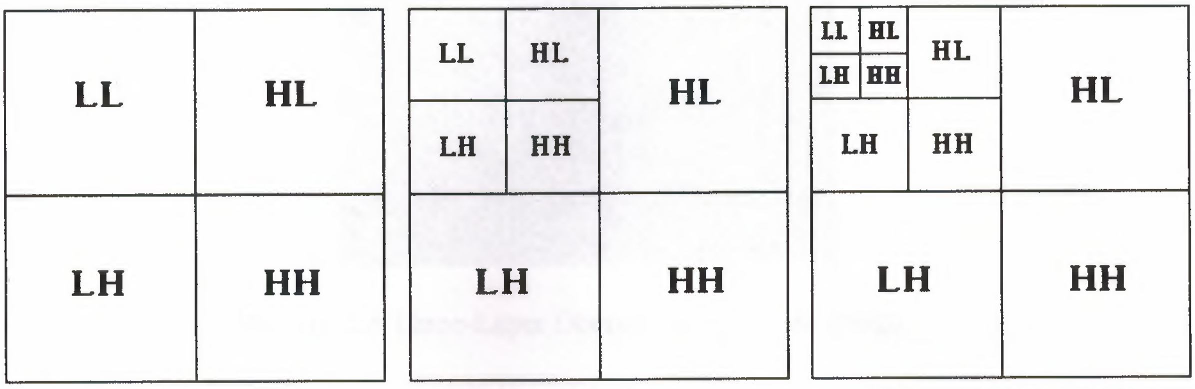
A two-dimensional array of coefficients represents an image, each coefficient representing the brightness level in that point. Most natural images consist of smooth color variations, with the fine details represented as sharp edges in between the smooth variations. Smooth variations in color define as low-frequency components, and the sharp variations as high-frequency components.

The low-frequency components constitute the base of an image, and the high-frequency components add upon them to refine the image, thereby giving a detailed image. Hence, the smooth variations demand more importance than the sharp variations.

2.5.2 The Discrete Wavelet Transform (DWT) of an Image

Select a low and a high-pass filter, called the Analysis Filter Pair, such that they exactly half the frequency range between themselves. First, the low-pass filter applies on each row of data, thereby getting the low-frequency components of the row. But as a half band filter, the output data contains frequencies only in the first half of the original frequency range. So, by Shannon's Sampling Theorem, they become sub-sampled by two, so that the output data now contains only half the original number of samples. Now, the high-pass filter applies on the same row of data, and similarly the high-pass components separate and placed along side of the low-pass components. This procedure repeats for all rows.

Next, the filtering occurs for each column of the intermediate data. The resulting two-dimensional array of coefficients contains four bands of data, each labeled as either LL (low-low), HL (high-low), LH (low-high), or HH (high-high). The LL band decomposes once again in the same manner, thereby producing more sub-bands. This may take pace up to any level, thereby resulting in a pyramidal decomposition as figure 2.2 demonstrates.



(a) Single Level Decomposition **(b) Two Level Decomposition** **(c) Three Level Decomposition**

Figure 2.2 Pyramidal Decomposition of an Image

The low-low band at the highest level classifies as most important, and the other detail bands classify as lesser importance, with the degree of importance decreasing from the top of the pyramid to the bands at the bottom.

Figure 2.3 Three-Layer Decomposition of an Image

2.6 The Inverse Discrete Wavelet Transform (DWT) of an Image

Just as a forward transform separates the image data into various classes of importance, a reverse transform reassembles the various classes of data into a reconstructed image. Here, the pair of high and low-pass filters defines as the Synthesis Filter Pair. The filtering procedure starts from the topmost level, applies the filters column and row-wise, and proceeds to the next level until the first level.

2.6.1 Bit Allocation

The first step in compressing an image consists of segregating the image data into different classes. Depending on the importance of the data it contains, each class receives an allocated portion of the total bit budget in order to minimize possible distortion within a compressed image.

The Rate-Distortion theory solves the problem of allocating bits to a set of classes, or for bit-rate control in general. This theory aims at reducing the distortion for a given target bit-rate by optimally allocating bits to the various classes of data. One approach to solve the problem of Optimal Bit Allocation using the Rate-Distortion theory includes:

1. Initially, all classes allocate a predefined maximum number of bits.
2. For each class, one bit reduces from its quota of allocated bits, and the distortion due to the reduction of that 1 bit calculates.
3. Of all the classes, the class with minimum distortion for a reduction of 1 bit receives notation, and 1 bit reduces from its quota of bits.
4. The total distortion for all classes D calculates.
5. The total rate for all the classes calculates as: R = p(i) * B(i) W Where: *p* indicates the Probability and *B* the Bit Allocation for each class.
6. Compare the target rate and distortion specifications with the values obtained above. If not optimal, go to step 2.

In the approach explained above, one bit at a time reduces until the achievement of optimality either in distortion or target rate, or both. An alternate approach involves starting with zero bits allocated for all classes, and to find the class most benefited by getting an additional bit. The benefit of a class defines as the decrease in distortion for that class.

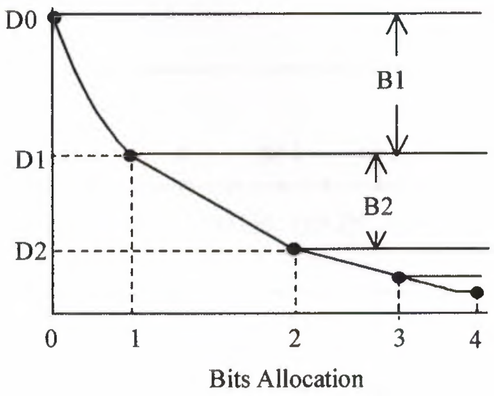


Figure 2.4 Decrease in Distortion Due to Receiving a Bit

2.6.2 Quantization

Quantization refers to the process of approximating the continuous set of values in the image data with a finite set of values. The original data forms the input to a quantizer, and the output always gives one among a finite number of levels. The quantizer process approximates, and a good quantizer represents the original signal with minimum loss or distortion.

Quantization includes two types: Scalar Quantization and Vector Quantization. In scalar quantization, each input symbol treats as separate in producing the output, while in vector quantization the input symbols club together in groups called vectors and process to give the output. This clubbing of data and treating them as a single unit increases the optimality of the vector quantizer; however, at the cost of increased computational complexity.

A quantizer categorizes by its input partitions and output levels, also called reproduction points. A uniform quantizer divides the input range into levels of equal spacing, yet a non-uniform quantizer does not. Implementing a uniform quantizer, presented in figure 2.5, proves easier than a non-uniform quantizer. If the input falls between n*r and (n+1)*r, the quantizer outputs the symbol n.

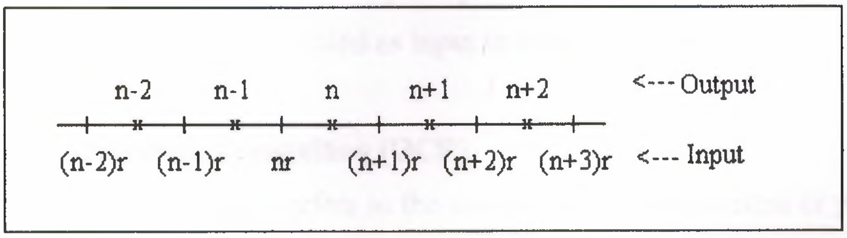


Figure 2.5 A Uniform Quantizer

Just the same way a quantizer partitions its input and outputs discrete levels, a de-quantizer receives the output levels of a quantizer and converts them into normal data by translating each level into a reproduction point in the actual range of data. The optimum quantizer, or encoder, and optimum de-quantizer, or decoder, must satisfy the following conditions.

- Centroid condition. Given the output levels or partitions of the encoder, the best decoder puts the reproduction points x' on the centers of mass of the partitions.
- Nearest neighbor condition. Given the reproduction points of the decoder, the best encoder puts the partition boundaries exactly in the middle of the reproduction points. In other words, each x translates to its nearest reproduction point.

The quantization error (x - x') denotes a measure of the optimality of the quantizer and de-quantizer.

2.7 Object Recognition

Object recognition consists of locating the positions, orientations, and scales of objects in an image. This may also assign a class label to a detected object. In most applications, artificial neural networks trained to locate individual objects based direction pixel data. Another less frequently used approach maps the contents of a window onto a feature space provided as input to a neural classifier.

2.7.1 Optical Character Recognition (OCR)

Optical Character Recognition refers to the recognition of handwritten or printed text by computer. Dynamic OCR comes into play when the input device, such as a digitizer tablet and other methods of pen-based computing, transmits the signal in real time or includes timing information together with pen position, as in signature capture. It also includes other methods of human-computer interaction, such as speech recognition. Image-based OCR employs when the input device captures the position of digital ink, as with a still camera or scanner.

Static OCR encompasses a range of problems that contain no counterpart in the recognition of spoken or signed language, usually collected under the heading of page decomposition or layout analysis. These include both the separation of linguistic material from photos, line drawings, and other non-linguistic information, establishing the local horizontal and vertical axes, and the appropriate grouping of titles, headers, footers, and other material set in a font different from the main body of the text. Another optical character recognition problem arises within different scripts, such as Kanji and Kana, or Cyrillic and Latin, in the same running text.

While the early experimental optical character recognition systems operated rule-based, by the eighties systems based on statistical pattern recognition replaced these. For clearly segmented printed materials, such techniques offer virtually error-free character recognition for the most important alphabetic systems including variants of the Latin, Greek, Cyrillic, and Hebrew alphabets. However, when an alphabet contains a large number of symbols, as in the Chinese or Korean writing systems, or the symbols connect to one another, as in Arabic or Devanagari print, these systems still cannot achieve the error rates of human readers. The gap between the two becomes more evident with the compromise of the quality of the image, such as by fax transmission. Until the resolution of these problems, optical character recognition can not play the pivotal role in the transmission of cultural heritage to the digital age.

2.8 Summary

This chapter provided basic information about image processing. Processing an image by analysis, forming pattern classes, classifying data, quantization, and performing the discrete wavelet transform was discussed. Object recognition and optical character recognition were accordingly presented.

3. IMAGE PROCESSING AND NEURAL NETWORKS

3.1 Overview

This chapter discusses neural networks when applied to image processing through the image processing algorithm, data reduction and feature extraction, image segregation, and real-life applications of neural networks.

3.2 Image Processing Algorithms

Traditional techniques from statistical pattern recognition, like the Bayesian discriminate and the Parzen windows, experienced popularity until the beginning of the 1990s. Since then, neural networks became an increasingly used as an alternative to classic pattern classifiers and clustering techniques. Non-parametric, feed-forward neural networks quickly became attractive, trainable machines for feature-based segmentation and object recognition. With the unavailability of a gold standard, the self-organizing feature map (SOM) constitutes an interesting alternative to supervised techniques. It may learn to discriminate, for example different textures, when provided with powerful features. The current use of neural networks in image processing exceeds the aforementioned traditional applications. The role of feed-forward neural networks and the self-organizing feature map extended to encompass also low-level image processing tasks, such as noise suppression and image enhancement. Hopfield neural networks received introduction as a tool for finding satisfactory solutions to complex optimization problems. This makes them an interesting alternative to traditional optimization algorithms for image processing tasks that can formulate as optimization problems. The deferent problems addressed in the field of digital image processing organize into the image processing chain. Figure 3.1 presents the distinctions in the processing steps within the image processing chain.

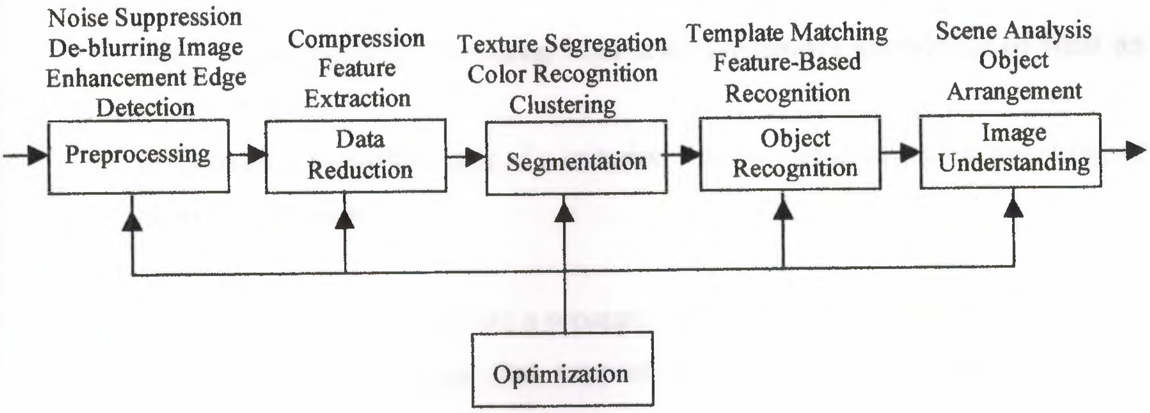

Noise Suppression De-blurring Image Enhancement Edge Detection
Compression Feature Extraction
Texture Segregation Color Recognition Clustering
Template Matching Feature-Based Recognition
Scene Analysis Object Arrangement
Preprocessing
Data Reduction
Segmentation
Object Recognition
Image Understanding
Optimization

Figure 3.1 Image Processing Chain

The image processing chain includes the five different tasks: preprocessing, data reduction, segmentation, object recognition and image understanding. Optimization techniques act as a set of auxiliary tools that makes itself available in all steps of the image processing chain.

1. Preprocessing and filtering. Operations that give as a result a modified image with the same dimensions as the original image such as contrast enhancement and noise reduction.
2. Data reduction and feature extraction. Any operation that extracts significant components from an image or window. The number of pixels in the input window generally exceeds the number of extracted features.
3. Segmentation. Any operation that partitions an image into regions coherent with respect to some criterion. One example includes the segregation of deferent textures.
4. Object detection and recognition. Determining the position and, possibly, also the orientation and scale of specific objects in an image, and classifying these objects.

5. Image understanding. Obtaining high level (semantic) knowledge of what an image shows.
6. Optimization. Minimization of a criterion function used for graph matching or object delineation.

Optimization techniques do not form a separate step in the image processing chain, but as a set of auxiliary techniques that support other steps. Besides the actual task performed by an algorithm, its processing capabilities partly determine by the abstraction level of the input data. The following abstraction levels distinguish as:

A. Pixel level. The intensities of individual pixels provide input to the algorithm.
B. Local feature level. A set of derived, pixel-based features constitutes the input.
C. Structure or edge level. The relative location of one or more perceptual features such as edges, corners, junctions, surfaces, etc.
D. Object level. Properties of individual objects.
E. Object set level. The mutual order and relative location of detected objects.
F. Scene characterization. A complete description of the scene possibly including lighting conditions, context, etc.

3.3 Neural Networks in Image Processing

Neural networks often apply to the various preprocessing procedures in the image processing chain rule. Such include image reconstruction, image restoration, and image enhancement.

3.3.1 Preprocessing

The first step in the image processing chain involves preprocessing. Loosely defined, preprocessing includes any operation of which the input consists of sensor data, and which outputs a full image. Preprocessing operations generally fall into one of three categories: image reconstruction that reconstructs an image from a number of sensor measurements, image restoration which removes any aberrations introduced by the sensor such as noise, and image enhancement that accentuates certain desired features which may facilitate later processing steps such as segmentation or object recognition.

3.3.2 Image Reconstruction

Image reconstruction problems often require quite complex computations and a unique approach for each application. An Adaptive Linear Element (ADALINE) network trains in order to perform an Electrical Impedance Tomography (EIT) reconstruction for the reconstruction of a 2-dimential image based on measurement on the circumference of the image. Srinivasan [9] trained a modified Hopfield network to perform the inverse radon transform for reconstruction of computerized tomography images. The Hopfield network contained summation layers to avoid having to interconnect all units. Meyer and Heindl [10] used regression feed-forward networks that learn the mapping E(y|x), with *x* the vector of input variables and *y* the desired output vector, to reconstruct images from electron holograms. Wang and Wahl trained a Hopfield network for reconstruction of 2-dimential images from pixel data obtained from projections [11].

3.3.3 Image Restoration

The majority of applications of neural networks in preprocessing originate in image restoration. In general, one wants to restore a distorted image by the physical measurement system. The system might introduce noise, motion blur, out-of-focus blur, distortion caused by low resolution, etc. Restoration uses all information about the nature of the distortions introduced by the system. The restoration problem appears ill-posed because conflicting criteria requires fulfillment: resolution versus smoothness.

In the most basic image restoration approach, simple filtering removes noise from an image. Greenhill and Davies [18] used a regression feed-forward network in a convolution-like way to suppress noise with a 5 x 5 pixel window as input and one output node. De Ridder built a modular feed-forward network approach that mimics the behavior of the Kuwahara filter, an edge-preserving smoothing filter [16]. Their experiments showed that the mean squared error used in network training may not representative of the problem at hand. Furthermore, unconstrained feed-forward networks often ended up in a linear approximation to the Kuwahara filter.

Chua and Yang [14, 15] used Cellular Neural Networks (CNN) for image processing. A system with locally connected nodes defines a cellular network. Each node contains a feedback and a control template, which to a large extent determine the functionality of the network. For noise suppression, the templates implement an averaging function; for edge detection, a Laplacian operator. The system operates locally, but multiple iterations allow it to distribute global information throughout the nodes.

Although quite fast in application, the parameters influencing the network behavior, the feedback and control templates, require hand setting.

Others proposed methods for training cellular networks such as using gradient descent or genetic algorithms grey-value images, proposed by Zamparelli. Cellular neural networks also applied for restoration of color images by Lee and Degyvez.

Another interesting neural network architecture includes the Generalized Adaptive Neural Filter (GANF) used for noise suppression. This consists of a set of neural operators based on stack a filter that uses binary decompositions of grey-value data. Finally, image restoration also applied fuzzy networks and neurochips. Traditional methods for more complex restoration problems, such as de-blurring and diminishing out-of-focus defects, include at maximum a posteriori estimation (MAP) and regularization. Utilizing these techniques entails solving high dimensional convex optimization tasks. The objective functions of the regularization problem can both map onto the energy function of the Hopfield network. Often, the network architectures require modification when mapping the problem proves difficult.

Image restoration also applied to other types of networks. Qian developed a hybrid system consisting of order statistic filters for noise removal and a Hopfield network for de-blurring by optimizing a criterion function. The modulation transfer function required measurement in advance. Guan developed a so-called network-of-networks for image restoration. The system consists of loosely coupled modules, where each module makes a separate network. Phoha and Oldham proposed a layered, competitive network to reconstruct a distorted image.

3.3.4 Image Enhancement

The goal of image enhancement consists of amplify specific perceptual features. Among the applications of neural networks developed for image enhancement, one would expect most applications to base on regression networks. However, several enhancement approaches rely on a classifier, typically resulting in a binary output image. The most well-known enhancement problem became edge detection. Pugmire reported a straightforward application of regression feed-forward networks trained to behave like edge detectors. Chandresakaran used a novel, feed-forward architecture to classify an input window as either containing an edge or not containing an edge. The weights of this network set manually instead of obtained from training. Formulating edge detection as an optimization problem made it possible for Tsai to train a Hopfield network for enhancement of endocardiac borders. Some enhancement approaches utilize other types of networks. Shih applied a new network for binary image enhancement. Moh and Shih describe a general approach for implementation of morphological image operations by a modified feed-forward networks using shunting mechanisms, such as neurons acting as switches. Waxman considered the application of a center-surround, shunting feed-forward network, initially proposed by Grossberg, for contrast enhancement and color night vision.

3.3.5 Applicability of Neural Networks in Preprocessing

Preprocessing contains three types of problems that require the application of neural networks:

- Optimization of an objective function defined by a traditional preprocessing problem.
- Approximation of a mathematical transformation used for image reconstruction by regression.
- Mapping by network trained to perform a certain task, usually based directly on pixel data, such as neighborhood input and pixel output.

To solve the first type of problem, a Hopfield network may replace traditional methods for optimization of some objective function.

For the approximation task, regression feed-forward neural networks apply. Although for some applications such networks proved successful, it would seem that these applications call for more traditional mathematical techniques, since processing needs a guaranteed worst-case performance.

In several other applications, regression, calculation, or mapping networks trained to perform image restoration or enhancement directly from pixel data. Preprocessing often used non-adaptive networks, such as cellular neural networks.

Secondly, with adaptive networks, their architectures usually differed much from those of the standard networks. Designing networks applied for image restoration or enhancement, those using shunting mechanisms to force a feed-forward network to make binary decisions required prior knowledge about the problem. The interest in non-adaptive networks indicates that the fast operation and the ease in which the network can embed in hardware presents important criteria when choosing for a neural implementation of a specific preprocessing operation. However, the ability to learn from data holds less importance in preprocessing. While constructing a linear filter with certain desired behavior by specifying its frequency profile seems easy, obtaining a large enough data set to teach the optimal function as a high-dimensional regression proves difficult. This holds especially when the desired network behavior only becomes critical for a small subset of all possible input patterns such as in edge detection. Moreover, choosing a suitable error measure for supervised training proves un-trivial, as simply minimizing the mean squared error might give undesirable results in an image processing setting.

The network parameters most likely turn to one type of image, such as a specific sensor, scene setting, scale, etc., this limits the applicability of the trained network. When the underlying conditional probability distributions $p(x|w_j)$ or $p(y|x)$, change, the classification or regression network-like all statistical models requires re-training.

3.4 Data Reduction and Feature Extraction

Two of the most important applications of data reduction include image compression and feature extraction. In general, an image compression algorithm, used for storing and transmitting images, contain two steps: encoding and decoding. Both these steps employ artificial neural networks. Subsequent segmentation or object recognition uses feature extraction. The kind of features one wants to extract often correspond to particular geometric or perceptual characteristics in an image, for instance, edges, corners and junctions, or application dependent ones, such as facial features.

3.4.1 Feature Extraction Applications

Feature extraction gives the impression of a special kind of data reduction that finds a subset of informative variables based on image data. Naturally high-dimensional, feature extraction for image data constitutes a necessary step for successful segmentation or object recognition. Besides lowering the computational cost, feature extraction indicates another means for controlling the so-called curse of dimensionality. When used as input for a subsequent segmentation algorithm, one wants to extract those features that preserve the class reparability well. There exists a wide class of neural networks able to receive training to perform mappings to a lower-dimensional space. A well-known feature-extraction network includes Oja's neural implementation of a one-dimensional Principal Component Analysis (PCA), later extended to multiple dimensions. Baldi and Homik proved that training three-layer auto-associator networks correspond to applying principal component analysis to the input data. Later, auto-associator networks with five layers obtained the ability to perform non-linear dimensionality reduction, such as finding principal surfaces. A mixture of linear subspaces to approximate a non-linear subspace includes another possibility. Another approach to feature extraction clusters the high-dimensional data and then uses the cluster centers as prototypes for the entire cluster. Among the networks trained to perform feature extraction, most applications use feed-forward networks.

Most of the networks trained for feature extractions obtain pixel data as input. Neural - network feature extraction performed for:

- Subsequent automatic target recognition in remote sensing accounting for orientation and character recognition.

- Subsequent segmentation of food images and of Magnetic Resonance (MR) images.
- Finding the orientation of objects by coping with rotation.
- Finding control points of deformable models.
- Clustering low-level features found by the Gabor filters in face recognition and wood defect detection.
- Subsequent stereo matching.
- Clustering the local content of an image before encoding.

In most applications, segmentation, image matching, or object recognition employed the extracted features. For anisotropic objects occurring at the same scale, rotation causes the largest amount of intra-class variation. Some feature extraction approaches designed to cope explicitly with changes in orientation of objects. Feature extraction requires the distinction between application of supervised and unsupervised networks. For a supervised auto-associator network, the information loss implied by the data reduction measures directly on the predicted output variables, however, not the case for unsupervised feature extraction. Both supervised and unsupervised network feature extraction methods contain advantages compared to traditional techniques. Feed-forward networks with several hidden layers can train to perform non-linear feature extraction, but lack a formal, statistical basis.

3.5 Image Segmentation

Segmentation includes the partitioning of an image into coherent parts according to some criterion. When considered as a classification task, segmentation assigns labels to individual pixels. Some neural-based approaches perform segmentation on the pixel data, obtained either from a convolution window, or the provided information to a neural classifier in the form of local features.

3.5.1 Image Segmentation Based on Pixel Data

Many neural network approaches segment images directly from the pixel data. Several deferent types of networks trained to perform pixel-based segmentation. Some include feed-forward networks, cellular networks, Hopfield networks, probabilistic networks, radical bias function networks, constraint satisfaction networks, etc. Also, the occurrence of biologically inspired neural-network approaches such as: the perception model developed by Grossberg, able to segment images from surfaces and their shading, and the brain-like networks proposed by Opara and Worgotter. Hierarchical segmentation approaches designed to combine networks on deferent abstraction levels. The guiding principles behind hierarchical approaches include specialization and bottom-up processing in which one or more networks dedicated to low-level feature extraction or segmentation, and their results combined at a higher abstraction level. There another neural classifier performs the final image segmentation. Reddick developed a pixel-based two-stage approach where a network trained to segment multi-spectral images. The segments subsequently classify into white matter, grey matter, etc., by a feed-forward network. Non-hierarchical, modular approaches also developed.

In general, pixel-based networks trained to classify the image content based on:

- Texture
- A combination of texture and local shape

Pre and post-processing steps in relation to segmentation developed networks for:

- Delineation of contours
- Connecting edge pixels
- Identification of surfaces
- Deciding whether a pixel occurs inside or outside a segment

- De-fuzzying the segmented image
- Clustering of pixels
- Motion segmentation

In most applications, neural networks trained as supervised classifiers to perform the desired segmentation. One feature that most pixel-based segmentation approaches lack includes a structured way of coping with variations in rotation and scale. This shortcoming may deteriorate the segmentation result.

3.6 Real-Life Applications of Neural Networks

Neural networks gradually found their way into a large range of commercial applications. Unfortunately, commercial and other considerations often impede publication of scientific and technical aspects of such systems. Some research programs gave an overview of commercial applications of neural networks, and one of its applications includes character recognition.

3.6.1. Character Recognition

Two essential components in a character recognition algorithm include the feature extractor and the classifier. Feature analysis determines the descriptors, or feature set, used to describe all characters. Given a character image, the feature extractor derives the features in the character. The derived features then become the input to the character classifier.

One of the most common classification methods embraces template matching, or matrix matching. Template matching uses individual image pixels as features. Classification performs by comparing an input character image with a set of templates from each character class. Each comparison results in a similarity measure between the input character and the template. One measure increases the amount of similarity when a pixel in the observed character matches the same pixel in the template image. If the pixels differ, the measure of similarity may decrease. After all templates compare with the observed character image, the character's identity assigns as the identity of the most similar template.

Template matching involves a trainable process because template characters may change. In many commercial systems, programmable read-only memories (PROMs) store templates containing single fonts. To retrain the algorithm, memories containing images of new font replace the current. Thus, if a suitable memory exists for a font, then template matching can train to recognize that font. The similarity measure of template matching may also modify, but commercial optical character recognition systems typically do not allow this.

Structural classification methods utilize structural features and decision rules to classify characters. Structural features defined either in terms of character strokes, character holes, or other character attributes such as concavities. For instance, the letter P describes as a vertical stroke with a hole attached on the upper right side. For a character image input, extraction of the structural features takes place and a rule-based system applies to classify the character. Another trainable method, structural methods, construct of a good feature set and a rule-base. However, this method consumes more time.

Character localization and segmentation then occurs. After the location of the document, the relative image portion quantizes into binary values according to an adaptive threshold established directly through a two-class clustering of tones. The characters segment by finding white areas between columns with higher density of black pixels, as illustrated in figure 3.2 and 3.3.

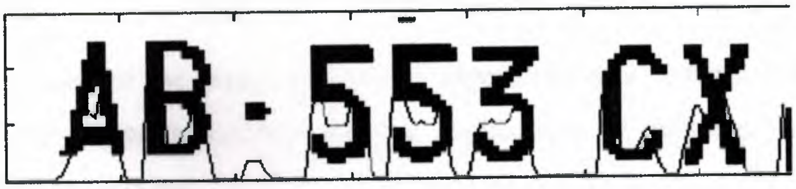


Figure 3.2 Character Localization

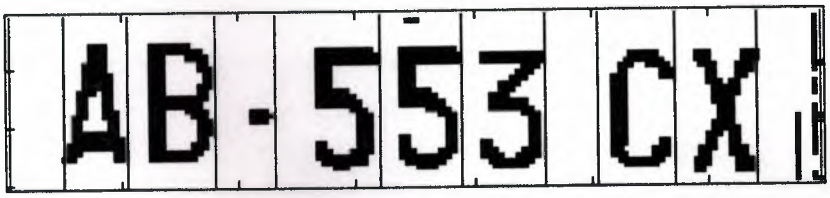


Figure 3.3 Character Segmentation

Isolated black pixels wipe out and the character resizes to the standard measure of 10 by 6 pixels after a factor-of-two decimation, as figure 3.4 shows.

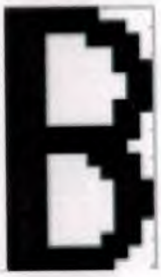


Figure 3.4 B Extracted and Digitized

3.7 Summary

This chapter presented the image processing algorithm, data reduction and feature extraction, image segmentation, and real-life applications of neural networks when applied to image processing.

4. THE CHARACTER RECOGNITION SYSTEM USING THE ARTIFICIAL NEURAL NETWORK

4.1 Overview

This chapter delves into the character recognition system that describes the processes in which to design character matrices and the network structure. In addition, the feed-forward and back-propagation algorithms are described, followed by compression of the character matrices.

4.2 Creating the Character Recognition System

The user must first create the character recognition system in order to prepare it for presentation into MatLab. Creating the matrices of each letter of the alphabet, along with the network structure, must first take place. In addition, one must understand how to separate the binary input code from the matrix, and how to create the binary target code, which the network requires for learning.

4.2.1 Character Matrices

An array of black and white pixels form a character matrix; the vector of one represented by a black pixel, and zero by a white pixel. Each character matrix represents a single character. The user creates them manually, in whatever size, font, or combination of fonts possible.

4.2.2 Creating a Character Matrix

Creating the character matrices first involve choosing an appropriate matrix size. With very small matrices, creating all the desired characters may not take place, especially when using more than one font. On the other hand, with large matrices, a few problems may arise: Despite the fact that the speed of computers double every third year, the network may not run in real time due to lack of processing power. Training may take days, and results may take hours. In addition, the computer's memory may not handle enough neurons in the hidden layer(s) needed to efficient and accurately process the information. After determining the size, the user fashions the matrices by designing the font shape from imagination. The vectors containing character information become shaded and receive a value of one, and all others take on a value of zero. Through these steps, two fonts containing twenty-six 20 x 20 matrices took shape. Figure 4.1 below shows two fonts of character A, Appendix I provides the complete list of the characters in dual fonts, and Appendix II presents the various fonts used for training in MatLab.

	First Font	Second Font
Character Matrix A	00000000000000000000 00000000011000000000 00000000111100000000 00000001111110000000 00000011111111000000 00000111000011100000 00001110000001110000 00011100000000111000 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00000000000000000000

Figure 4.1 Dual Fonts of Character A

4.2.3 Choosing a Suitable Network Structure

A neural network contains an input, hidden(s), and output layer. The number of neurons inside each layer decided corresponds to the particular matrix and criteria of information to process. In this case, a 400-52-26 network—400 inputs, 52 hidden neurons, 26 outputs—formed based on 20 x 20 matrices of upper case letters of the

English alphabet. Figure 4.2 illustrates the structure of this particular neural network.

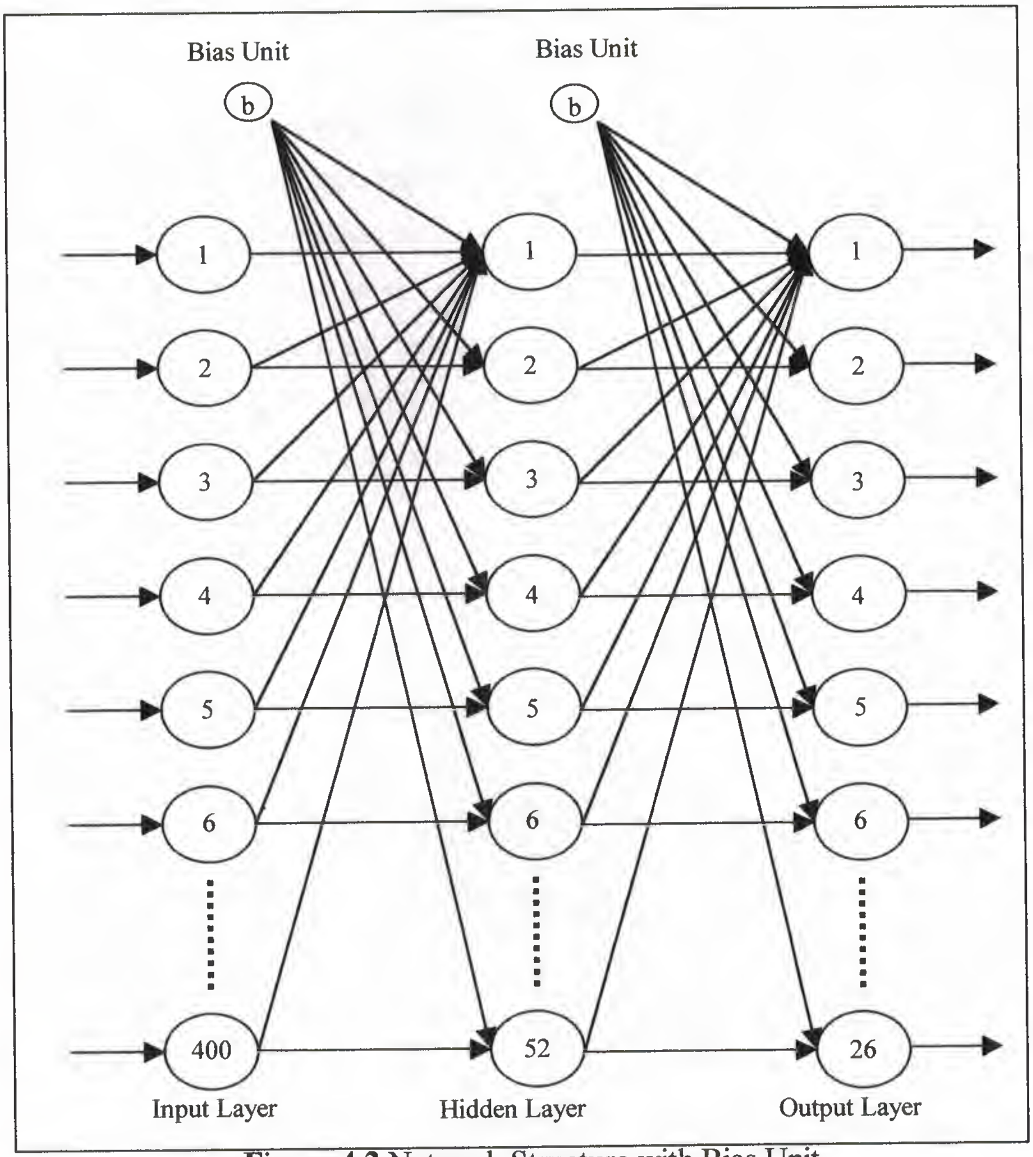


Figure 4.2 Network Structure with Bias Unit

Input Layer

The nodes of the input layer depend on the size of the character matrix, for example, a character matrix of 20 x 20 results in 400 neurons within the input layer.

Hidden Layer

Two factors come into play regarding the hidden layer: (1) how many hidden layers to include inside of the network, and (2) the number of neurons within each of those layers.

Neural networks may consist of none, one, or two hidden layers. Currently no theoretical reason to use more than two took place. Networks with no hidden layers suffer from limited capabilities of computing very small and simple data with only two possible outputs, such as yes and no questions. A network employed 99% of the time uses one hidden layer, idea for basic and practical problems. Problems that require two hidden layers within a neural network rarely surface. Figure 4.3 gives basic insight into determining the number of hidden layers to use.

Number of Hidden Layers	**Result**
0	Only capable of representing linear separable functions or decisions
1	Can approximate any functions which contain a continuous mapping from one finite space to another
2	Represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy

Figure 4.3 Determining the Number of Hidden Layers

Character recognition requires only one hidden layer, but the number of neurons inside depends on the size of the matrices and result from experimentation. Using too few neurons in the hidden layer may result in under-fitting. This occurs when the hidden layer(s) contain too few neurons to detect the signals, or weights, in a complicated data set. On the other hand, using too many may result in several problems: Over-fitting occurs when the neural network boasts so much information processing capacity that the limited amount of information contained in the training set cannot train all of the neurons in the hidden layer(s). This results in useless neurons containing weighted connections of zero, in which removing them becomes necessary. A second problem may occur even with the presence of sufficient training data; an inordinately large number may increase the time it takes to train the neural network enough that training the network becomes impossible. Obviously, some compromise between too few and too many neurons must take place.

A few rule-of-thumb-methods evolved for determining the correct number of hidden nodes to use. The number of hidden neurons, generally, should range between the size of the input layer and the size of the output layer. This also applies to very large matrices, with the addition of factoring in the network and the computer's memory and processing capabilities. For very small matrices, such as 5 x 7, the number of hidden layer nodes should bear a smaller amount than twice the size of the input layer.

Although these methods help to consume less time, ultimately the selection of the architecture of the hidden layer must result from trial and error. Two trial and error approaches further the experimentation process, called the "forward" and "backward" selection methods. The forward selection method begins by selecting a smaller number of hidden neurons, and then training and testing of the network occurs. The numbers of hidden neurons are slowly increased, and the process repeats until the overall results discontinue improving. Figure 4.4 demonstrates this process.

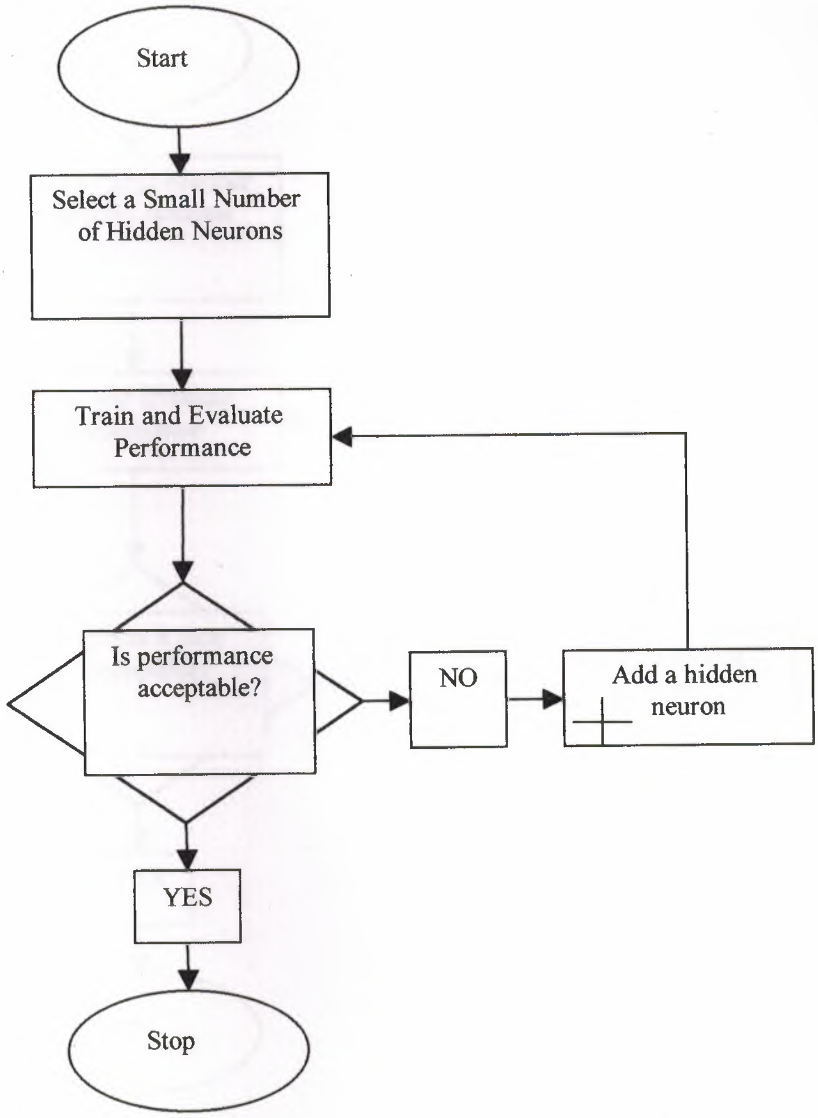

Start
Select a Small Number of Hidden Neurons
Train and Evaluate Performance
Is performance acceptable?
NO
Add a hidden neuron
YES
Stop

Figure 4.4 Forward Selection Method

The second method, the backward selection method, begins by using a large number of neurons. This process behaves in the same manor as the first; the user repeatedly trains and tests the network, each time subtracting a neuron, however, so long as the overall results continue to improve. Figure 4.5 exhibits the backward selection method.

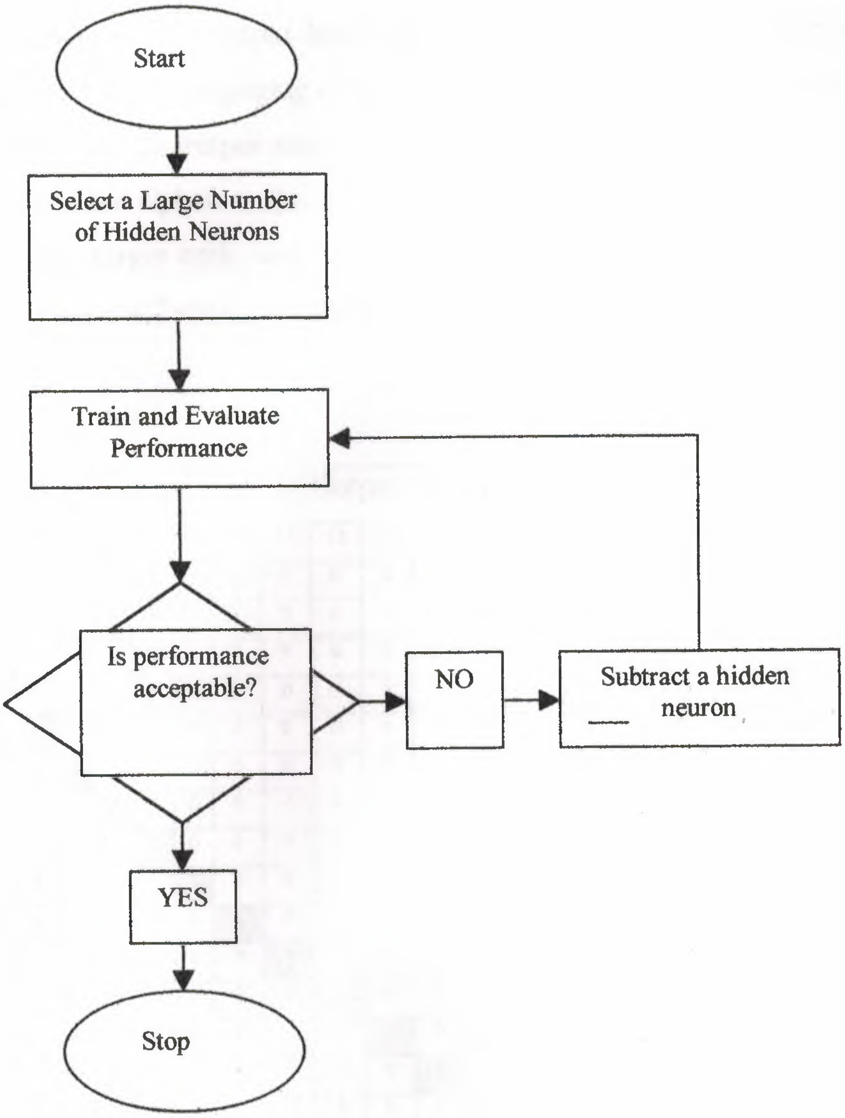

Start
Select a Large Number of Hidden Neurons
Train and Evaluate Performance
Is performance acceptable?
NO
Subtract a hidden neuron
YES
Stop

Figure 4.5 Backward Selection Method

An additional step may employ after conducting any method of selecting hidden layer neurons – pruning. Pruning in its simplest form evaluates the weighted connections between the layers. The user manually deletes any hidden neurons that contain weights of zero within the network. This solves the condition of over-fitting as previously discussed.

Output Layer

The number of nodes in the output layer results from the amount of information received. For example, if recognizing characters based on the English alphabet, the network will encompass 26 output nodes; 25 of which result as 0's, and a single 1 in place of the letter in the alphabet, shown in figure 4.6. All output values combined, known as the binary target code, tells the computer which output corresponds to a particular input. Figure 4.7 shows the binary target code for each character.

	Output Layer Neurons																									
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 4.6 Outputs for Each Character

Character	Binary Target Code
A	10000000000000000000000000
B	01000000000000000000000000
C	00100000000000000000000000
D	00010000000000000000000000
E	00001000000000000000000000
F	00000100000000000000000000
G	00000010000000000000000000
H	00000001000000000000000000
I	00000000100000000000000000
J	00000000010000000000000000
K	00000000001000000000000000
L	00000000000100000000000000
M	00000000000010000000000000
N	00000000000001000000000000
O	00000000000000100000000000
P	00000000000000010000000000
Q	00000000000000001000000000
R	00000000000000000100000000
S	00000000000000000010000000
T	00000000000000000001000000
U	00000000000000000000100000
V	00000000000000000000010000
W	00000000000000000000001000
X	00000000000000000000000100
Y	00000000000000000000000010
Z	00000000000000000000000001

Figure 4.7 Binary Target Code for Each Character

Four hundred inputs, one for each vector of the 20 x 20 matrix, comprises the input layer. After completing the steps above, 52 hidden nodes within a single hidden layer proved most suitable. As previously explained, the output layer contains 26 neurons, one for each letter of the alphabet. This constitutes a two-layer network, excluding the input layer as a layer since no calculations take place. Refer back to figure 4.2.

4.2.4 Deriving the Input from a Character Matrix

For humans, the general overview of the matrix with its black and white pixels forms the input that results in recognition. Nevertheless, computers use the array of zeroes and ones, known as the binary input code. Therefore, simply separate the numerical information contained within each vector to form a single code; for example, a 20 x 20 matrix will boast a 400-digit binary code, such as in figure 4.8.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0

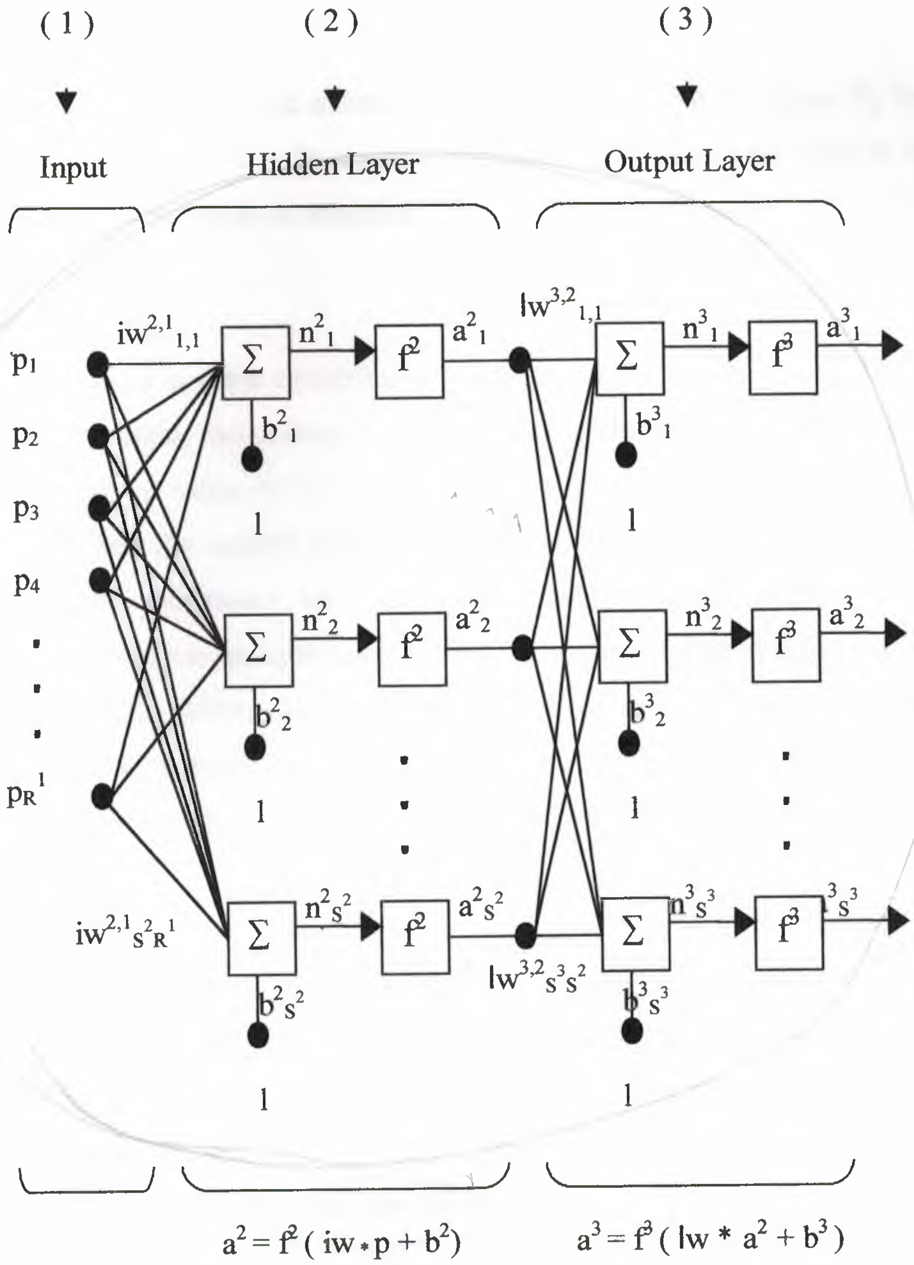
={00000000000000000000000000000110000000000000000011110000000000000001111111000000
000000011111111000000000000011100001110000000001110000001110000000011100000000111000
0011100000000000111000011100000000000111000011100000000000111000011111111111111111000
011111111111111111000011100000000000111000011100000000000111000011100000000000111000
0
111000000000001110000111000000000001110000111000000000001110000111000000000000011100}

Figure 4.8 Binary Input Code for Character A

4.3 The Feed-Forward Algorithm

The multi-layer, feed-forward neural network characterizes the most commonly used network architecture. The term, "feed-forward" describes how the neural network processes and recalls a pattern. When using a feed-forward neural network, the neurons of each layer include restrictions of only forward connections. Each layer of the neural network contains only connectionism to the next layer; more specifically, each node from a layer will connect to all nodes of the next layer. These "links" connect the input to the hidden layer, and hidden to the output layer. No connections may exist between neurons and those in the previous layers, nor between neurons and themselves. Additionally, neurons may not connect to those beyond the next layer as figure 4.9 displays. Therefore, the layers of a feed-forward neural network only receive inputs from previous layers.

The input layer's nodes assert output functions that deliver data to the hidden layer's nodes. The actual computation begins in the hidden layer. Each node in a hidden layer computes a sum based on its input from the input layer, the input-layer weights, and the bias. Their sum equals the overall unit activation value. This sum then becomes "compacted" by a sigmoid function, which squeezes the sum into a more a limited and manageable range. If the unit activation value exceeds the threshold, or bias, value; then the neuron passes on its data. This output sum from the hidden layer passes on to the final processing layer where again calculation takes place, in much the same way. This layer outputs the final network result.



where R^1 represents the number of inputs into the interface layer, S^2 the number of neurons in the first layer, and S^3 the number of neurons in the second layer
AND
where superscripts indicate the layer number and subscripts the node number in a layer

Figure 4.9 Multi-Layer Feed-Forward Network Topology

4.3.1 Input (P_R)

Input *P* represents the binary input matrix. Defined as the number of inputs, P_R thus becomes P_{400} concerning a 20 x 20 matrix. Calculation does not take place in the input layer, thus serving only as an interface to the external environment

4.3.2 Weights (*iw* , *lw*)

Weights refer to the connection strength of a link between two neurons in a network. The strength, or weight, determines the effect that neurons hold on each other. These weights store a value of that strength. The first superscript indicates the destination layer, and the second identifies the source layer. Additionally the first subscript shows the destination node and the second the source neuron. For example, $iw^{2,1}{}_{1,1}$ represents an input-layer weight. The superscripts indicate that the weight connects to the hidden layer, represented by the number 2, and originates from the input layer, represented by the number 1. Furthermore, the subscripts indicate a connection to the first node in the hidden layer from the first node in the input layer.

A two-layer network contains two sets of weights: input layer weights (*iw*) that connect the input to the hidden layer, and hidden layer weights (*lw*) that connect the hidden to the output layer. These two types of weights model the "memory" of the neural network because initial weights first generate randomly as a value between zero and one; and then go through updating during back-propagation as explained later in section 4.4.

4.3.3 Bias Unit (b)

A neuron parameter summed with the neuron's weighted inputs and passed through the neuron's transfer function to generate the neuron's output summarizes the purpose of the bias unit. Most networks employ a bias as part of every layer but the input layer, and this unit holds an initial activation value at one. Each bias neuron connects to all units in the next higher layer and provides a constant term in the weighted sum of the units. The back-propagation process refers to the bias as the node threshold.

4.3.4 Sum Function (∑)

The individual input elements $p_1, p_2, \ldots p_R$ multiplied by weights $w_{1,1}, w_{1,2}, \ldots w_{1,R}$ form the sum *Wp*. The weighted values fed into the sum function and added together with the bias form the net input *n*.

The sum function (∑), when specifically applied in the hidden layer equals the sum of the product of the input and the input layer weight plus the bias. When applied to the output layer, it equals the sum of the product of the output of the first layer and the hidden layer weights plus the bias. These equations written in mathematical form:

Hidden Layer Sum Function

$\sum$ = [Input($p_R{}^1$) * Input Layer Weights($iw^{2,1}{}_{S}{}^{2}{}_{,R}{}^{1}$)] + Bias Unit($b^2{}_{s}{}^{2}$)

Simplified: $\sum = (iw) (p) + b$

Output Layer Sum Function

$\sum$ = [Hidden Layer Output($a^2{}_{s}{}^{2}$)* Hidden Layer Weights($lw^{3,2}{}_{S}{}^{3}{}_{,R}{}^{2}$)] +

Bias Unit($b^3 s^3$)

Simplified: $\sum = (lw) (p) + b$

4.3.5 Net Input (n)

Basically, the value of the sum function constitutes the net input (*n*) and represents the overall unit activation value. Each neuron owns a bias *b*, which summed together with the weighted inputs forms *n*. This sum, *n*, makes up the argument of the transfer function *f*.

Net Input Equation

$n = w_{1,1}\, p_1 + w_{1,2}\, p_2 + \ldots + w_{1,R}\, p_R + b$

4.3.6 Transfer Function (f)

Multi-layer networks, especially those employing the character recognition system, often use the log-sigmoid transfer function. As a non-linear function constraining all numbers from negative to positive infinity to produce simply a zero or one, the sigmoid function works well with the character recognition problem. This occurs because the log-sigmoid transfer function actually behaves as the step function in figure 4.10; all numbers less than zero take on a value of zero, and those zero and above take on a value of one. However, only the neuron with the greatest activation receives a non-zero value.

The transfer function (f) accepts the net input (n) and outputs zeroes for all neurons except for the "winner", who receives a value of one. The function designates the neuron associated with the most positive element as the winner.

Sigmoid Function Equation

$$f(x) = \frac{1}{1 + e^{\sum_{i=1}^{-n} oi\,\mathrm{wi}}} \qquad (4.1)$$

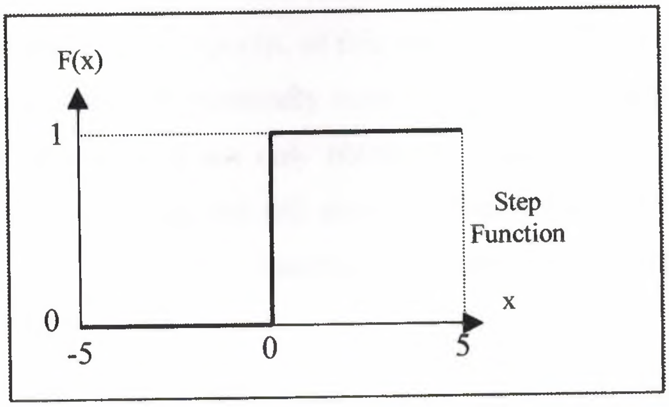


Figure 4.10 Sigmoid Function's Behavior as Step Function

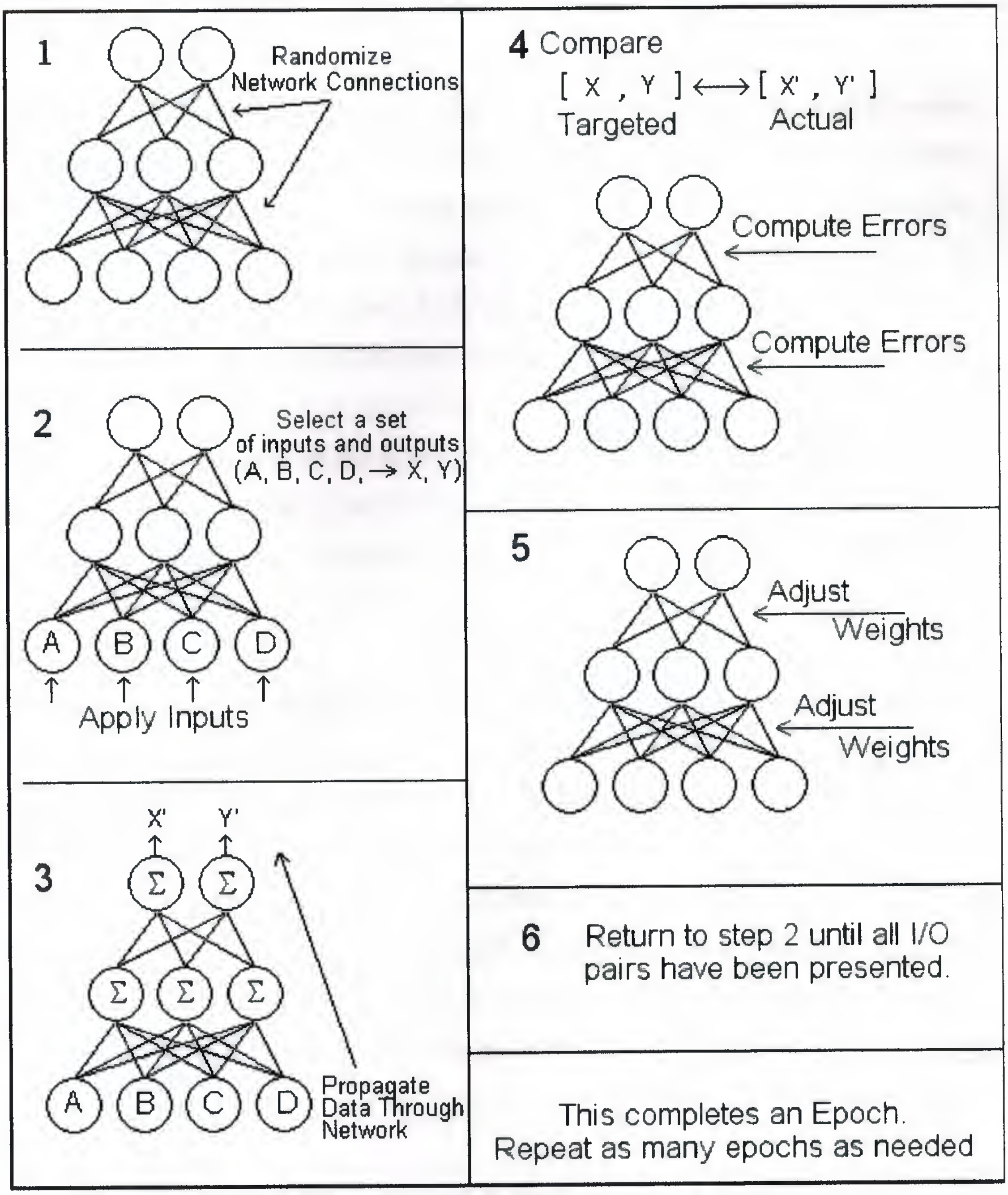
4.3.7 Output (a)

Output a represents the result of all calculations by the sigmoid as previously indicated in figure 4.9. The hidden layer output becomes the input for the neurons of the next layer, and the output layer's output denotes the final network result.

4.4 The Back-Propagation Algorithm

Back-propagation rose to become the most commonly used training algorithm in combination with feed-forward neural networks. Back-propagation refers to a specific type of supervised learning in which errors propagate backwards through the network and distribute evenly to the weights.

As a supervised learning algorithm, it becomes mandatory for the programmer to give the network examples of inputs and their target output so that the network may compare what it outputted against what it should output. If the actual output does not match the target output, then weights automatically adjust to minimize the error and to produce the desired output. Initially, the network produces random and incorrect responses. When the neural network produces an incorrect decision, the connections within the network weaken so as not to produce that answer again. Similarly, when a network produces a correct decision, the connections inside the network strengthen so that it will become more likely to produce that answer. Through many iterations, or epochs, of this process and by giving the network many examples, the network will eventually learn to classify all characters presented. A trained neural network will not only obtain the capability to identify and classify previously encountered data, but will also generalize similar data not yet presented, such as distorted and noisy characters. Figure 4.11 summarizes the back-propagation process.



where an epoch signifies a complete training cycle of all 26 training patterns

Note: Step 5 also includes adjusting node thresholds

Figure 4.11 The Back-Propagation Process

4.4.1 Training

Before use, the network requires training. The network begins with random weight and node threshold values between zero and one. The network learns the initial behavior through exposure to training data presented into the input layer together with known output data, known as patterns. During a forward pass through the network, steps 3 and 4 in figure 4.11; the nodes in the network accumulate the necessary changes in weight and error calculates between the actual output of the network and the output specified in the training set. On the backward pass phase, step 5, each connection weight amends to adjust the behavior of the network. The forward and backward pass phase repeat for a predefined number either of epochs or until the learning starts to saturate. Through this careful adjustment of weights, the network learns.

The first step consists of training the network until it correctly recognizes the inputs within a single epoch, which may take up to 500 epochs of training to reach. Second, each epoch should present randomly ordered inputs. This will take more epochs to learn in this manor, however, the error will decrease. The third step involves totaling the error for each epoch and running it until that error precedes a desired threshold as subsequently explained in Chapter 5. This may take up to 2000 epochs. At this point, the network owns adequate training and may even recognize imperfect data, such as distorted and noisy characters.

Over-training

If the network suffers from more than enough training, it may of just memorized already presented patterns. When this occurs, the network may not handle noise or distorted data. To prevent the occurrence of over-training, employ one font only for training and one only for testing. If the network seems unable to recognize the second font, then over-training occurred and re-initialization must take place.

4.4.2 Computing Error

The degree in which the output from the neural network matches the anticipated output, as specified in the training patterns, defines the error. The trick of back-propagation consists of assessing the blame for an error and dividing it among the contributing weights. Weight updating occurs during each iteration, and the network learns while iterating repeatedly until a achieving a net minimum error value. The error shown during the learning procedure inflicts the greatest error of all the patterns.

Output Unit Error

$\Delta_o = a_o(1 - a_o)(T_o - a_o)$

where T_o represents the target Output Activation and a_o the actual Output Activation at output unit o

Hidden Unit Error

$\Delta_h = a_h(1 - a_h) \sum \delta_o$

where δ_o stands for the Error at unit o (output layer) to which a connection points from hidden unit h.

Network Error Using Sum Square Error

$\delta = (T - r)^2$

where δ specifies the Error, T the Target, and r the Responsive Value

4.4.3 Adjusting Weights

Modifying weights begins at the output layer and works backwards to the hidden layer. The following iterations repeat until convergence in terms of the selected error criterion. An iteration includes presenting an instance by submitting training patterns to the network, calculating activations, and adjusting weights.

Calculation of Activation

An instance or training set, presented to the network determines the activation level. The activation corresponds directly to the node threshold, whose initial set value changes. The idea urges teaching the network when to pass on the data to the next layer by adjusting its value during back-propagation, because only when the activation reaches the threshold value data passes on.

The below formulas for the level of activation match those defined as the sum function above, with the exception of a node threshold negative in this case. This results from experimentation, which found the results of the later to yield less error.

Hidden Layer Activation Level---

Hidden Layer Output($a^{2}{}_{S}{}^{2}$) = Hidden Layer Sigmoid Function(f^{2}) {Sum Function($\sum$) [Input-Layer Weight($iw^{2,1}{}_{S}{}^{2}{}_{R}{}^{1}$) * Input($p_{R}{}^{1}$) – Hidden Layer Node Threshold(θ_{h})] }

Simplified: $a_{h} = f \{ \sum [(iw) (p) - \theta_{h}]\}$

where subscript “$_{h}$” refers to the hidden layer

Adjusting the Hidden Layer Node Threshold

$\theta_{h} = \theta_{h} + (\alpha)(\delta_{h})$

where θ_h symbolizes the Hidden Layer Node Threshold, α the Learning Rate as later discussed in Chapter 5, and δ_h the Hidden Layer Error

Output Layer Activation Level---

Output($a^{3}{}_{S}{}^{3}$) = Function(f^{3}) {Sum Function($\sum$) [Hidden-Layer Weight($w^{3,2}{}_{S}{}^{3}{}_{S}{}^{2}$) * Hidden Layer Output($a^{2}{}_{S}{}^{2}$) – Output Layer Node Threshold(θ_{o})] }

Simplified: $a_{o} = f \{ \sum [(lw) (a_{h}) - \theta_{h}]\}$

where subscript “$_{h}$” refers to the hidden layer and subscript “$_{o}$” to the output layer

Adjusting the Output Layer Node Threshold

$\theta_{o} = \theta_{o} + (\alpha)(\delta_{o})$

where θ_o signifies the Output Layer Node Threshold, α the Learning Rate, and δ_o the Output Layer Error

4.4.4 Weight Modifications

Input Weight Adjustment---

$\mathrm{iw}(t+1) = \mathrm{iw}(t) + \Delta \mathrm{iw}$

where *iw* represents the input weight at time *t*, or the "*t*"th epoch; and Δ*iw* the input weight change

Input Weight Change

Input Weight Change($\triangle \mathrm{iw}$) = Learning Rate(α) * Hidden Layer Error(δ_h) * Input($p_R{}^1$)

Simplified: $\triangle \mathrm{iw} = \alpha\ \delta_h p$

Hidden Layer Weight Adjustment---

$\mathrm{lw}(t+1) = \mathrm{lw}(t) + \Delta \mathrm{lw}$

where lw represents the hidden-layer weight at time t. or the "t"th epoch; and Δlw the hidden-layer weight change

Hidden Layer Weight Change

Hidden Layer Weight Change($\triangle \mathrm{lw}$) = *Learning Rate*(α) * *Output Layer Error*(δ_o) * *Hidden Layer Output*($a^2 s^2$)

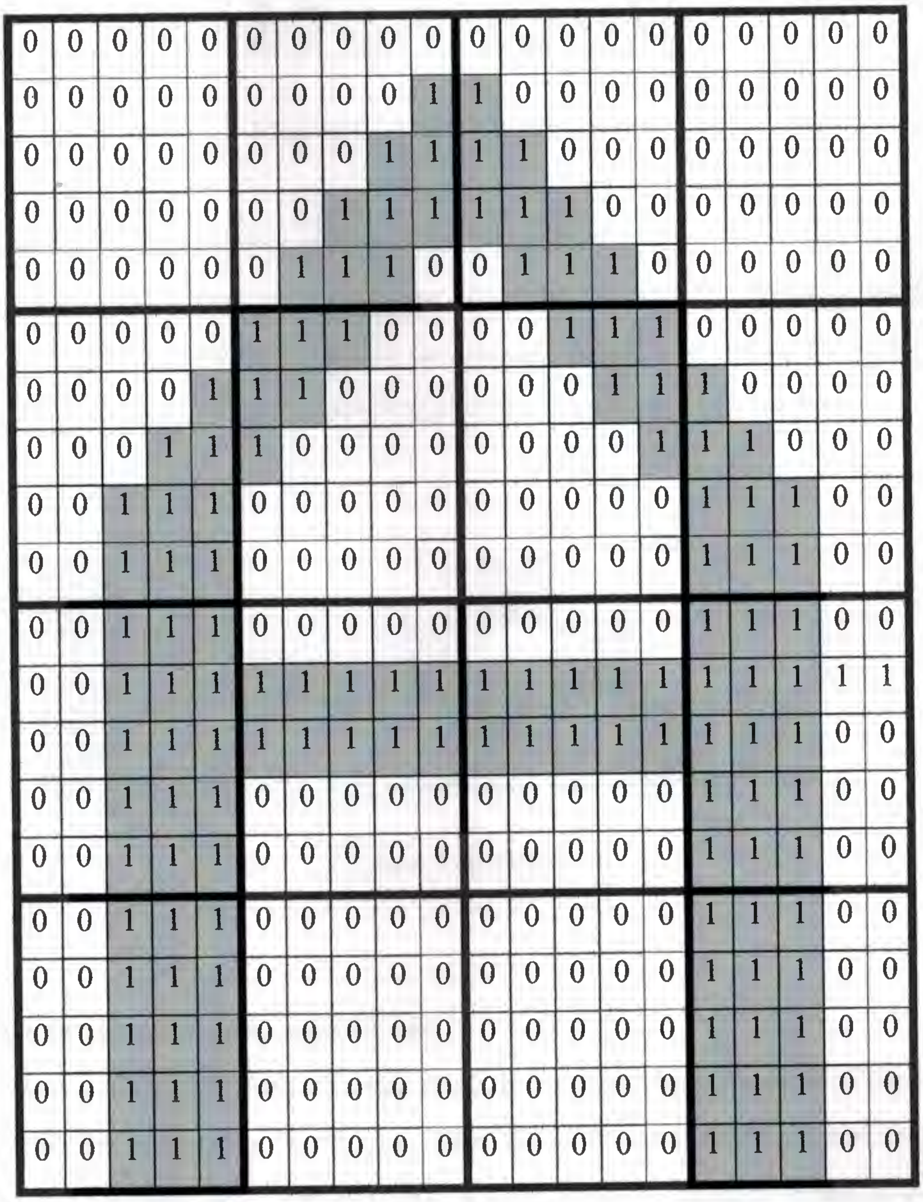
Simplified: $\triangle \mathrm{lw} = \alpha\ \delta_o a_h$

4.5 Manual Compression

A large matrix size may not train and run in real time due to limited memory. To solve this problem, pre-processing in order to cut down the size of the vectors must take place in order to help generalization.

4.5.1 Creating Sub-Matrices

The character matrices brake down into equal-sized sub-matrices. For example, a 20 x 20 matrix brakes down into sixteen 5 x 5 sub-matrices as shown in figure 4.12. Additionally, figure 4.13 shows the number of each sub-matrix.



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0

F

Figure 4.12 Character A Divided into equal Sub-Matrices

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 4.13 Sub-Matrix Number

4.5.2 Generalizing Features within Each Sub-Matrix

To generalize the matrices, it becomes important to note the features inside each sub-matrix. These combined create a compressed binary code. The features to look for include horizontal, vertical, and diagonal lines as shown in figure 4.14.

	1	
	1	
	1	

1	1	1

1		
	1	
		1

		1
	1	
1		

Figure 4.14 Features

4.5.3 Compressed Binary Input Code

To create the compressed binary input code, begin by looking for each of the four features within each of the 16 regions. Then those features present record as a one, and those absent as a zero. For example, region 2 of the character *A* matrix in figure 4.13 contains only a right-slanted diagonal feature. Thus, the sequence code for vector 2 becomes (0,0,0,1) as shown in figure 4.15. This search process repeats for all 16 regions.

Sub-Matrix No.	1	2	3	4	5	6	7	8
Feature	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /
Code Sequence	0000	0001	0010	0000	1001	0001	0010	1010
Sub-Matrix No.	9	10	11	12	13	14	15	16
Feature	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /	| – \ /
Code Sequence	1000	0100	0100	1000	1000	0000	0000	1000

Figure 4.15 Code Sequence for each Sub-Matrix within Character A

The combined code sequences of each sub-matrix creates the compressed binary input code, and replaces the 400-input vector derived from the original non-compressed matrices. Hence, 16 sub-matrices multiplied by 4 features results in a 64-bit compressed binary code
0000000100100000100100010010101010000100010010001000000000001000

Slightly distorted letters still map into the same feature-based binary code, thus revealing the caliber of this technique. For example, the following distortion of character *A* creates the same 64-vector input code:

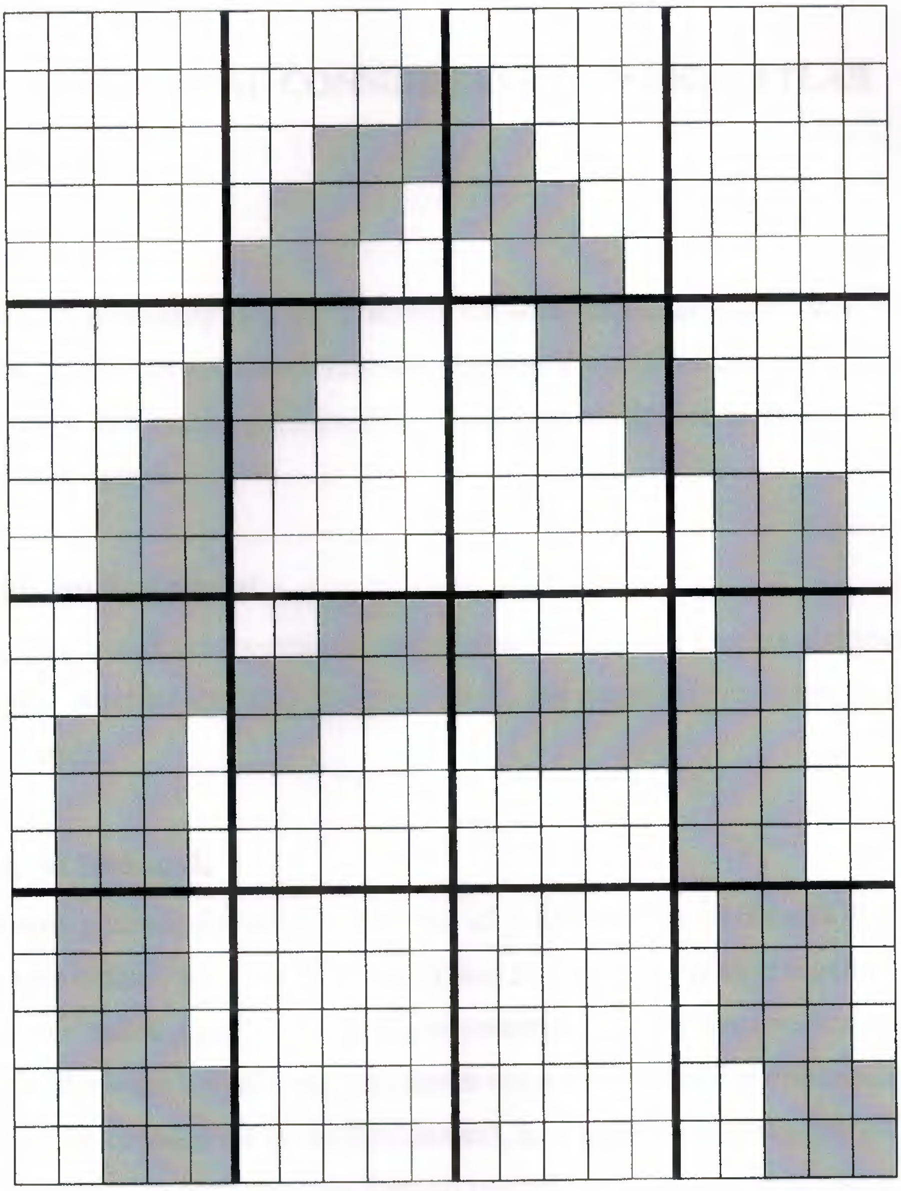


Figure 4.16 Distorted A

4.6 Summary

This chapter explained all necessary steps to design and implement a character recognition system by creating character matrices and network topology. The feed-forward and back-propagation algorithms in addition to training using a compression scheme were also discussed.

5. PRACTICAL CONSIDERATION USING MATLAB

5.1 Overview

This chapter practically applies a neural network implemented in MatLab to the character recognition problem. The neural network's architecture, training with and without noise, and system performance is discussed in addition to the presentation of the MatLab program.

5.2 Problem Statement

To design a network to recognize 26, upper-case letters of the English alphabet; and to give perfect classification of ideal input vectors, and reasonably accurate classification of noisy vectors.

5.3 Neural Network

The network receives the values within the 20 x 20 character matrix as a 400-element input vector defined as *p1* for A, *p2* for B, and so on. This presents together with the target vector called *targets*. Each target comprises 26 elements with a one in the position of the letter it represents, and zeroes everywhere else. For example, the target for the letter A contains a 1 in the first element, as A constitutes as the first letter of the alphabet, and 0's in elements two through twenty-six. After the networks trains, the output passes through the competitive transfer function *compet*. This makes sure that the output corresponding to the letter most like the noisy input vector takes on a value of 1, and all others receive a value of 0. The network should respond with the correct character matrix, with and without noise.

5.4 Architecture

The neural network consists of 400 inputs and 26 neurons in its output layer to identify the letters. The network constitutes a two-layer, log-sigmoid network with 52 hidden nodes. Selecting the non-linear log-sigmoid transfer function proved ideal for character recognition, especially when training with noise, because it limits outputs of the network to a small range whereas linear neurons can output any value. Figure 5.1 illustrates the network topology; refer to the previous chapter for full explanation of all variables.

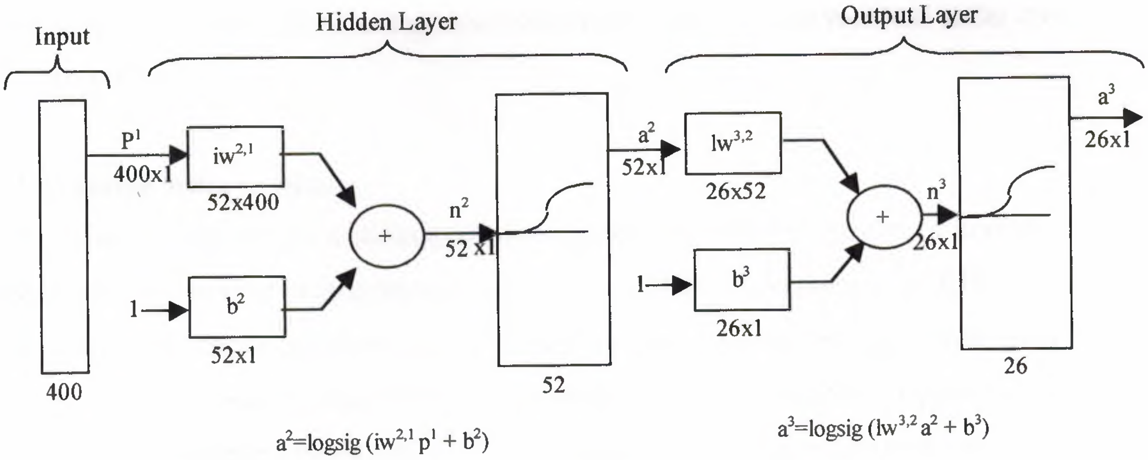


Figure 5.1 Neural Network Architecture

5.5 Initialization

The command *newff* creates a new feed-forward network. In addition, *R* indicates the number of inputs, *S1* the hidden layer neurons, *S2* and *Q* the nodes in the output layer which the initial script file *prprob* defines. The function *'traingdx'* employs fast back-propagation with a momentum term that speeds up the calculation of error.

```
S1 = 52;
[R,Q] = size(p1);
```

```
[S2,Q] = size(targets);
P = p1;
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

5.6 Training

To teach a network to handle noisy input vectors, the network receives training on both ideal and noisy vectors. To do this, the network first trains on ideal vectors until it reaches a low sum-squared error. Then, the network trains on 10 sets of ideal and noisy vectors. Finally, the network again trains on just ideal vectors to ensure perfect response to ideal letters. All training uses back-propagation with momentum under the function *'traingdx'*.

5.6.1 Training without Noise

The network initially trains without noise for a maximum of 5000 epochs, or until the network sum-squared error falls beneath a goal of 0.1. The sum-square error difference employs to adjust the connection weights of the neurons. During training, the network generates an output pattern which then compares with the target pattern. Depending on the difference between output and target, the sum-square error value computes. A learning rate of 0.003 multiplies by the negative of the error gradient to control the changes to the weights and biases, and a momentum constant of 0.85 is added to make calculation faster.

```
P = p1;
T = targets;
net.performFcn = 'sse';               Sum-squared error performance function
net.trainParam.goal = 0.1;            Sum-squared error goal.
net.trainParam.show = 20;             Frequency of progress displays in epochs.
net.trainParam.epochs = 5000;         Maximum number of epochs to train
net.trainParam.mc = 0.85;             Momentum constant
net.trainParam.lr = 0.003;          Learning rate
[net,tr] = train(net,P,T);
```

5.6.2 Training with Noise

To obtain a network not sensitive to noise, training occurs on two ideal copies and two noisy copies of the vectors in *p1*. The target vectors consist of four copies of the vectors in *targets*. The noisy vectors contain noise of mean 0.1 and 0.2 added to them. This forces the neuron to learn how to properly identify noisy letters, while requiring that it still respond well to ideal vectors. To train with noise, the maximum number of epochs reduces to 300 and the error goal increases to 0.6, reflecting the expectation of a higher error due to the presentation of more vectors.

```
 netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
P = [p1, p1, ...
    (p1 + randn(R,Q)*0.1), ...
    (p1 + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
```

5.7 System Performance

The reliability of the neural network pattern recognition system measures by testing the network with many input vectors with varying quantities of noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5, adds to input vectors. At each noise level, each letter receives 100 presentations of different noisy versions.

5.8 MatLab Program

MatLab used the following program to train and test the network. Figure 5.2 displays the performance goal, and figure 5.3 the percentage of recognition errors of two networks trained on various levels of noise. Notice that Network 1, trained without noise, contains slightly more errors resulting from noise than does Network 2, which trained with noise.

5.8.1 Defining and Initialization

The number of layers and nodes within each layer will be defined in the network in addition with the input, target, activation function, and learning algorithm. The script file *proprob* defines a matrix *alphabet* which contains the bit maps of the 26 letters of the alphabet, and target vectors *targets* for each letter.

```
%DEFINING THE MODEL PROBLEM
[alphabet,targets] = prprob;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
% DEFINING THE NETWORK
S1 = 52;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
```

5.8.2 Training the Initial Network without Noise

The initial network will train on ideal character until it reaches the error goal and produces figure 5.2 which shows the time, or number of epochs, that it took the network to reach the error goal.

```
net.performFcn = 'sse';          % Sum-squared error performance function.
net.trainParam.goal = 0.1;       % Sum-squared error goal.
net.trainParam.show = 20;        % Frequency of progress displays (in epochs).
net.trainParam.epochs = 5000;    % Maximum number of epochs to train.
net.trainParam.mc = 0.85;        % Momentum constant.
net.trainParam.lr = 0.003;       % Learning rate.
P = alphabet;
T = targets;
[net,tr] = train(net,P,T);
% Produces figure5.2
```

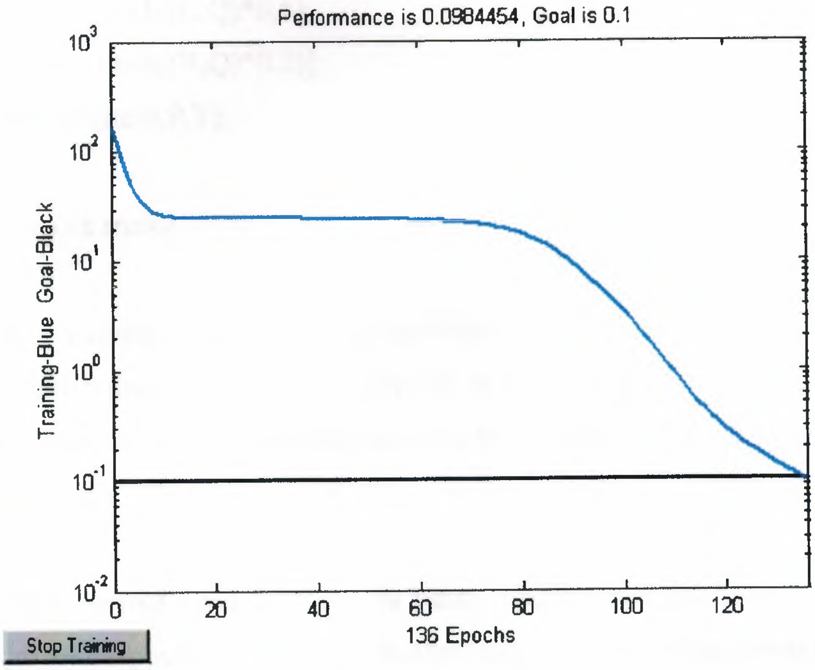


Figure 5.2 Training Performance

5.8.3 Training Network 2 with Noise

A copy of the initial network, called *Network 2*, will train with 10 noisy examples of the letters of the alphabet. The original initial network trained only on ideal vectors is called *Network 1*.

```
netn = net;
netn.trainParam.goal = 0.6;          % Mean-squared error goal.
netn.trainParam.epochs = 300;        % Maximum number of epochs to train.
T = [targets targets targets targets];
for pass = 1:10
fprintf('Pass = %.0f\n',pass);
P = [alphabet, alphabet, ...
    (alphabet + randn(R,Q)*0.1), ...
    (alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
% The network trains
```

5.8.4 Re-Training Network 2 without Noise

Network 2 is re-trained on ideal characters, as it was first trained without noise in the initial network, to ensure that it did not learn to classify noisy vectors at the expense of ideal ones.

```
netn.trainParam.goal = 0.1;          % Mean-squared error goal.
netn.trainParam.epochs = 500;        % Maximum number of epochs to train.
net.trainParam.show = 5;             % Frequency of progress displays (in epochs).
P = alphabet;
T = targets;
```

```
[netn,tr] = train(netn,P,T);
% The network trains
```

5.8.5 Testing Network 1 and 2 on Various Levels of Noise

Both networks, Network 1 trained only on ideal vectors, and Network 2 trained on ideal and noisy vectors, are tested to compare how well they perform when presented with various levels of noise. Figure 5.3 displays the results.

```
% SET TESTING PARAMETERS
noise_range = 0:.05:.5;                % Defines the various levels of noise
max_test = 100;
network1 = [];
network2 = [];
T = targets;
% INITIALIZE TESTING
for noiselevel = noise_range
fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
errors1 = 0;
errors2 = 0;
for i=1:max_test
P = alphabet + randn(400,26)*noiselevel;
% TEST NETWORK 1
A = sim(net,P);
AA = compet(A);
errors1 = errors1 + sum(sum(abs(AA-T)))/2;
% TEST NETWORK 2
An = sim(netn,P);
AAn = compet(An);
```

```
errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
end
% AVERAGE ERRORS FOR 100 SETS OF 26 TARGET VECTORS.
Network1 = [network1 errors1/26/100];
network2 = [network2 errors2/26/100];
end
% DISPLAY RESULTS
clf
plot(noise_range,network1*100,’—‘,noise_range,network2*100);
title(‘Percentage of Recognition Errors’);
xlabel(‘Noise Level’);
ylabel(‘Network 1 - -   Network 2 ---‘);
% Produces figure 5.3
```

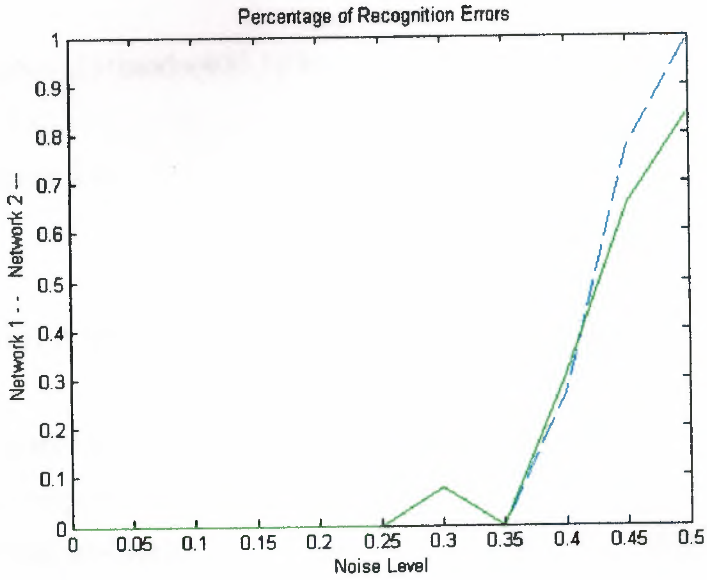


Figure 5.3 Percentages of Recognition Errors for Networks with and without Noise

5.8.6 Displaying Characters

After training and testing, the network is asked to answer with ideal and noisy characters as defined by the user. When the program examined it has been noted that when the noise level is given bigger than the maximum level the answer of the program is wrong recognition. To overcome this problem the following small mfile is added to compare the entered noise level with the maximum noise value. And if the noise level entered bigger than the maximum range (k) it will give a warning that the NOISE LEVEL IS TOO HIGH.

```
% CHARACTER A WITH NOISE
k=0.5
if(k>0.5)
disp('Noise Is Too High')
else
noisyA = alphabet(:,1)+randn(400,1)*k ;
plotchar(noisyA);
% Produces figure 5.4
```

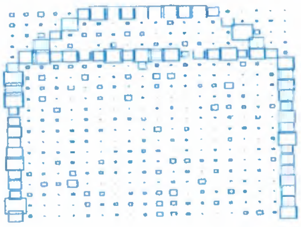


Figure 5.4 Character A with Noise

```
% CHARACTER A WITHOUT NOISE
figure;
A2 = sim(net,noisyA);
answer = find(compet(A2) == 1);
plotchar(alphabet(:,answer));
end
% Produces figure 5.5
```

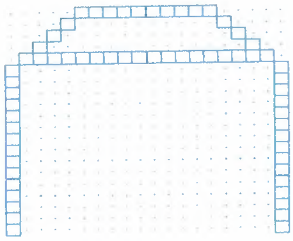


Figure 5.5 Character A without Noise

```
% CHARACTER Z WITH NOISE
k=0.5
if(k>0.5)
disp('Noise Is Too High')
else
noisyZ = alphabet(:,26)+randn(400,1)*k ;
plotchar(noisyZ);
% Produces figure 5.6
```

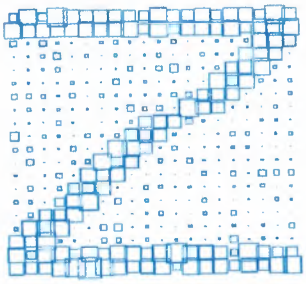


Figure 5.6 Character Z with Noise

```
% CHARACTER Z WITHOUT NOISE
figure;
A2 = sim(net,noisyZ);
answer = find(compet(A2) == 1);
plotchar(alphabet(:,answer));
end
% Produces figure 5.7
```

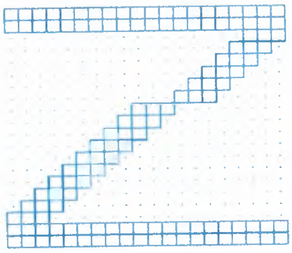


Figure 5.7 Character Z without Noise

5.9 Neural Network Final Parameters and the Learning Rate

The performance of the algorithm 'traingdx' is very sensitive to the proper setting of the learning rate. Table 5.1 shows particular final parameters as a result of the set learning rate. If the learning rate is set too high, the algorithm may become unstable. In this case, learning stops abruptly without reaching the maximum number of epochs nor the error goal. On the other hand, if the learning rate is too small, the algorithm will either take too long to converge or not converge at all – possibly exceeding the maximum number of epochs before the error goal has been reached. It is not practical to determine the optimal setting for the learning rate before training, several training sessions with different learning rate settings must first take place in order to determine the best to use. For this particular program, a learning rate of 0.003 proved best due to its fastest convergence to the error goal in the least epochs.

Input Layer	Hidden Layer	Output Layer	Momentum Rate	Learning Rate	Error Value	Epochs	Result
400	52	26	0.85	0.03	26	35	Unstable
400	52	26	0.85	0.006	0.095	145	OK
400	52	26	0.85	0.003	0.098	136	Optimal
400	52	26	0.85	0.0001	0.096	207	OK
400	52	26	0.85	0	168	5000	Does Not Converge

Table 5.1 Learning Rate Dependent Neural Network Final Parameters

5.10 Summary

This chapter presented a general MatLab program to train and recognize character with and without noise. The architecture, training, and system performance of the particular neural network was discussed.

CONCLUSION

In this thesis, character recognition has been studied and implemented by developing a MatLab program for all training, testing, and plotting functions. A suitable neural network topology was designed for the particular character recognition problem employing 20 x 20 matrices designed to represent the English alphabet in multiple fonts.

A 2-layer, feed-forward network designed in the MatLab software trained using back-propagation of errors to recognize the 26 upper-case, alpha characters of the English alphabet. The network topology consisted of 400 input nodes, one hidden layer with 52 nodes, and an output layer with 26 nodes. The squashing function 'logsig' applied because it produces either a 0 or 1 for all values between negative and positive infinity. Due its high performance, fast convergence, and use of the back-propagation algorithm, 'traingdx' defined the training function.

The neural network was trained with and without noise by using the back-propagation algorithm in order to teach the neural network to correctly classify ideal and noisy characters. The neural network trained and learned to correctly classify characters with and without noise. After defining the network and model problem, the network initially gave random and incorrect answers. During learning, this dissolved as the network "taught" itself to give the desired output by internally adjusting its weights and biases. Thus, endowing the computer with some sort of brain-like function proved successful.

The performance was compared between a network trained on only ideal vectors, and that trained on both ideal and noisy vectors. Network 1 trained only on ideal vectors, whereas Network 2 trained on both ideal and noisy vectors in order to compare performance results when presented with noise. Although not significantly different, Network 2 performed slightly better. This indicates that the network acquired the ability to estimate when classifying characters.

Teaching a machine to recognize characters when presented with noise may additionally be applied to future applications and developments of character recognition, such as Optical Character Recognition in scanners and faxes. Scanners and faxes must deal with noisy or distorted data since transmission leaves documents often blurry or out of line. This requires some level of tolerance of imperfect characters on part of these systems.

APPENDIX I

	First Font	Second Font
Character A	00000000000000000000 00000000011000000000 00000000111100000000 00000001111110000000 00000011111111000000 00000111000011100000 00001110000001110000 00011100000000111000 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00000000000000000000
Character B	00000000000000000000 00111111111111000000 00111111111111100000 00111111111111110000 00111000000000111000 00111000000000011100 00111000000000111000 00111000000000110000 00111111111111100000 00111111111111000000 00111111111111100000 00111000000001110000 00111000000000111000 00111000000000011100 00111000000000111000 00111000000000110000 00111000000001110000 00111111111111100000 00111111111111000000 00000000000000000000	00000000000000000000 00111111111111000000 00111111111111110000 00111000000111111000 00111000000011111100 00111000000001111100 00111000000000111100 00111000000000001f1100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000111100 00111000000001111100 00111000000011111000 00111111111111110000 00111111111111100000 00000000000000000000

	First Font	Second Font
Character C	00000000000000000000 00000001111110000000 00000011111111000000 00000111111111100000 00001110000001111000 00011100000000111100 00111000000000001100 00111000000000000100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000100 00111000000000001100 00111000000000011100 00111000000001111000 00111111111111111000 00111111111111110000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00000000000000000000
Character D	00000000000000000000 00111111111000000000 00111111111100000000 00111111111110000000 00111000000111000000 00111000000011100000 00111000000001110000 00111000000000111000 00111000000000011100 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000011100 00111000000000111000 00111000000001110000 00111111111111100000 00111111111111000000 00111111111100000000 00000000000000000000	00000000000000000000 00111111111111100000 00111111111111110000 00111111111111111000 00111000000011111100 00111000000001111100 00111000000000111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000111100 00111000000001111100 00111000000011111100 00111111111111111000 00111111111111110000 00111111111111100000 00000000000000000000 00000000000000000000 00000000000000000000

	First Font	**Second Font**
Character E	00000000000000000000 00111111111111111100 00111111111111111110 00111111111111111100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111100000000 00111111111110000000 00111111111100000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111100000 00111111111111110000 00111111111111100000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111111100 00111111111111111100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00000000000000000000
Character F	00000000000000000000 00111111111111111000 00111111111111111100 00111111111111111000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111000000 00111111111111100000 00111111111111000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00010000000000000000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111111100 00111111111111111100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00000000000000000000

	First Font	Second Font
Character G	00000000000000000000 00000001111110000000 00000011111111000000 00000111111111100000 00001110000001110000 00011100000000111000 00111000000000111000 00111000000000111000 00111000000000010000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000111111000 00111000000111111000 00111100000000111000 00011111000000111000 00011111111111110000 00000111111111000000 00000011111110000000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000111111100 00111000000111111100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00000000000000000000
Character H	00000000000000000000 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00000000000000000000	00000000000000000000 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00000000000000011100 00000000000000011100 00000000000000011100 00000000000000000000

	First Font	**Second Font**
Character I	00000000000000000000	00000000000000000000
	00000111111111000000	00000000111000000000
	00000111111111000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000111111111000000	00000111111111000000
	00000111111111000000	00000111111111000000
	00000000000000000000	00000000000000000000
	00000000000000000000	00000000000000000000
Character J	00000000000000000000	00000000000000000000
	00000111111111000000	00000111111111000000
	00000111111111000000	00000111111111000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00000000111000000000	00000000111000000000
	00010000111000000000	00000000111000000000
	00111000111000000000	00000000111000000000
	00011111111000000000	00111111111000000000
	00001111111000000000	00111111111000000000
	00000000000000000000	00000000000000000000
	00000000000000000000	00000000000000000000

	First Font	**Second Font**
Character K	00000000000000000000 00111000000011110000 00111000000111100000 00111000001110000000 00111000011100000000 00111000111000000000 00111001110000000000 00111011100000000000 00111111000000000000 00111111100000000000 00111111110000000000 00111001111000000000 00111000111000000000 00111000011100000000 00111000001110000000 00111000000111000000 00111000000011100000 00111000000001110000 00111000000000111000 00000000000000000000	00000000000000000000 00111000000001111100 00111000000011111100 00111000000111000000 00111000001110000000 00111000011100000000 00111000111000000000 00111001110000000000 00111111110000000000 00111111110000000000 00111111110000000000 00111000111100000000 00111000011100000000 00111000001110000000 00111000000111000000 00111000000011100000 00111000000001110000 00111000000000111110 00111000000000011110 00000000000000000000
Character I	00000000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000110 00111111111111111100 00111111111111111000 00111111111111110000 00000000000000000000	00000000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111000000000000000 00111111111111111110 00111111111111111110 00000000000000000000

	First Font	Second Font
Character M	00000000000000000000 00111100000000011110 00111110000000111110 00111111000001111110 00111011100011101110 00111001110111001110 00111000111110001110 00111000011100001110 00111000001000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00000000000000000000 00000000000000000000	00000000000000000000 00111100000000011110 00111100000000011110 00111111000001111110 00111011000001101110 00111011100011101110 00111001110011001110 00111001111111001110 00111000111110001110 00111000011100001110 00111000001000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00000000000000000000 00000000000000000000
Character N	00000000000000000000 00111000000000001110 00111100000000001110 00111110000000001110 00111110000000001110 00111011100000001110 00111001110000001110 00111001110000001110 00111000111000001110 00111000111000001110 00111000011100001110 00111000001110001110 00111000000111001110 00111000000111001110 00111000000011101110 00111000000001111110 00111000000001111110 00111000000000111110 00000000000000000000 00000000000000000000	00000000000000000000 00111111000000001110 00111111000000001110 00111111100000001110 00111011100000001110 00111001110000001110 00111001110000001110 00111000111000001110 00111000111000001110 00111000011100001110 00111000011100001110 00111000000111001110 00111000000111001110 00111000000011101110 00111000000011101110 00111000000001111110 00111000000001111110 00111000000000111110 00000000000000000000 00000000000000000000

	First Font	Second Font
Character Q	00000000000000000000 00000111111111110000 00001111111111111000 00011111111111111100 00111111000001111110 00111110000000111110 00111100000000011110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111100001100001110 00111110000111111110 00111111000111111110 00011111111111111100 00001111111111111110 00000111111100001111 00000000000000000000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111100001100011100 00111110000111011100 00111111000011111100 00111111111111111100 00111111111111111110 00000000000000001111 00000000000000000000 00000000000000000000
Character R	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00111111000000000000 00111011100000000000 00111001110000000000 00111000111000000000 00111000011100000000 00111000001110000000 00111000000111000000 00111000000011100000 00000000000000000000	00000000000000000000 00111111111111100000 00111111111111110000 00111111111111111000 00111000000011111100 00111000000001111100 00111000000000111100 00111000000001111100 00111000000011111000 00111111111111110000 00111111111111100000 00111111000000000000 00111011100000000000 00111001110000000000 00111000111000000000 00111000011100000000 00111000001110000000 00111000000111110000 00111000000011110000 00000000000000000000

	First Font	**Second Font**
Character S	00000000000000000000 00000000000000000000 00000011111111110000 00000111111111111000 00001110000000011100 00011100000000001000 00111000000000000000 00111000000000000000 00111100000000000000 00011111111111100000 00001111111111111000 00000000000000011100 00000000000000001110 00000000000000011100 00010000000000111000 00111000000001110000 00111111111111100000 00011111111111000000 00000000000000000000 00000000000000000000	00000000000000000000 00000000000000000000 00111111111111111100 00111111111111111100 00111110000000011100 00011100000000001000 00111000000000000000 00111000000000000000 00111100000000000000 00111111111111111100 00111111111111111100 00000000000000011100 00000000000000011100 00000000000000011100 00010000000000011100 00111000000000011100 00111111111111111100 00111111111111111100 00000000000000000000 00000000000000000000
Character T	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000000000000000 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00110000111000001100 00110000111000001100 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000000111000000000 00000111111111000000 00000111111111000000 00000000000000000000

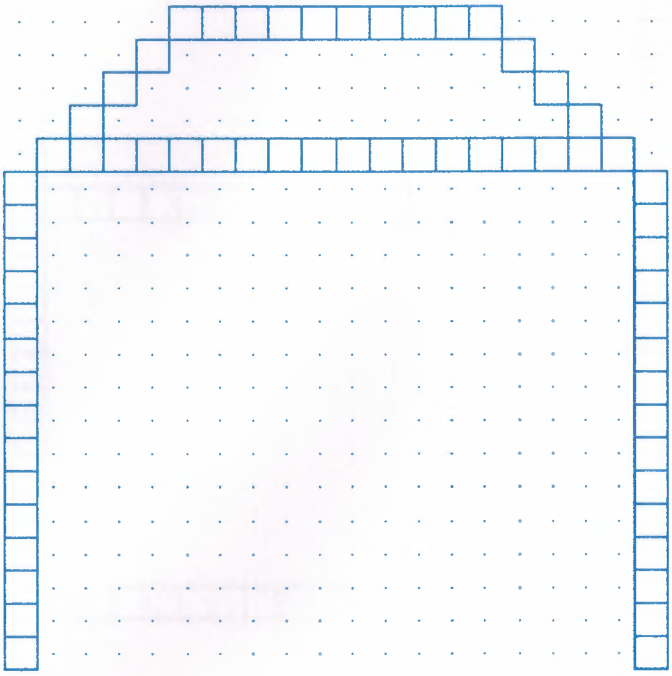
	First Font	Second Font
Character U	00000000000000000000	00000000000000000000
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111000000000011100
	00111000000000011100	00111100000000111100
	00111000000000011100	00111110000001111100
	00111000000000011100	00011111000011111000
	00111111111111111100	00001111111111110000
	00111111111111111100	00000111111111100000
	00000000000000000000	00000000000000000000
Character V	00000000000000000000	00000000000000000000
	00111000000000001110	00111000000000001110
	00111000000000001110	00111000000000001110
	00111000000000001110	00111000000000001110
	00011100000000011100	00111100000000011110
	00011100000000011100	00111100000000011110
	00001110000000111000	00011100000000011110
	00001110000000111000	00011110000000111110
	00000111000001110000	00011110000000111100
	00000111000001110000	00011110000000111100
	00000011100011100000	00011110000000111000
	00000011100011100000	00001111000001111000
	00000001110111000000	00001111000001111000
	00000001111111000000	00001111100011110000
	00000000111110000000	00000111100011100000
	00000000111110000000	00000011110111100000
	00000000011100000000	00000011111111000000
	00000000011100000000	00000001111110000000
	00000000001000000000	00000000011000000000
	00000000000000000000	00000000000000000000

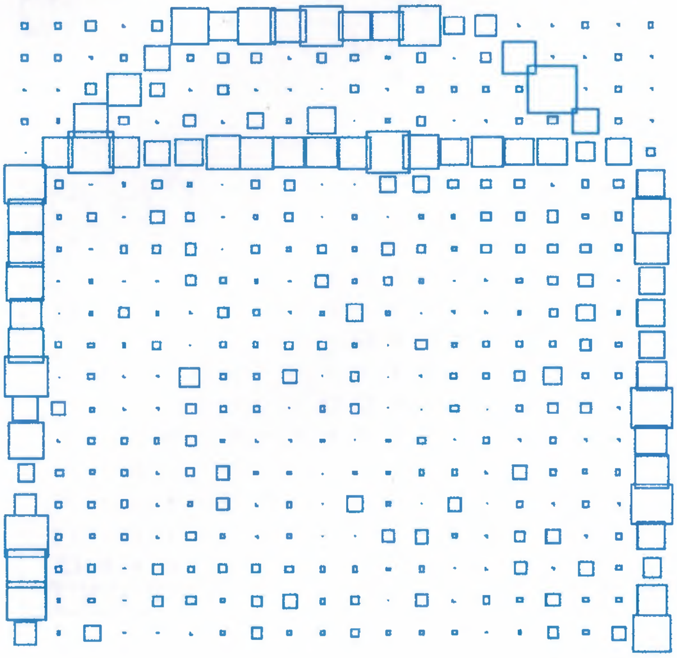
	First Font	Second Font
Character W	00000000000000000000	00000000000000000000
	01100000011000000110	01110000011000001110
	01100000011000000110	01110000011000001110
	01100000011000000110	01110000011000001110
	01100000011000000110	01110000011000001110
	00110000111100001100	01110000111100001110
	00110000111100001100	00111000111100011100
	00110000111100001100	00111000111100011100
	00110000111100001100	00111000111100011100
	00011001100110011000	00111001100110011100
	00011001100110011000	00111001100110011100
	00011001100110011000	00111001100110011100
	00011001100110011000	00111001100110011100
	00011111100111111000	00111111100111111100
	00001111000011110000	00011111000011111000
	00000110000001100000	00001110000001110000
	00000110000001100000	00001110000001100000
	00000000000000000000	00000100000000000000
	00000000000000000000	00000000000000000000
	00000000000000000000	00000000000000000000
Character X	00000000000000000000	00000000000000000000
	00111000000000001110	00111000000000001110
	00011100000000011100	00011100000000011100
	00001110000000111000	00001110000000111000
	00000111000001110000	00000111000001110000
	00000011100011100000	00000011100011100000
	00000001111111000000	00000001110111000000
	00000000111110000000	00000000111110000000
	00000000111110000000	00000000011100000000
	00000000111110000000	00000000111110000000
	00000001111111000000	00000001110111000000
	00000011100011100000	00000011100011100000
	00000111000001110000	00000111000001110000
	00001110000000111000	00001110000000111000
	00011100000000011100	00011100000000011100
	00111000000000001110	00111000000000001110
	01110000000000000110	01110000000000000111
	01110000000000000000	01110000000000000111
	00000000000000000000	00000000000000000000
	00000000000000000000	00000000000000000000

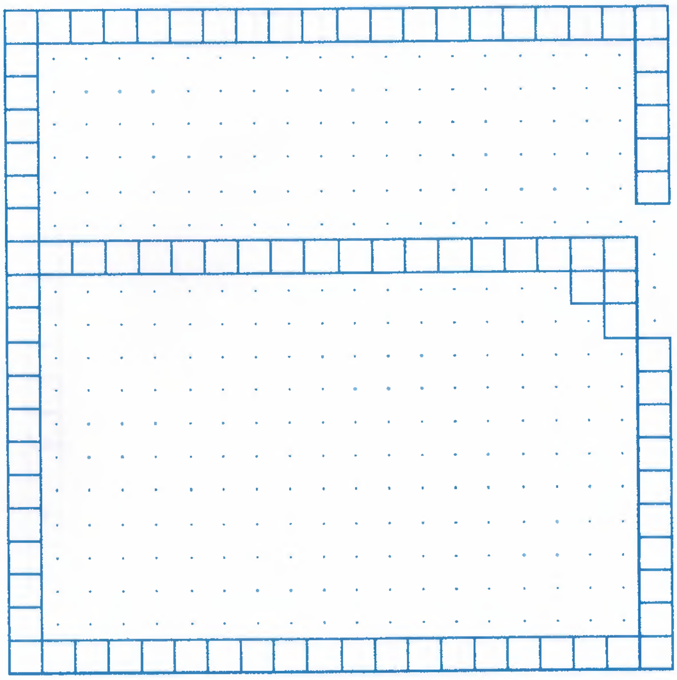
	First Font	**Second Font**
Character Y	00111100000000011110	00000000000000000000
	00011110000000011110	00111000000000001110
	00001111000000011110	00011100000000011100
	00000111100000011100	00001110000000111000
	00000011110001111000	00000111000001110000
	00000001111011111000	00000011100011100000
	00000000111111110000	00000011111111000000
	00000000011111100000	00000001111110000000
	00000000011111000000	00000000111100000000
	00000000011111000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000011110000000	00000000111100000000
	00000000000000000000	00000000111100000000
	00000000000000000000	00000000111100000000
	00000000000000000000	00000000000000000000
Character Z	00000000000000000000	00000000000000000000
	00111111111111111110	00011111111111111110
	00111111111111111110	00111111111111111110
	00111111111111111110	00011111111111111110
	00000000000000011110	00000000000000111110
	00000000000000111000	00000000000001111000
	00000000000001110000	00000000000011110000
	00000000000011100000	00000000000111100000
	00000000000111000000	00000000001111000000
	00000000001110000000	00000000011110000000
	00000000011100000000	00000000111100000000
	00000000111000000000	00000001111000000000
	00000001110000000000	00000011110000000000
	00000011100000000000	00000111100000000000
	00000111000000000000	00001111000000000000
	00001110000000000000	00111110000000000000
	00111111111111111110	00111111111111111100
	00111111111111111110	00111111111111111110
	00111111111111111110	00111111111111111100
	00000000000000000000	00000000000000000000

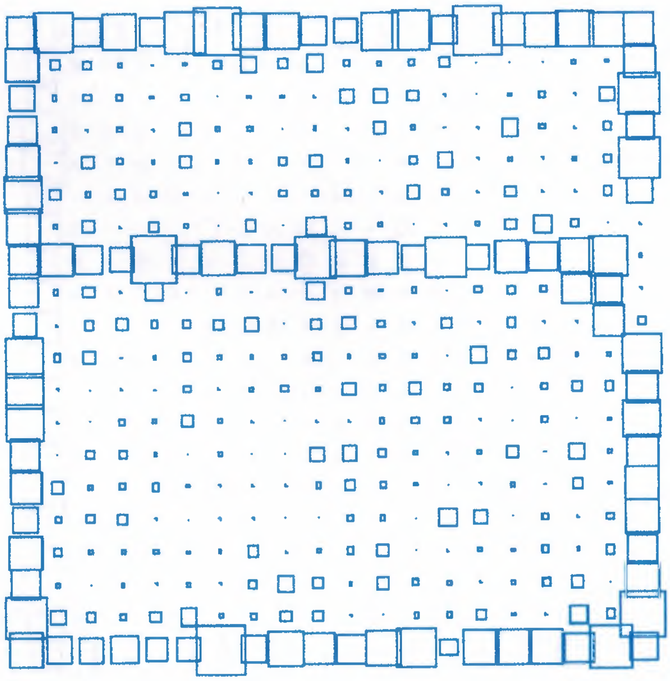
Character	Target Vector
A	10000000000000000000000000
B	01000000000000000000000000
C	00100000000000000000000000
D	00010000000000000000000000
E	00001000000000000000000000
F	00000100000000000000000000
G	00000010000000000000000000
H	00000001000000000000000000
I	00000000100000000000000000
J	00000000010000000000000000
K	00000000001000000000000000
L	00000000000100000000000000
M	00000000000010000000000000
N	00000000000001000000000000
O	00000000000000100000000000
P	00000000000000010000000000
Q	00000000000000001000000000
R	00000000000000000100000000
S	00000000000000000010000000
T	00000000000000000001000000
U	00000000000000000000100000
V	00000000000000000000010000
W	00000000000000000000001000
X	00000000000000000000000100
Y	00000000000000000000000010
Z	00000000000000000000000001

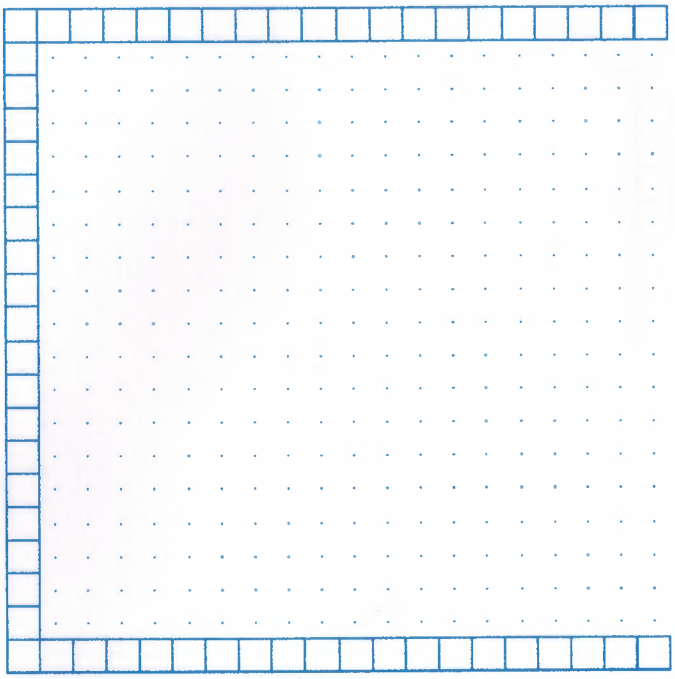
APPENDIX II

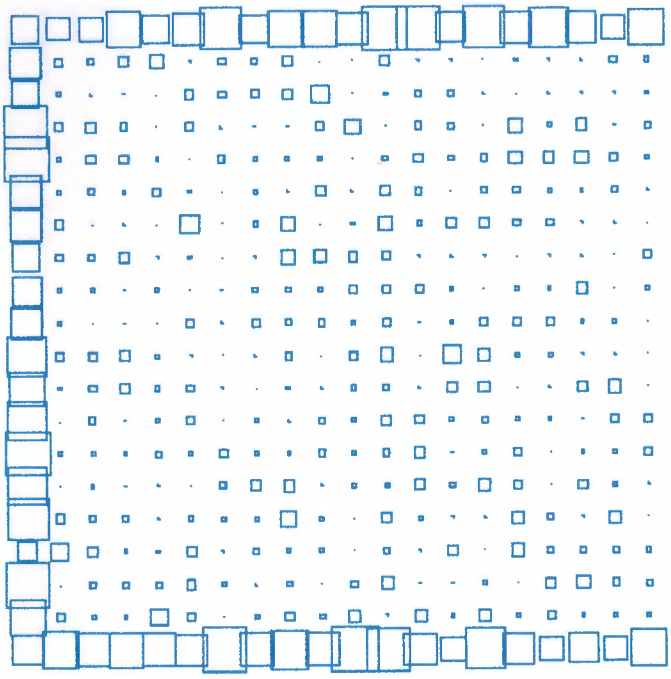


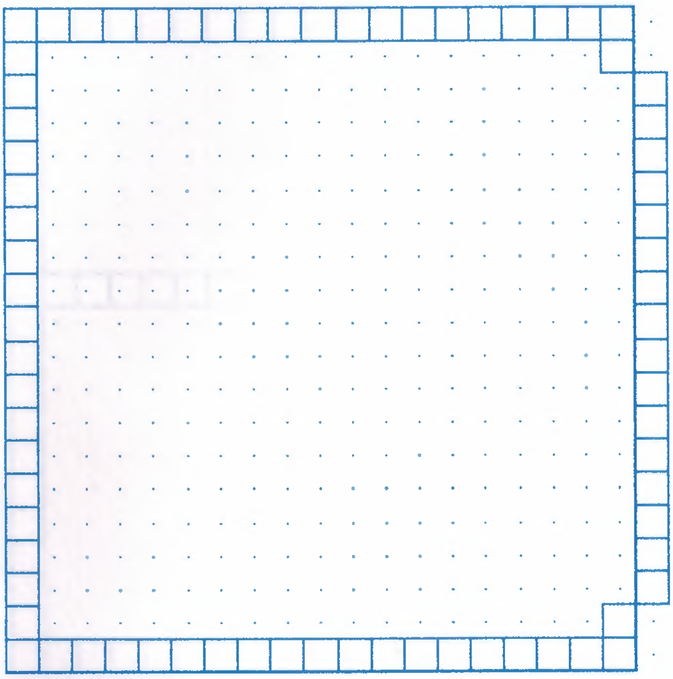


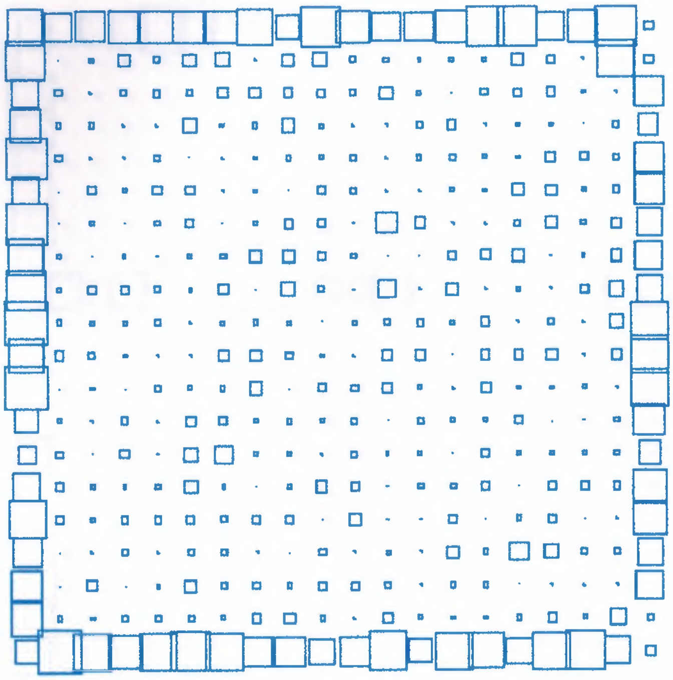


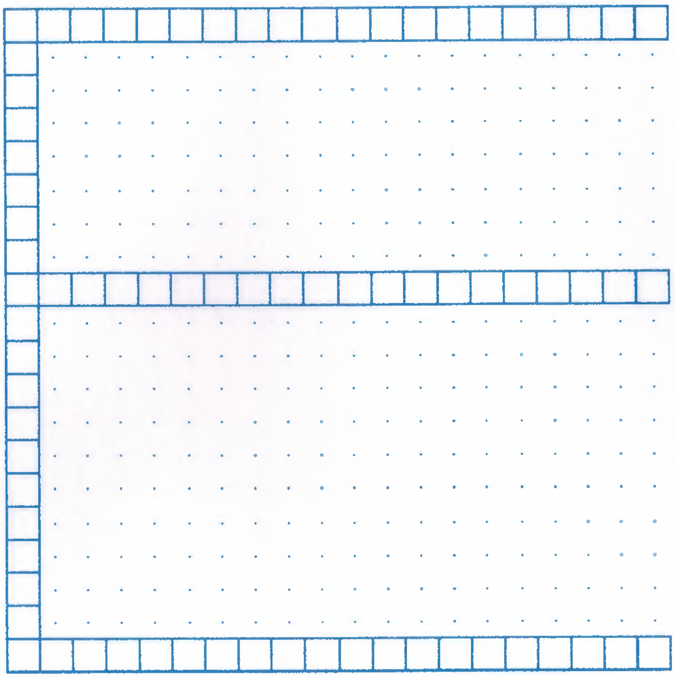


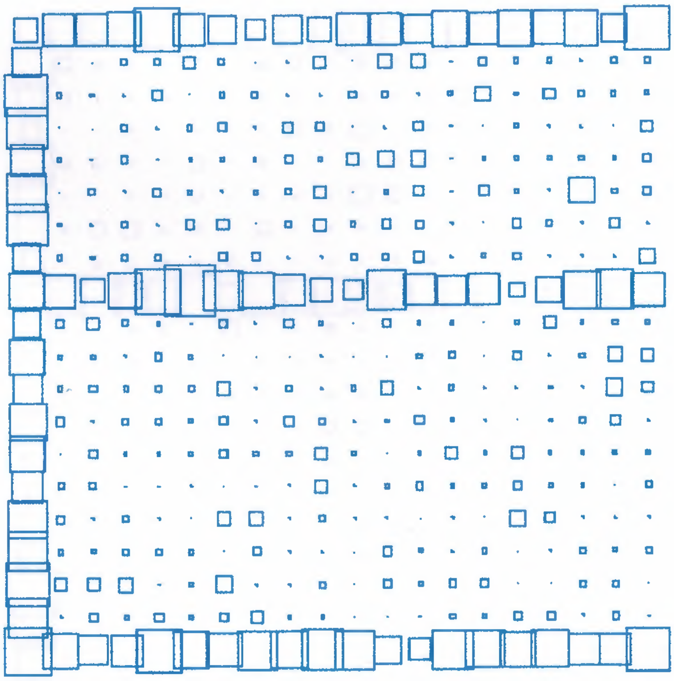


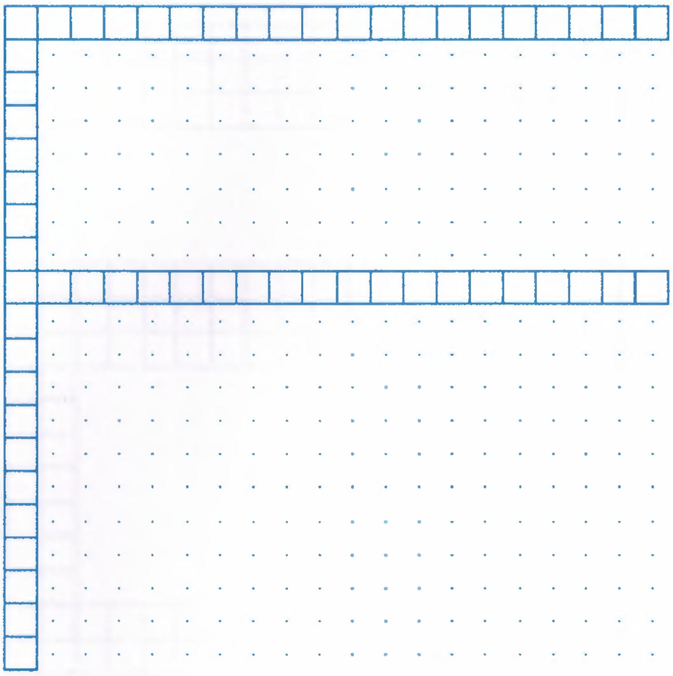


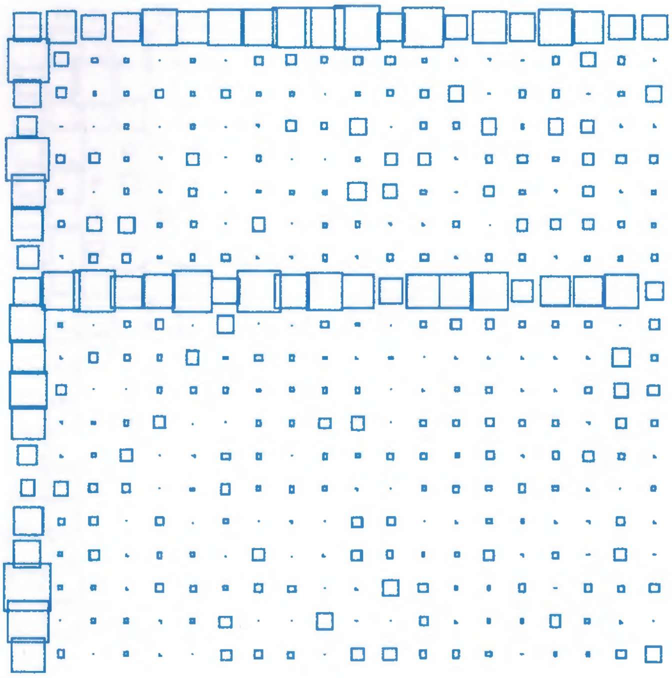


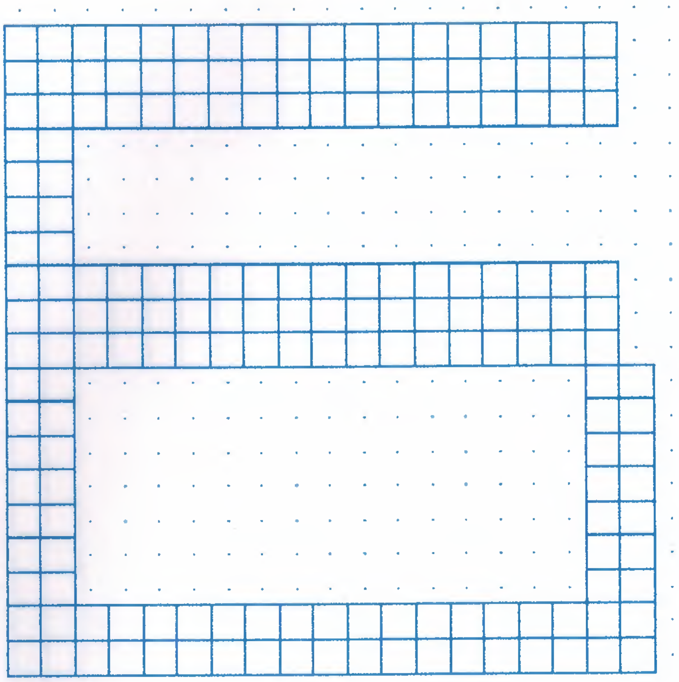


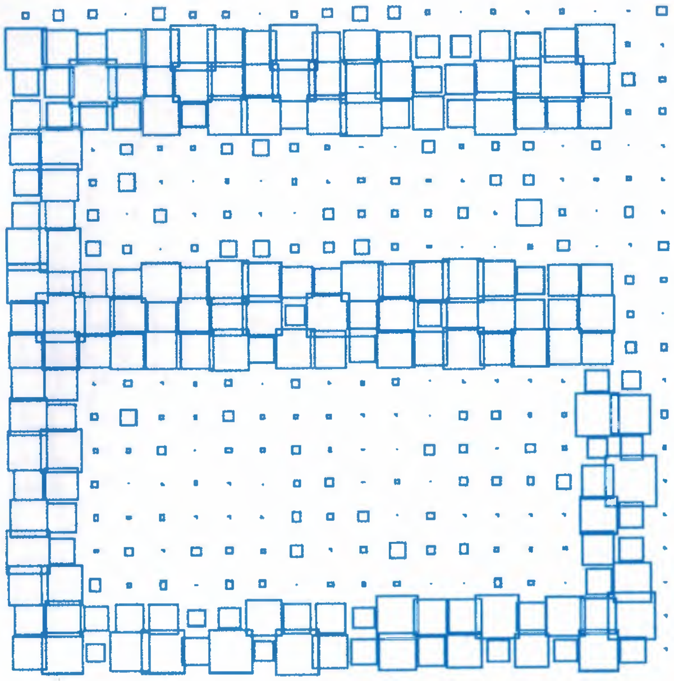


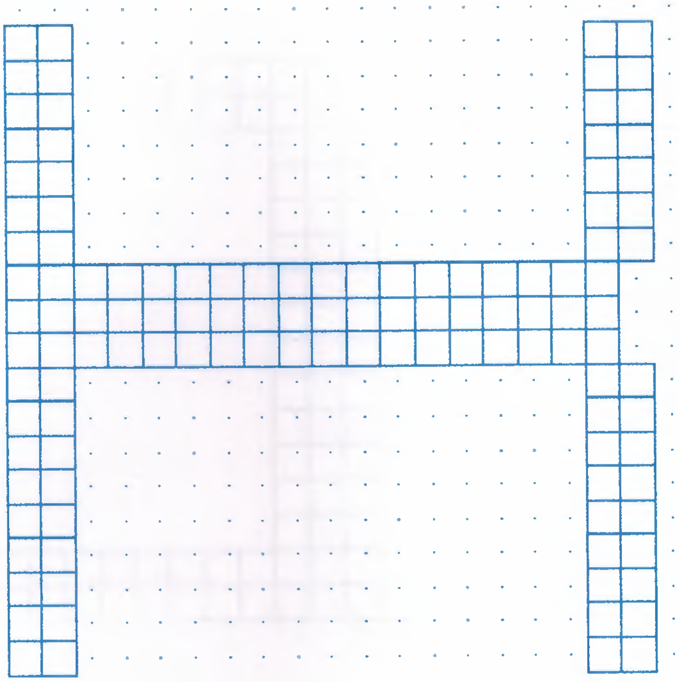


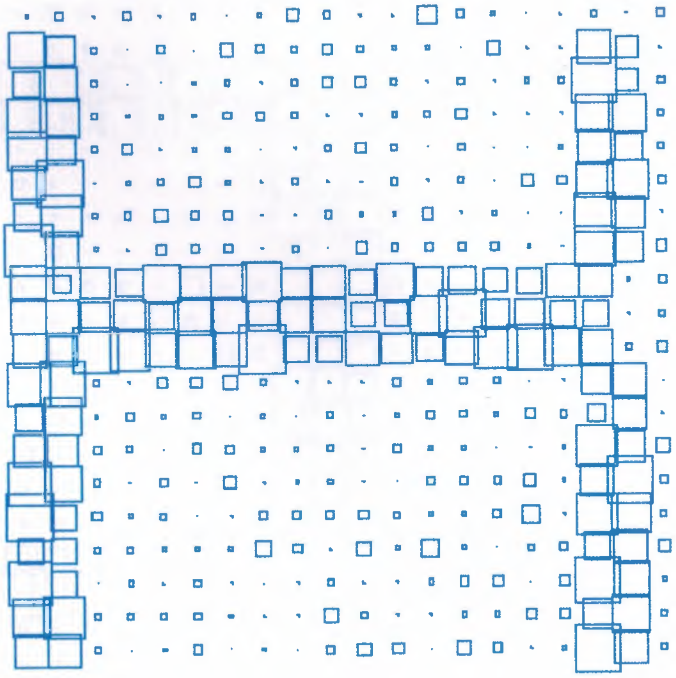


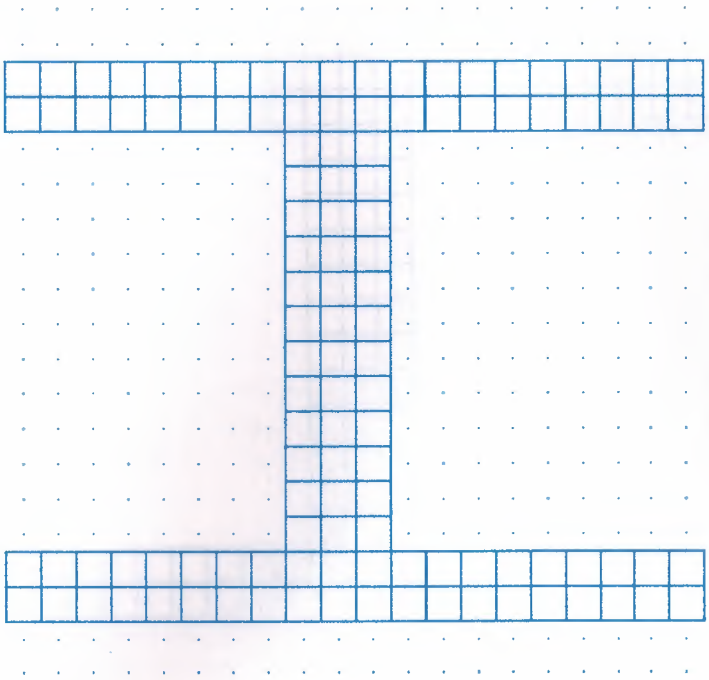


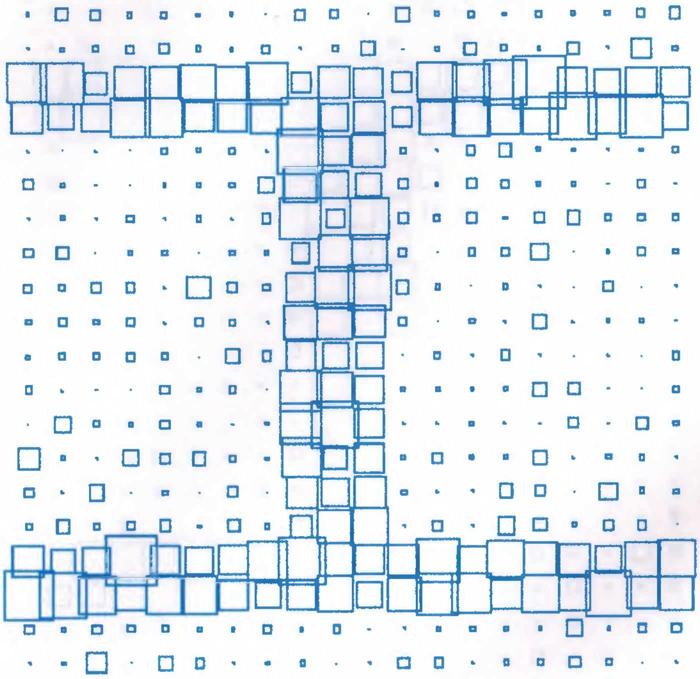


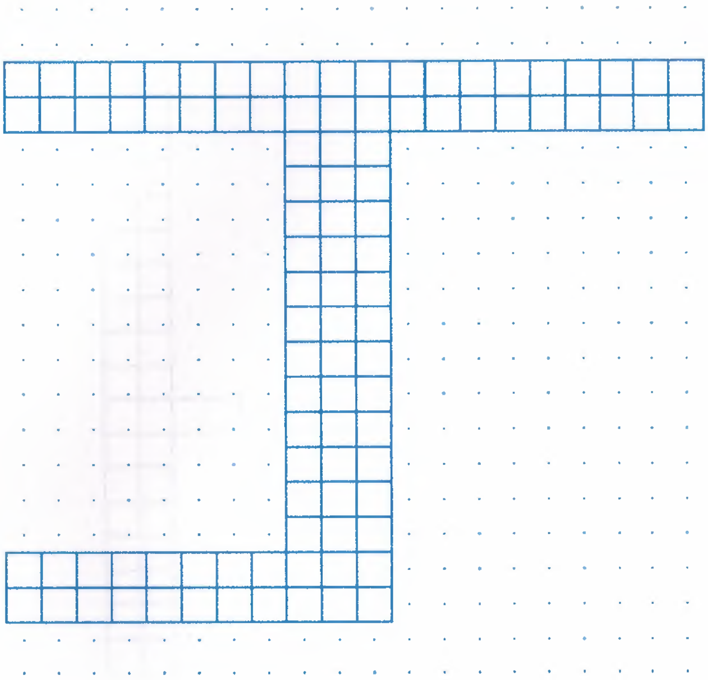


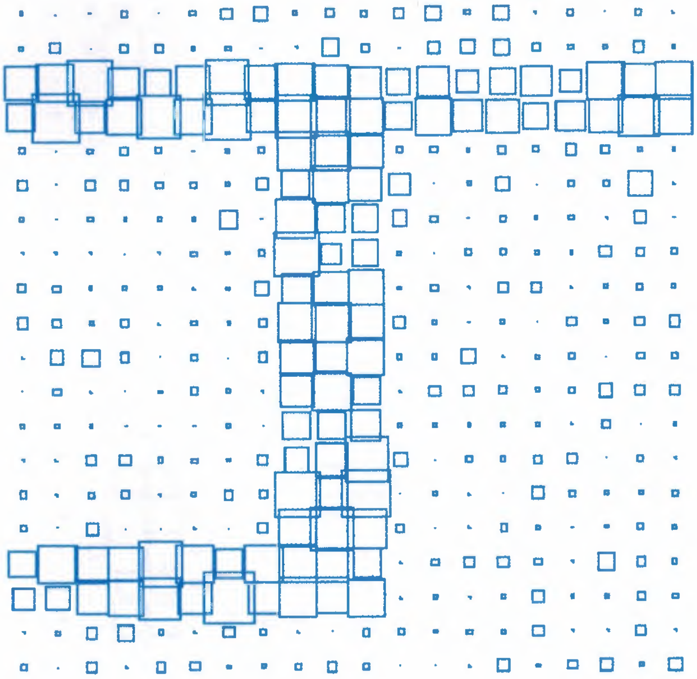


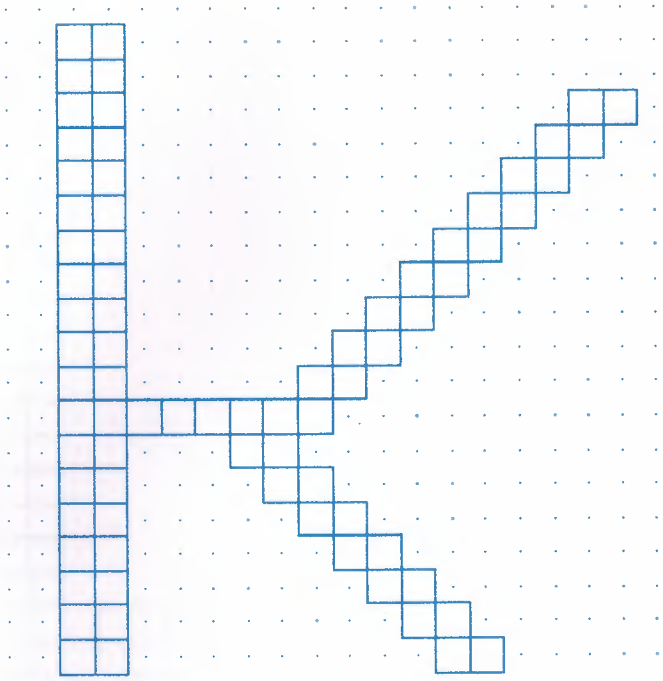


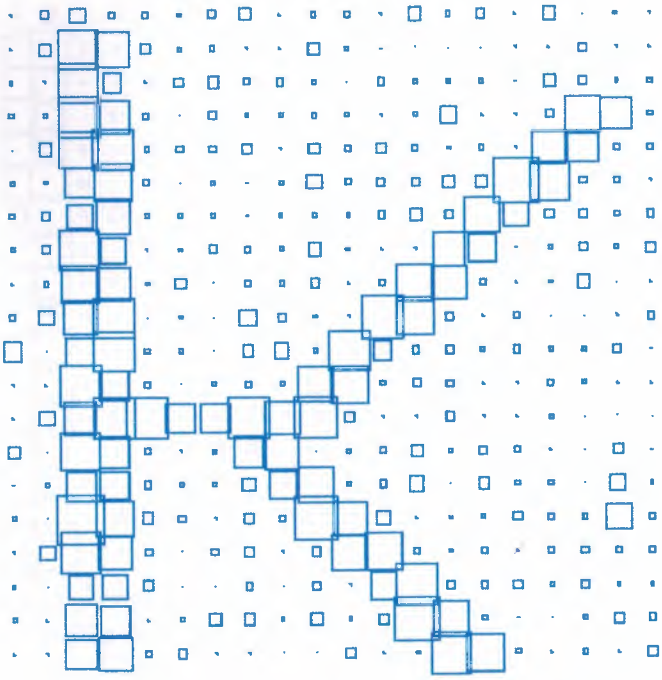


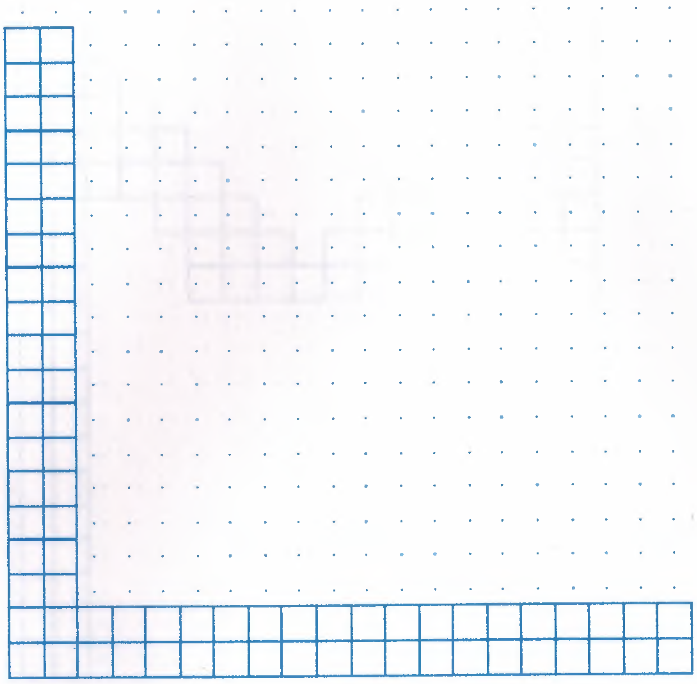


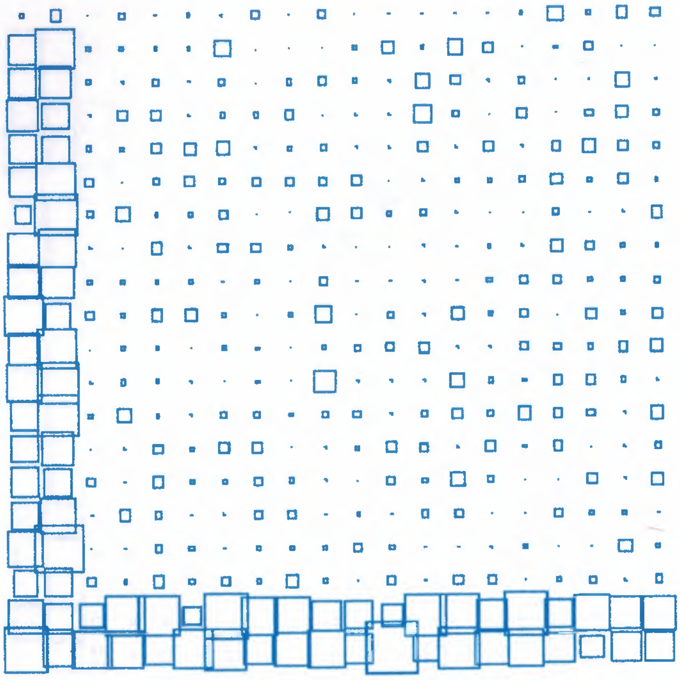


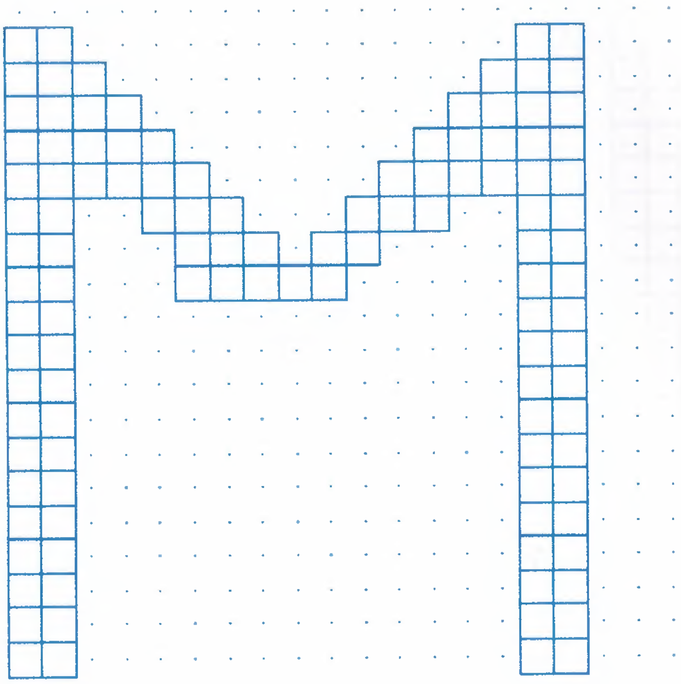


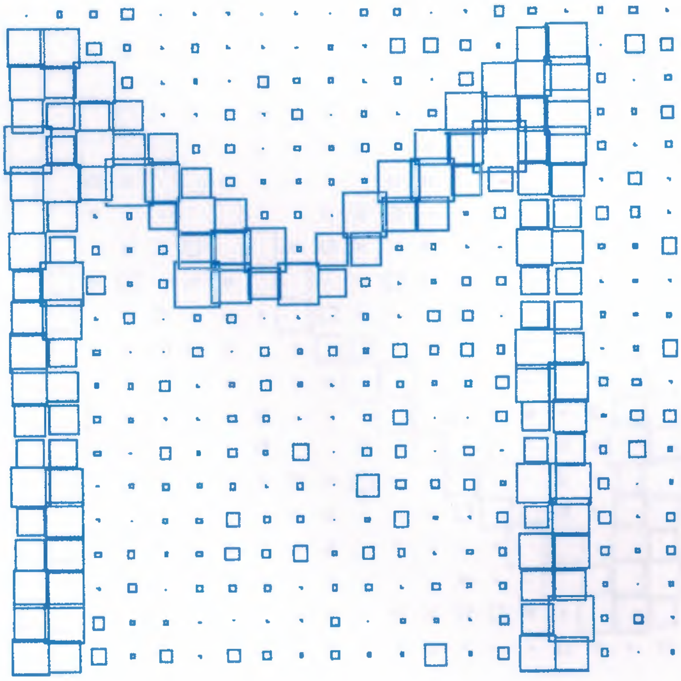


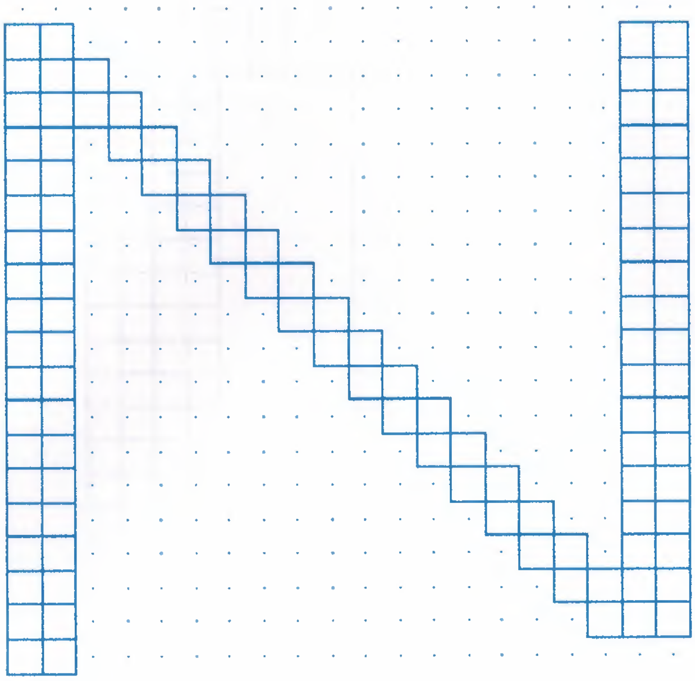


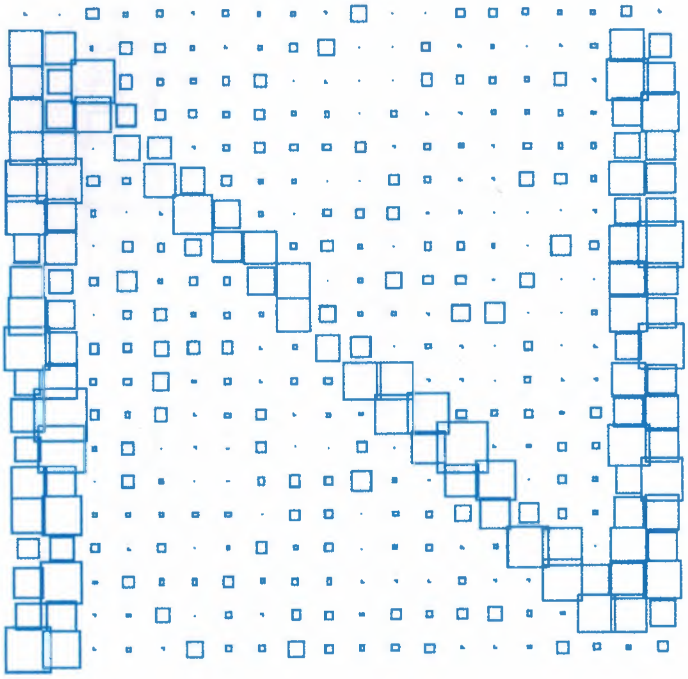


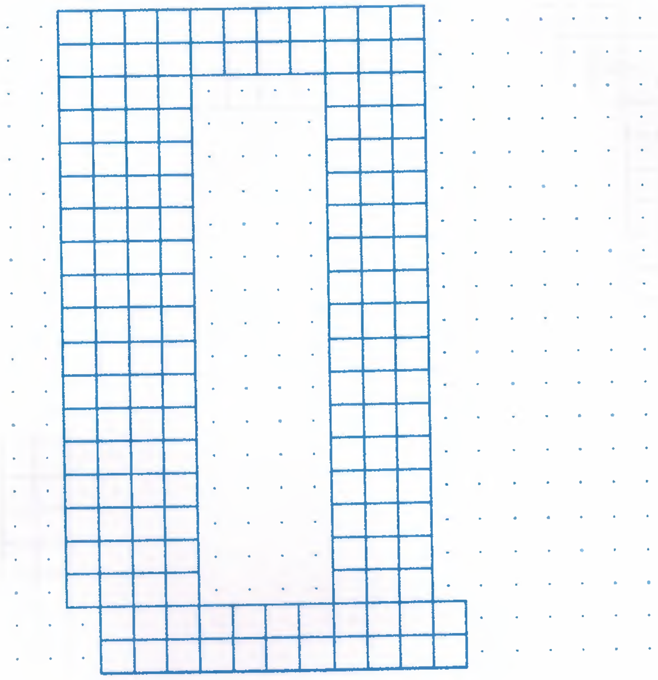


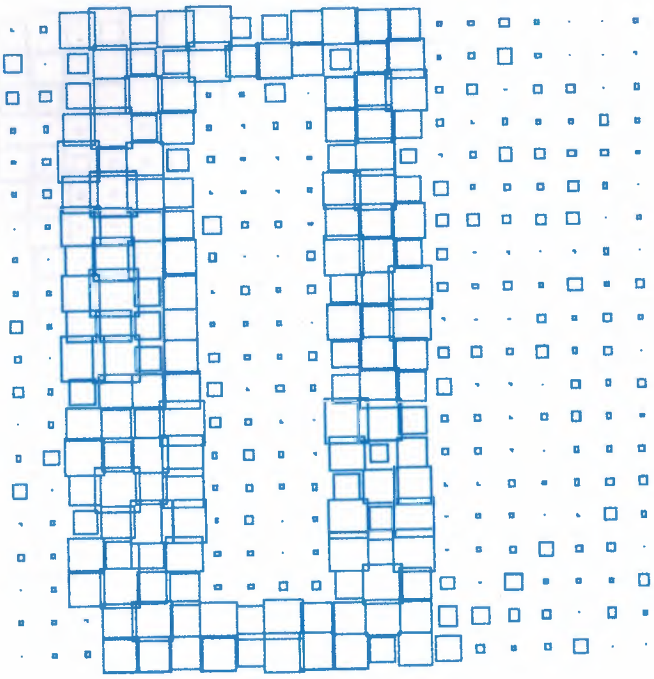


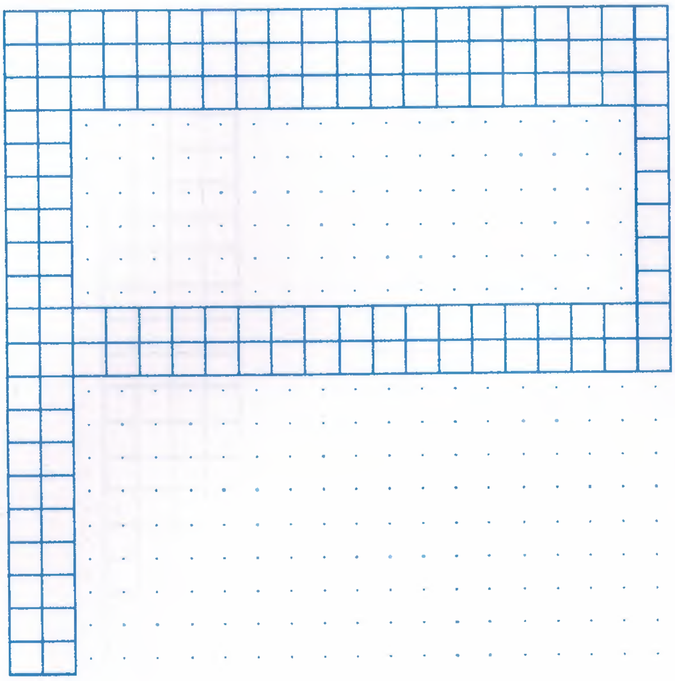


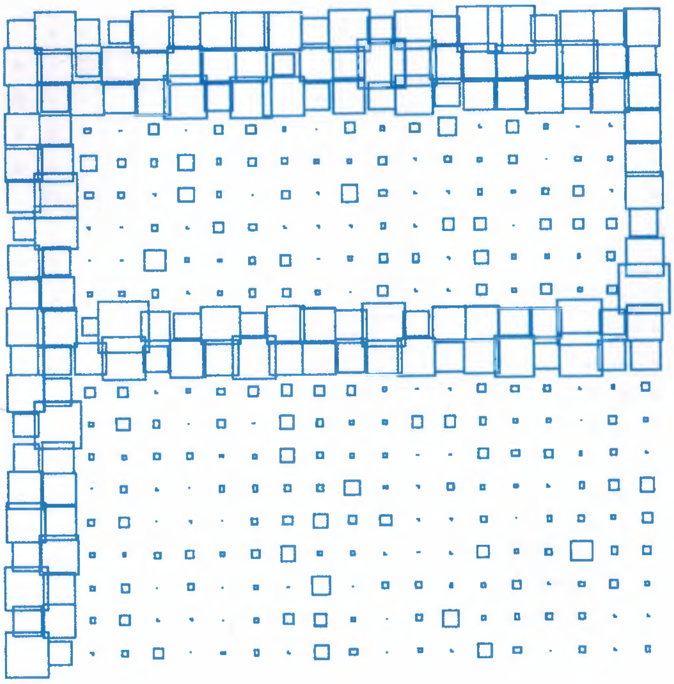


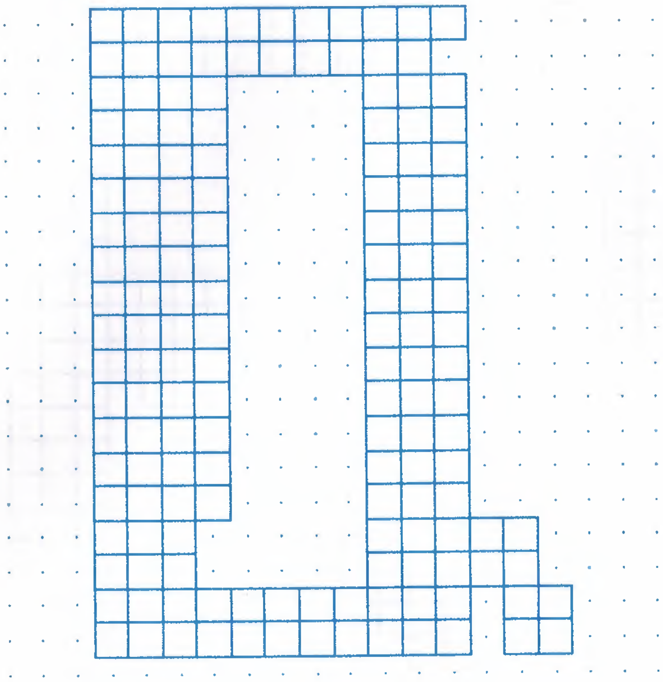


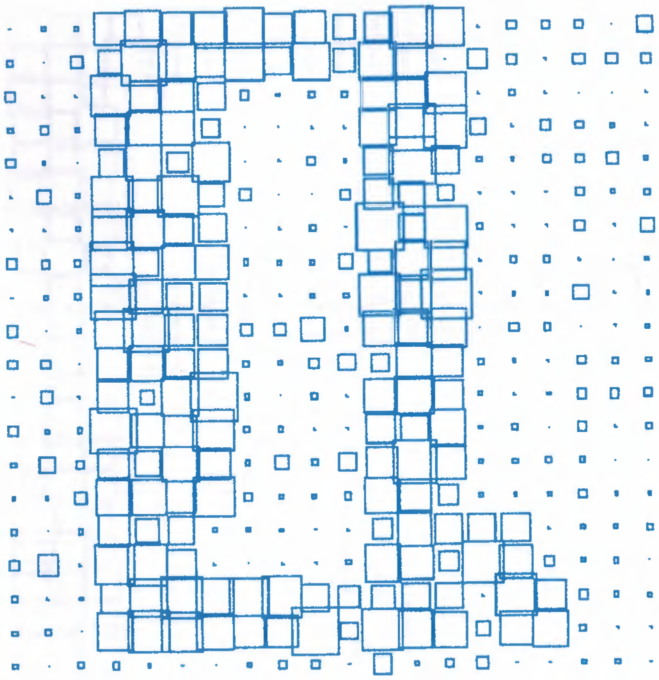


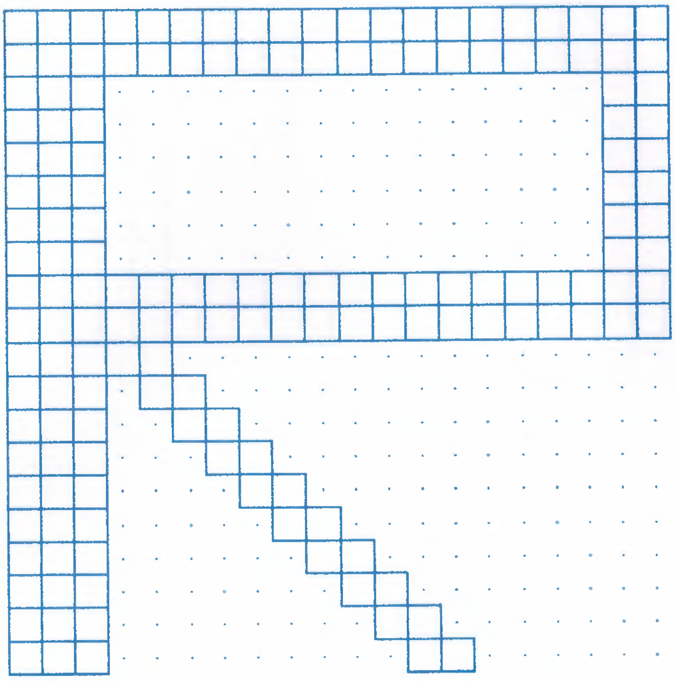


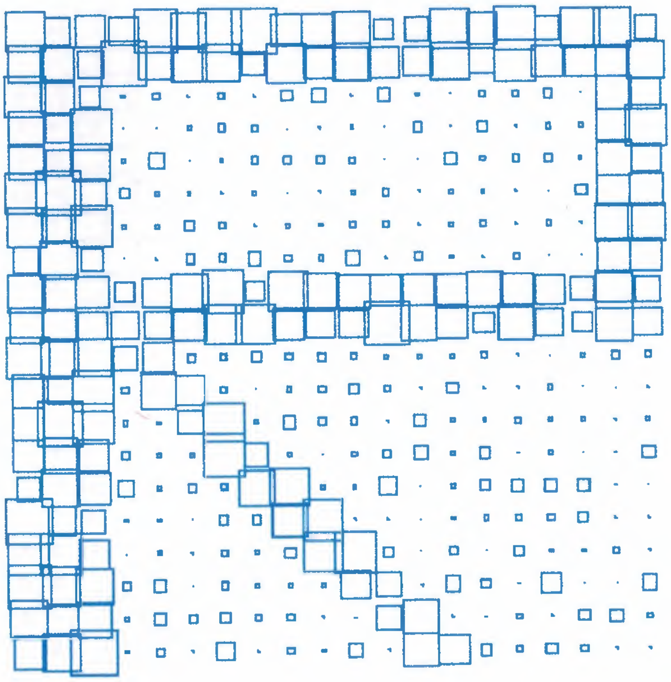


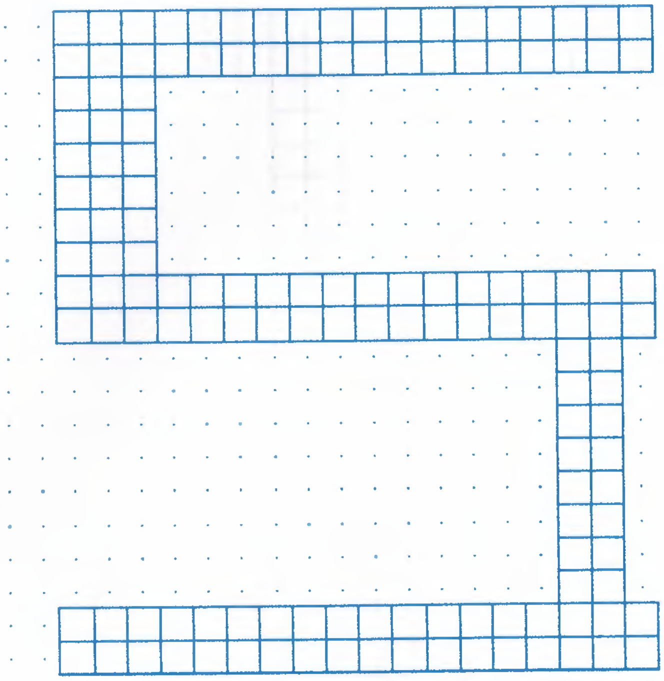


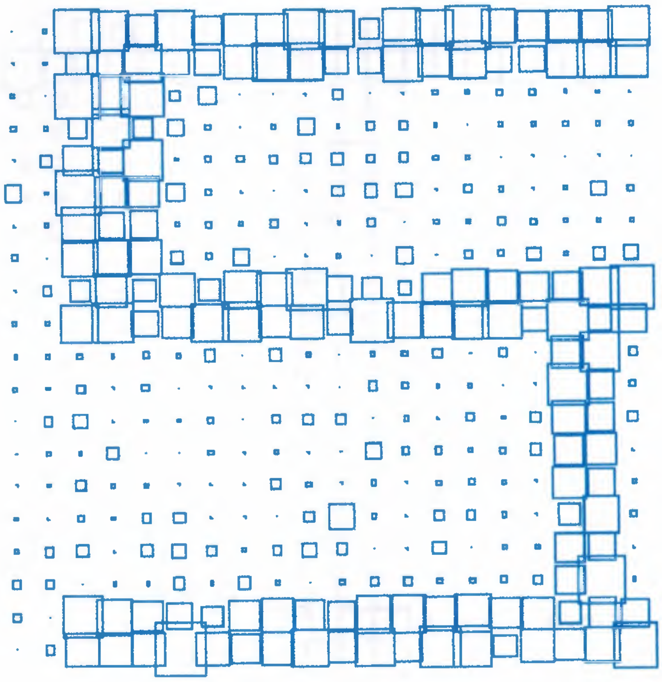


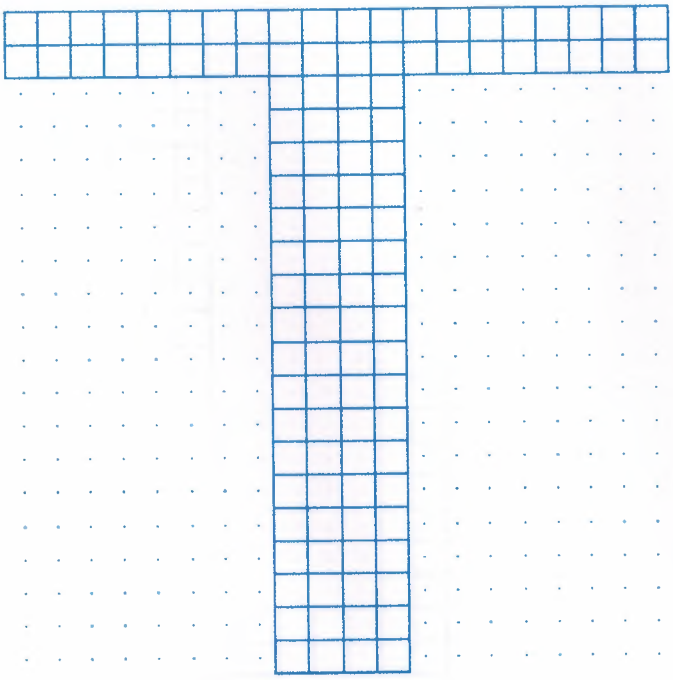


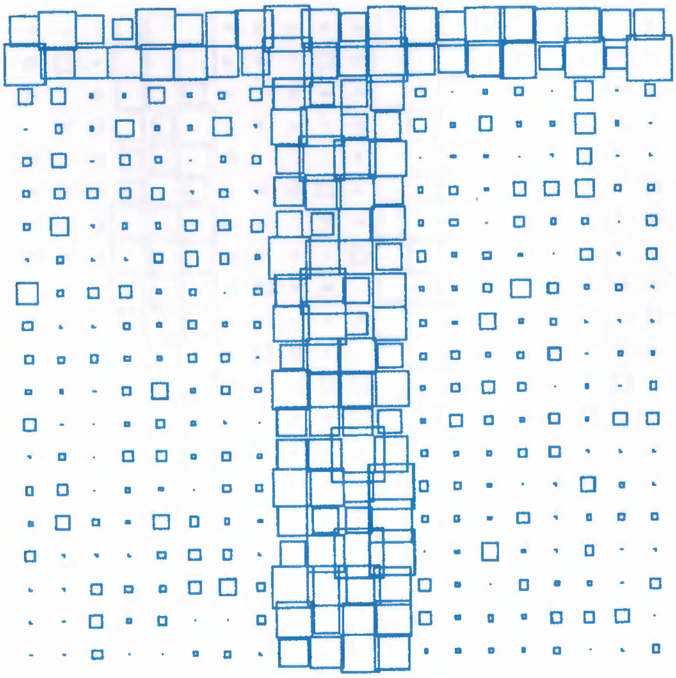


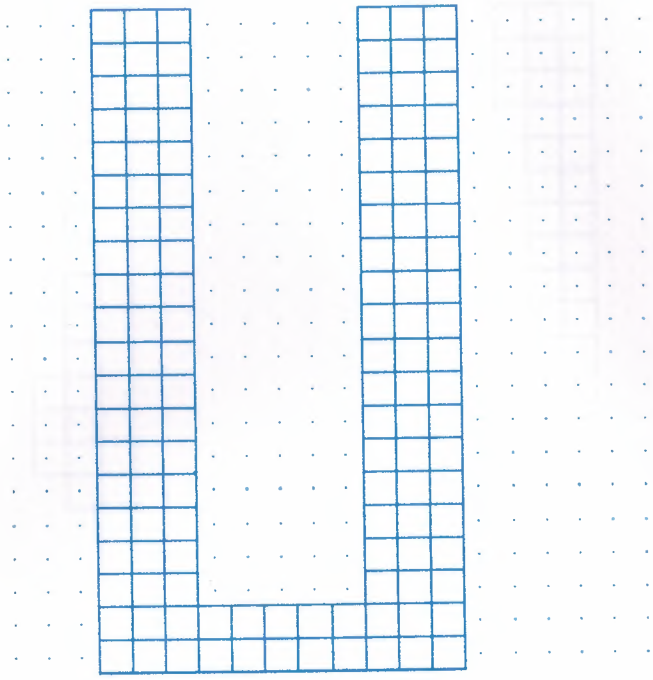


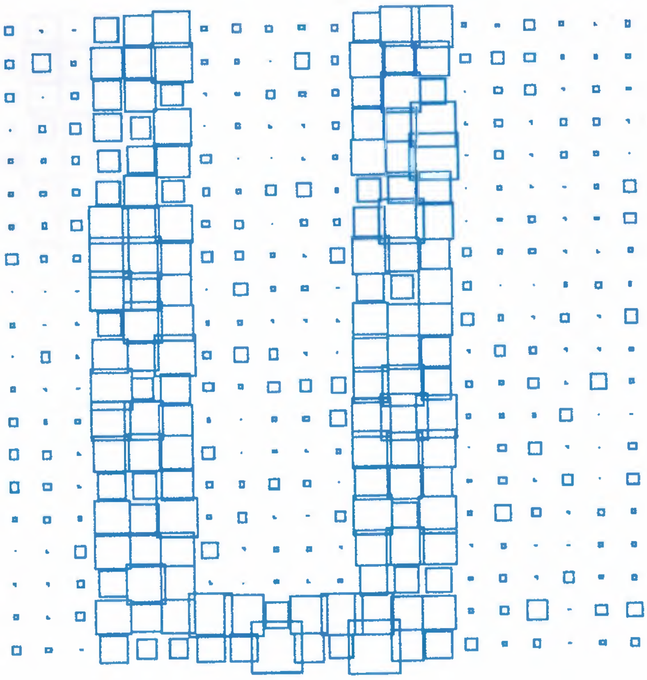


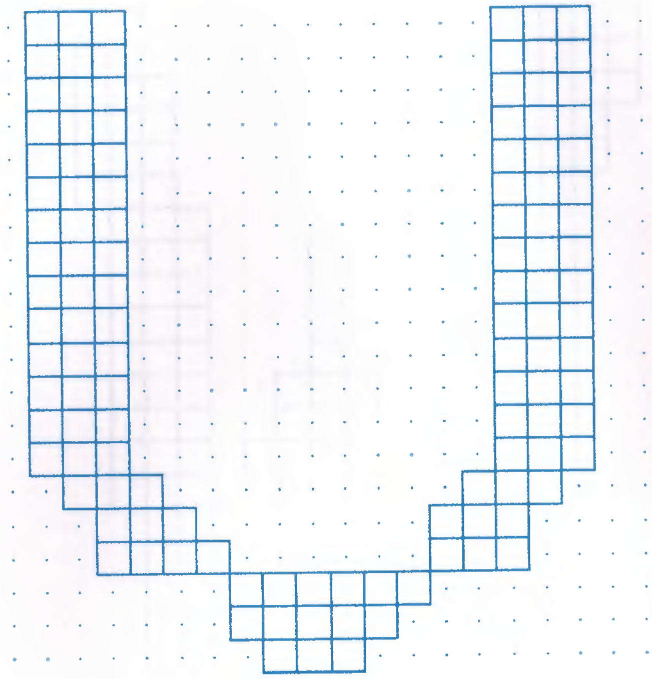


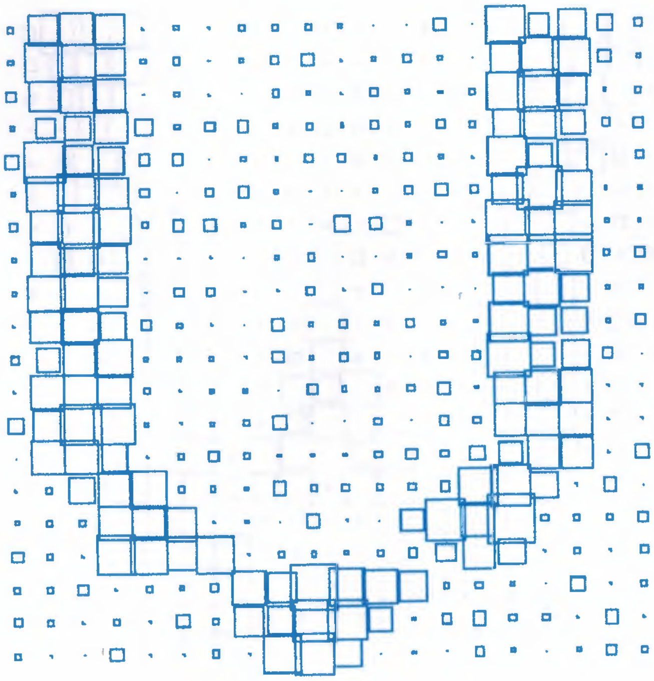


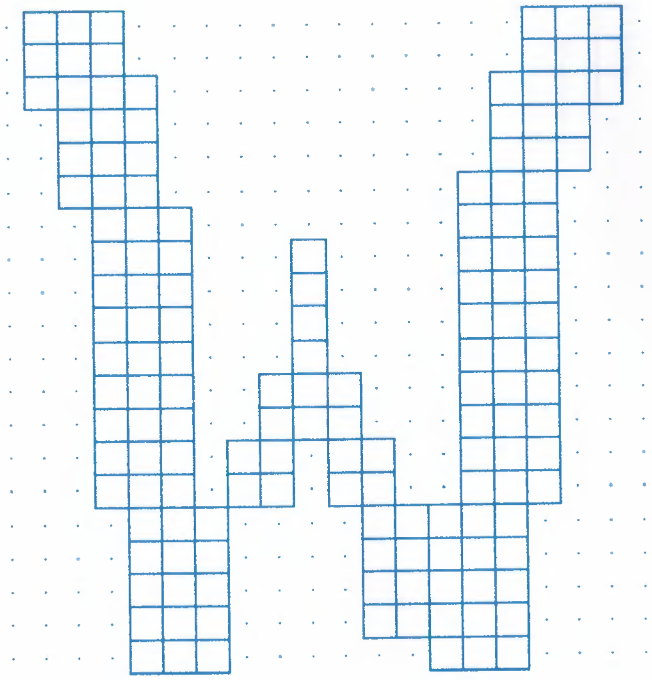


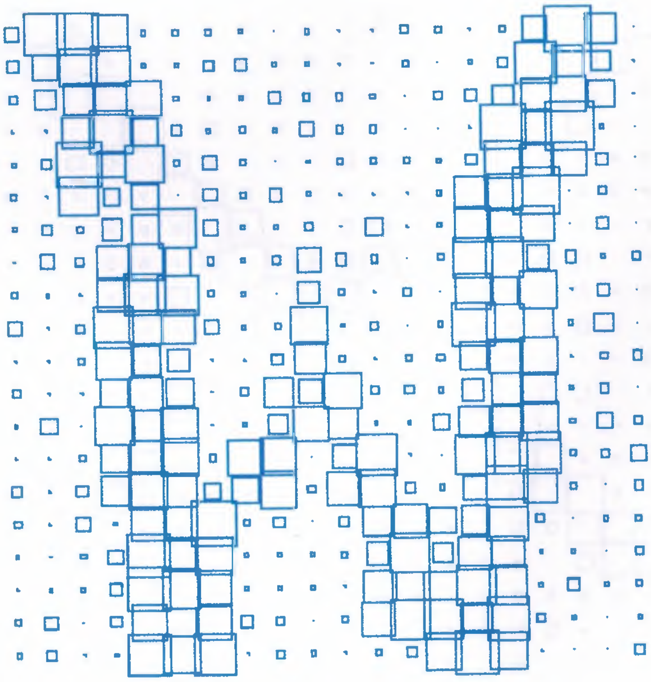


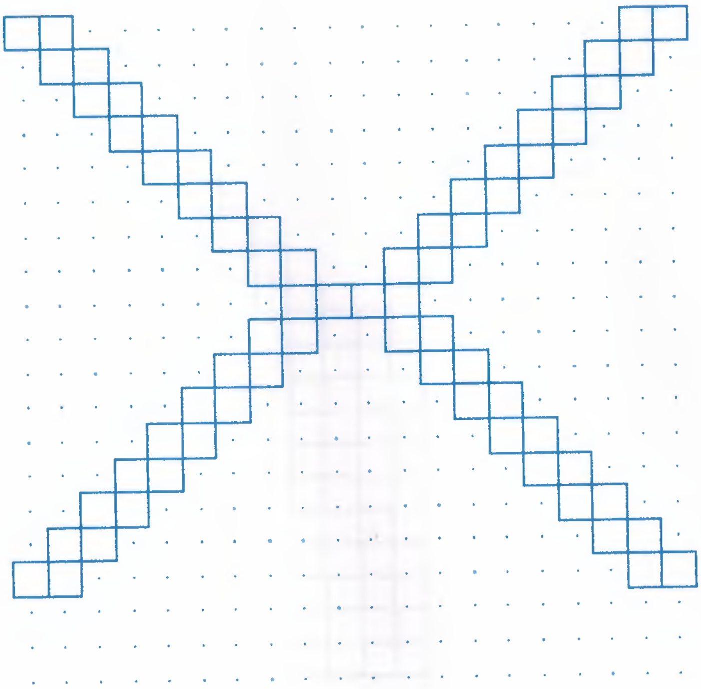


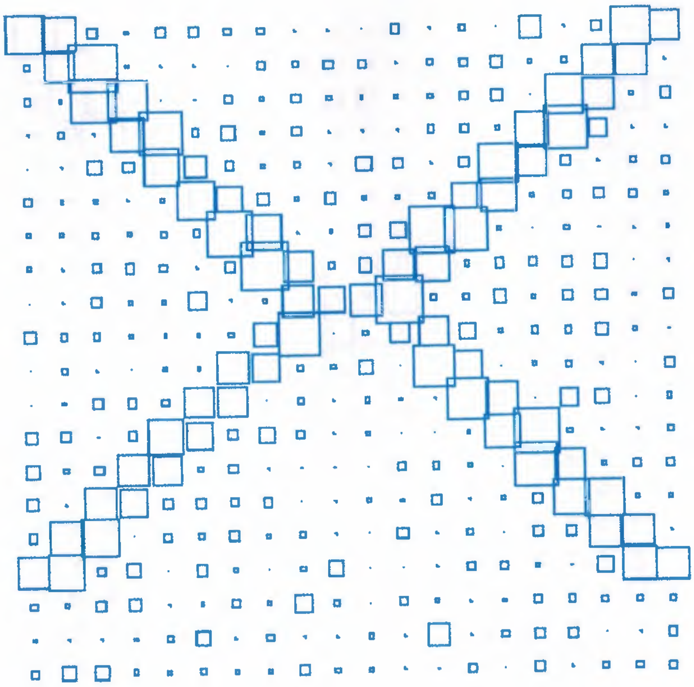


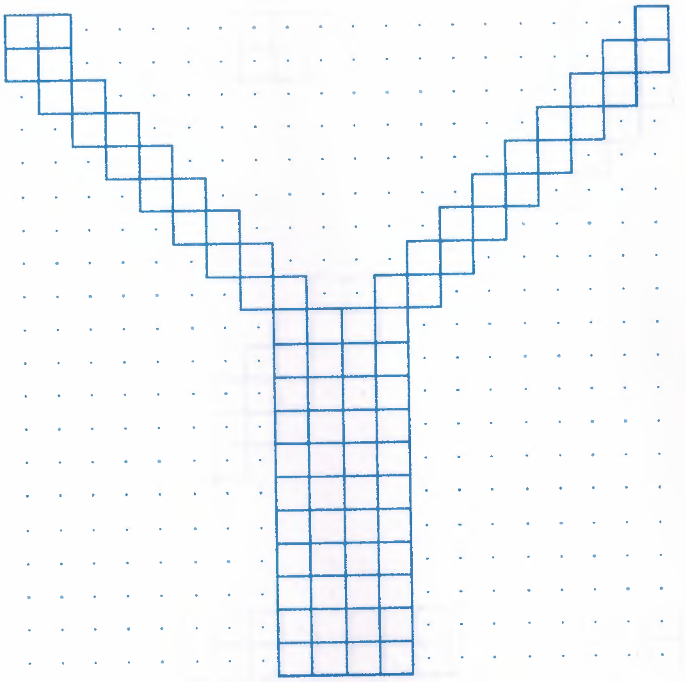


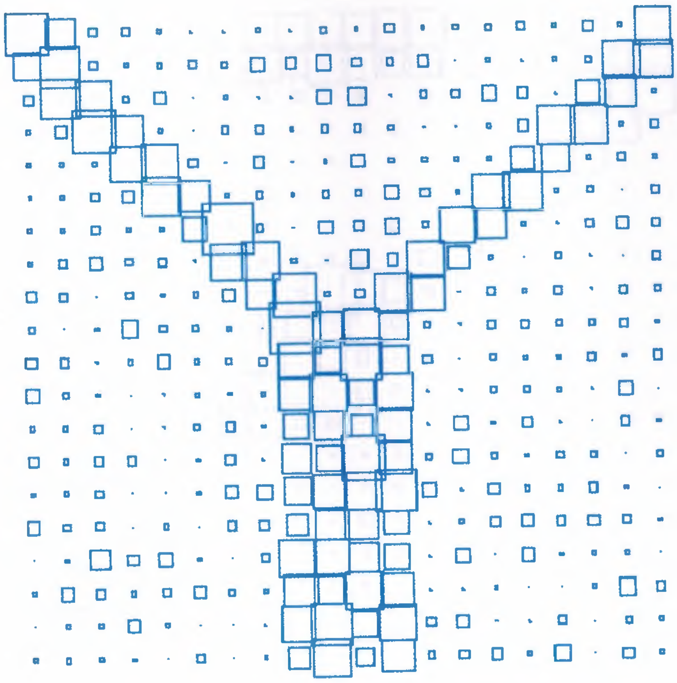


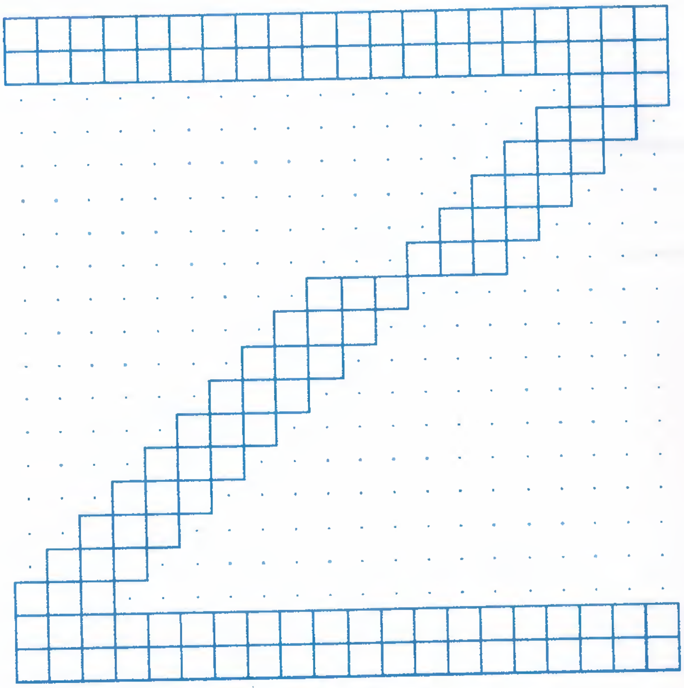


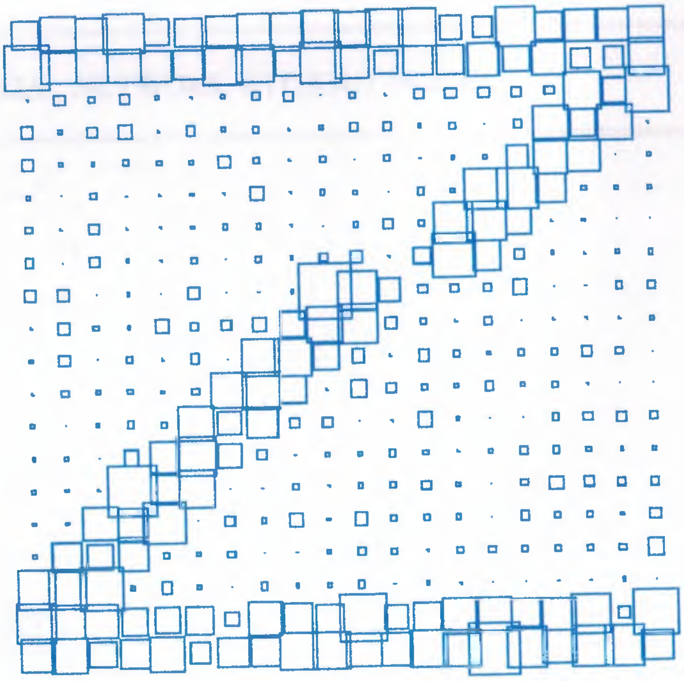












APPENDIX III

```
%==========================================================================
% DEFINING THE MODEL PROBLEM
%==========================================================================
 [alphabet,targets] = prprob;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);

% DEFINING THE NETWORK
S1 = 52;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;

%==========================================================================
% TRAINING INITIAL NETWORK WITHOUT NOISE
%==========================================================================
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.85;
net.trainParam.lr = 0.003;
P = alphabet;
T = targets;
[net,tr] = train(net,P,T);
```

```
% ==========================================================
% TRAINING NETWORK 2 WITH NOISE
% ==========================================================
netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
fprintf('Pass = %.0f\n',pass);
P = [alphabet, alphabet, ...
    (alphabet + randn(R,Q)*0.1), ...
    (alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end

% ==========================================================
% RE-TRAINING NETWORK 2 WITHOUT NOISE
% ==========================================================
netn.trainParam.goal = 0.1;
netn.trainParam.epochs = 500;
net.trainParam.show = 5;
P = alphabet;
T = targets;
[netn,tr] = train(netn,P,T);

% ==========================================================
% TESTING NETWORK 1 AND 2 WITH VARIOUS LEVELS OF NOISE
% ==========================================================
% SET TESTING PARAMETERS
k = 0:.05:.5;
max_test = 100;
```

```
network1 = [];
network2 = [];
T = targets;
% INITIALIZE TESTING
for noiselevel = k
fprintf('Testing networks with noise level of%.2f.\n',noiselevel);
errors1 = 0;
errors2 = 0;
for i=1:max_test
P = alphabet + randn(400,26)*noiselevel;
% TEST NETWORK 1
A = sim(net,P);
AA = compet(A);
errors1 = errors1 + sum(sum(abs(AA-T)))/2;
% TEST NETWORK 2
An = sim(netn,P);
AAn = compet(An);
errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
end
% AVERAGE ERRORS FOR 100 SETS OF 26 TARGET VECTORS.
network1 = [network1 errors1/26/100];
network2 = [network2 errors2/26/100];
end
% DISPLAY RESULTS
clf
plot(k,network1*100,'--',k,network2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
ylabel('Network 1 - -   Network 2 ---');
```

```
% ==========================================================
% DISPLAY CHARACTERS
% ==========================================================
% CHARACTER A WITH NOISE
k=0.5
if(k>0.5)
disp('Noise Is Too High')
else
noisyA = alphabet(:,1)+randn(400,1)*k ;
plotchar(noisyA);

% CHARACTER A WITHOUT NOISE
figure;
A2 = sim(net,noisyA);
answer = find(compet(A2) == 1);
plotchar(alphabet(:,answer));
emd
```

REFERENCES

[1]. Pratt, William K. Digital Image Processing. New York: John Wiley & Sons, Inc., 1991. p. 634.

[2]. Horn, Berthold P. K., Robot Vision. New York: McGraw-Hill, 1986. pp. 73-77.

[3]. Pratt, William K. Digital Image Processing. New York: John Wiley & Sons, Inc., 1991. p. 633.

[4]. Haralick, Robert M., and Linda G. Shapiro. Computer and Robot Vision, Volume I. Addison-Wesley, 1992.

[5]. Ardeshir Goshtasby, Piecewise linear mapping functions for image registration, Pattern Recognition, Vol 19, pp. 459-466, 1986.

[6]. Ardeshir Goshtasby, Image registration by local approximation methods, Image and Vision Computing, Vol 6, p. 255-261, 1988.

[7]. Jain, Anil K. Fundamentals of Digital Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1989. pp. 150-153.

[8]. Pennebaker, William B., and Joan L. Mitchell. JPEG: Still Image Data Compression Standard. Van Nostrand Reinhold, 1993.

[9]. Gonzalez, Rafael C., and Richard E. Woods. Digital Image Processing. Addison-Wesley, 1992. p. 518.

[10]. Haralick, Robert M., and Linda G. Shapiro. Computer and Robot Vision, Volume I. Addison-Wesley, 1992. p. 158.

[11]. Floyd, R. W. and L. Steinberg. "An Adaptive Algorithm for Spatial Gray Scale," International Symposium Digest of Technical Papers. Society for Information Displays, 1975. p. 36.

[12]. Lim, Jae S. Two-Dimensional Signal and Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1990. pp. 469-476.

[13]. Canny, John. "A Computational Approach to Edge Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986. Vol. PAMI-8, No. 6, pp. 679-698.

[14]. Lim, Jae S. Two-Dimensional Signal and Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1990. pp. 478-488.

[15]. Parker, James R. Algorithms for Image Processing and Computer Vision. New York: John Wiley & Sons, Inc., 1997. pp. 23-29.

[16]. Gonzalez, Rafael C., and Richard E. Woods. Digital Image Processing. Addison-Wesley, 1992. p. 518.

[17]. Haralick, Robert M., and Linda G. Shapiro. Computer and Robot Vision, Volume I. Addison-Wesley, 1992. p. 158.

[18]. Jain, Anil K. Fundamentals of Digital Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1989. pp. 150-153.

[19]. Pennebaker, William B., and Joan L. Mitchell. JPEG: Still Image Data Compression Standard. New York: Van Nostrand Reinhold, 1993.

[20]. Robert M. Haralick and Linda G. Shapiro, Computer and Robot Vision, vol. I, Addison-Wesley, 1992, pp. 158-205.

[21]. van den Boomgaard and van Balen, "Image Transforms Using Bitmapped Binary Images," Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 54, no. 3, May, 1992, pp. 254-258.

[22]. Kak, Avinash C., and Malcolm Slaney, Principles of Computerized Tomographic Imaging. New York: IEEE Press.

[23]. J. P. Lewis, "Fast Normalized Cross-Correlation", Industrial Light & Magic. http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html

[24]. Robert M. Haralick and Linda G. Shapiro, Computer and Robot Vision, Volume II, Addison-Wesley, 1992, pp. 316-317.

[25]. Bracewell, Ronald N. Two-Dimensional Imaging. Englewood Cliffs, NJ: Prentice Hall, 1995. pp. 505-537.

[26]. Lim, Jae S. Two-Dimensional Signal and Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1990. pp. 42-45.

[27]. Rein van den Boomgard and Richard van Balen, "Methods for Fast Morphological Image Transforms Using Bitmapped Images," Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 54, no. 3, May 1992, pp. 252-254.

[28]. Rolf Adams, "Radial Decomposition of Discs and Spheres," Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, vol. 55, no. 5, September 1993, pp. 325-332.

[29]. Ronald Jones and Pierre Soille, "Periodic lines: Definition, cascades, and application to granulometrie," Pattern Recognition Letters, vol. 17, 1996, 1057-1063.