NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Electrical & Electronics Engineering

Character Recognition Using Neural Networks

EE 400

Student: Sakeb Hussein (20034081)

Supervisor: Mr. Jamal Fathi

Lefkosa - 2007

Dedicated To My Mother

ACKNOWLEDGEMENTS

I could not have prepared this project without the generous help of my supervisor, colleaques, friends, family and especially my mother and brothers Hasheem, Bakir and Muatasim.Also I would like to thank my friend Mutleq Qa'aqorah.

My deepest thanks are to my supervisor Mr. Jamal fathi for his help and answering any question I asked him.

I would like to express my gratitude to Prof. Dr. Fakhraddin Mamedov.

Also I would like to express my gratitude to Mr. Tayseer Alshanableh and his family.

Finally, I could never have prepared this project without the encouragement and support of my mum, brothers and sister.

ABSTRACT

In this project we will see that the neural network behaves like a baby because it is learning from what we are teaching it such as the examples which are giving to it. Also we will show how a neural network can recognize the alphabet letters as we teach it.

CONTENTS

1. ARTIFICIAL NEURAL NETWORKS

1.1 Overview

This chapter presents an overview of neural networks, its history, simple structure, biological analogy and the Back propagation algorithm.

In both the Perceptron Algorithm and the Back propagation Producer, the correct output for the current input is required for learning. This type of learning is called **supervised learning**. Two other types of learning are essential in the evolution of biological intelligence: **unsupervised learning** and reinforcement learning. In unsupervised learning a system is only presented with a set of exemplars as inputs. The system is not given any external indication as to what the correct responses should be nor whether the generated responses are right or wrong. Statistical clustering methods, without knowledge of the number clusters, are examples of *unsupervised learning.* **Reinforcement learning** is somewhere between *supervised learning*, in which the system is provided with the desired output, and *unsupervised learning*, in which the system gets no feedback at all on how it is doing. In reinforcement learning the system receivers a feedback that tells the system whether its output response is right or wrong, but no information on what the right output should be is provided.[27]

1.2 Neural Network Definition

First of all, when we are talking about a neural network, we should more properly say "artificial neural network" (ANN) because that is what we mean most of the time. Biological neural networks are much more complicated than the mathematical models we use for ANNs, but it is customary to be lazy and drop the "A" or the "artificial".

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems

involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

- **Definition:**

  A machine that is designed to model the way in which the brain preference a particular taste or function. The neural network is usually implemented using electronic components or simulated as software.

- **Simulated:**

  A neural network is a massive, parallel-distributed processor made up of simple processing units, which has neural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

  1. The network from its environment through a learning process acquires knowledge.
  2. Interneuron connection strength, known as synaptic weights, are used to store the acquired knowledge.

- **Simulated:**

  A neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

- **Simulated:**

  A neural network is a massive, parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

  1. Knowledge is acquired by the network through a learning process.
  2. Interneuron connection strengths, known as synaptic weights are used to store the knowledge.

- **Simulated:**

  A neural network is a computational model that shares some of the properties of the brain. It consists of many simple units working in parallel with no central control; the connections between units have numeric weights that can be modified by the learning element.

- **Simulated:**

  A new form of computing inspired by biological models, a mathematical model composed of a large number of processing elements organized into layers.
  "A computing system made up of a number of simple ,highly interconnected elements, which processes information by its dynamic state response to external inputs"

Neural networks go by many aliases. Although by no means synonyms the names listed in figure 1.1 below.

- Parallel distributed processing models
- Connectivist /connectionism models
- Adaptive systems
- Self-organizing systems
- Neurocomputing
- Neuromorphic systems

Figure 1.1 Neural Network Aliases

All refer to this new form of information processing; some of these terms again when we talk about implementations and models. In general though we will continue to use the words "neural networks" to mean the broad class of artificial neural systems. This appears to be the one most commonly used

1.3 History of Neural Networks

1.3.1 Conception (1890-1949)

Alan Turing was the first to use the brine as a computing paradigm, a way of looking at the world of computing. That was in 1936. In 1943, a Warren McCulloch, a neurophysiologist, and Walter Pitts, an eighteen–year old mathematician, wrote a paper about how neurons might work. They modeled a simple neural network with electrical circuits. John von Neumann used it in teaching the theory of computing machines. Researchers began to look to anatomy and physiology for clues about creating intelligent machines.

Another important book was Donald Hebb's the Organization of Behavior (1949) [2], which highlights the connection between psychology and physiology, pointing out that a neural pathway is reinforced each time it is used. Hebb's "Learning Rule", as it is sometime known, is still used and quoted today.

1.3.2 Gestation (1950s)

Improvements in hardware and software in the 1950s ushered in the age of computer simulation. It became possible to test theories about nervous system functions. Research expanded and neural network terminology came into its own.

1.3.3 Birth (1956)

The Dartmouth Summer Research Project on Artificial Intelligence (AI) in the summer of 1956 provided momentum for both the field of AI and neural computing. Putting together some of the best minds of the time unleashed a whole raft of new work. Some efforts took the "high-level" (AI) approach in trying to create computer programs that could be described as "intelligent" machine behavior; other directions used mechanisms modeled after "low-level" (neural network) processes of the brain to achieve "intelligence".[7]

1.3.4 Early Infancy (Late 1950s-1960s)

The year following the Dartmouth Project, John von Neumann wrote material for his book The Computer and the Brain (Yale University Press, 1958). Here he makes such suggestions as imitating simple neuron function by using telegraph relays or vacuum. The Perceptron, a neural network model about which we will hear more later, built in hardware, is the oldest neural network and still has use today in various form for applications such as character recognition.

In 1959, Bernard Widrow and Marcian Hoff (Stanford) developed models for ADALINE, then MADALINE (Multiple Adaptive Liner Elements). This was the first neural network applied to real-world problem-adaptive filers to eliminate echoes on phone lines. As we mentioned before, this application has been in commercial use for several decades.

One of the major players in the neural network reach from to the 1960s to current time is Stephen Grossberg (Boston University). He has done considerable writing (much of it tedious) on his extensive physiological research to develop neural network models. His 1967 network, Avalanche, uses a class of networks to perform activities such as continuous-speech recognition and teaching motor commands to robotic arms.[10]

1.3.5 Excessive Hype

Some people exaggerated the potential of neural networks. Biological comparisons were blown out of proportion in the October 1987 issue of the "Neural Network Review", newsletter editor Craig Will quoted Frank Rosenblatt from a 1958 issue of the "New Yorker".

1.3.6 Stunted Growth (1969-1981)

In 1969 in the midst of such outrageous claims, respected voices of critique were raised that brought a halt too much of the funding for neural network research. Many researchers turned their attention to AI, which looked more promising at the time.

- Amari (1972) independently introduced the additive model of a neural and used it to study the dynamic behavior of randomly connected neuron like elements.
- Wilson and Cowan (1972) derived coupled nonlinear differential equations for the dynamic of spatially localized populations containing both excitatory and inhibitory model neurons.

- Little and Shaw (1975) described a probabilistic of a neuron, either firing or not firing an action potential and used the model to develop a theory of short term memory.
- Anderson Silverstein Ritz and Jones (1977) proposed the brain state in a box (BSB) model consisting of simple associative network coupled to nonlinear dynamics. [14]

1.3.7 Late Infancy (1982 –Present)

Important development in 1982 was the publication of Kohonen's paper on self-organizing maps "Kohonen 1982", which used a one or two dimensional lattice structure.

In 1983,Kirkpatrick, Gelatt, and Vecchi described a new procedure called simulated annealing, for solving combinatorial optimization problems. Simulated annealing is rooted in statistical mechanics.

Jordan (1996) by used a mean-field theory a technique also in statistical mechanics.

A paper by Bator, Sutton and Anderson on reinforcement learning was published in 1983. Although, they were not the first to use reinforcement learning (Minsky considered it in his 1954 Ph.D. thesis for example).

In 1984 Braitenberg's book, Vehicles: Experiments in Synthetic Psychology, was published.

In 1986 the development of the back-propagation algorithm was reported by Rumelhart Hinton and Williams (1986).

In 1988 Linsker described a new principle for self-organization in a perceptual network (Linsker, 1988a) Also in 1988, Broomhead and Lowe described a procedure for the design of layered feed-forward networks using radial basis functions (RBF) which provide an alter native to multiplayer perceptrons.

In 1989 Mead's book, Analog VLSI and Neural Systems, was published. This book provides an unusual mix of concepts drawn from neurobiology and VLSI technology.

In the early 1990s, Vapnik and coworkers invented a computationally powerful class of supervised leaning networks called Support Vector Machines, for solving pattern recognition regression, and the density estimation problem. "Boser, Guyon and Vapnik, 1992, Cortes and Vapnik, 1995; Vapnik, 1995,1998."

In 1982 the time was rip for renewed interest in neural networks. Several events converged to make this a pivotal year.

John Hopfield (Caltech) presented his neural network paper to the National Academy of Sciences. Abstract ideas became the focuse as he pulled together previous work on neural networks.

But there were other threads pulling at the neural network picture as well. Also in 1982 the U.S. – Japan Joint Conference on Cooperative Competitive Neural Network, was held in Kyoto Japan.

In 1985 the American Institute of Physics began what has become an annual Neural Networks for computing meeting. This was the first of many more conference to come in 1987 the institute of Electrical and Electronic Engineers (IEEE). The first international conference on neural networks drew more than 1800 attendees and 19 vendors (although there were few products yet to show). Later the same year, the International Neural Network Society (INNS), was formed under the leadership of Grossberg in the U.S., Kohonen in Finland, and Amari in Japan.

AI though there were two competing conferences in 1988, the spirit of cooperation in this new technology has resulted in joint spontional Joint Conference on Neural Networks (IJCNN) held in Japan in 1989 which produce 430 papers, 63 of which focused on application development. January 1990 IJCNN in Washington, D.C. clouded an hour's concert of music generated by neural networks. The Neural Networks for Defense meeting, held in conjunction with the June 1989 IJCNN above, gathered more than 160 represntives of government defense and defense contractors giving presentations on neural network efforts. When the U.S. Department of Defense announced its 1990 Small Business Innovation Program 16 topics specifically targeted neural networks. An additional 13 topics mentioned the possibility of using neural network approaches. The year of 1989 was of unfolding application possibilities. On

September 27, 1989, the IEEE and the Learning Neural Networks Capabilities created applications for today and the Future.

The ICNN in 1987 included attendees from computer science electrical engineering, physiology cognitive psychology, medicine and even a philosopher of two. In May of 1988 the North Texas Commission Regional Technology Program convened a study group for the purpose of reviewing the opportunities for developing the field of computational neuroscience. Their report of October 1988 concluder that the present is a critical time to establish such a center. [1]

Believing that a better scientific understanding of the brain and the subsequent application to computing technology could have significant impact. They assess their regional strength in electronics and biomedical science and their goals are both academic and economic. You can sense excitement and commitment in their plans.

Hecht-Nielsen (1991) attributes a conspiratorial motive to Minsky and Papert. Namely, that the MIT AI Laboratory had just been set up and was focussing on LISP based AI, and needed to spike other consumers of grants. A good story, whatever the truth, and given extra spice by the coincidence that Minsky and Rosenblatt attended the same class in high-school. Moreover, any bitterness is probably justified because neural network researchers spent the best part of 20 years in the wilderness.

Work did not stop however, and the current upsurge of interest began in 1986 with the famous PDP books which announced the invention of a viable training algorithm (back propagation) for multilayer networks (Rumelhart and McClelland, 1986). [23]

Table 1.1. Summarize the history of the development of N.N.

Table 1.1 Development of N.N.

Present	Late 80s to now	Interest explodes with conferences, articles, simulation, new companies, and government funded research.
Late Infancy	1982	Hopfiled at National Academy of Sciences
Stunted Growth	1969	Minsky & Papert's critique Perceptrons
Early Infancy	Late 50s, 60s	Excessive Hype Research efforts expand
Birth	1956	AI & Neural computing Fields launched Dartmouth Summer Research Project
Gestation	1950s	Age of computer simulation
	1949	Hebb, the Organization of Behavior
	1943	McCulloch & Pitts paper on neurons
	1936	Turing uses brain as computing paradigm
Conception	1890	James, Psychology (Briefer Curse)

1.4 Analogy to the Brain

The human nervous system may be viewed as a three stage system, as depicted in the block diagram of the block diagram representation of the nervous system.

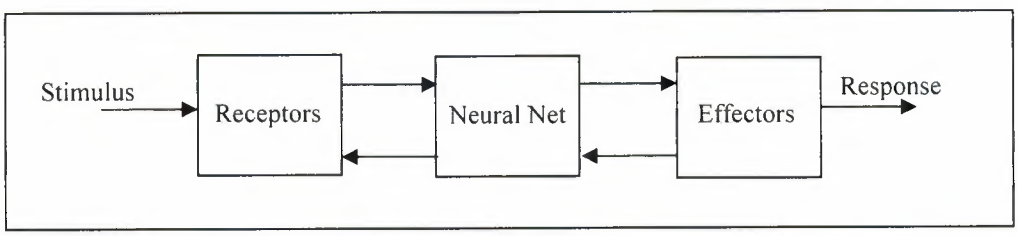


Figure 1.2 Block Diagram of the Nervous System.

(Arbib,1987) Central to the system is the brain, represented by the neural (nerve) network which continually receives information, perceives if, and makes appropriate decisions. Two sets of arrows are shown in the block diagram. 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 neural network (brain). The effectors convert electrical impulses by the neural network into discernible responses as system outputs.

1.4.1 Natural Neuron

A neuron is a nerve cell with all of its processes. Neurons are one of the main distinctions of animals (plants do not have nerve cells). Between seven and one hundred different classes of neurons have been identified in humans. The wide variation is related to how restrictively a class is defined. We tend to think of them as being microscopic, but some neurons in your legs are as long three meters. The type of neuron found in the retina is shown in figure 1.3.

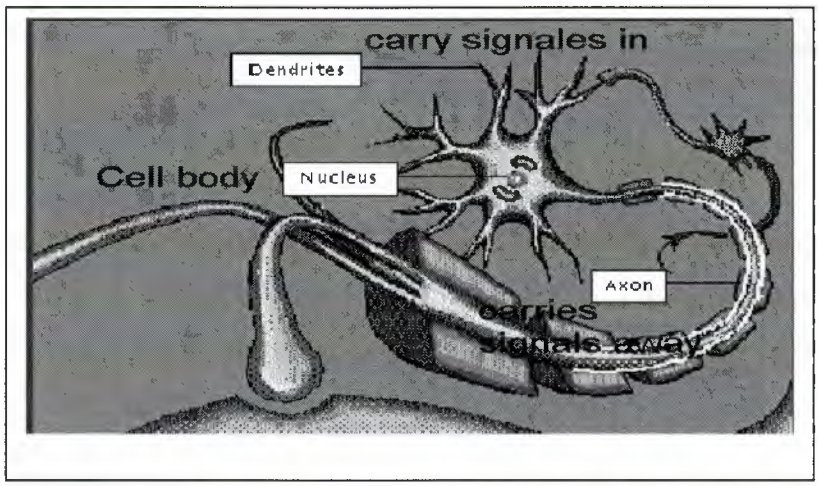


Figure 1.3 Neuron Natural. [23]

An example is a bipolar neuron. Its name implies that has two processes. The cell body contains the nucleus, and leading into the nucleus are one or more dendrites. These branching, tapering processes of the nerve cell, as a rule, conduct impulses toward the cell body. The axon is the nerve cell process that conducts the impulse type of neurons. This one gives us the functionality and vocabulary we need to make analogies.

1.4.2 Artificial Neuron

Our paper and pencil model starts by copying the simplest element the neuron call our artificial neuron a processing element or PE for short. The word node is also used for this simple building block, which is represented by circle in the figure 1.4 "a single mode or processing element PE or Artificial Neuron"

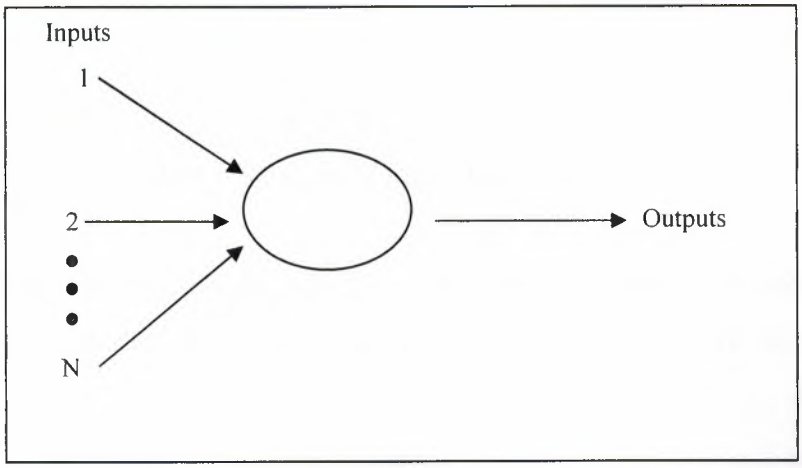


Figure 1.4 Artificial Neuron

The PE handles several basic functions: (1) Evaluates the input signals and determines the strength of each one, Calculates the total for the combined input signals and compare that total to some threshold level, and (3) Determines what the output should be.

Input and Output: Just as there are many inputs (stimulation levels) to a neuron there should be many input signals to our PE. All of them should come into our PE simultaneously. In response a neuron either "fires" or "doesn't fire" depending on some threshold level. The PE will be allowed a single output signal just as is present in a biological neuron. There are many inputs and only one output.

Weighting Factors: Each input will be given a relative weighting which will affect the impact of that input. In figure 1.5, "a single mode or processing element PE or Artificial Neuron" with weighted inputs.

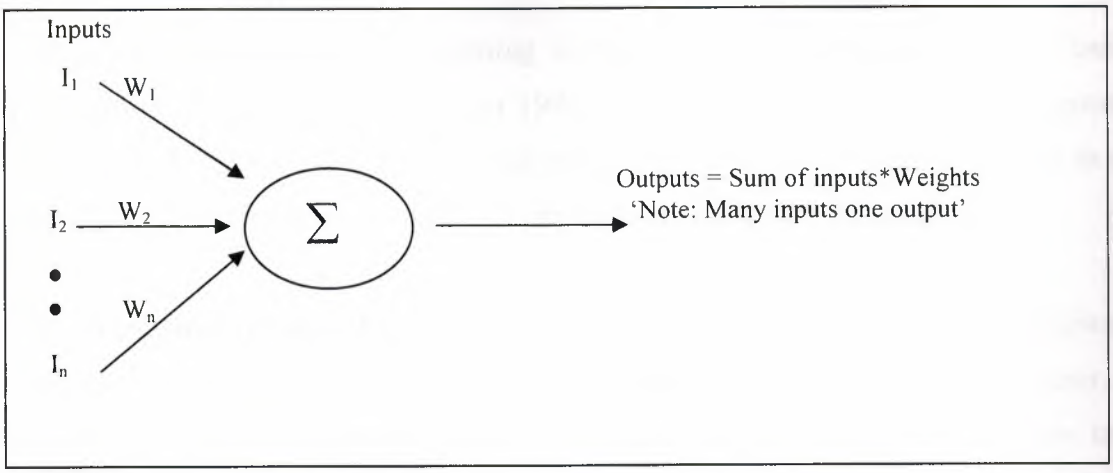


Figure 1.5 Single Mode Artificial Neuron

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

1.5 Model of a Neuron

The neuron is the basic processor in neural networks. Each neuron has one output, which generally related to the state of the neuron –its activation, which may fan out to several other neurons. Each neuron receives several inputs over these connections, called synapses. The inputs are the activations of the neuron. This is computed by applying a threshold function to this product. An abstract model of the neuron is shown in figure 1.6.

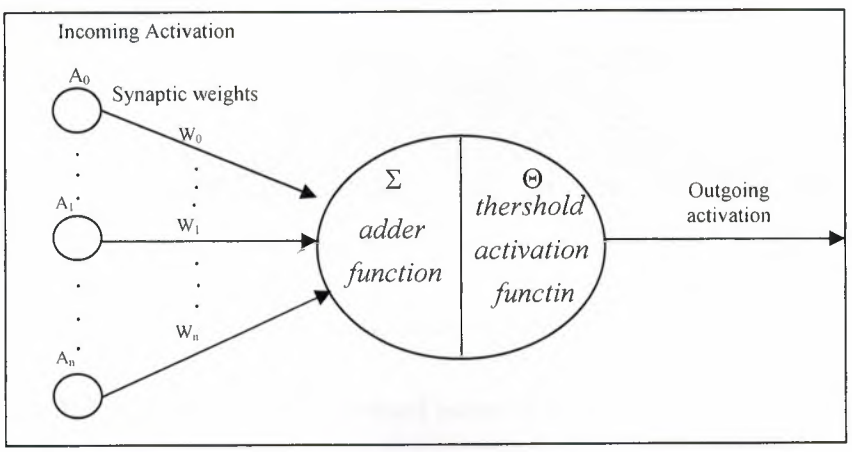


Figure 1.6 Diagram of Abstract Neuron Model. [23]

1.6 Back-Propagation

The most popular method for learning in the multiplayer network is called "back-propagation." It was first invented in 1996 by Bryson, but was more or less ignored until the mid-1980s. The reason for this may be sociological, but may also have to do with the computational requirements of the algorithm on nontrivial problems.

The **back-propagation** learning algorithm works on multiplayer feed-forward networks, using gradient descent in weight space to minimize the output error. It converges to a locally optimal solution, and has been used with some success in a variety of applications. As with all hill-climbing techniques, however, there is no guarantee that it will find a global solution. Furthermore, its converge is often very slow.

1.6.1 Back-Propagation Learning

Suppose we want to construct a network for the restaurant problem. So we will try a two-layer network. We have ten attributes describing each example, so we will need ten input units. In figure 1.7, we show a network with four hidden nits. This turns out to be about right for this problem.

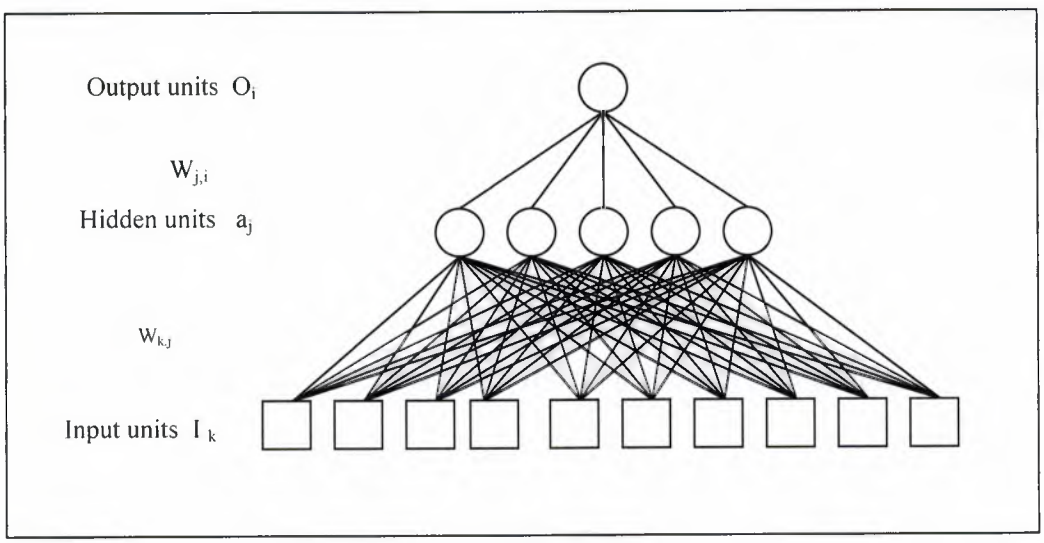


Figure 1.7 A two layer feed forward network for the restaurant problem. [23]

Example inputs are presented to the network, and if the network computes an output vector that matches the target, nothing is done. If there is an error (a difference between the output and target), then weights are adjusted to reduce this error. The trick is to assess the blame for an error and divide it among the contributing weights. In Perceptrons, this is easy, because there is only one weight connecting each input and output. But in multiplayer networks, there are many weights connecting each input to an output and each of these weights contributes to more than one output.

The back-propagation algorithm is a sensible approach to dividing the contribution of each weight. As in the Perceptron Learning Algorithm, we try to minimize the error between each target output and the output actually computed by the network. At the output layer the weight update rule is very similar to the rule for the perceptron. However, there are two differences. The activation of the hidden unit a_j is used instead of the input value; and the rule contains a term for the gradient of the activation function. If Err_i is the error (T_i-O_i) at the output node, then the weight update rule for the link from unit j to unit i is

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

Where g' is the derivative of the activation g will find it convenient to define a new error term Δ_i which for output node is defined as $\Delta_i = Err_i g'(in_i)$. 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, we need to define a quantity analogous to the error term for output node. The propagation rule so the following:

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

Now the weight update rule for the weights between the inputs and the hidden layer is almost identical 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, α) returns a network with modified weights.
Inputs: network, a multiplayer network
Examples, asset of input/output pairs α, 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 has converged
return network

Figure 1.8 Back propagation algorithm for updating weights in a multiplayer network

Back-propagation provides a way of dividing the calculation of the gradient among the unit so the change in each weight can be calculated by the unit to which the weight is attached using only local information.

We use the sum of squared errors over the output values:

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

The key insight again is that the output values O_i are a function of the weights for general two-layer network, we can write:

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

$$E(W) = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} g(\sum W_{k,j} I_k)))^2 \quad (1.7)$$

1.7 Learning Processes

Learning is a process by which the free parameters of a neural network are a adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by a manner in which the parameter change takes place.

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

- The neural network is stimulated by an environment.
- 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 changes that have occurred in its internal structure.

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

Basically learning algorithms differ from each other in the way in which the adjustment to a synaptic weight of neurons is formulated. Another factor to be considered is the manner in which a neural network (learning machine) is made up of a set of interconnected neurons. Learning paradigm refers to a model of the environment in whish the neural network operates.

1.7.1 Memory-Based Learning

In memory-based learning, all (or most) of the past experiences are explicitly stored in a large memory of correctly classified input-output examples.

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

Where xi denotes an input vector and di denotes the corresponding desired response.

1.7.2 Hebbian Learning

When an axon of cell A is near 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 cell such that A is efficiency as one of the cells firing B is increased.

1. If two neurons on either side of a synapse are selectively (connection) activated simultaneously (i.e. then the strength of that synapse is selectively increased).
2. If two neurons on either side of a synapse are active asynchronously, then that synapse is selectively weakended or eliminated.

The following are four key mechanisms that characterize a Hebbian Synapse:

1. Time-dependent mechanism. This mechanism refers to the fact that the modification in the Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals.
2. Local mechanism. By its nature a synapse is the transmission site where information-bearing signals (representing ongoing activity in the presynaptic and postsynaptic units) are in spatiotemporal congtiguity.
3. Interactive mechanism. The occurrence of a change in the Hebbian synapse depends on signals on both sides of the synapse.
4. Conjunctional or correlational mechanism. One interpretation of Hebb's postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and posynaptic signals.

1.7.2.1 Synaptic Enhancement and Depression

The conception of a Hebbian modification by is recognizing that positively correlated activity produces synaptic weakening; synaptic for depression may also be of a noninteractive type. The classification of modifications such as Hebbian, anti-Hebbian, and non-Hebbian, according to this scheme, increases its strength when these signals are either uncorrelated or negatively correlated.

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, is expressed in the general form:

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

Where F(.,.) is a function of both postsynaptic and presynaptic signals the signals $x_j(n)$ and $y_k(n)$ are often treated as dimensionless.

1.7.2.3 Hebbian Hypothesis

The simplest form of Hebbian learning is described by:

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

Where η is a positive constant that determine the rate of learning, it clearly emphasizes the correlational nature of a Hebbian synapse. It is sometimes referred to as the activity product rule. (The top curve of figure 1.9).

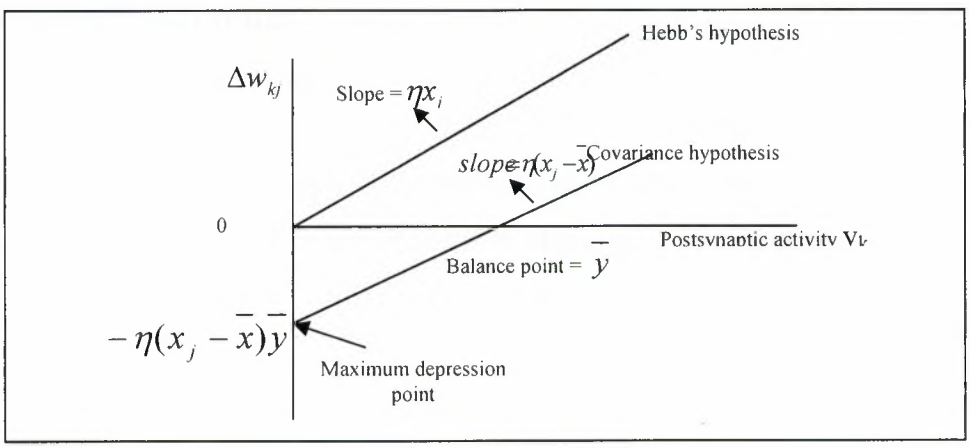


Figure 1.9 Illustration of Hebb's Hypothesis and the Covariance Hypothesis. [23]

With the change Δw_{kj} plotted, versus the output signal (postsynaptic activity) y_k, therefore exponential growth finally drives the synaptic connection into staturation. At that point no information will be stored in the synapse and selectivity is lost.

Covariance hypothesis: One way of overcoming the limitation of Hebb's hypothesis is to use covariance hypothesis introduced by Sejnowski. In this hypothesis, the presynaptic and postsynaptic signals in are replaced by the departure of presynaptic and postsynaptic signals from their respective values over a certain time interval. 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} is defined by:

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

Where η is the learning rate parameter, 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 (fired). The several output neurons may be active simultaneously in completive learning; only a signal output neuron is active at any time. It is this features that may be used to classify a set of input patterns.
The three basic elements to a competitive learning rule.

- A set of neurons that are all the same 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; such that only one output neurons.

In the simplest form of competitive learning the neuronal network has a single layer of output neurons. Each of which is fully connected to the input nodes. The network may include feedback connection among the neurons as indicated in figure 1.10.

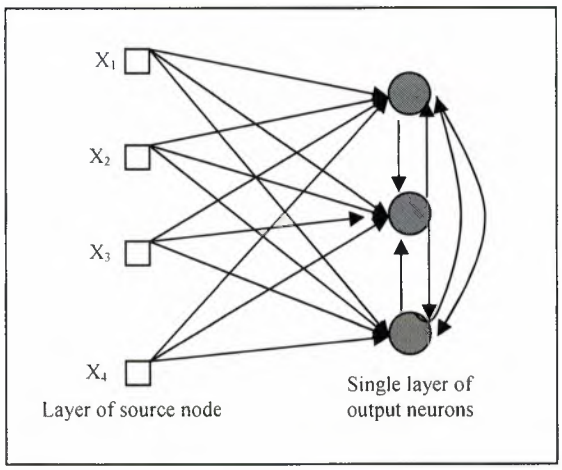


Figure 1.10 Feedback Connections Among the Neurons. [23]

For a neuron k, to be the winning neuron, its induced local field v_k for a specified input pattern. X must be the largest among all the neurons in the network. The output signal y_k, of winning neurons k is set equal to one. The output signals of all the neurons that lose the competition are set equal to zero. We thus write:

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

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 neurons is allotted a fixed amount of synaptic weight, which is distributed among its input node that is:

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

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

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

Where η is the learning rate parameter this has 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 named in honor of Ludwig Boltzmann is a stochastic learning algorithm derived from ideas rooted in statistical mechanics. In a Boltzmann machine, the neurons constitute a recurrent structure and they operate in a binary manner. Since, for example, they are either in an on state denoted by +1 or in an off state denoted of which is determined by the particular states occupied by the individual neurons of the machine as shown by:

$$E = -\frac{1}{2}\sum_j \sum_k w_{kj} x_k x_j \tag{1.15}$$

Where x_j is the state of neuron j and w_{kj} is 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 has self feedback. 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.

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

Where ΔE_k is the energy change resulting from such a flip notice that T is 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. There are two modes of operation to be considered.

- Clamped condition in which the visible neurons are all clamped onto specific states determined by the environment.
- Free running condition in which all the neurons visible and hidden are allowed to 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:

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

Where η is a learning rate parameter, note that both p_{kj}^{+} and p_{kj}^{-} range in value from –1 to +1.

1.8 Learning Tasks

In this context we will identify six learning tasks that apply to the use of neural network in one form or another.

a. Pattern Association

An associative memory is a brain-like, distributed memory that learns by association. Association has been known to be a prominent feature of human memory since Aristotle and all models of cognition use in one form or another as the basic operation. There are two phases involved in the operation of an associative memory:

- 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 are good at pattern recognition. We receive data from the world around us via our senses and are able to recognize the source of the data.

Pattern recognition is formally defined as the process whereby a received pattern/signal is assigned to one of a prescribed number of classes (categories).

c. Function Approximation

The third learning task of interest is that of function approximation.

d. Control

The control of a plant is another learning task that can be done by a neural network; by a plant we mean a process or critical part of a system that is to be maintained in a controlled condition.

e. Filtering

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

f. Beamforming

Beamforming is a spatial form of filtering and is used to distinguish between the spatial properties of a target signal and background noise. The device used to do the beamforming is called a "beamformer."

1.9 Activation Functions

This threshold function is generally some form of nonlinear function. One simple nonlinear function that is appropriate for discrete neural nets is the step function. One variant of the step function is:

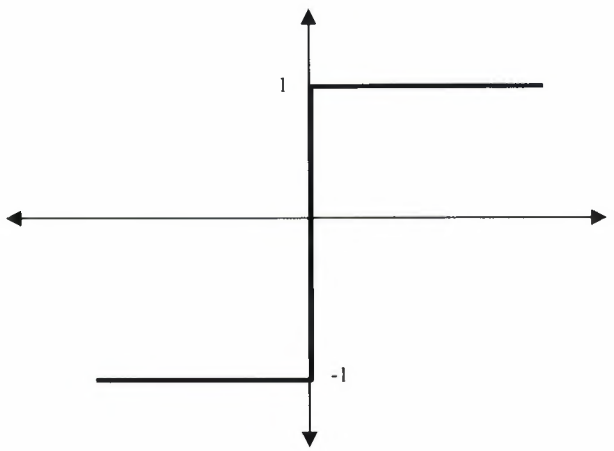


Figure 1.11 Hard Activation Functions

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

Where f' (x) refers to the previous value of f(x) (that is the activation of the neuron will not change)

Where x is the summation (over all the incoming neurons) of the product of the incoming neuron's activation, and the connection:

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

The number of incoming neurons, is A the vector of incoming neurons and w is the vector of synaptic weights connecting the incoming neurons to the neurons we are examining. One more appropriate to analog is the sigmoid, or squashing, function; an example is the logistic functions illustrated in figure 1.12.

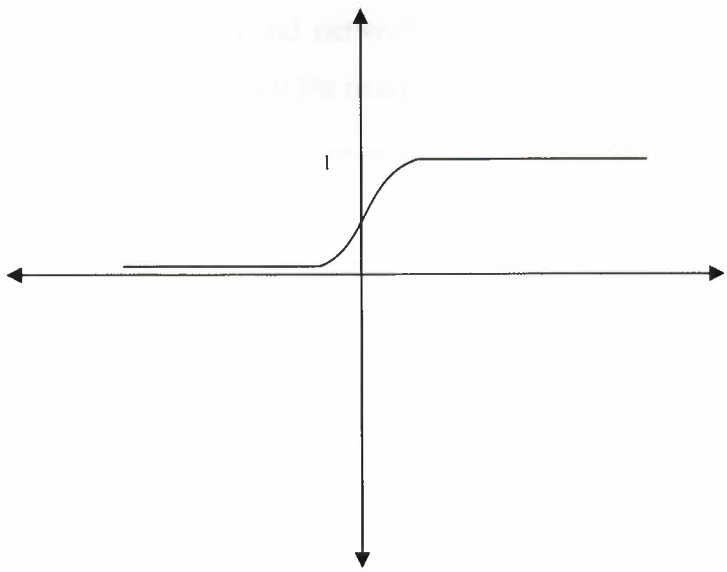


Figure 1.12 Sigmoid Functions

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

Another popular alternative is:

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

The most important characteristic of our activation function is that it is nonlinear. If we wish to use activation function as a multiplayer network, the activation function must be nonlinear, or the computational will be equivalent to a single-layer network.

1.9.1 A.N.N.

All of the knowledge that a neural network possesses is stored in the synapses. The weights of the connections between the neurons of diagram of the synapse layer model.

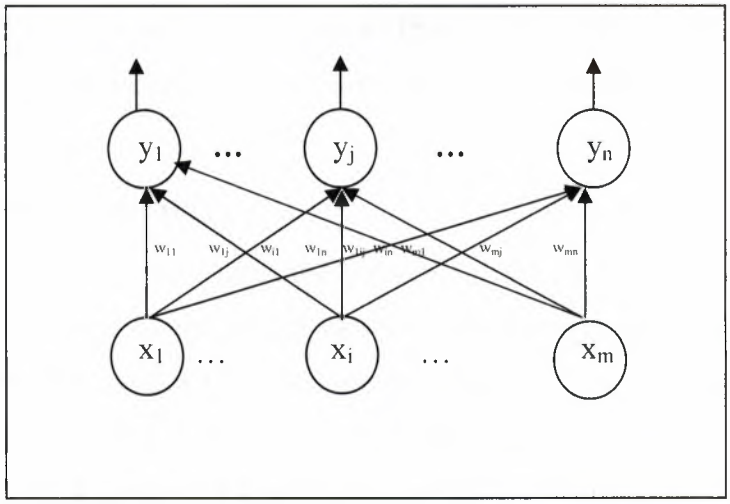


Figure 1.13 Diagram of Synapse Layer Model

However the network acquires that knowledge, this happens during training$\alpha\alpha$ g pattern associations are presented to the network in sequence, and the weights are adjusted to capture this knowledge. The weight adjustment scheme is known as the "learning law". One of the first learning methods formulated was Hebbian Learning.

Donald Hebb, in his organization of behavior formulated the concept of "correlation learning". This is the idea that the weight of a connection is adjusted based on the values of the neurons its connects:

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

Where α is the learning rate a_i is the activation of the ith neuron in one neuron layer, a_j is the activation of the jth neuron in another layer, and w_{ij} is the connection strength between the two neurons. A variant of this learning rule is the signal Hebbian Law:

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

S is a sigmoid

1.9.2 Unsupervised learning

One method of learning is the unsupervised learning method. In general, an unsupervised learning method is one in which weight adjustments are not made based on comparison with some target output. There is no teaching signal feed into the weight adjustments. This property is also known as self – organization.

1.9.3 Supervised learning

In many models, learning takes the form of supervised training. I present input pattern one after the other to the neural network and observe the recalled output pattern in comparison with our desired result, there is needed some way of adjusting the weights which takes into account any error in the output pattern.

An example of a supervised learning law is the Error Correction Law:

$$\Delta w_{ij} = \alpha a_i \left[c_j - b_j \right] \tag{1.24}$$

A before α is again the learning rate, ai the activation of the ith neuron, bj is the activation of the jth neuron in the recalled pattern, and cj is the deired activation of the jth 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 is similar to supervised learning except that each ouput neuron gets an error value. Only one error value is computed for each ouput neuron. The weight adjustment formula is then:

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

Again α is the learning rate, v is the single value indicting the total error of the output pattern, and Θ is the threshold value for the jth output neuron. We need to spread out this generalized error for the jth output neuron to each of the incoming i neurons, is a value representing the eligibility of the weight for updating. This may be computed as:

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

Where g_i is the probability of the output being correct given the input from the ith incoming neuron. (This is vague description; the probability function is of necessity a heuristic estimate and manifests itself differently from specific model to specific model).

1.10 Back propagation Model

Back propagation of errors is a relatively generic concept. The Back propagation model is applicable to a wide class of problems. It is certainly the predominant supervised training algorithm. Supervised learning implies that we must have a set of good pattern associations to train with. The back propagation model presented in figure 1.14.

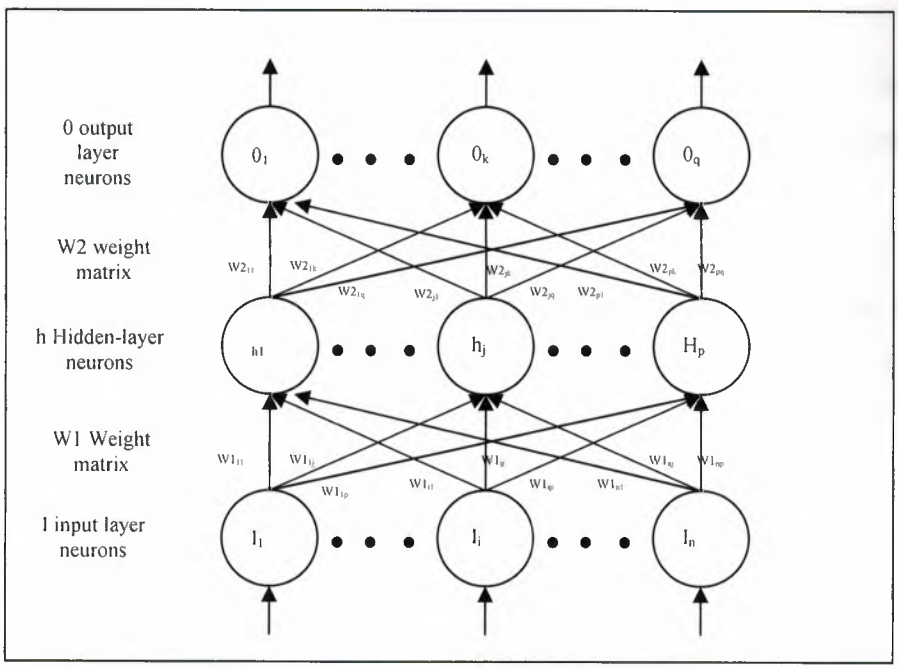


Figure 1.14 Diagram of Back propagation Topology. [23]

It has three layers of neurons: an input layer, a hidden layer, and an output layer. There are two layers of synaptic weights. There is a learning rate term, α in the subsequent formulas indicating how much of the weight changed to effect on each pass this is typically a number between 0 and 1. There is a momentum term Θ indicating how much a previous weight change should influence the current weight change. There is also a term indicating within what tolernce we can accept an output as good.

1.10.1 Back Propagation Algorithm

Assign random values between –1 and +1 to the weghts between the input and hidden layers, the weights between the hidden and output layers, and the thershold for the hidden layer and output layer neurnos train the network by preforming the following procedure for all pattern pairs:

Forward Pass.

1. Computer the hidden layer neuron activations:

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

   Where h is the vector of hidden layer neurons i is 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 FO is a sigmoid activation function we will use the logistic function:

   $$f(x)=\frac{1}{1+e^{-x}} \tag{1.29}$$

   Backward Pass.

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

   $$d=O(1-O)(O-t) \tag{1.30}$$

   Where d is the vector of errors for each output neuron, o is the output layer, and t is the target correct activation of the output layer.

4. Compute the hidden layer error:

$$e = h(1-h)W2d \tag{1.31}$$

Where is e is the vector of errors for each hidden layer neuron.

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

$$W2 = W2 + \Delta W2 \tag{1.32}$$

Where $\Delta W2$ is a matrix representing the change in matrix $W2$. It is computed as follows:

$$\Delta W2_t = \alpha hd + \Theta \Delta W2_{t-1} \tag{1.33}$$

Where α is the learning rate, and Θ is 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 is somehow incorporated into the mode. It means only that a weight adjustment has been made. This could also be called a cycle.

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

$$W1 = W1 + W1_t \tag{1.34}$$

Where

$$W1_t = \alpha ie + \Theta \Delta W1_{t-1} \tag{1.35}$$

Repeat step 1 to 6 on all pattern pairs until the output layer error (vector d) is within the specified tolerance for each pattern and for each neuron.

Recall:

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

- Compute the hidden layer activation:

$$h = F(W1i) \tag{1.36}$$

- Computer the output layer:

$$O = F(W2h) \tag{1.37}$$

The vector o is our recalled pattern.

1.10.2 Strengths and Weaknesses

The Back Propagation Network has the ability to learn any arbitrarily complex nonlinear mapping this is due to the introduction of the hidden layer. It also has a capacity much greater than the dimensionality of its input and output layers as we will see later. This is not true of all neural net models.

However Back propagation can involve extremely long and potentially infinite training time. If you have a strong relationship between input and outputs and you are willing to accept results within a relatively broad time, your training time may be reasonable.

1.11 Summary

In this chapter the followings were discussed Perceptron Algorithm, supervised and unsupervised algorithms, Neural network definition, some history of the Neural network, Natural Neuron, Artificial Neuron, the Back propagation algorithm and their models, Learning processes and their tasks, and the Activation function.

2. IMAGE PROCESSING

2.1 Overview

This chapter presents an overview of image processing, image analysis systems, dividing the spectrum of techniques in image analysis into three basic areas is conceptually useful. Finally, *high-level processing* involves recognition and interpretation, the principal subjects of this chapter.

2.2 Introduction

Image analysis is a process of discovering, identifying, and understanding patterns that are relevant to the performance of an image-based task. One of the principal goals of image analysis by computer is to endow a machine with the capability to approximate, in so me sense, a similar capability in human beings. For example, in a system for automatically reading images of typed documents, the patterns of interest are alphanumeric characters, and the goal is to achieve character recognition accuracy that is as close as possible to the superb capability exhibited by human beings for performing such tasks.

Thus an automated image analysis system should be capable of exhibiting various degrees of intelligence. The concept of *intelligence* is somewhat vague, particularly with reference to a machine. However, conceptualizing various types of behavior generally associated with intelligence is not difficult. Several characteristics come immediately to mind: (1) the ability to extract pertinent information from a background of irrelevant details; (2) the capability to learn from examples and to generalize this knowledge so that it will apply in new and different circumstances; and (3) the ability to make inferences from incomplete information.

Image analysis systems with these characteristics can be designed and implemented for *limited* operational environments. However, we do not yet know how to endow these systems with a level of performance that comes even close to emulating human capabilities in performing general image analysis functions. Research in biological and computational systems continually is uncovering new and promising theories to explain

human visual cognition. However, the state of the art in computerized image analysis for the most part is based on heuristic formulations tailored to solve specific problems. For example, some machines are capable of reading printed, properly formatted documents at speeds that are orders of magnitude faster than the speed that the most skilled human reader could achieve. However, systems of this type are highly specialized and thus have little or no extendability.

2.3 Elements of Image Analysis

Dividing the spectrum of techniques in image analysis into three basic areas is conceptually useful. These areas are (1) low-level processing, (2) intermediate level processing, and (3) high-level processing. Although these subdivisions have no definitive boundaries, they do provide a useful framework for categorizing the various processes that are inherent components 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 exist For example, thresholding may be viewed as an enhancement (preprocessing) or a segmentation tool, depending on the application.

Low-level processing deals with functions that may be viewed as automatic reactions, requiring no intelligence on the part of the image analysis system. We treat image acquisition and preprocessing as low-level functions. This classification encompasses activities from the image formation process itself to compensations, such as noise reduction or image deblurring. Low-level functions may be compared 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 a suitable image is available. The process followed by the brain in adapting the visual system to produce such an image is an automatic, unconscious reaction.

Intermediate-level processing deals with the task of extracting and characterizing components (say, regions) in an image resulting from a low-level process. As figure 2.1 indicates, intermediate-level processes encompass segmentation and description, using techniques. Some capabilities for intelligent behavior have to be built into flexible segmentation procedures. For example, bridging small gaps in a segmented boundary

involves more sophisticated elements of problem solving than mere low-level automatic reactions.

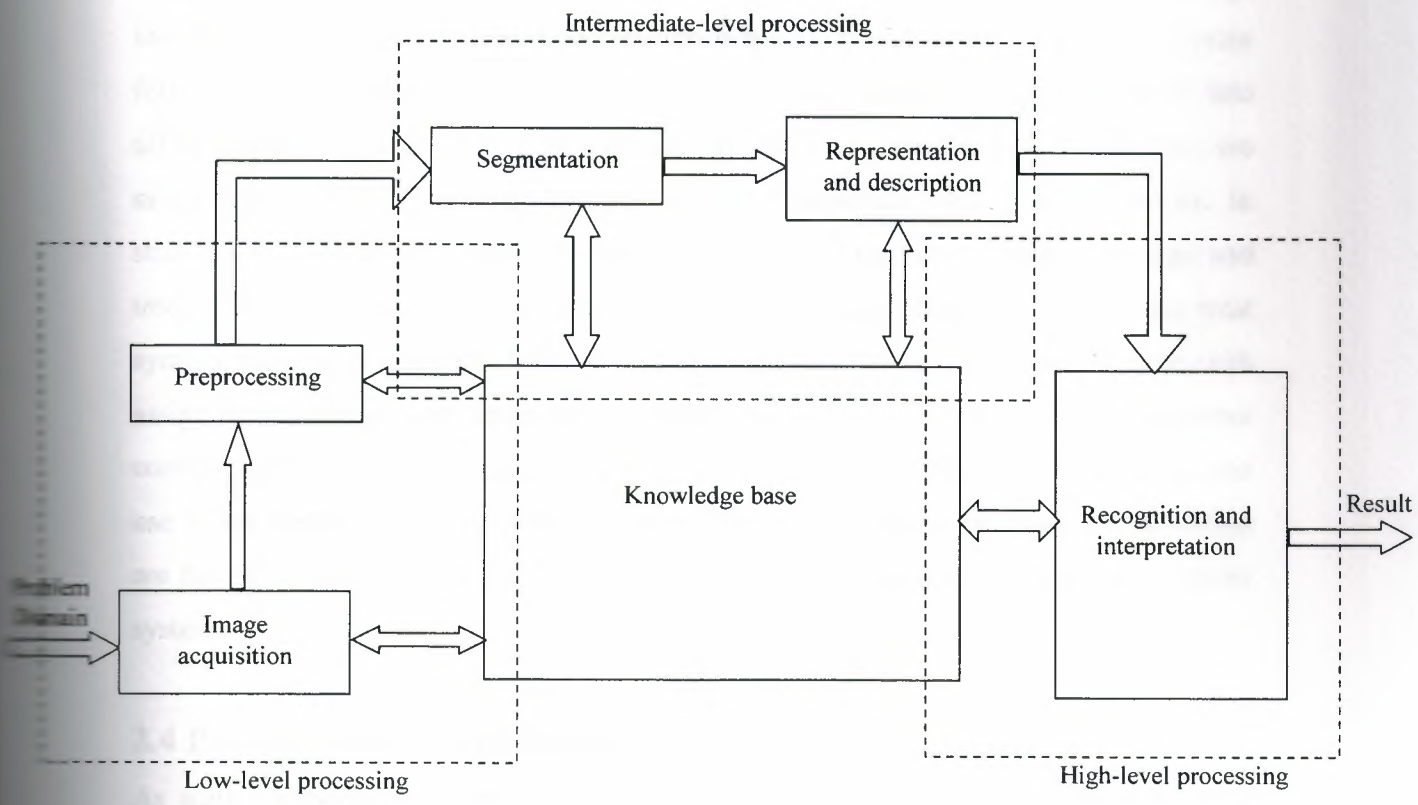


Figure 2.1 Elements of Image Analysis

Finally, *high-level processing* involves recognition and interpretation, the principal subjects of this chapter. These two processes have a stronger resemblance to what generally is meant by 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, as we venture into recognition, and especially into interpretation, our knowledge and understanding of fundamental principles becomes far less precise and much 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 end product is a system with highly specialized operational capabilities.

The material in the following sections deals with: (1) decision-theoretic methods for recognition, (2) structural methods for recognition, and (3) methods for image interpretation. Decision-theoretic recognition is based on representing patterns in vector form and then seeking approaches for grouping and assigning pattern vectors into different pattern classes. The principal approaches to decision-theoretic recognition are minimum distance classifiers, correlators, Bayes classifiers, and neural networks. In structural recognition, patterns are represented in symbolic form (such as strings and trees), and recognition methods are based on symbol matching or on models that treat symbol patterns as sentences from an artificial language. Image interpretation deals with assigning meaning to an ensemble of recognized image elements. The predominant concept underlying image interpretation methodologies is the effective organization and use of knowledge about a problem domain. Current techniques for image interpretation are based on predicate logic, semantic networks, and production (in particular, *expert)* systems.

2.4 Patterns and Pattern Classes

As stated in Section 2.2, the ability to perform pattern recognition at some level is fundamental to image analysis. Here, a *pattern* is a quantitative or structural description of an object or some other entity of interest in an image. In general, a pattern is formed by one or more descriptors. In other words, a pattern is an arrangement of descriptors. (The name *features* is of ten used in the pattern recognition literature to denote descriptors.) A *pattern class* is a family of patterns that share some common properties. Pattern classes are denoted ω_1, ω_2, ..., ω_M where M is 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.

2.5 Error Matrics

Two of the error metrics used to compare the various image compression techniques are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The MSE is the cumulative squared error between the compressed and the original image, whereas PSNR is a measure of the peak error. The mathematical formulae for the two are

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

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

Where I(x,y) is the original image, I'(x,y) is the approximated version (which is actually the decompressed image) and M,N are the dimensions of the images. A lower value for MSE means lesser error, and as seen from the inverse relation between the MSE and PSNR, this translates to a high value of PSNR. Logically, a higher value of PSNR is good because it means that the ratio of Signal to Noise is higher. Here, the 'signal' is the original image, and the 'noise' is the error in reconstruction. So, if you find a compression scheme having a lower MSE (and a high PSNR), you can recognize that it is a better one.

2.6 The Outline

We'll take a close look at compressing grey scale images. The algorithms explained can be easily extended to color images, either by processing each of the color planes separately, or by transforming the image from RGB representation to other convenient representations like YUV in which the processing is much easier.

The usual steps involved in compressing an image are

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, such that the distortion is a minimum.
4. Quantize each class separately using the bit allocation information derived in step 3.
5. Encode each class separately using an entropy coder and write to the file.

Remember, this is how 'most' image compression techniques work. But there are exceptions. One example is the Fractal Image Compression technique, where possible self similarity within the image is identified and used to reduce the amount of data required to reproduce the image. Traditionally these methods have been time consuming, but some latest methods promise to speed up the process.

Reconstructing the image from the compressed data is usually a faster process than compression. The steps involved are

1. Read in the quantized data from the file, using an entropy decoder. (Reverse of step 5).
2. Dequantize the data. (Reverse of step 4).
3. Rebuild the image. (Reverse of step 2).

2.6.1 Classifying Image Data

An image is represented as a two-dimensional array of coefficients, each coefficient representing the brightness level in that point. When looking from a higher perspective, we can't differentiate between coefficients as more important ones, and lesser important ones. But thinking more intuitively, we can. Most natural images have smooth color variations, with the fine details being represented as sharp edges in between the smooth variations. Technically, the smooth variations in color can be termed as low frequency variations and the sharp variations as high frequency variations.

The low frequency components (smooth variations) constitute the base of an image, and the high frequency components (the edges which give the detail) add upon them to refine the image, thereby giving a detailed image. Hence, the smooth variations are demanding more importance than the details.

Separating the smooth variations and details of the image can be done in many ways. One such way is the decomposition of the image using a Discrete Wavelet Transform (DWT).

2.6.2 The DWT of an Image

The procedure goes like this. A low pass filter and a high pass filter are chosen, such that they exactly halve the frequency range between themselves. This filter pair is called the Analysis Filter pair. First, the low pass filter is applied for each row of data, thereby getting the low frequency components of the row. But since the LPF is 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 can be sub-sampled by two, so that the output data now contains only half the original number of samples. Now, the high pass filter is applied for the same row of data, and similarly the high pass components are separated, and placed by the side of the low pass components. This procedure is done for all rows.

Next, the filtering is done for each column of the intermediate data. The resulting two-dimensional array of coefficients contains four bands of data, each labeled as LL (low-low), HL (high-low), LH (low-high) and HH (high-high). The LL band can be decomposed once again in the same manner, thereby producing even more sub-bands. This can be done up to any level, thereby resulting in a pyramidal decomposition as shown below.

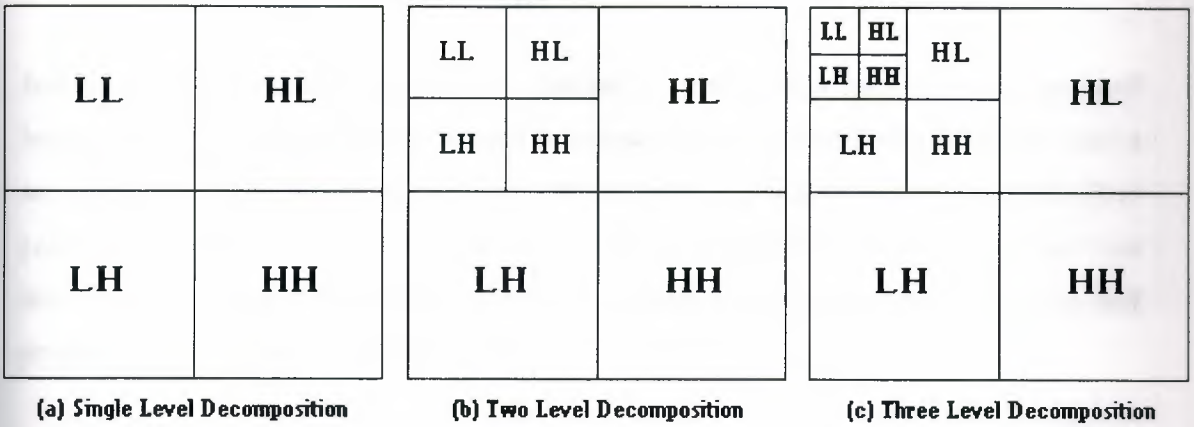


Figure 2.2 Pyramidal Decomposition of an Image

As mentioned above, the LL band at the highest level can be classified as most important, and the other 'detail' bands can be classified as of lesser importance, with the degree of importance decreasing from the top of the pyramid to the bands at the bottom.

Figure 2.3 The Three Layer Decomposition of the 'Lena' Image.

2.7 The Inverse DWT of an Image

Just as a forward transform used to separate the image data into various classes of importance, a reverse transform is used to reassemble the various classes of data into a reconstructed image. A pair of high pass and low pass filters is used here also. This filter pair is called the Synthesis Filter pair. The filtering procedure is just the opposite - we start from the topmost level, apply the filters column-wise first and then row-wise, and proceed to the next level, till we reach the first level.

2.7.1 Bit Allocation

The first step in compressing an image is to segregate the image data into different classes. Depending on the importance of the data it contains, each class is allocated a portion of the total bit budget, such that the compressed image has the minimum possible distortion. This procedure is called Bit Allocation.

The Rate-Distortion theory is often used for solving the problem of allocating bits to a set of classes, or for bit-rate control in general. The 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 is given in [1], which is explained below.

1. Initially, all classes are allocated a predefined maximum number of bits.
2. For each class, one bit is reduced from its quota of allocated bits, and the distortion due to the reduction of that 1 bit is calculated.
3. Of all the classes, the class with minimum distortion for a reduction of 1 bit is noted, and 1 bit is reduced from its quota of bits.
4. The total distortion for all classes D is calculated.
5. The total rate for all the classes is calculated as R = p(i) * B(i), where p is the probability and B is 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, we keep on reducing one bit at a time till we achieve optimality either in distortion or target rate, or both. An alternate approach which is also mentioned in [1] is to initially start with zero bits allocated for all classes, and to find the class which is most 'benefited' by getting an additional bit. The 'benefit' of a class is defined as the decrease in distortion for that class.

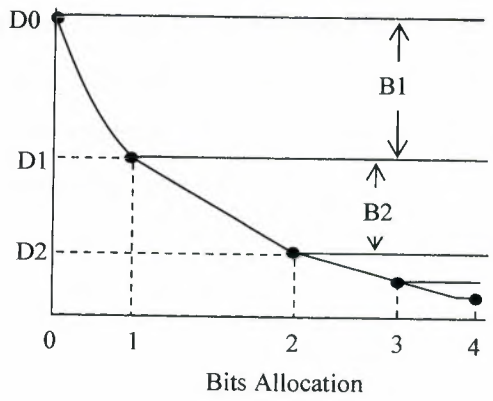


Figure 2.4 'Benefit' of a Bit is the Decrease in Distortion Due to Receiving that Bit.

As shown above, the benefit of a bit is a decreasing function of the number of bits allocated previously to the same class. Both approaches mentioned above can be used to the Bit Allocation problem.

2.7.2 Quantization

Quantization refers to the process of approximating the continuous set of values in the image data with a finite (preferably small) set of values. The input to a quantizer is the original data, and the output is always one among a finite number of levels. The quantizer is a function whose set of output values are discrete, and usually finite. Obviously, this is a process of approximation, and a good quantizer is one which represents the original signal with minimum loss or distortion.

There are two types of quantization - Scalar Quantization and Vector Quantization. In scalar quantization, each input symbol is treated separately in producing the output, while in vector quantization the input symbols are clubbed together in groups called vectors, and processed to give the output. This clubbing of data and treating them as a single unit increases the optimality of the vector quantizer, but at the cost of increased computational complexity. Here, we'll take a look at scalar quantization.

A quantizer can be specified by its input partitions and output levels (also called reproduction points). If the input range is divided into levels of equal spacing, then the quantizer is termed as a Uniform Quantizer, and if not, it is termed as a Non-Uniform Quantizer. A uniform quantizer can be easily specified by its lower bound and the step size. Also, implementing a uniform quantizer is easier than a non-uniform quantizer. Take a look at the uniform quantizer shown below. If the input falls between n*r and (n+1)*r, the quantizer outputs the symbol n.

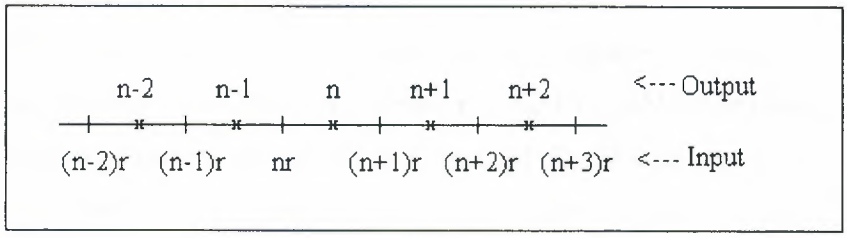


Figure 2.5 a Uniform Quantizer

Just the same way a quantizer partitions its input and outputs discrete levels, a Dequantizer is one which 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. It can be seen from literature, that the optimum quantizer (encoder) and optimum dequantizer (decoder) must satisfy the following conditions.

- Given the output levels or partitions of the encoder, the best decoder is one that puts the reproduction points x' on the centers of mass of the partitions. This is known as *centroid condition.*
- Given the reproduction points of the decoder, the best encoder is one that puts the partition boundaries exactly in the middle of the reproduction points, i.e. each **x** is

translated to its nearest reproduction point. This is known as *nearest neighbour condition.*

The quantization error (x - x') is used as a measure of the optimality of the quantizer and dequantizer.

2.8 Object Recognition

Object recognition consists of locating the positions and possibly orientations and scales of instances of objects in an image. The purpose may also be to assign a class label to a detected object. Our survey of the literature on object recognition using ANNs indicates that in most applications, ANNs have been trained to locate individual objects based direction pixel data. Another less frequently used approach is to map the contents of a window onto a feature space that is provided as input to a neural classifier.

2.8.1 Optical Character Recognition

The recognition of handwritten or printed text by computer is referred to as Optical Character Recognition. When the input device is a digitizer tablet that transmits the signal in real time (as in pen-based computers and personal digital assistants) or includes timing information together with pen position (as in signature capture) we speak of dynamic recognition. When the input device is a still camera or a scanner, which captures the position of digital ink on the page but not the order in which it was laid down, we speak of static or image-based OCR.

Dynamic OCR is an increasingly important modality in Human Computer I interaction, and the difficulties encountered in the process are largely similar to those found in other HCI modalities, in particular, Speech Recognition. The stream of position/pen pressure values output by the digitizer tablet is analogous to the stream of speech signal vectors output by the audio processing front end, and the same kinds of lossy data compression techniques, including cepstral analysis, linear predictive coding, and vector quantization, are widely employed for both.

Static OCR encompasses a range of problems that have 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 (deskewing), and the appropriate grouping of titles, headers, footers, and other material set in a font different from the main body of the text. Another OCR-specific problem is that we often find different scripts, such as Kanji and Kana, or Cyrillic and Latin, in the same running text.

While the early experimental OCR systems were often rule-based, by the eighties these have been completely replaced by systems based on statistical, Pattern Recognition. For clearly segmented printed materials such techniques offer virtually error-free OCR for the most important alphabetic systems including variants of the Latin, Greek, Cyrillic, and Hebrew alphabets.
However, when the number of symbols is large, as in the Chinese or Korean writing systems, or the symbols are not separated from one another, as in Arabic or Devanagari print, OCR systems are still far from the error rates of human readers, and the gap between the two is also evident when the quality of the image is compromised e.g. by fax transmission. Until these problems are resolved, OCR can not play the pivotal role in the transmission of cultural heritage to the digital age that it is often assumed to have.

In the recognition of handprint, algorithms with successive segmentation, classification, and identification (language modeling) stages are still in the lead, as shown in the later chapters.

2.9 Summary

This chapter presented an introduction to the image processing, Elements of image analysis, Patterns and pattern classes, Classifying of image data, The DWT of an image Bit allocation, Quantization, Optical Character Recognition, and Character Recognition.

3. IMAGE PROCESSING AND NEURAL NETWORKS

3.1 Overview

This chapter presents an overview of image processing with neural networks, history, taxonomy for image processing algorithms, neural networks in image processing, preprocessing, image reconstruction, image restoration, image enhancement, applicability of neural networks in preprocessing, data reduction and feature extraction, image compression applications, feature extraction applications, image segmentation, image segmentation based on pixel data, image segmentation based on features, open issues in applications of neural networks, segmentation by ANNs, object recognition, and real-world.

3.2 Introduction

Techniques from statistical pattern recognition have, since the revival of neural networks, obtained a widespread use in digital image processing. Initially, part recognition problems were often solved by linear and quadratic discriminants [3] or the (nonparametric) k-nearest neighbor classifier and the Parzen density estimator [4,5]. In the mid-eighties, the PDP group together with others introduced the back-propagation learning algorithm for neural networks. This algorithm for the first time made it feasible .to train a non-linear neural network equipped with layers of the so-called hidden nodes. Since then, neural networks with one or more hidden layers can, in theory, be trained to perform virtually any regression or discrimination task. Moreover, no assumptions are made as with respect to the type of underlying (parametric) distribution of the input variables, which may be nominal, ordinal, real or any combination hereof 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 turned out to be right. In this review article, we survey applications of neural networks developed to solve different problems in image processing (for a review of neural networks

used for ID signal processing, [1]). There are two central questions which we win try to answer in this review article:

1. What are major applications of neural networks in image processing now and in the nearby future?
2. Which are the major strengths and weaknesses of neural networks for solving image processing tasks?

To facilitate a systematic review of neural networks in image processing, we propose a two-dimensional taxonomy for image processing techniques. This taxonomy establishes a framework in which the advantages and unresolved problems can be structured in relation to the application of neural networks in image processing.

3.3 Image Processing Algorithms

Traditional techniques from statistical pattern recognition like the Bayesian discriminant and the Parzen windows were popular until the beginning of the 1990s. Since then, neural networks (ANNs) have increasingly been used as an alternative to classic pattern classifiers and clustering techniques. Non-parametric feed-forward ANNs quickly turned out to be attractive trainable machines for feature-based segmentation and object recognition. When no gold standard is available, the self-organizing feature map (SOM) is an interesting alternative to supervised techniques. It may learn to discriminate, e.g., deferent textures when provided with powerful features. The current use of ANNs in image processing exceeds the aforementioned traditional applications. The role of feed-forward ANNs and SOMs has been extended to encompass also low-level image processing tasks such as noise suppression and image enhancement. Hopfield ANNs were introduced as a tool for finding satisfactory solutions to complex (NP-complete) optimization problems. This makes them an interesting alternative to traditional optimization algorithms for image processing tasks that can be formulated as optimization problems. The deferent problems addressed in the field of digital image processing can be organized into what we have chosen to call the image processing chain. We make the following distinction between steps in the image processing chain as in figure 3.1.

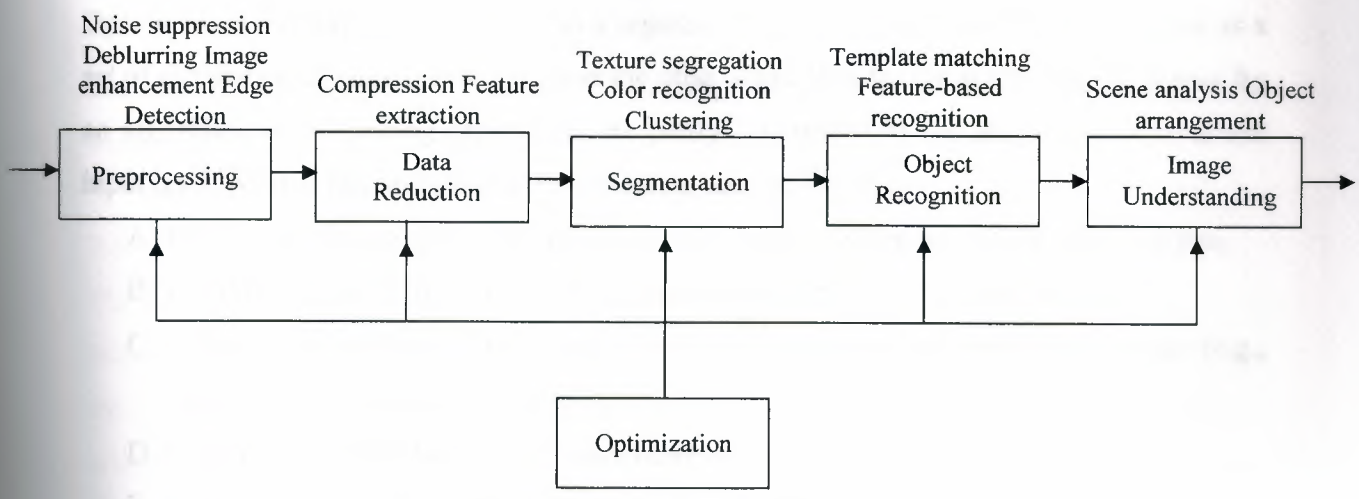


Figure 3.1 Graphs Matching Automatic Thresholding

Figure 3.1. The image processing chain containing the five deferent tasks: preprocessing, data reduction, segmentation, object recognition and image understanding. Optimization techniques are used as a set of auxiliary tools that are available in all steps of the image processing chain.

1. Preprocessing/filtering. Operations that give as a result a modified image with the same dimensions as the original image (e.g., contrast enhancement and noise reduction).
2. Data reduction/feature extraction. Any operation that extracts significant components from an image (window). The number of extracted features is generally smaller than the number of pixels in the input window.
3. Segmentation. Any operation that partitions an image into regions that are coherent with respect to some criterion. One example is 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 which may be used for, e.g., graph matching or object delineation.

Optimization techniques are not seen as a separate step in the image processing chain but as a set of auxiliary techniques, which support the other steps. Besides the actual task performed by an algorithm, its processing capabilities are partly determined by the abstraction level of the input data. We distinguish between the following abstraction levels:

A. Pixel level. The intensities of individual pixels are provided as input to the algorithm.

B. Local feature level. A set of derived, pixel-based features constitutes the input.

C. Structure (edge) level. The relative location of one or more perceptual features (e.g., 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.

Table 3.1 contains the taxonomy of image processing algorithms that results from combining the steps of the image processing chain with the abstraction level of the input data.

Table 3.1 Image Processing Tasks Categorized into a Two-Dimensional Taxonomy

	Preprocessing	Compression/feature	Segmentation	Recognition	Image understanding	Optimization
Pixel	26	25	39	51	3	5
Feature	4	2	19	38	2	3
Structure			2	6		5
Object						1
Object set				2	2	
Scene						

3.4 Neural Networks in Image Processing

In this section, we will review neural networks trained to perform one of the six tasks in the image processing chain (3.1-3.6).

3.4.1 Preprocessing

The first step in the image processing chain consists of preprocessing. Loosely defined, by preprocessing we mean any operation of which the input consists of sensor data, and of which the output is a full image. Preprocessing operations generally fall into one of three categories: image reconstruction (to reconstruct an image from a number of sensor measurements), image restoration (to remove any aberrations introduced by the sensor, including noise) and image enhancement (accentuation of certain desired features, which may facilitate later processing steps such as segmentation or object recognition).

Applications of ANNs in these three preprocessing categories win be discussed separately below. The majority of the ANNs were applied directly to pixel data (level A); only four networks were applied to more high-Level data (features, level B).

3.4.2 Image Reconstruction

Image reconstruction problems often require quite complex computations and a unique approach is needed for each application. An ADALINE network is trained to perform an electrical impedance tomography (EIT) reconstruction, i.e., a reconstruction of a 2D, image based on ID measurement on the circumference of the image. Srinivasan et al. [9] trained a modified Hopfield network to perform the inverse Radon transform (e.g., 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 image s from electron holograms. Wang and Wahl trained a Hopfield ANN for reconstruction of 2D images from pixel data obtained from projections [11].

3.4.3 Image Restoration

The majority of applications of ANNs in preprocessing can be found in image restoration. In general, one wants to restore an image that is distorted by the (physical) measurement system. The system might introduce noise, motion blur, out-of-focus blur, distortion caused by low resolution, etc. Restoration can employ all information about the nature of the distortions introduced by the system, e.g., the point spread function. The restoration problem is ill-posed because conflicting criteria need to be fulfilled: resolution versus smoothness.

The neural-network applications we reviewed had various designs ranging from relatively straightforward to highly complex, modular approaches. In the most basic image restoration approach, noise is removed from an image by simple filtering. Greenhil 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 et al. built a modular feed-forward ANN 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 ANN training may not be 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 (CNNs) for image processing. A CNN is a system in which nodes are locally connected. Each node contains a feedback template 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, a disadvantage is that the parameters influencing the network behavior (the feedback and control templates) have to be set by hand.

Others have proposed methods for training CNNs, e.g., using gradient descent or genetic algorithms (grey-value images, Zamparelli). CNNs were also applied for restoration of color images by Lee and Degyvez.

Another interesting ANN architecture is the generalized adaptive neural filter (GANF) which has been used for noise suppression. A GANF consists of a set of neural operators, based on stack a filter that uses binary decompositions of grey-value data. Finally, fuzzy ANNs and the neurochips have been applied to image restoration as well. Traditional methods for more complex restoration problems such as deblurring and diminishing out-of-focus defects are maximum a posteriori estimation (MAP) and regularization. Applying these techniques entails solving high dimensional convex optimization tasks. The objective functions of MAP estimation or the regularization problem can both be mapped onto the energy function of the Hopfield network. Often, mapping the problem turned out to be difficult, so in some cases the network architecture had to be modified as well.

Other types of networks have also been applied to image restoration. Qian et al. developed a hybrid system consisting of order statistic filters for noise removal and a Hopfield network for deblurring (by optimizing a criterion function). The modulation transfer function had to be measured in advance. Guan et al. developed a so-called network-of-networks for image restoration. Their system consists of loosely coupled modules, where each module is a separate ANN. Phoha and Oldham proposed a layered, competitive network to reconstruct a distorted image.

3.4.4 Image Enhancement

The goal of image enhancement is to amplify specific (perceptual) features. Among the applications where ANNs have been developed for image enhancement, one would expect most applications to be based on regression ANNs. However, several enhancement approaches rely on a classifier, typically resulting in a binary output image. The most well-known enhancement problem is edge detection. A straightforward application of regression feed-

forward ANNs, trained to behave like edge detectors, was reported by Pugmire et al. Chandresakaran et al. used a novel feed-forward architecture to classify an input window as containing an edge or not. The weights of the network were set manually instead of being obtained from training. A number of more complex, modular systems were also proposed. Formulating edge detection as an optimization problem made it possible for Tsai et al. to train a Hopfield network for enhancement of end cardiac borders. Some enhancement approaches utilize other types of ANNs. Shih et al. applied an ART network for binary image enhancement. Moh and Shih describe a general approach for implementation of morphological image operations by a modified feed-forward ANN using shunting mechanisms, i.e., neurons acting as switches. Waxman et al. consider the application of a centre-surround shunting feed-forward ANN (proposed by Grossberg) for contrast enhancement and color night vision.

3.4.5 Applicability of Neural Networks in Preprocessing

There seem to be three types of problems in preprocessing (unrelated to the three possible operation types) to which ANNs can be applied:

- Optimization of an objective function defined by a traditional preprocessing problem;
- Approximation of a mathematical transformation used for image reconstruction, e.g., by regression;
- Mapping by an ANN trained to perform a certain task, usually based directly on pixel data (neighborhood input, pixel output).

To solve the first type of problems, traditional methods for optimization of some objective function may be replaced by a Hopfield network. For a further discussion of the suitability of Hopfield networks for solving optimization problems.

For the approximation task, regression (feed-forward) ANNs could be applied. Although some applications such ANNs were indeed successful, it would seem that these applications call for more traditional mathematical techniques, because a guaranteed (worst-case) performance is crucial in preprocessing.

In several other applications, regression or classilcation (mapping) networks were trained to perform image restoration or enhancement directly from pixel data. A remarkable finding was that non-adaptive ANNs (e.g., CNNs) were often used for preprocessing.

Secondly, when networks were adaptive, their architectures usually differed much from those of the standard ANNs: prior knowledge about the problem was used to design the networks that were applied for image restoration or enhancement (e.g., by using shunting mechanisms to force a feed-forward ANN to make binary decisions). The interest in non adaptive ANNs indicates that the fast, parallel operation and the ease with which ANNs can be embedded in hardware may be important criteria when choosing for a neural implementation of a specific preprocessing operation. However, the ability to learn from data is apparently of less importance in preprocessing. While it is relatively easy to construct a linear filter with a certain, desired behavior, e.g., by specifying its frequency profile, it is much harder to obtain a large enough data set to learn the optimal function as a high-dimensional regression problem. This holds especially when the desired network behavior is only critical for a small subset of all possible input patterns (e.g., in edge detection). Moreover, it is not at all trivial to choose a suitable error measure for supervised training, as simply minimizing the mean squared error might give undesirable results in an image processing setting.

An important caveat is that the network parameters are likely to become tuned to one type of image (e.g., a specific sensor, scene setting, scale, etc.), which limits the applicability of the trained ANN. When the underlying conditional probability distributions, $p(x|w_j)$ or $p(y|x)$, change, the classifieation or regression network-like all statistical models-needs to be retrained.

3.5 Data Reduction and Feature Extraction

Two of the most important applications of data reduction are image compression and feature extraction. In general, an image compression algorithm, used for storing and transmitting images, contains two steps: encoding and decoding. For both these steps, ANNs have been used. Feature extraction is used for subsequent segmentation or object recognition. The kind of features one wants to extract often correspond to particular geometric or perceptual characteristics in an image (edges, comers and junctions), or application dependent ones, e.g., facial features.

3.5.1 Feature Extraction Applications

Feature extraction can be seen as a special kind of data reduction of which the goal is to find a subset of informative variables based on image data. Since image data are by nature very high dimensional, feature extraction is often a necessary step for segmentation or object recognition to be successful. Besides lowering the computational cost, feature extraction is also a 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 separability well. There is a wide class of ANNs that can be trained to perform mappings to a lower-dimensional space. A well-known feature-extraction ANN is Oja's neural Implementation of a one-dimensional principal component analysis (PCA), later extended to multiple dimensions. In, Baldi and Homik proved that training three-layer auto-associator networks corresponds to applying PEA to the input data. Later, auto-associator networks with five layers were shown to be able to perform non-linear dimensionality reduction (i.e., finding principal surfaces). It is also possible to use a mixture of linear subspaces to approximate a non-linear subspace. Another approach to feature extraction is first to cluster the high-dimensional data, e.g., by a SOM, and then use the cluster centers as prototypes for the entire cluster. Among the ANNs that have been trained to perform feature extraction, feed-forward ANNs have been used in most of the reviewed applications. SOMs and Hopfield ANNs have also been trained to perform feature extraction.

Most of the ANNs trained for feature extractions obtain pixel data as input. Neural network feature extraction was 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 (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 it is encoded.

In most applications, the extracted features were used for segmentation, image matching or object recognition. For (anisotropic) objects occurring at the same scale, rotation causes the largest amount of intra-class variation. Some feature extraction approaches were designed to cope explicitly with (changes in) orientation of objects. It is important to make a distinction between application of supervised and unsupervised ANNs for feature extraction. For a supervised auto-associator ANN, the information loss implied by the data reduction can be measured directly on the predicted output variables, which is not the cage for unsupervised feature extraction by the SOM. Both supervised and unsupervised ANN feature extraction methods have advantages compared to traditional techniques such as PCA Feed-forward ANNs with several hidden layers can be trained to perform non-linear feature extraction, but lack a formal, statistical basis.

3.6 Image Segmentation

Segmentation is the partitioning of an image into parts that are coherent according to some criterion. When considered as a classification task, the purpose of segmentation is to assign labels to individual pixels or voxels. Some neural-based approaches perform segmentation directly on the pixel data, obtained either from a convolution window (occasionally from more bands as present in, e.g., remote sensing and MR images), or the information is provided to a neural classifier in the form of local features.

3.6.1 Image Segmentation Based on Pixel Data

Many ANN approaches have been presented that segment images directly from pixel or voxel data. Several deferent types of ANNs have been trained to perform pixel based segmentation: feed-forward ANNs. SOMs Hopfield networks, probabilistic ANNs, radial basis function networks, CNNs, constraint satisfaction ANNs and RAM-networks. A self-organizing architecture with fuzziness measures was used in. Also, biologically inspired neural-network approaches have been proposed: the perception model developed by Grossberg, which is able to segment images from surfaces and their shading, and the brain-like networks proposed by Opara and Worgotter. Hierarchical segmentation approaches have been designed to combine ANNs on deferent abstraction levels. The guiding principles behind hierarchical approaches are specialization and bottom-up processing: one or more ANNs are dedicated to low level feature extraction/segmentation, and their results are combined at a higher abstraction level where another (neural) classifier performs the final image segmentation. Reddick et al. developed a pixel-based two-stage approach where a SOM is trained to segment multi-spectral MR images. The segments are subsequently classified into white matter, grey matter, etc., by a feed-forward ANN. Non-hierarchical, modular approaches have also been develoed.

In general, pixel-based (often supervised) ANNs have been trained to classify the image content based on

- Texture;
- A combination of texture and local shape.

ANNs have also been developed for pre- and post processing steps in relation to segmentation, e.g., for

- Delineation of contours;
- Connecting edge pixels;
- Identification of surfaces;
- Deciding whether a pixel occurs inside or outside a segment;
- Defuzzifying the segmented image; and for
- Clustering of pixels;
- Motion segmentation.

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

3.7 Real-Life Applications of Neural Networks

This review has concentrated on applications of ANNs to image processing problems, which were reported in scientific literature. However, as the field matured, ANNs have 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. In some research programmes, an overview of commercial applications of ANNs has been given, and one of its applications is, character recognition.

3.7.1 Character Recognition

Two essential components in a character recognition algorithm are 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 that the character possesses. The derived features are then used as input to the character classifier.

Template matching, or matrix matching, is one of the most common classification methods. In template matching, individual image pixels are used as features. Classification is performed by comparing an input character image with a set of templates (or prototypes) 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 is identical to the same pixel in the template image. If the pixels differ the measure of similarity may be decreased. After all templates have been compared with the observed character image, the character's identity is assigned as the identity of the most similar template.

Template matching is a trainable process because template characters may be changed. In many commercial systems, PROMs (programmable read-only memory) store templates containing single fonts. To retrain the algorithm the current PROMs are replaced with PROMs that contain images of a new font. Thus, if a suitable PROM exists for a font then template matching can be trained to recognize that font. The similarity measure of template matching may also be modified, but commercial OCR systems typically do not allow this.

Structural classification methods utilize structural features and decision rules to classify characters. Structural features may be defined in terms of character strokes, character holes, or other character attributes such as concavities. For instance, the letter P may be described as a vertical stroke with a hole attached on the upper right side. For a character image input, the structural features are extracted and a rule-based system is applied to classify the character. Structural methods are also trainable but construction of a good feature set and a good rule-base can be time-consuming.

Character localization and segmentation. After the document has been located, the relative image portion is quantized to binary values according to an adaptive threshold established directly through a two-class clustering of tones. The characters are segmented by finding white areas between columns with higher density of black pixels as illustrated in figures 3.2, and 3.3.

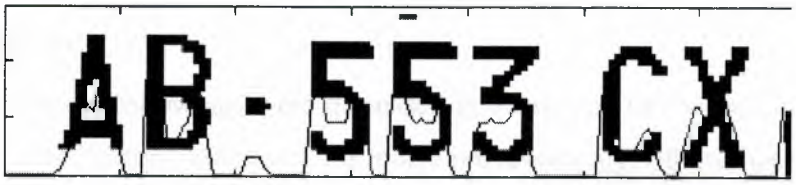


Figure 3.2 Character Localization

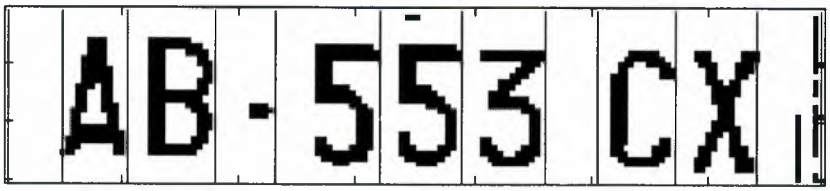


Figure 3.2 Character Segmentation

Isolated black pixels are wiped out and the character is resized to the standard measure of (10 by 6) pixels after a factor-of-two decimation as shown in figure 3.3.

Figure 3.2 'B' extracted and digitized.

The next chapter describes an algorithm that attempts to work with a subset of the features in a character that a human would typically see for the identification of machine-printed English characters.

3.8 Summary

In this chapter the followings were discussed in details; an overview f image processing with neural networks, history, taxonomy for image processing algorithms, neural networks in image processing, preprocessing, image reconstruction, image restoration, image enhancement, applicability of neural networks in preprocessing, and real-world applications of neural networks.

4. CHARACTER RECOGNITION SYSTEM USING NEURAL NETWORK

4.1 Overview

This chapter reviewed the work developed by the author. The architecture of the neural network used for character recognition. Data encoding & decoding presentation of the alphabets to the Neural Network.

N.N. Classification Method.

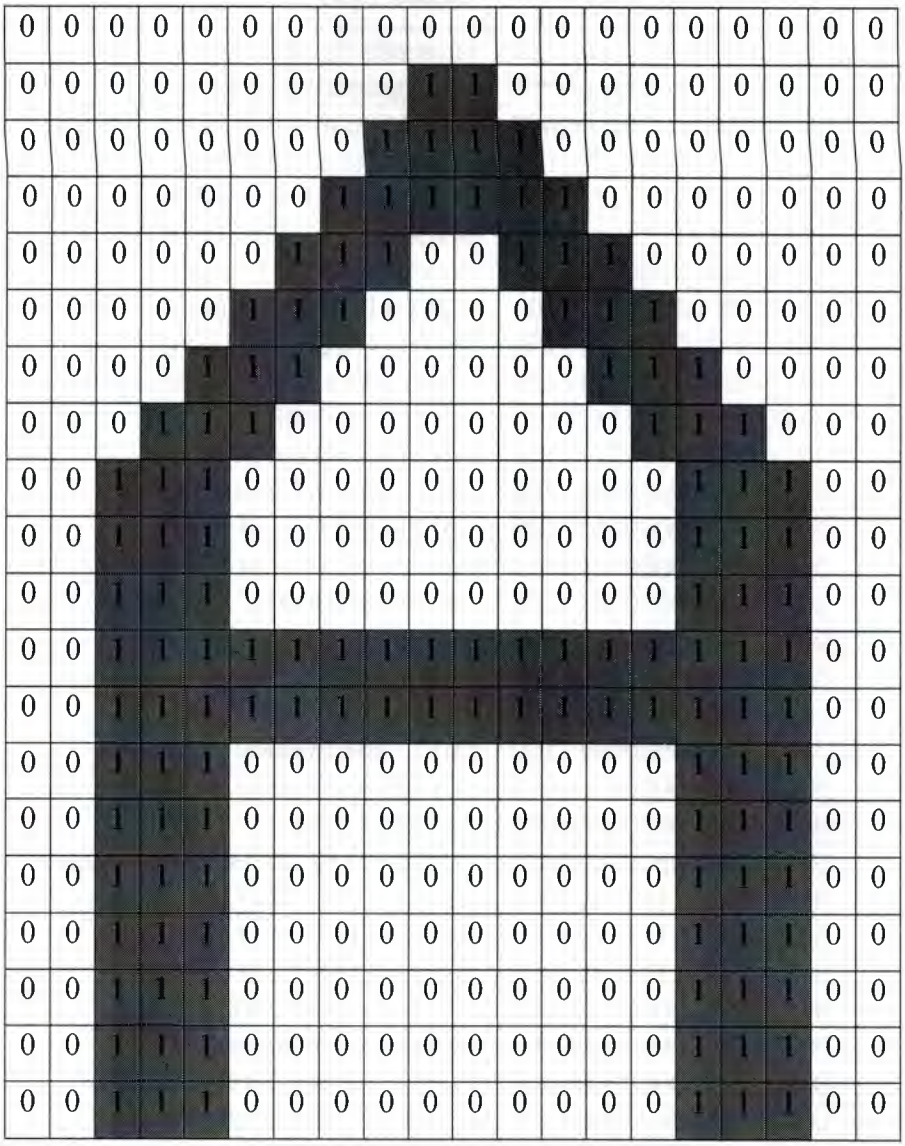
Let us consider the design at a computer program that must translate 20x20 matrixes in binary file which represent all the alphabets

{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}

The most common training scenarios utilize supervised learning during which the network is presented with an input pattern together with the target output for the correct answer, or correct classification for the input pattern.

4.2 Input Data Presentation

The input it's the file of DAT A to Z matrix 20x20 which is an array of black white pixels is used to represent individual characters. Each of these characters can be represented as 0/1 vector. In total 400 values representing a character will be ready to be presented to the input layers of the N.N.



The input at an array of character A it can be as an array input ={0000000000000000000000000000011000000000000000001111000000000000000
1111110000000000000111111110000000000011100001110000000001110000001110
0000001110000000011100000111000000000011100001110000000000111000011100
0000000011100001111111111111111000011111111111111110000111000000000011
1000011100000000001110000111000000000011100001110000000000111000011100
0000000011100001110000000000111000011100000000011100}

Table 4.1 The Patterns of these Characters.

	The matrix of character for the first trainings	The matrix of character for the second trainings
This is for code 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 00000000000000000000
This is for code Z	00000000000000000000 00111111111111111110 00111111111111111110 00111111111111111110 00000000000000011110 00000000000000111000 00000000000001110000 00000000000011100000 00000000000111000000 00000000001110000000 00000000011100000000 00000000111000000000 00000001110000000000 00000011100000000000 00000111000000000000 00001110000000000000 00111111111111111110 00111111111111111110 00111111111111111110 00000000000000000000	00000000000000000000 00011111111111111110 00111111111111111110 00011111111111111110 00000000000000111110 00000000000001111000 00000000000011110000 00000000000111100000 00000000001111000000 00000000011110000000 00000000111100000000 00000001111000000000 00000011110000000000 00000111100000000000 00001111000000000000 00111110000000000000 00111111111111111100 00111111111111111110 00111111111111111100 00000000000000000000

4.3 Output Data Presentation

Output of the neurons will be matched with the different types of patterns of the network and it will identify whether it is A or B or C and so on up to Z. The output presentation will be binary, where 26 values of input/output will identify each character as shown in table 4.2.

Table 4.2 Binary Identification for Each Character

	the matrix of character for the target matching trainings
The target of the output to be compared	123456.....26 A10000000000000000000000000 B01000000000000000000000000 C00100000000000000000000000 D00010000000000000000000000 E00001000000000000000000000 F00000100000000000000000000 G00000010000000000000000000 H00000001000000000000000000 J00000000100000000000000000 K00000000010000000000000000 L00000000001000000000000000 M00000000000100000000000000 N00000000000010000000000000 O00000000000001000000000000 P00000000000000100000000000 Q00000000000000010000000000 R00000000000000001000000000 S00000000000000000100000000 T00000000000000000010000000 U00000000000000000001000000 V00000000000000000000100000 W00000000000000000000010000 X00000000000000000000001000 Y00000000000000000000000100 Z00000000000000000000000010

Table 4.3 The Output of the Neurons how it will be Match the Patterns

	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

4.4 Neural Network Design

Neural Network Design is shown in figure 4.1 below which shows the number of input, hidden & output layers.

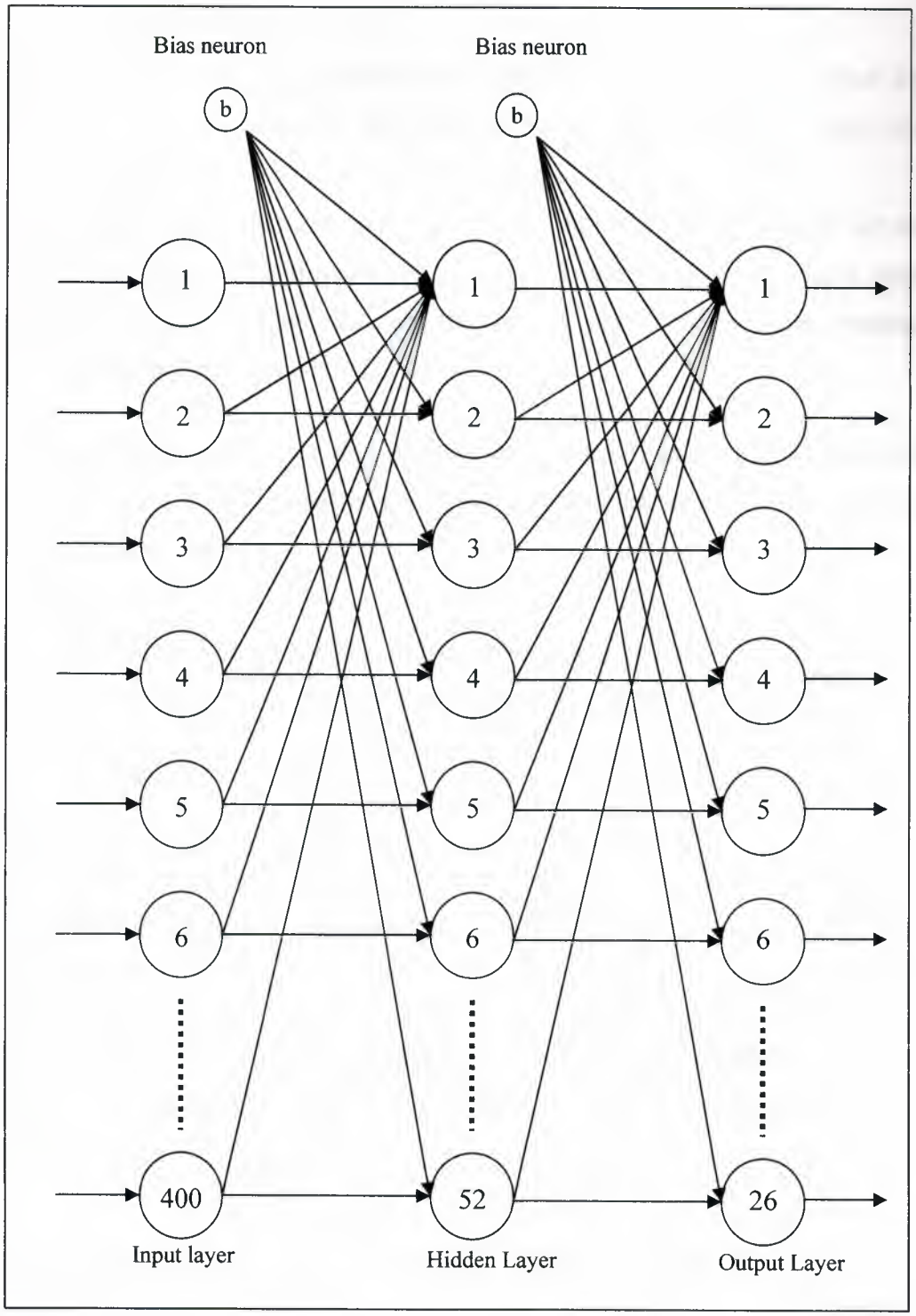


Figure 4.1 Back Propagation of Network Structure

4.5 Setting the Weights

There are two sets of weights; input-hidden layer weights and hidden-output layer weights. These weights represent the memory of the neural network, where final training weights can be used when running the network.

Initial weights are generated randomly there, after; weights are updated using the error (difference) between the actual output of the network and the desired (target) output. Weight updating occurs each iteration, and the network learns while iterating repeatedly until a net minimum error value is achieved.

First we must define notion for the patterns to be stored Pattern p. a vector of 0/1 usually binary–valued. Additional layers of weights may be added but the additional layers are unable to adopt.

Inputs arrive from the left and each incoming interconnection has an associated weight, w_{ji}. The perception processing unit performs a weighted sum at its input value.

The sum takes the form net $= \sum_{i=1}^{n} O_i W_i$

Weights associated with each inter connection are adjusted during learning .The weight to unit J from unit j from unit I is denoted as w_i after learning is completed the weights are fixed from 0 to 1.

There is a matrix of weight values that corresponds to each layer at inter connections as shown bellow These matrices are indexed with superscripts to distinguish weights in different layers.

Weights

0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7.4,0.7,0.8,0.2,0.4,0.0,3,0.2
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.7,0.5,0.1,0.2,0.5,0.8
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7.4,0.7,0.8,0.2,0.4,0.0,3,0.2
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9
0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.2,0.3
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2
0.7,0.8,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2
0.5,0.8,0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9

4.6 Bias Unit

Some back-propagation networks employ a bias unit as part of every layer but the output layer, this unit has a constant activation value at 1. Each bias unit is connected to all units in the next higher layer, and its weights to them are adjusted during the back-error propagation. The bias units provide a constant term in the weighted sum of the units in the next layer. The bios unit also provides a others hold effect on each unit it targets.

4.7 Training the N.N.

4.7.1 Forward- pass

The forward – pass phase is initiated when an input pattern is presented to the network, each input unit corresponds to an entry in the input pattern vector, and each unit takes on the value of this entry.

Incoming connection to unit J are at the left and originate at units in the layer below. The function F(x), a sigmoid curve is illustrated as bellows.

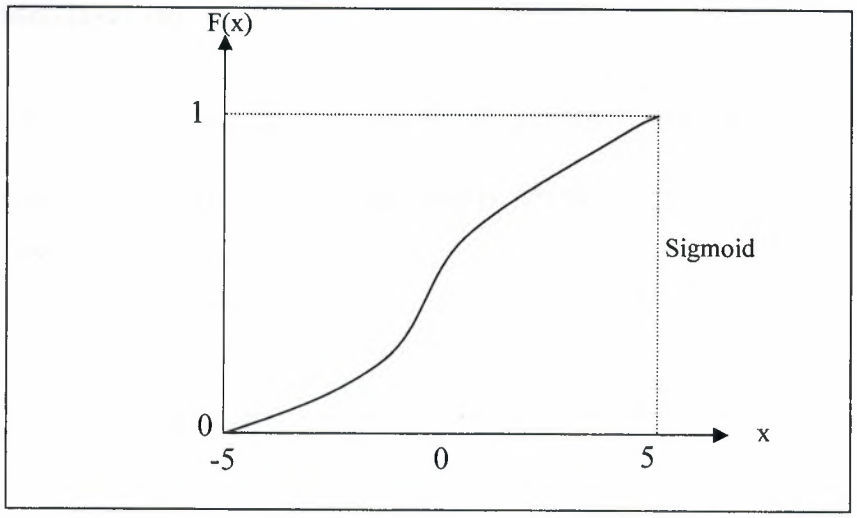


Figure 4.2 Sigmoid Curve

There is a transition from 0 to 1 that takes place when x is approximately $(-3 < \chi < 3)$ the sigmoid function performs assort at soft threshold that is rounded as shown in figure 4.3 bellow.

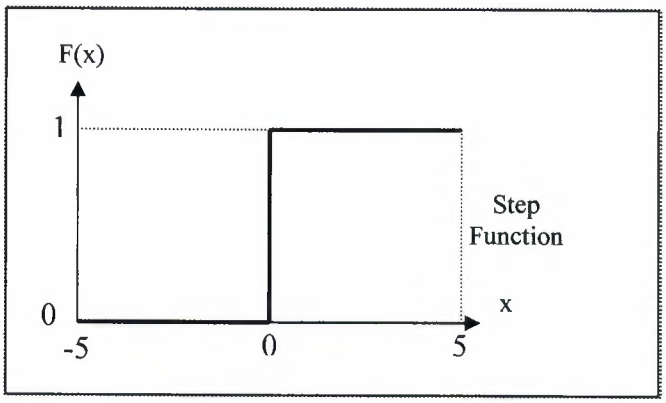


Figure 4.3 Step Function

The equation for the sigmoid function is

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

a. Input layer (i)

For input we have 26 inputs will be saved by the DAT file.

Input Layer at neuron = output layer of neuron $I_i = O_i$

b. Hidden layer (h)

Hidden–Layer input $h = I_k = \sum_i W_{ki} O_i$ as we have suggest that our weight is this and we are taking the value at our input at character is A*I*S

Where A is the input-matrix,

I is the hidden-layer input matrix

And

S is the Sigmoid function matrix.

$$Weights = W_{hi}$$

0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7.4,0.7,0.8,0.2,0.4,0.0,3,0.2
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.7,0.5,0.1,0.2,0.5,0.8
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7.4,0.7,0.8,0.2,0.4,0.0,3,0.2
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9
0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.2,0.3,0.4,0.5,0.6,0.7,0.9,0.2,0.4,0.5,0.9,0.7,0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.8
0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2,0.5,0.8
0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.2,0.3
0.2,0.3,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2
0.7,0.8,0.2,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.4,0.8,0.0,0.7,0.5,0.1,0.2
0.5,0.8,0.4,0.5,0.6,0.7,0.8,0.2,0.4,0.5,0.9,0.9,0.0,0.4,0.7,0.8,0.2,0.4,0.5,0.9
0.4,0.7,0.8,0.2,0.4,0.5,0.9,0.7,0.7,0.8,0.2,0.4,0.0,0.3,0.2,0.5,0.6,0.4,0.7,0.9

$$A \text{ is } Input = O_i \Rightarrow \begin{Bmatrix} 00000000000000000000 \\ 00000000011000000000 \\ 00000000111100000000 \\ 00000001111110000000 \\ 00000011111111000000 \\ 00000111000011100000 \\ 00001110000001110000 \\ 00011100000000111000 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111111111111111100 \\ 00111111111111111100 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111000000000011100 \\ 00111000000000011100 \end{Bmatrix}$$

I_h= $\sum$(0.2x0)+(0.3x0)+(0.4x0)+(0.5x0)+(0.6x0)+(0.7x0)+(0.9x0)+(0.2x0)+(0.4x0)+(0.5x0)+(0.9x0)+(0.7x0)+(0.4x0)+(0.7x0)+(0.8x0)+(0.2x0)+(0.4x0)+(0.0x0)+(0.3x0)+(0.2x0)+(2x0)+(0.3x0)+(0.2x0)+(0.9x0)+(0.0x0)+(0.4x0)+(0.7x0)+(0.8x0)+(0.2x0)+(0.4x1)+(0.5x1)+(0.9x0)+(0.7x0)+(0.4x0)+(0.7x0)+(0.8x0)+(0.2x0)+(0.4x0)+(0.0x0)+(0.2x0)

.....

.....

(0.4x0)+(0.7x0)+(0.8x1)+(0.2x1)+(0.4x1)+(0.5x0)+(0.9x0)+(0.7x0)+(0.7x0)+(0.8x0)+(0.2x0)+(0.4x0)+(0.0x0)+(0.3x0)+(0.2x0)+(0.5x1)+(0.6x1)+(0.4x1)+(0.7x0)+(0.9x0)}

So after calculation we will get the value of I_h.

Each output at a hidden neuron is calculated using the sigmoid function.

Hidden-layer output $h = O_h = \frac{1}{1+e^{-I_h}}$ this calculation is for one neuron and the summation for the other output layer (j).

Now we are going to the second neuron which is the neuron in the output layer that is equal to the sum at all the output at the hidden layer neurons multiplied by their associated connection weights plus the bias weights at each neuron.

4.8 Summary

This chapter represents the most common training scenarios utilize supervised learning during which the network is presented with an input pattern together with the target output for the correct answer, or correct classification for the input pattern as will be seen in the later chapters.

5. PRACTICAL CONSIDERATION USING MATLAB

5.1 Overview

It is often useful to have a machine perform pattern recognition. In particular, machines that can read symbols are very cost effective. A machine that reads banking checks can process many more checks than a human being in the same time. This kind of application saves time and money, and eliminates the requirement that a human perform such a repetitive task.

5.2 Problem Statement

A network is to be designed and trained to recognize the 26 letters of the alphabet. An imaging system that digitizes each letter centered in the system's field of vision is available. The result is that each letter is represented as a 20 by 20 grid of Boolean values.

Perfect classification of ideal input vectors is required, and reasonably accurate classification of noisy vectors. The target vectors are also defined in this file with a variable called targets. Each target vector is a 26-element vector with a 1 in the position of the letter it represents, and 0's everywhere else. For example, the letter A is to be represented by a 1 in the first element (as A is the first letter of the alphabet), and 0's in elements two through twenty-six as shown in table 4.3.

5.3 Neural Network

The network receives the 400 Boolean values as a 400-element input vector. It is then required to identify the letter by responding with a 26-element output vector. The 26 elements of the output vector each represent a letter. To operate correctly, the network should respond with a 1 in the position of the letter being presented to the network. All other values in the output vector should be 0. In addition, the network should be able to handle noise. In practice, the network does not receive a perfect Boolean vector as input. Specifically, the network should make as few mistakes as possible when classifying vectors with noise of mean 0 and standard deviation of 0.2 or less.

5.4 Architecture

The neural network needs 400 inputs and 26 neurons in its output layer to identify the letters. The network is a two-layer log-sigmoid/log-sigmoid network. The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output Boolean values.

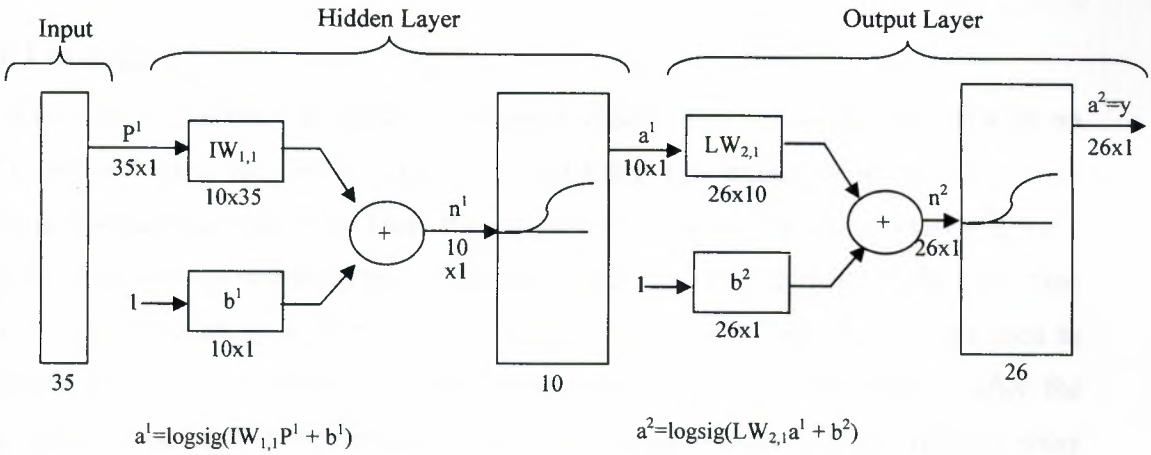


Figure 5.1 Neural Network Architecture

The hidden (first) layer has 10 neurons. This number was picked by guesswork and experience. If the network has trouble learning, then neurons can be added to this layer. The network is trained to output a 1 in the correct position of the output vector and to fill the rest of the output vector with 0's. However, noisy input vectors may result in the network not creating perfect 1's and 0's. After the network is trained the output is passed 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 have a value of 0. The result of this post-processing is the output that is actually used.

5.5 Initialization

The two-layer network is created with newff.

```
S1 = 10;
[R,Q] = size(p1);
[S2,Q] = size(targets);
P = p1;
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

5.6 Training

To create a network that can handle noisy input vectors it is best to train the network on both ideal and noisy vectors. To do this, the network is first trained on ideal vectors until it has a low sum-squared error. Then, the network is trained on 10 sets of ideal and noisy vectors. The network is trained on two copies of the noise-free alphabet at the same time as it is trained on noisy vectors. The two copies of the noise-free alphabet are used to maintain the network's ability to classify ideal input vectors. Unfortunately, after the training described above the network may have learned to classify some difficult noisy vectors at the expense of properly classifying a noise-free vector. Therefore, the network is again trained on just ideal vectors. This ensures that the network responds perfectly when presented with an ideal letter. All training is done using backpropagation with both adaptive learning rate and momentum with the function trainbpx.

5.6.1 Training without Noise

The network is initially trained without noise for a maximum of 5000 epochs or until the network sum-squared error falls beneath 0.1. P = p1;

```
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);
```

5.6.2 Training with Noise

To obtain a network not sensitive to noise, we trained with two ideal copies and two noisy copies of the vectors in alphabet. The target vectors consist of four copies of the vectors in target. The noisy vectors have 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 can still respond well to ideal vectors. To train with noise, the maximum number of epochs is reduced to 300 and the error goal is increased to 0.6, reflecting that higher error is expected because more vectors (including some with noise), are being presented.

```
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 is measured by testing the network with hundreds of input vectors with varying quantities of noise. The script file appcr1 tests the network at various noise levels, and then graphs the percentage of network errors versus noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5 is added to input vectors. At each noise level, 100 presentations of different noisy versions of each letter are made and the network's output is calculated. The output is then passed through the competitive transfer function so that only one of the 26 outputs (representing the letters of the alphabet), has a value of 1.

5.8 MATLAB Program

```
[p1,targets] = prprob;
[R,Q] = size(p1);
[S2,Q] = size(targets);
S1 = 10;
net = newff(minmax(p1),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
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 = 0.95;     % Momentum constant.
%   Training begins...please wait...
P = p1;
T= targets;
 [net,tr] = train(net,P,T);

%   TRAINING THE NETWORK WITH NOISE
%   ================================
%   A copy of the network will now be made.  This copy will
%   be trained with noisy examples of letters of the alphabet.
netn = net;
netn.trainParam.goal = 0.6;   % Mean-squared error goal.
netn.trainParam.epochs = 300; % Maximum number of epochs to train.

%   The network will be trained on 10 sets of noisy data.
%   Training begins...please wait...
T = [targets targets targets targets];
for pass = 1:10
  fprintf('Pass = %.0f\n',pass);
```

```
Pass = 1
 P = [alphabet, p1, ...
    (p1+ randn(R,Q)*0.1), ...
    (p1 + randn(R,Q)*0.2)];
 [netn,tr] = train(netn,P,T);

%   TRAINING THE SECOND NETWORK WITHOUT NOISE
%   =========================================
%   The second network is now retrained without noise to
%   insure that it correctly categorizes non-noizy letters.
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).

%   Training begins...please wait...
P = p1;
T = targets;
[netn,tr] = train(netn,P,T);

%   TRAINING THE NETWORK
%   ====================
% SET TESTING PARAMETERS
noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
T = targets;
```

```
% PERFORM THE TEST
for noiselevel = noise_range
 fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
% Testing networks with noise level of 0.00.
 errors1 = 0;
 errors2 = 0;
 for i=1:max_test
  P = p1 + randn(35,26)*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;

%   DISPLAY RESULTS
%   ===============
%   Here is a plot showing the percentage of errors for
%   the two networks for varying levels of noise.
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
ylabel('Network 1 - -  Network 2 ---');
```

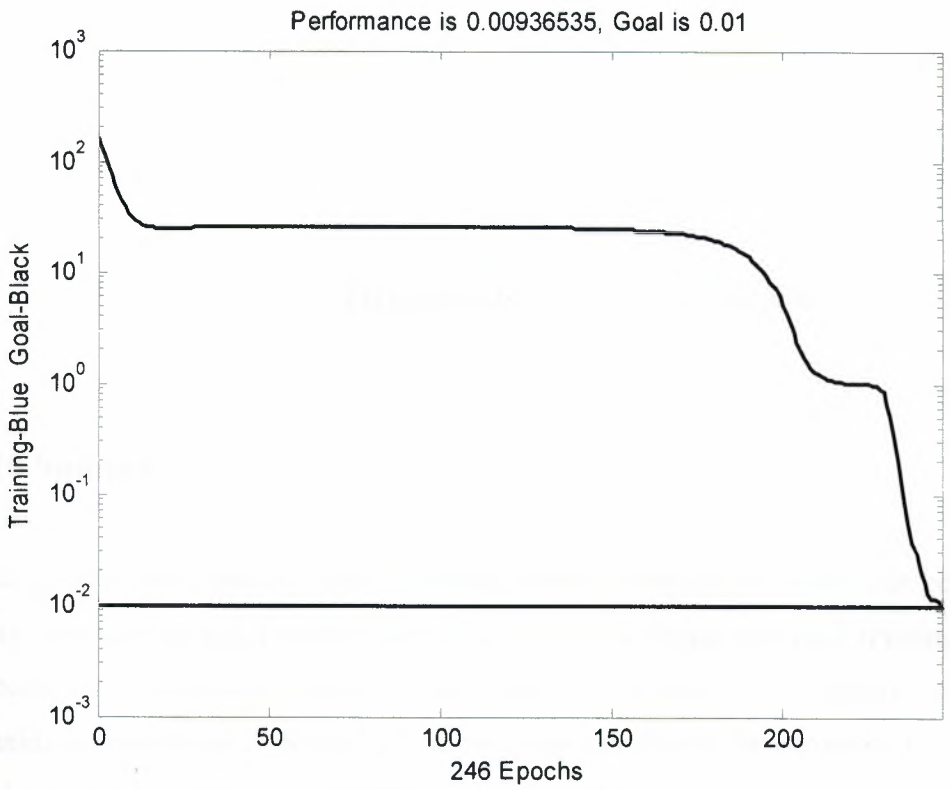
5.9 Practical Example

The following input matrix represents the letter R inserted to the Neural Network shown in figure 5.1.

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1;
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0;
 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0];
```

The above input was trained, and tested with the following specifications

Learning Rate	Maximum Epochs	Reached Epochs	Alpha	Goal	Result
0.75	5000	584	0.04	0.01	Performance Goal met
0.75	5000	246	0.03	0.01	Performance Goal met



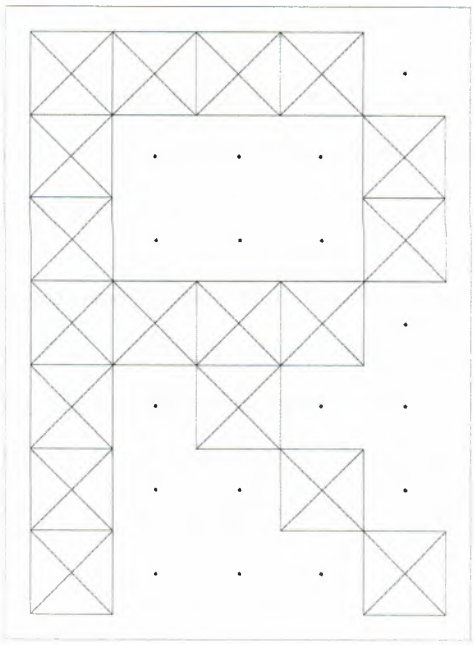


Figure 5.2 Result of Tested Character

5.10 Summary

This problem demonstrates how a simple pattern recognition system can be designed. Note that the training process did not consist of a single call to a training function. Instead, the network was trained several times on various input vectors. In this case, training a network on different sets of noisy vectors forced the network to learn how to deal with noise, a common problem in the real world.

CONCLUSION

In this thesis, we have presented the basic concept of back propagation in order to obtain progressive compression of character recognition. We also presented a specific technique. Some practical results are also included.

Chapter one presented as an overview of N.N's. it history simple structure, biological analogy and the Back propagation Producer also noted that the Back Propagation Network has the ability to learn any arbitrarily complex nonlinear mapping this is due to the introduction of the hidden layer. It also has a capacity much greater than the dimensionality of its input and output layers as we will see later. This is not true of all neural net models.

However Back propagation can involve extremely long and potentially infinite training time. If you have a strong relationships between input and outputs and you are willing to accept results within a relatively broad time, your training time may be reasonable.

Chapter two has been noticed that Multimedia has rapidly become a buzzword of the '90s, and, with it, digital video has gained enormous exposure. In digital video, two technologies stand out. Resizing reduces the amount of video data required to transmit or store, it allows users the option to choose what size they view video images at, and it may also enhance compression. Compression, on the other hand, brings the bandwidth requirements for digital video to more manageable proportions. Together, the two technologies make digital video a reality.

Chapter four has been noticed that this technology is demonstrated in the ICR/OCR/OMR/MICR demo application (which is installed with the product download) and may be found within the MATLAB product.

APPENDIX I

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code A	0000000000000000000 0000000011000000000 0000000111100000000 0000001111110000000 0000011111111000000 0000111000011100000 0001110000001110000 0011100000000011100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100	0000000000000000000 0011111111111111100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0000000000000000000
This is for code B	0000000000000000000 0011111111111000000 0011111111111100000 0011111111111110000 0011100000000111000 0011100000000011100 0011100000000111000 0011100000000110000 0011111111111100000 0011111111111000000 0011111111111100000 0011100000001110000 0011100000000111000 0011100000000011100 0011100000000111000 0011100000000110000 0011100000001110000 0011111111111100000 0011111111111000000 0000000000000000000	0000000000000000000 0011111111111000000 0011111111111110000 0011100000011111000 0011100000011111100 0011100000001111100 0011100000000111100 00111000000001f1100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000111100 0011100000001111100 0011100000011111000 0011111111111110000 0011111111111100000 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code C	000000000000000000 000000011111100000 000000111111110000 000001111111100000 000011100000011100 000111000000001100 001110000000000100 001110000000000000 001110000000000000 001110000000000000 001110000000000000 001110000000000000 001110000000000100 001110000000001100 001110000000011100 001110000000111000 001111111111111000 001111111111110000 000000000000000000	000000000000000000 001111111111111100 001111111111111100 001110000000011100 001110000000011100 001110000000011100 001110000000011100 000000000000000000 000000000000000000 000000000000000000 000000000000000000 000000000000000000 001110000000011100 001110000000011100 001110000000011100 001110000000011100 001111111111111100 001111111111111100 000000000000000000
This is for code D	000000000000000000 001111111111000000 001111111111100000 001111111111110000 001110000001110000 001110000000111000 001110000000011100 001110000000001110 001110000000001110 001110000000001110 001110000000001110 001110000000011100 001110000000011100 001110000000111000 001110000000111000 001111111111111100000 001111111111111000000 001111111111100000000 000000000000000000	000000000000000000 001111111111111100000 001111111111111110000 001111111111111111000 001110000000111111100 001110000000011111100 001110000000000111100 001110000000000111100 001110000000000111100 001110000000000111100 001110000000000111100 001110000000000111100 001110000000000111100 001110000000011111100 001111111111111111000 001111111111111110000 001111111111111100000 000000000000000000000 000000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code E	0000000000000000000 0011111111111111100 0011111111111111110 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111110000000 0011111111100000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111100000 0011111111111110000 0011111111111100000 0000000000000000000	0000000000000000000 0011111111111111100 0011111111111111100 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111111100 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111111100 0011111111111111100 0011111111111111100 0000000000000000000
This is for code F	0000000000000000000 0011111111111111000 0011111111111111100 0011111111111111000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111000000 0011111111111100000 0011111111111000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0001000000000000000 0000000000000000000	0000000000000000000 0011111111111111100 0011111111111111100 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111111100 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code G	0000000000000000000 0000000111111000000 0000001111111000000 0000011111111100000 0000111000000111000 0001110000000111000 0011100000000111000 0011100000000111000 0011100000000010000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000111111000 0011100000111111000 0011110000000111000 0001111100000111000 0001111111111110000 0000011111111000000 0000001111111000000 0000000000000000000	0000000000000000000 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000011111100 0011100000011111100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0000000000000000000
This is for code H	0000000000000000000 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0000000000000000000	0000000000000000000 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0000000000000011100 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code I	0000000000000000000 0000011111111000000 0000011111111000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000011111111000000 0000011111111000000 0000000000000000000 0000000000000000000	0000000000000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000011111111000000 0000011111111000000 0000000000000000000 0000000000000000000
This is for code J	0000000000000000000 0000011111111000000 0000011111111000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0001000011100000000 0011100011100000000 0001111111100000000 0000111111100000000 0000000000000000000 0000000000000000000	0000000000000000000 0000011111111000000 0000011111111000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0000000111000000000 0011111111100000000 0011111111100000000 0000000000000000000 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code K	0000000000000000000 0011100000011110000 0011100000111100000 0011100001110000000 0011100011100000000 0011100111000000000 0011001110000000000 0011011100000000000 0011111100000000000 0011111110000000000 0011111110000000000 0011001111000000000 0011100011100000000 0011100001110000000 0011100000111000000 0011100000011100000 0011100000001110000 0011100000000111000 0000000000000000000	0000000000000000000 0011100000001111100 0011100000011111100 0011100000111000000 0011100001110000000 0011100011100000000 0011100111000000000 0011001110000000000 0011111110000000000 0011111110000000000 0011100011100000000 0011100001110000000 0011100000111000000 0011100000011100000 0011100000001110000 0011100000000111110 0011100000000011110 0000000000000000000
This is for code I	0000000000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000110 0011111111111111100 0011111111111111000 0011111111111110000 0000000000000000000	0000000000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011111111111111110 0011111111111111110 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code 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 00000000000000000000 00000000000000000000
This is for code 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

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code O	0000000000000000000 0000011111111110000 0000111111111111000 0001111111111111100 0011111000001111110 0011111000000111110 0011110000000011110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011111000000111110 0011111000001111110 0001111111111111100 0000111111111111000 0000011111111110000 0000000000000000000	0000000000000000000 0011111111111111110 0011111111111111110 0011111111111111110 0011111000001111110 0011111000000111110 0011110000000011110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011100000000001110 0011111000000111110 0011111000001111110 0011111111111111110 0011111111111111110 0011111111111111110 0000000000000000000
This is for code P	0000000000000000000 0011111111111100000 0011111111111110000 0011111111111111000 0011100000011111100 0011100000001111100 0011100000000111100 0011100000001111100 0011100000011111100 0011111111111110000 0011111111111100000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0000000000000000000	0000000000000000000 0011111111111111100 0011111111111111100 0011111111111111100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011100000000011100 0011111111111111100 0011111111111111100 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0011100000000000000 0000000000000000000

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code Q	00000000000000000000 00000111111111110000 00001111111111111000 00011111111111111100 00111111000001111110 00111110000000111110 00111100000000011110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111000000000001110 00111100001100001110 00111110000111111110 00111111000111111110 00011111111111111100 00001111111111111110 00000111111100001111 00000000000000000000	00000000000000000000 00111111111111111100 00111111111111111100 00111111111111111100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111000000000011100 00111100001100011100 00111110000111011100 00111111000011111100 00111111111111111100 00111111111111111110 00000000000000001111 00000000000000000000
This is for code 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

	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code 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
This is for code 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 00000111111111000000 00000111111111000000 00000000000000000000

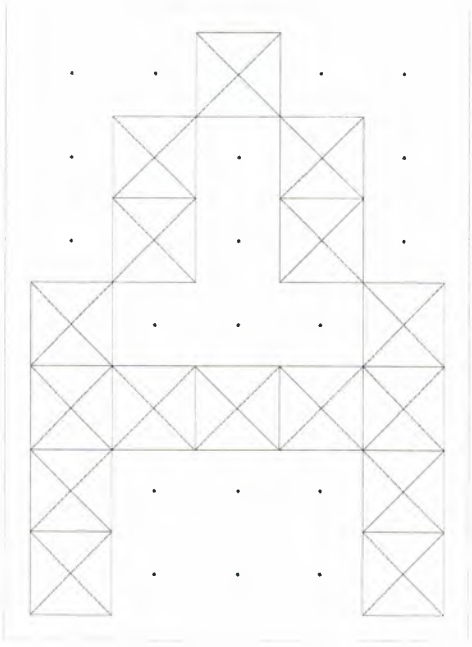
	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code U	0000000000000000000	0000000000000000000
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011100000000011100
	0011100000000011100	0011110000000111100
	0011100000000011100	0011111000000111100
	0011100000000011100	0001111100001111000
	0011111111111111100	0000111111111110000
	0011111111111111100	0000011111111100000
	0000000000000000000	0000000000000000000
This is for code V	0000000000000000000	0000000000000000000
	0011100000000001110	0011100000000001110
	0011100000000001110	0011100000000001110
	0011100000000001110	0011100000000001110
	0001110000000011100	0011110000000011110
	0001110000000011100	0011110000000011110
	0000111000000111000	0001110000000011110
	0000111000000111000	0001111000000111110
	0000011100001110000	0001111000000111100
	0000011100001110000	0001111000000111100
	0000001110011100000	0001111000000111000
	0000001110011100000	0000111100000111000
	0000001110111000000	0000111100000111000
	0000001111111000000	0000111110011110000
	0000000111110000000	0000011110011100000
	0000000111110000000	0000001111011110000
	0000000011100000000	0000001111111000000
	0000000011100000000	0000000111111000000
	0000000001000000000	0000000011000000000
	0000000000000000000	0000000000000000000

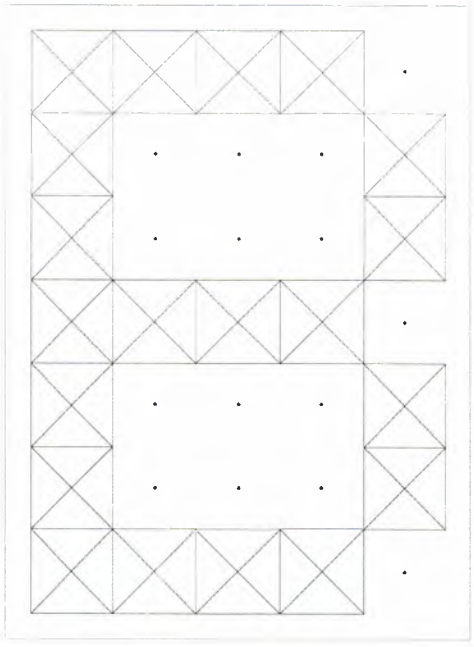
	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code w	0000000000000000000 0110000001100000110 0110000001100000110 0110000001100000110 0110000001100000110 0011000011110000110 0011000011110000110 0011000011110000110 0011000011110000110 0001100110011000 0001100110011000 0001100110011000 0001100110011000 0001111110011111000 0000111100001111000 0000011000000110000 0000011000000110000 0000000000000000000 0000000000000000000 0000000000000000000	0000000000000000000 0111000001100000111 0111000001100000111 0111000001100000111 0111000001100000111 0111000001100000111 0011100011110001100 0011100011110001100 0011100011110001100 0011100011110001100 0011100110011001100 0011100110011001100 0011100110011001100 0011100110011001100 0011111110011111100 0001111100001111000 0000111000000111000 0000111000000110000 0000010000000000000 0000000000000000000 0000000000000000000
This is for code X	0000000000000000000 0011100000000000110 0001110000000001100 0000111000000011000 0000011100000110000 0000001110001100000 0000000111111000000 0000000011111000000 0000000011111000000 0000000111111000000 0000001110001100000 0000011100000110000 0000111000000011000 0001110000000001100 0011100000000000110 0111000000000000110 0111000000000000110 0000000000000000000 0000000000000000000	0000000000000000000 0011100000000000110 0001110000000001100 0000111000000011000 0000011100000110000 0000001110001100000 0000000111011000000 0000000011111000000 0000000011100000000 0000000011111000000 0000000111011000000 0000001110001100000 0000011100000110000 0000111000000011000 0001110000000001100 0011100000000000110 0111000000000000111 0111000000000000111 0000000000000000000 0000000000000000000

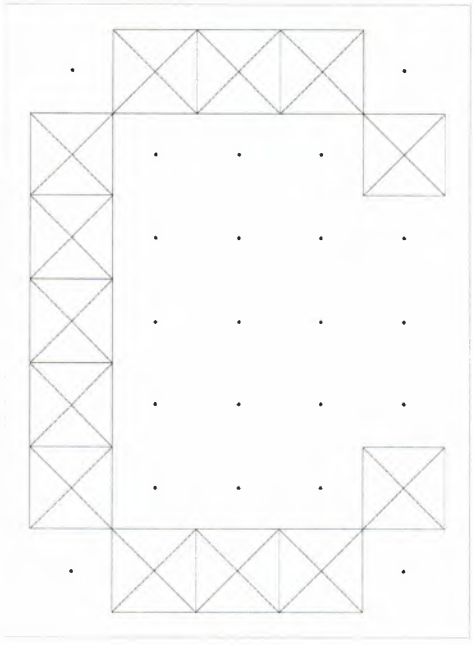
	The matrix of character for the first trainings	**The matrix of character for the second trainings**
This is for code Y	00111100000000011110 00011110000000011110 00001111000000011110 00000111100000011100 00000011110001111000 00000001111011111000 00000000111111110000 00000000011111100000 00000000011111000000 00000000011111000000 00000000011110000000 00000000011110000000 00000000011110000000 00000000011110000000 00000000011110000000 00000000011110000000 00000000000000000000 00000000000000000000	00000000000000000000 00111000000000001110 00011100000000011100 00001110000000111000 00000111000001110000 00000011100011100000 00000011111111000000 00000001111110000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000111100000000 00000000000000000000
This is for code Z	00000000000000000000 00111111111111111110 00111111111111111110 00111111111111111110 00000000000000011110 00000000000000111000 00000000000001110000 00000000000011100000 00000000000111000000 00000000001110000000 00000000011100000000 00000000111000000000 00000001110000000000 00000011100000000000 00000111000000000000 00001110000000000000 00111111111111111110 00111111111111111110 00111111111111111110 00000000000000000000	00000000000000000000 00011111111111111110 00111111111111111110 00011111111111111110 00000000000000111110 00000000000001111000 00000000000011110000 00000000000111100000 00000000001111000000 00000000011110000000 00000000111100000000 00000001111000000000 00000011110000000000 00000111100000000000 00001111000000000000 00111110000000000000 00111111111111111100 00111111111111111110 00111111111111111100 00000000000000000000

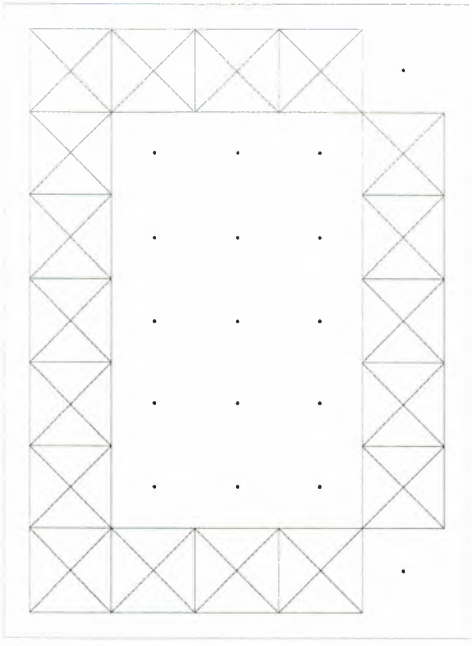
	the matrix of character for the target matching trainings
The target of the output to be compared	123456.....26 A10000000000000000000000000 B01000000000000000000000000 C00100000000000000000000000 D00010000000000000000000000 E00001000000000000000000000 F00000100000000000000000000 G00000010000000000000000000 H00000001000000000000000000 J00000000100000000000000000 K00000000010000000000000000 L00000000001000000000000000 M00000000000100000000000000 N00000000000010000000000000 O00000000000001000000000000 P00000000000000100000000000 Q00000000000000010000000000 R00000000000000001000000000 S00000000000000000100000000 T00000000000000000010000000 U00000000000000000001000000 V00000000000000000000100000 W00000000000000000000010000 X00000000000000000000001000 Y00000000000000000000000100 Z00000000000000000000000010

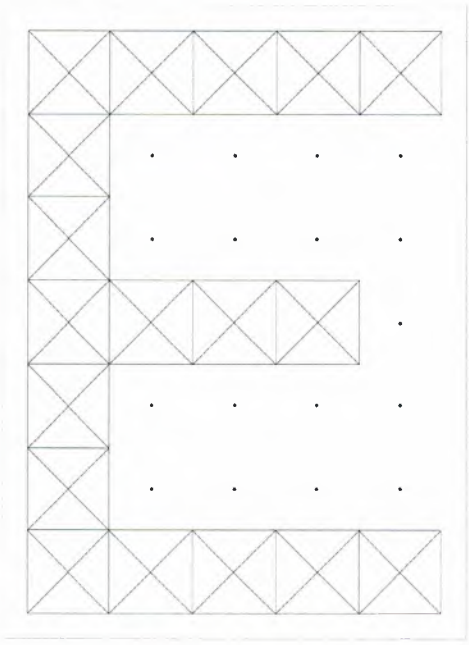
APPENDIX II

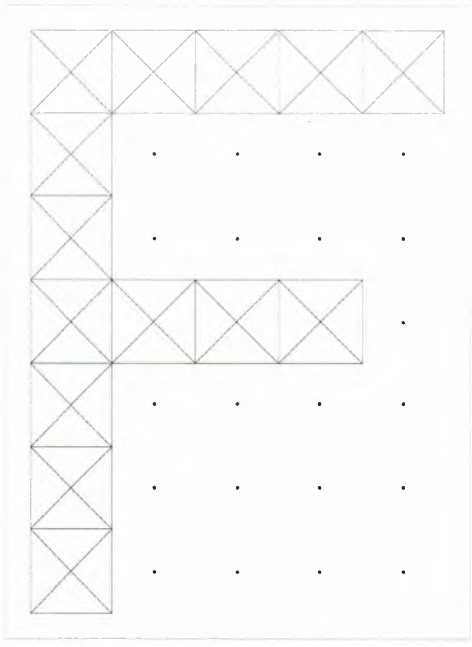


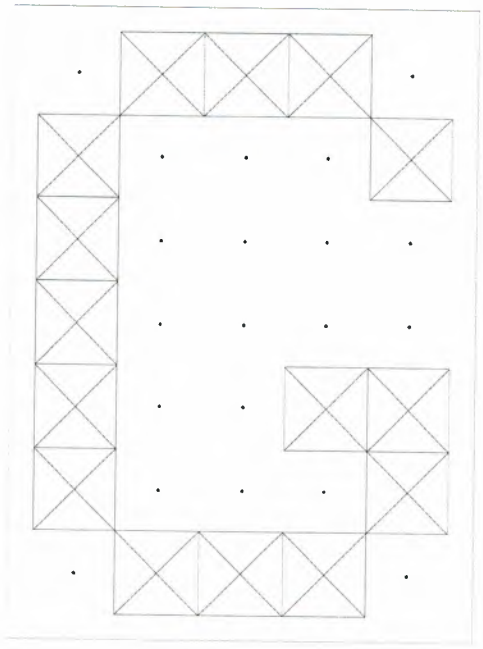


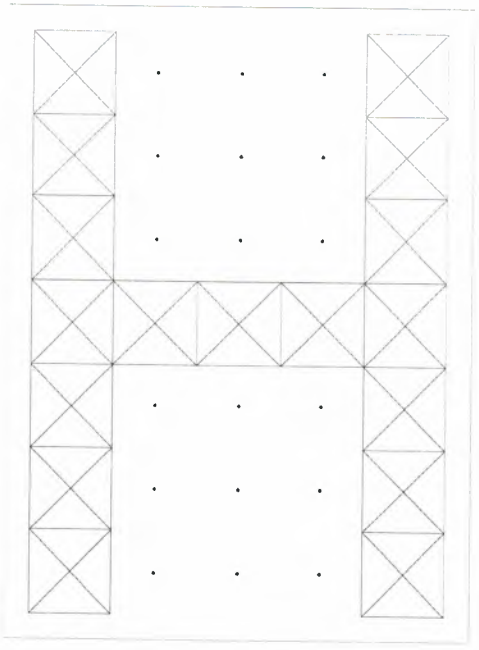


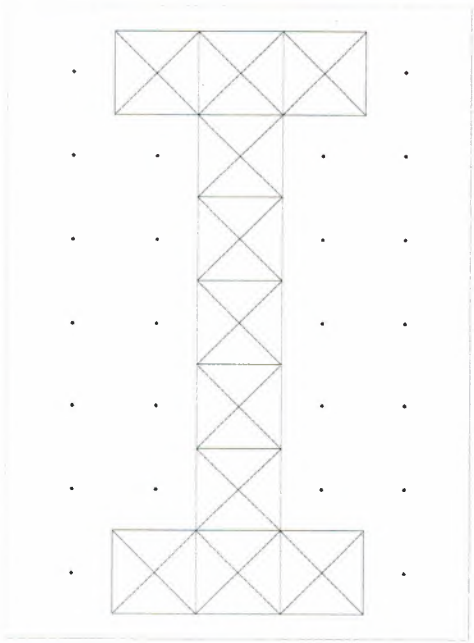


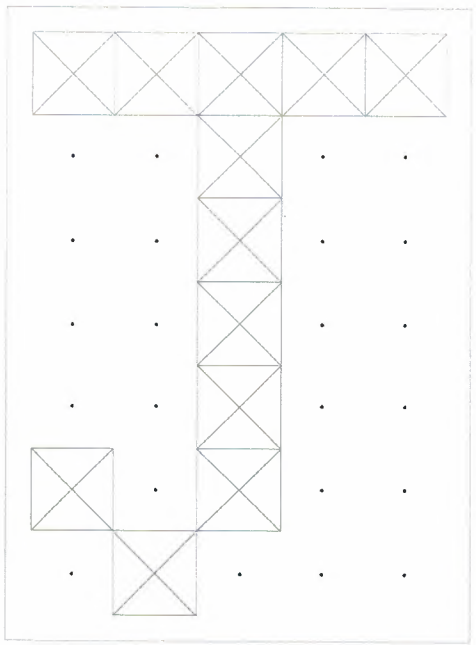


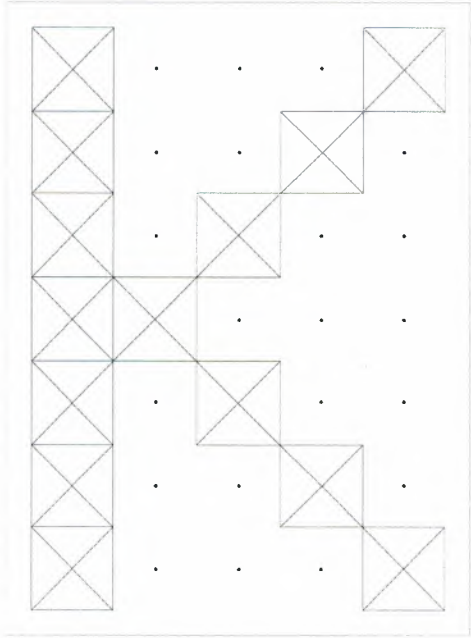


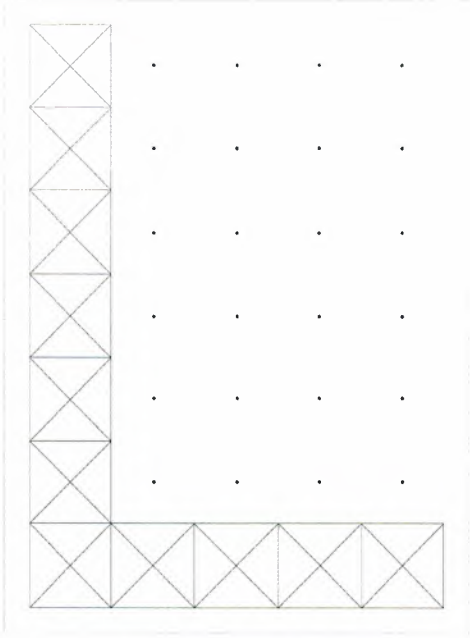


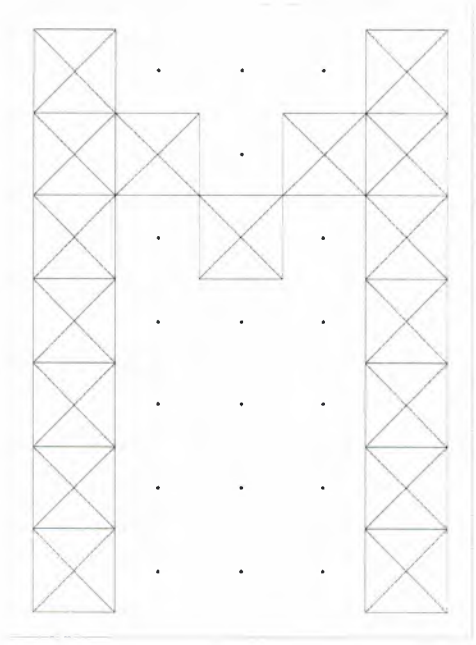


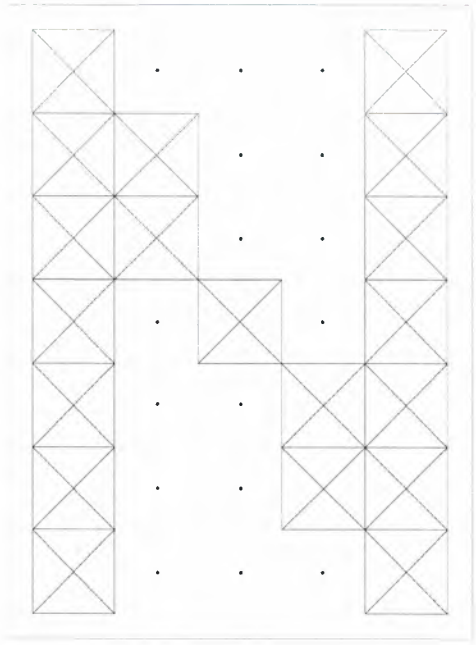


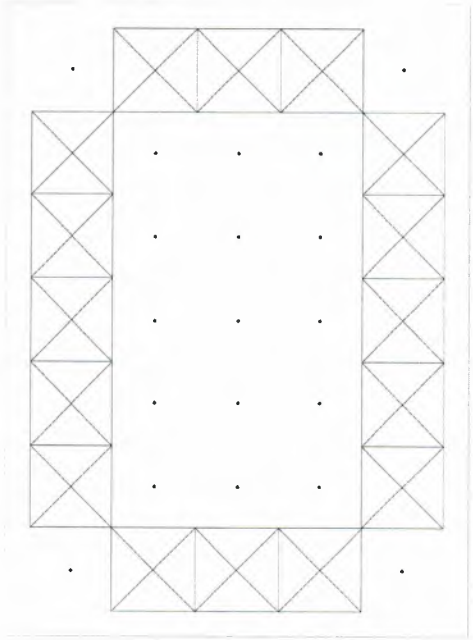


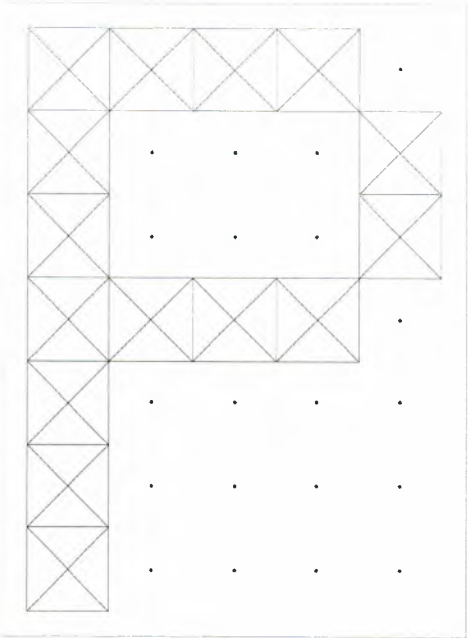


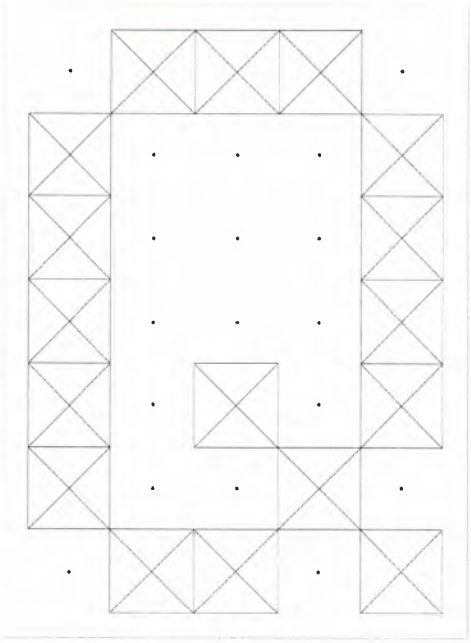


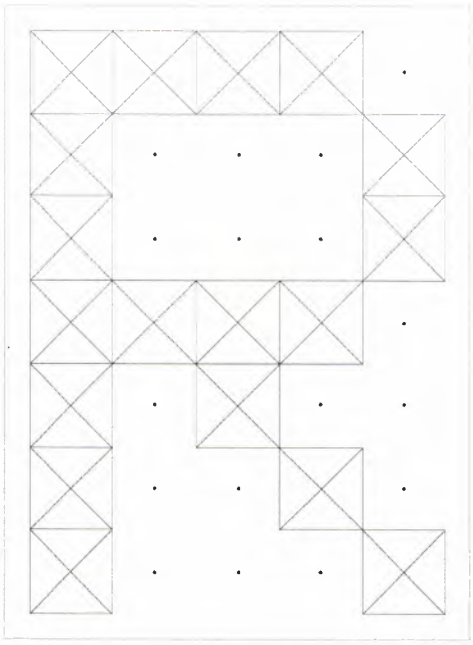


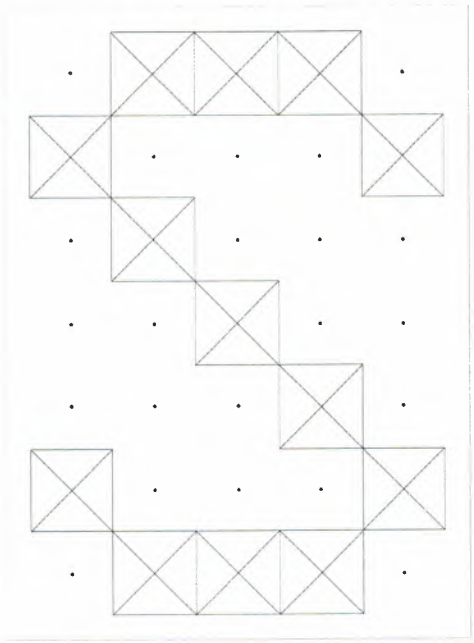


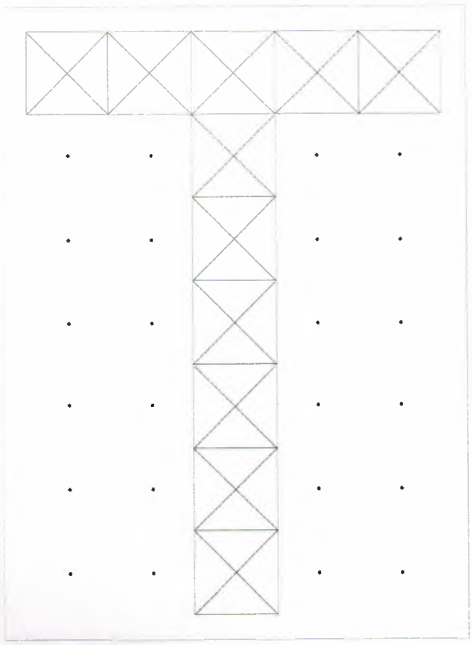


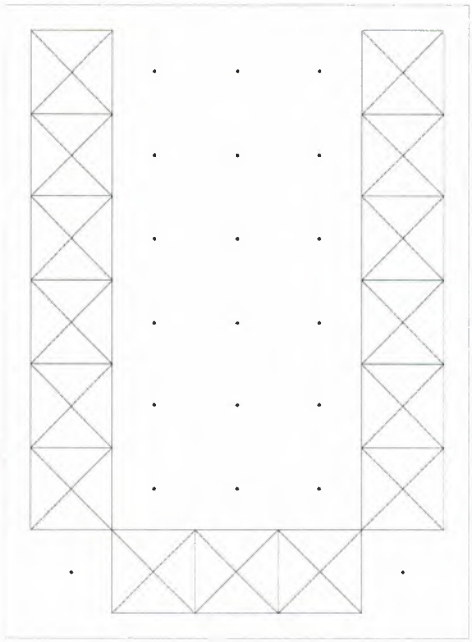


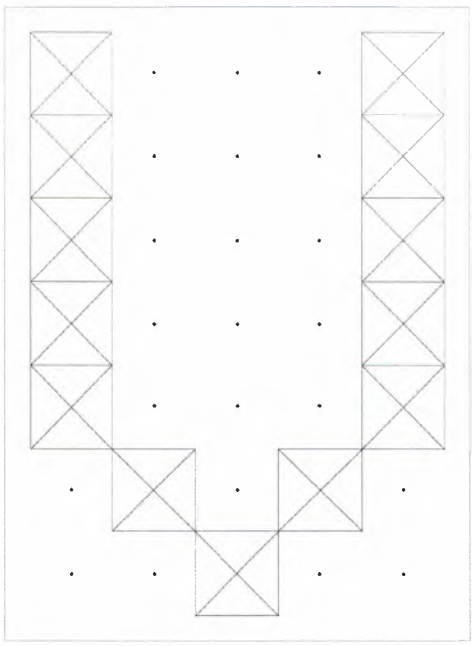


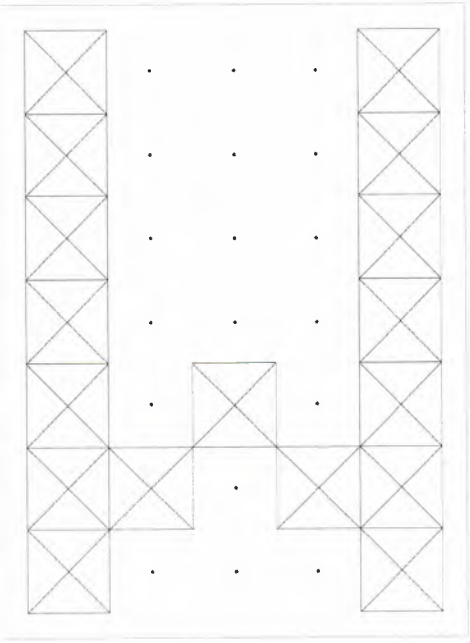


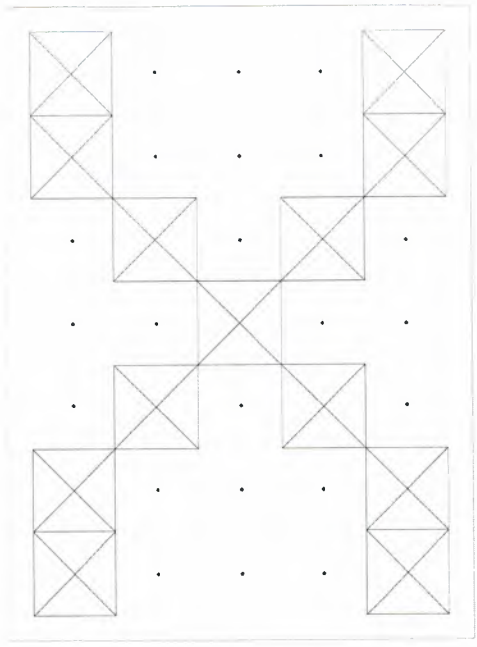


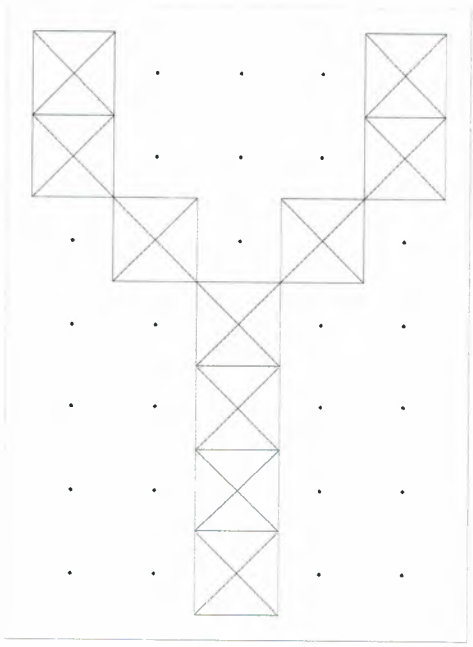


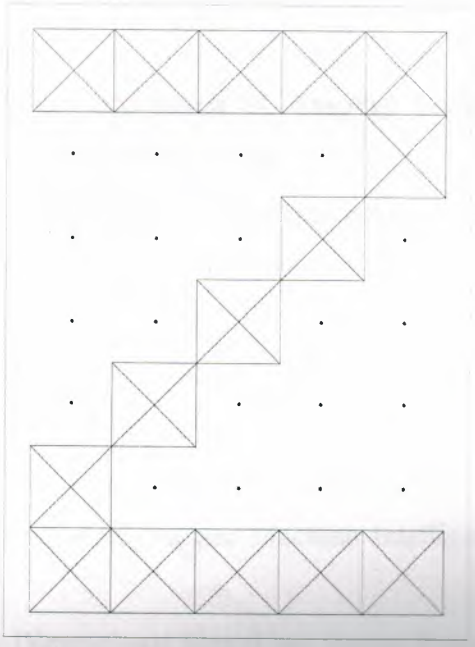












REFERENCES

[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.