



NEAR EAST UNIVERSITY

**GRADUATE SCHOOL OF APPLIED AND SOCIAL
SCIENCES**

**INTELLIGENT BANKNOTE IDENTIFICATION
SYSTEM (IBIS)**

Boran Şekeroğlu

Master Thesis

Department of Computer Engineering

Nicosia - 2004



Boran Şekeroğlu: Intelligent Banknote Identification System

**Approval of the Graduate School of
Applied and Social Sciences**

**Prof. Dr. Fahreddin M. Sadıkoğlu
Director**

**We certify that this thesis is satisfactory for the award of the
degree of Master of Science in Computer Engineering**

Examining Committee in Charge:

Prof. Dr. Parviz G. Alizade,

Chairman of Committee,

Assoc. Prof. Dr. Rahib Abiyev,

**Computer Engineering
Department, NEU**

Assist. Prof. Dr. Kadri Bürüncük,

**Electrical and Electronics
Engineering Department,
NEU**

Assoc. Prof. Dr. Adnan Khashman

Supervisor

NEU

JURY REPORT

DEPARTMENT OF
COMPUTER ENGINEERING

Academic Year: 2003-2004

STUDENT INFORMATION

Full Name	Boran Şekeroğlu		
Undergraduate degree	BSc.	Date Received	Spring 2001
University	Near East University	CGPA	3.13


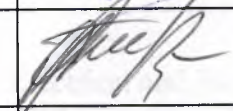
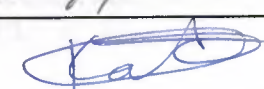
THESIS

Title	Intelligent Banknote Identification System		
Description	The aim of this thesis is development of banknote recognition system and to implement the Average Pixel / Node Approach in pattern recognition.		
Supervisor	Assoc.Prof.Dr.Adnan Khashman	Department	Computer Engineering


DECISION OF EXAMINING COMMITTEE

The jury has decided to accept / reject the student's thesis. The decision was taken unanimously / by majority .

COMMITTEE MEMBERS

Number Attending	3	Date	19/04/2004
Name		Signature	
Prof. Dr. Parviz G. Alizada, Chairman of the Jury			
Assoc. Prof. Dr. Rahib Abiyev, Member			
Assist. Prof. Dr. Kadri Bürüncük, Member			

APPROVALS

Date 19/04/2004	 Chairman of Department Prof. Dr. Doğan İbrahim
--------------------	---

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date:19/04/2004

Subject: Completion of M.Sc. Thesis

Participants: Prof. Dr. Parviz G. Alizada, Prof. Dr. Doğan İbrahim, Assoc. Prof. Dr. Rahib Abiyev, Assoc. Prof. Dr. Adnan Khashman, Assist. Prof. Dr. Kadri Bürüncük, Kamil Dimililer, Cemal Kavalcıoğlu, Ali Özgen, Alaa Eleyan

DECISION

We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

CGPA

970065

Boran Şekeroğlu

3.50

Prof. Dr. Parviz G. Alizada,

Committee Chairman, Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Rahib Abiyev,

Committee Member, Computer Engineering Department, NEU

Assist. Prof. Dr. Kadri Bürüncük,

Committee Member, Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Adnan Khashman,

Supervisor, Chairman of Electrical and Electronic Engineering Department, NEU

Chairman of Department
Prof. Dr. Doğan İbrahim

ACKNOWLEDGEMENT

First, I would like to thank my supervisor Assoc. Prof. Dr. Adnan Khashman for his invaluable advice and belief in my work and myself over the course of this MSc. Degree.

Second, I would like to express my gratitude to Near East University for the assistantship that made the work possible.

Third, I thank my family for their constant encouragement, support and patient during the preparation of this thesis.

Finally, I would also thank to my mother Şadan Şekeroğlu, for her unique ideas and advices.

Boran Şekeroğlu

ABSTRACT

In the real life, there are such image recognition problems that need construction of system with high accuracy and very quick decision-making mechanism. One of approaches to solve these problems is Neural Networks. For these reason the thesis is devoted one of the important problem, Intelligent Banknote Identification System.

Neural network applications have lately become a common feature with varying degrees of success and usability. Image processing has also gained an important part in engineering applications and machine automation. The combination of both image processing and neural networks can provide sufficient and robust solutions to problems where automation is required.

Back Propagation is a popular algorithm employed for training multilayer connectionist learning systems with nonlinear activation function – sigmoid.

The back – propagation learning algorithm has been implemented and tested on a real – life application, banknote recognition. Experimental results including training time and recognition accuracy are given.

The research work presented within this thesis describes the development of an intelligent system (Intelligent Banknote Identification System) that recognizes automatically any desired banknote (Turkish Lira) in this work. IBIS can be used for recognizing any banknote and is not intended to be used for detection of counterfeit banknotes or indeed for counting banknotes. IBIS uses image processing and a neural network simulated using the C – programming language.

CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT	ii
CONTENTS	iii
LIST OF FIGURES	vii
LIST OF TABLES	x
INTRODUCTION	xi
1. FUNDAMENTALS OF NEURAL NETWORKS	
1.1 Overview	1
1.2 Origins of Neural Networks	1
1.3 Biological Neuron	2
1.4 Artificial Neuron Models	4
1.4.1 An Artificial Neuron.....	4
1.4.2 Major Components of Artificial Neurons	5
1.5 Comparing Neural Networks and Traditional Computing	10
1.6 Network Layers	10
1.7 Communication Types and Connections	11
1.7.1 Inter – Layer Connections	11
1.7.2 Intra – Layer Connections	13
1.8 A Simple Artificial Neural Network	14
1.9 Learning	16
1.9.1 Unsupervised Learning	17
1.9.2 Supervised Learning	17
1.10 Off-line and On-line Learning	17
1.10.1 Off-line Learning	17
1.10.2 On-line Learning	17
1.11 Learning Laws	18
1.12 Network Selection	18
1.13 Summary	19

2. LEARNING METHODS IN NEURAL NETWORKS

2.1 Overview	20
2.2 Learning Methods	20
2.2.1 Unsupervised Learning	20
2.2.1.1 Unsupervised Learners	20
2.2.2 Supervised Learning	22
2.2.2.1 Supervised Learners	23
2.3 Why Back Propagation Neural Networks?.....	32
2.4 Summary.....	32

3. IMAGE PROCESSING FUNDAMENTALS

3.1 Overview	33
3.2 What is Image Processing ?	33
3.3 Image Processing Applications	33
3.3.1 Science and Space	33
3.3.2 Movies	33
3.3.3 Medical Industry	34
3.3.4 Machine Vision	34
3.3.5 Law Enforcement	34
3.4 How Your Computer Stores Image?	34
3.5 Image File Formats	36
3.5.1 BMP.....	36
3.5.2 JPEG	36
3.5.3 GIF	36
3.5.4 TIF	37
3.5.5 RAW	37
3.5.6 PNG	37
3.6 Image Segmentation	38
3.6.1 Image Segmentation Methods.....	38
3.7 Image Compression	39
3.7.1 Types of Compressions	39

3.7.2	Compression Formats	40
3.8	Edge Detection	41
3.8.1	Edge Definition	42
3.8.2	Properties of Edge Detectors	43
3.9	Image Restoration	45
3.9.1	Image Restoration Techniques	45
3.9.2	A Priori Knowledge	46
3.9.3	A Posteriori Knowledge	46
3.10	Image Enhancement	46
3.10.1	Gray Scale Modification	49
3.11	Summary	50

4. INTELLIGENT BANKNOTE IDENTIFICATION SYSTEM

4.1	Overview	51
4.2	IBIS	51
4.3	Image Processing	51
4.3.1	Preparing Image for Program	52
4.3.2	Data Preparation Program	52
4.4	Neural Networks	56
4.4.1	Training of Neural Network	57
4.4.2	Neural Network Generalization.....	60
4.5	Additional Information	61
4.6	Summary	61

5. EXPERIMENTAL RESULTS

5.1	Overview	62
5.2	Training Sets	62
5.3	Target & Actual Outputs	63
5.4	Tolerance	64

5.5	Testing Sets	64
5.6	Results	65
5.6.1	Results of First Experiment	65
5.6.2	Results of Second Experiment	68
5.6.3	Results of Third Experiment	70
5.6.4	Performance of IBIS with Corrupted Data	72
5.7	Summary	73
 CONCLUSION.....		74
REFERENCES.....		75
APPENDIX 1		78
APPENDIX 2		80
APPENDIX 3		85
APPENDIX 4		92

List of Figures

<u>SUBJECT</u>	<u>PAGE</u>
CHAPTER 1	
Figure 1.1 - Schematic of Biological Neuron	3
Figure 1.2 - The Synapse	3
Figure 1.3 - A Simple Artificial Neuron	4
Figure 1.4 - Sample Transfer Functions	7
Figure 1.5 - Fully Connected Neural Networks	11
Figure 1.6 - Partially Connected Neural Networks	12
Figure 1.7 - Feed Forward Neural Networks	12
Figure 1.8 - Classic Hierarchical Connection	13
Figure 1.9 - Simple Neural Network Diagram	15
Figure 1.10 - Simple Network with Feedback and Competition	16
CHAPTER 2	
Figure 2.1 - Unsupervised Learning	20
Figure 2.2 - Kohonen's Learning	21
Figure 2.3 - Competitive Learning	22
Figure 2.4 - Supervised Learning	22
Figure 2.5 - Basic Perceptron	23
Figure 2.6 - The Perceptron Model	23
Figure 2.7 - Hopfield Network	24
Figure 2.8 - Hamming Network.....	24
Figure 2.9 - Artificial Neuron	25
Figure 2.10- Back Propagation Network Structure	27
Figure 2.11- An Input Layer Neuron	27
Figure 2.12- A Hidden Layer Neuron	28
Figure 2.13- An Output Layer Neuron	28
CHAPTER 3	
Figure 3.1 - A 1-bit Image	35
Figure 3.2 - A 4-bit Image	35
Figure 3.3 - An 8-bit Image	35
Figure 3.4 - A 16-bit Image	35
Figure 3.5 - Step Edge Profile	43
Figure 3.6 - Edge Detection	44
Figure 3.7 - The Image Enhancement Process	48
Figure 3.8 - Image Enhancement	48
Figure 3.9 - Original and Modified Image	49
Figure 3.10- Original and Modified Image	50

CHAPTER 4

Figure 4.1 - Training Phase of IBIS	51
Figure 4.2 - Average Pixel / Node Approach	54
Figure 4.3 - Steps of Image Processing	54
Figure 4.4 - Storing Image Algorithm of IBIS	55
Figure 4.5 - Trimming Algorithm of IBIS	55
Figure 4.6 - Compression Algorithm of IBIS	56
Figure 4.7 - Neural Network Topology	56
Figure 4.8 - General Topology of IBIS	58
Figure 4.9 - Flowchart of IBIS Training	59
Figure 4.10 - Running Part of IBIS	60

CHAPTER 5

Figure 5.1 - Error Level Graph of Experiment I	66
Figure 5.2 - Error Level Graph of Experiment II	68
Figure 5.3 - Error Level Graph of Experiment III	70
Figure 5.4 - Corrupted Images	72

APPENDIX II

Figure A2.1 - 500,000 TL Front	80
Figure A2.2 - 500,000 TL Back	80
Figure A2.3 - 1,000,000 TL Front	80
Figure A2.4 - 1,000,000 TL Back	80
Figure A2.5 - 5,000,000 TL Front	81
Figure A2.6 - 5,000,000 TL Back	81
Figure A2.7 - 10,000,000 TL Front... ..	81
Figure A2.8 - 10,000,000 TL Front	81
Figure A2.9 - 20,000,000 TL Front	81
Figure A2.10 - 20,000,000 TL Back... ..	82
Figure A2.11 - 500,000 TL Front Reversed	82
Figure A2.12 - 500,000 TL Back Reversed	82
Figure A2.13 - 1,000,000 TL Front Reversed... ..	82
Figure A2.14 - 1,000,000 TL Back Reversed	83
Figure A2.15 - 5,000,000 TL Front Reversed	83
Figure A2.16 - 5,000,000 TL Back Reversed	83
Figure A2.17 - 10,000,000 TL Front Reversed.....	83
Figure A2.18 - 10,000,000 TL Front Reversed.....	84
Figure A2.19 - 20,000,000 TL Front Reversed.....	84
Figure A2.20 - 20,000,000 TL Back Reversed.....	84

APPENDIX III

Figure A3.1 - 50 AUS Dollar Front	85
Figure A3.2 - 50 AUS Dollar Back	85
Figure A3.3 - 100 AUS Dollar Front	85
Figure A3.4 - 100 AUS Dollar Front	85
Figure A3.5 - 5 AUS Dollar Front	86
Figure A3.6 - 50 AUS Dollar Back	86
Figure A3.7 - 100 Russian Ruble Front	86
Figure A3.8 - 100 Russian Ruble Back.	86
Figure A3.9 - 5000 Russian Ruble Front	86
Figure A3.10 - 5000 Russian Ruble Back	87
Figure A3.11 - 10,000 Russian Ruble Front	87
Figure A3.12 - 10,000 Russian Ruble Back	87
Figure A3.13 - 5 Pakistan Rupie Front	87
Figure A3.14 - 5 Pakistan Rupie Back	87
Figure A3.15 - 5 Canada Dollar Front.....	88
Figure A3.16 - 5 Canada Dollar Back.....	88
Figure A3.17 - 10 Canada Dollar Front.....	88
Figure A3.18 - 10 Canada Dollar Back	88
Figure A3.19 - 500,000 Romania Lei Front.....	88
Figure A3.20 - 500,000 Romania Lei Back.....	89
Figure A3.21 - 50,000 Romania Lei Front.....	89
Figure A3.22 - 50,000 Romania Lei Back.....	89
Figure A3.23 - 100,000 Romania Lei Front.....	89
Figure A3.24 - 100,000 Romania Lei Back.....	89
Figure A3.25 - 5 GB Sterling Front	90
Figure A3.26 - 5 GB Sterling Back ..	90
Figure A3.27 - 10 GB Sterling Front	90
Figure A3.28 - 10 GB Sterling Back ..	90
Figure A3.29 - 20 GB Sterling Front	90
Figure A3.30 - 20 GB Sterling Back	91
Figure A3.31 - 1 CYP Pound Front	91
Figure A3.32 - 1 CYP Pound Back	91
Figure A3.33 - 5 CYP Pound Front	91
Figure A3.34 - 5 CYP Pound Back	91
Figure A3.33 - 10 CYP Pound Front	92
Figure A3.34 - 10 CYP Pound Back	92

List of Tables

<u>SUBJECT</u>	<u>PAGE</u>
Table 1.1 - Comparison of Computing Approaches	10
Table 1.2 - Network Selector Table	19
Table 5.1 - Training Set I	62
Table 5.2 - Training Set II	62
Table 5.3 - Training Set III	63
Table 5.4 - Target Outputs	63
Table 5.5 - Example of Actual Outputs	64
Table 5.6 - Tolerance	64
Table 5.7 - Testing Set I	64
Table 5.8 - Testing Set II	65
Table 5.9 - Training Results of Experiment I	66
Table 5.10 - Results of Experiment I for Testing Set I	66
Table 5.11 - Results of Experiment I for Testing Set II	67
Table 5.12 - Training Results of Experiment II	68
Table 5.13 - Results of Experiment II for Testing Set I	69
Table 5.14 - Results of Experiment II for Testing Set II	69
Table 5.15 - Training Results of Experiment III	70
Table 5.16 - Results of Experiment III for Testing Set I	71
Table 5.17 - Results of Experiment III for Testing Set II	72
Table A1 - Brain Properties	78
Table A2 - Brain Weights of Some Alive.....	78
Table A3 - Biological Neuron vs. Artificial Neuron	79

INTRODUCTION

In worldwide banking systems, there are several kinds of banknotes then it takes much time to classify all banknote kinds by human. Thus, automatic banknote readers and sorters machines are needed.

For this reason, many recognition techniques such as image processing techniques and a neural network (NN) technique have been proposed for the reader and sorter machines. Based on several previous research studies, it seems that the NN technique is more suitable and robust for recognition than any other techniques because of its self-organization and generalization abilities. Therefore, the NN technique has been applied to invent recognition system for banknote readers and sorters machines. However, using only the NN technique for recognition is not effective for real world banking system. Consequently, image processing techniques and Average Pixel / Node Approach also have been applied to improve the recognition ability of such system. Therefore, this recognition system is composed of three core techniques that are image processes, the NN technique, Average Pixel / Node Approach.

This recognition system has been applied to recognize many kinds of banknotes such as Turkish Lira and Greek Cyprus Pound. In this report, these banknotes have been proposed as objects of recognition.

For Turkish banknotes (Turkish Lira = TL), there are 5 types that are 500,000 TL, 1,000,000 TL, 5,000,000 TL, 10,000,000 TL and 20,000,000 TL. Figure A2.1-A2.10 shows all types of Turkish banknotes in 2 sides (front and back). Each banknote has approximately same width, about 7.6 cm, and has approximately same length, about 16.2 cm. But each banknote and both sides of each banknote has different color and figures on it. For example, 20,000,000 TL is green while 10,000,000 TL is red.

Furthermore, there are 4 conveyed directions of each type for inserting them to the banking machine which are front upright, front reversed, back upright and back reversed. Figure

A2.1-A2.20 shows 4 conveyed directions. Therefore, there are totally 20 patterns for all Turkish banknotes.

Greek Cyprus pounds have been used to show the flexibility of this system. For Greek Cyprus banknotes (Cyprus Pounds = CYPP), there are 4 types that are 1 CYPP, 5 CYPP, 10 CYPP and 20 CYPP. Each banknote has different width, length, color and figures on it.

Several years ago, many researchers applied various recognition techniques to invent banknote recognition machines such as a discriminative inequalities technique and a NN technique [20]. For the discriminative inequalities technique, it requires discriminative points, which contain characteristics of each banknote, and threshold values to distinguish purpose banknote from other banknotes. This technique requires expert engineering designers to manually determine those required values by trail and error that take much time to design banknote recognition machines. Therefore, the NN technique, which is widely used in several fields of engineering such as pattern recognition, system identification and control problem, has been applied for banknote recognition. The NN is applied for banknote recognition, which is the part of pattern recognition, because of its abilities of self-organization and generalization [20]. By these 2 abilities, the NN can recognize patterns effectively and robustly.

According to previous researches of Prof. Takeda, a three-layers NN has been applied to recognize banknotes by inputting FFT data of banknotes, which extracted from 4 sensors to the NN [20]. However, this kind of inputs made a large scale of the NN and then made a scale problem for inventing banknote recognition machines. In this work, basic image compression methods, image segmentation methods, Back Propagation Learning Algorithm and Average Pixel / Node Approach has been applied. The reasons, advantages and disadvantages of using these methods will be explained.

The aims of the work presented within this thesis:

- To investigate image-processing applications using Neural Networks.
- To provide an intelligent system (Intelligent Banknote Identification System) that identifies banknotes.

- To design and simulate a neural network for banknote identification using C – Language.
- To confirm the success of IBIS real – life application.
- To show average pixel/node approach can be implemented to represent image.

This work consists of 5 chapters and a conclusion. First three chapters give an introduction about the background of this work, Neural Networks and Image Processing and the last two chapters explain the work done.

In Chapter 1, an introduction about the general neural networks, development of neural networks, biological neural networks, artificial models and methods of neural networks are presented. Also major components of artificial neurons, connection types, learning laws and comparison of neural networks with traditional computing will be explained.

In Chapter 2, all learning methods and their algorithms will be described in details. The training method usually consists of one of two schemes:

- Unsupervised learning
- Supervised Learning

In Chapter 3, the image processing fundamentals will be explained. After brief introduction about file formats; Image components will be described.

In Chapter 4, all steps of Intelligent Banknote Identification System (IBIS) will be described. From getting image to the actual outputs of Intelligent Banknote Identification System (IBIS) will be explained in two parts: Image Processing and Neural Networks.

In Chapter 5, experimental results will be described in details. Also the efficiency and success of IBIS will be shown.

In Conclusion, the general results of this work will be explained.

CHAPTER 1

FUNDAMENTALS OF NEURAL NETWORKS

1.1 Overview

This chapter presents an introduction about the general neural networks, development of neural networks, biological neural networks, artificial models and methods of neural networks. Also major components of artificial neurons, connection types, learning laws and comparison of neural networks with traditional computing will be explained.

1.2 Origins of Neural Networks

In the early 1940s, Warren McCulloch (1899-1969, was an American neurophysiologist and cybernetician¹) and Walter Pitts (1928-late 1960's, was logician who worked in the field of Cognitive Psychology) published a seminar paper titled "A Logical Calculus of the Ideas Immanent in Nervous Activity". In it, they proposed a mathematical model of a neuron, which could perform computations. This artificial neuron, or neurode (some call them neurons), was a simple device, which could receive input from other such devices [1].

The neurode's output was either a 1 or a 0, reflecting the *all-or-none* theory of biological neurons. When the total input reached a certain critical level, the neurode would send its output to other neurodes with which it was connected. This method is called *threshold logic*. In basic propositional logic, something can be either true or false. Since a neurode's state is either a 1 or a 0, it can be represented by a proposition. If you organize simple neurodes into a network, they can combine to form more complex propositions. This theory was so influential that this type of neurode is called the *McCulloch- Pitts neuron*. Some modern neural networks use neurodes, which are essentially extensions of the McCulloch-Pitts neuron.

The concept of neural networks has been around since the early 1950s, but was mostly dormant until the mid 1980s. One of the first neural networks developed was the *perceptron*. Created by a psychologist named Frank Rosenblatt in 1958, *the perceptron* was a very simple system that used interconnected neurodes to analyze data, usually visual patterns. Rosenblatt published a series of papers, which generated a great deal of interest in the perceptron. Many people researched and developed further the perceptron model, even implementing it in hardware. The perceptron was widely and unrealistically praised by researchers. Rosenblatt and other scientists claimed that

¹: **Cybernetics** is a theory of the communication and control of regulatory feedback. The term *cybernetics* stems from the Greek *kybernetis* meaning "steersman".

eventually, with enough complexity and speed, the perceptron would be able to solve almost any problem.

This was far from the truth. In 1969, Marvin Minsky (1927, is an American scientist in the field of Artificial Intelligence) and Seymour Papert (1928, is an MIT¹ mathematician. He is one of the pioneers of AI) published an influential book titled "Perceptrons". In it, they proved several theorems, which showed that the perceptron could never solve a class of simple problems, and hinted at several other serious, fundamental flaws in the model. After "Perceptrons", scientists working on neural network type devices found it almost impossible to receive funding.

1.3 Biological Neuron

The brain is a collection of about 10 billion interconnected neurons [1]. Each neuron is a cell that uses biochemical reactions to receive, process and transmit information. A neuron's dendritic tree is connected to a thousand neighboring neurons. When one of those neurons fire, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation. Spatial summation occurs when several weak signals are converted into a single large one, while temporal summation converts a rapid series of weak pulses from one source into one large signal. The aggregate input is then passed to the soma (cell body). The soma and the enclosed nucleus don't play a significant role in the processing of incoming and outgoing data. Their primary function is to perform the continuous maintenance required to keep the neuron functional. The part of the soma that does concern itself with the signal is the axon hillock. If the aggregate input is greater than the axon hillock's threshold value, then the neuron *fires*, and an output signal is transmitted down the axon. The strength of the output is constant, regardless of whether the input was just above the threshold, or a hundred times as great. The output strength is unaffected by the many divisions in the axon; it reaches each terminal button with the same intensity it had at the axon hillock. This uniformity is critical in an analogue device such as a brain where small errors can snowball, and where error correction is more difficult than in a digital system (Figure 1.1).

Each terminal button is connected to other neurons across a small gap called a synapse [1] (Figure 1.2). The physical and neurochemical characteristics of each synapse determines the strength and polarity of the new input signal. This is where the brain is

¹ : MIT : Massachussets Institute of Technology

the most flexible, and the most vulnerable. Changing the constitution of various neurotransmitter chemicals can increase or decrease the amount of stimulation that the firing axon imparts on the neighboring dendrite. Altering the neurotransmitters can also change whether the stimulation is excitatory or inhibitory. Many drugs such as alcohol and LSD have dramatic effects on the production or destruction of these critical chemicals. The infamous nerve gas sarin can kill because it neutralizes a chemical (acetyl cholinesterase) that is normally responsible for the destruction of a neurotransmitter (acetylcholine). This means that once a neuron fires, it keeps on triggering all the neurons in the vicinity. One no longer has control over muscles, and suffocation ensues.

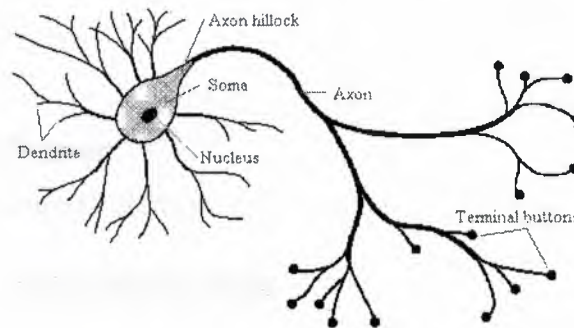


Figure 1.1 Schematic of Biological Neuron

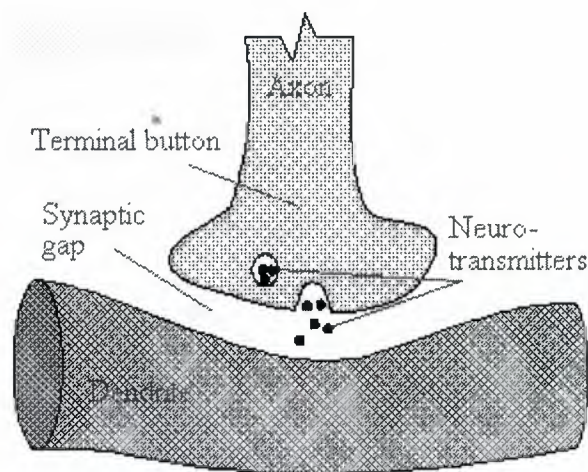


Figure 1.2 The synapse

1.4 Artificial Neuron Models

Computational neurobiologists have constructed very elaborate computer models of neurons in order to run detailed simulations of particular circuits in the brain. As Computer Scientists, we are more interested in the general properties of neural networks; independent of how they are actually "implemented" in the brain. This means that we can use much simpler, abstract "neurons", which (hopefully) capture the essence of neural computation even if they leave out much of the details of how biological neurons work.

People have implemented model neurons in hardware as electronic circuits, often integrated on VLSI chips. Remember though that computers run much faster than brains - we can therefore run fairly large networks of simple model neurons as software simulations in reasonable time. This has obvious advantages over having to use special "neural" computer hardware.

1.4.1 An Artificial Neuron

Basic computational element (model neuron) is often called a node or unit (Figure 1.3). It receives input from some other units, or perhaps from an external source. Each input has an associated weight w , which can be modified so as to model synaptic learning. The unit computes some function f of the weighted sum of its inputs:

$$y_i = f\left(\sum_j w_{ij} y_j\right)$$

Its output, in turn, can serve as input to other units.

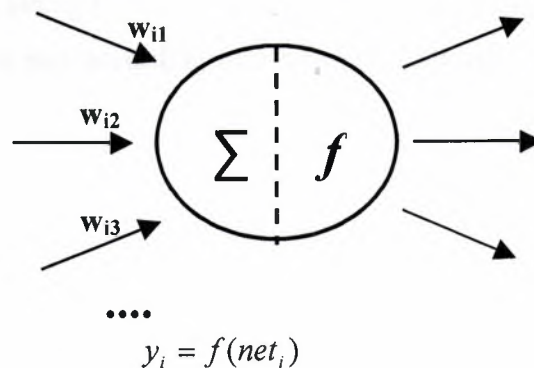


Figure 1.3 A Simple Artificial Neuron

- The weighted sum $\sum_j w_{ij} y_j$ is called the *net input* to unit i , often written net_i .
- Note that w_{ij} refers to the weight from unit j to unit i (not the other way around).
- The function f is the unit's *activation function*. In the simplest case, f is the identity function, and the unit's output is just its net input. This is called a *linear unit*.

1.4.2 Major Components of Artificial Neurons

Major components of an artificial neuron are described as shown below [1]. These components are valid whether the neuron is used for input, output, or is in one of the hidden layers.

1.4.2.1 Weighting Factors

A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input the impact that it needs on the processing element's summation function. These weights perform the same type of function, as do the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response. Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

1.4.2.2 Summation Function

The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as (i_1, i_2, \dots, i_n) and (w_1, w_2, \dots, w_n) . The total input signal is the dot, or inner, product of these two vectors. This

simplistic summation function is found by multiplying each component of the i vector by the corresponding component of the w vector and then adding up all the products. $Input_1 = i_1 * w_1$, $input_2 = i_2 * w_2$, etc., are added as $input_1 + input_2 + \dots + input_n$. The result is a single number, not a multi-element vector. Geometrically, the inner product of two vectors can be considered a measure of their similarity. If the vectors point in the same direction, the inner product is maximum; if the vectors point in opposite direction (180 degrees out of phase), their inner product is minimum. The summation function can be more complex than just the simple input and weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to a simple product summing, the summation function can select the minimum, maximum, majority, product, or several normalizing algorithms. The specific algorithm for combining neural inputs is determined by the chosen network architecture and paradigm.

Some summation functions have an additional process applied to the result before it is passed on to the transfer function. This process is sometimes called the *activation function*. The purpose of utilizing an activation function is to allow the summation output to vary with respect to time. Activation functions currently are pretty much confined to research. Most of the current network implementations use an "identity" activation function, which is equivalent to not having one. Additionally, such a function is likely to be a component of the network as a whole rather than of each individual processing element component.

1.4.2.3 Transfer Function

The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal. If the sum of the input and weight products is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant. The threshold, or transfer function, is generally non-linear. Linear (straight-line)

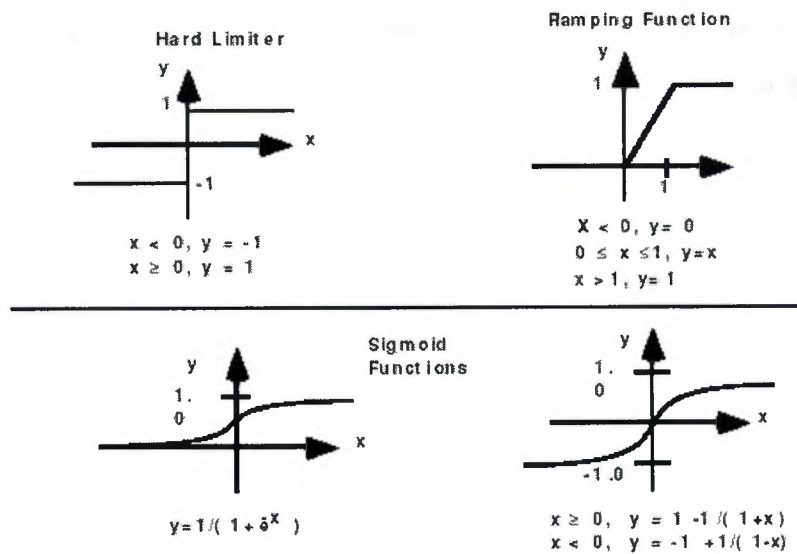


Figure 1.4 Sample Transfer Functions

functions are limited because the output is simply proportional to the input. Linear functions are not very useful. That was the problem in the earliest network models as noted in Minsky and Papert's book *Perceptrons*. The transfer function could be something as simple as depending upon whether the result of the summation function is positive or negative. The network could output zero and one, one and minus one, or other numeric combinations. The transfer function would then be a "hard limiter" or step function. See Figure 1.4 for sample transfer functions.

Another type of transfer function, the threshold or ramping function, could mirror the input within a given range and still act as a hard limiter outside that range. It is a linear function that has been clipped to minimum and maximum values, making it non-linear. Yet another option would be a sigmoid or S-shaped curve. That curve approaches a minimum and maximum value at the asymptotes. It is common for this curve to be called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent when it ranges between -1 and 1. Mathematically, the exciting feature of these curves is that both the function and its derivatives are continuous. This option works fairly well and is often the transfer function of choice. Other transfer functions are dedicated to specific network architectures. Prior to applying the transfer function, uniformly distributed random noise may be added. The source and amount of this noise is determined by the learning

mode of a given network paradigm. This noise is normally referred to as "temperature" of the artificial neurons. The name, temperature, is derived from the physical phenomenon that as people become too hot or cold their ability to think is affected. Electronically, this process is simulated by adding noise. Indeed, by adding different levels of noise to the summation result, more brain-like transfer functions are realized. To more closely mimic nature's characteristics, some experimenters are using a gaussian noise source. Gaussian noise is similar to uniformly distributed noise except that the distribution of random numbers within the temperature range is along a bell curve. The use of temperature is an ongoing research area and is not being applied to many engineering applications.

NASA announced a network topology, which uses what it calls a *temperature coefficient* in a new feed-forward, back-propagation learning function. But this temperature coefficient is a global term that is applied to the gain of the transfer function. It should not be confused with the more common term, *temperature*, which is simple noise being added to individual neurons. In contrast, the global temperature coefficient allows the transfer function to have a learning variable much like the synaptic input weights. This concept is claimed to create a network, which has a significantly faster (by several order of magnitudes) learning rate and provides more accurate results than other feed forward, back-propagation networks.

1.4.2.4 Scaling and Limiting

After the processing element's transfer function, the result can pass through additional processes, which scale and limit. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism, which insures that the scaled result does not exceed an upper, or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed. This type of scaling and limiting is mainly used in topologies to test biological neuron models, such as James Anderson's brain-state-in-the-box [1].

1.4.2.5 Output Function (Competition)

Each processing element is allowed one output signal, which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

1.4.2.6 Error Function and Back-Propagated Value

In most learning networks the difference between the current output and the desired output is calculated. This raw error is then transformed by the error function to match particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error. The current error is typically propagated backwards to a previous layer. Yet, this back-propagated value can be either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle.

1.4.2.7 Learning Function

The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the *adaptation function*, as well

as the learning mode. There are two types of learning: *supervised* and *unsupervised*. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is learning by doing. In Chapter 2, you can see the details of Supervised and Unsupervised Learning .

1.5 Comparing Neural Networks and Traditional Computing

Neural networks offer a different way to analyze data, and to recognize patterns within that data, than traditional computing methods. However, they are not a solution for all computing problems. Traditional computing methods work well for problems that can be well characterized. Balancing checkbooks, keeping ledgers, and keeping tabs of inventory are well defined and do not require the special characteristics of neural networks. Table 1.1 identifies the basic differences between the two computing approaches. Traditional computers are ideal for many applications. They can process data, track inventories, network results, and protect equipment. These applications do not need the special characteristics of neural networks.

1.6 Network Layers

Basically, all artificial neural networks have simple topologic structures. Some neuron are used to get inputs from real world and some other neurons are used to form the real world at the output of the network. All remained neurons are called hidden neurons because of their invisibility.

Table 1.1 Comparison of Computing Approaches

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing Style	Sequential	Parallel
Functions	Logically (left brained) via Rules Concepts Calculations	Gestalt (right brained) via Images Pictures Controls
Learning Method	by rules	by example
Applications	Accounting, word processing math, inventory, digital communications	Sensor processing, speech recognition, pattern recognition, text recognition

When an inputs reach to input layer, neurons produce outputs that are the inputs of other layers.

The number of the hidden neurons is so important because if we use too much hidden neurons in our network, we cannot reach to desired output. And it means, there is a generalization in our network.

1.7 Communication and Types of Connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there might however be a two-way connection between two neurons, because there may be another path in reverse direction. A neuron receives input from many neurons, but produce a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

1.7.1 Inter-Layer Connections

There are different types of connections used between layers; these connections between layers are called inter-layer connections [3].

1.7.1.1 Fully Connected

Each neuron on the first layer is connected to every neuron on the second layer (Figure 1.5) [3].

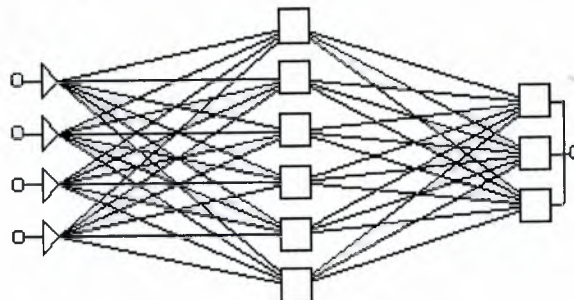


Figure 1.5 Fully Connected Neural Networks

1.7.1.2 Partially Connected

A neuron of the first layer does not have to be connected to all neurons on the second layer (Figure 1.6) [3].

1.7.1.3 Feed Forward

The neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back from the neurons on the second layer (Figure 1.7) [3].

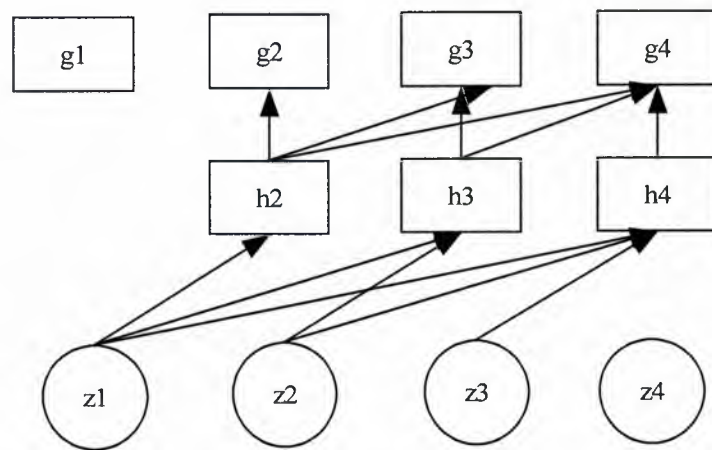


Figure 1.6 Partially Connected Neural Networks

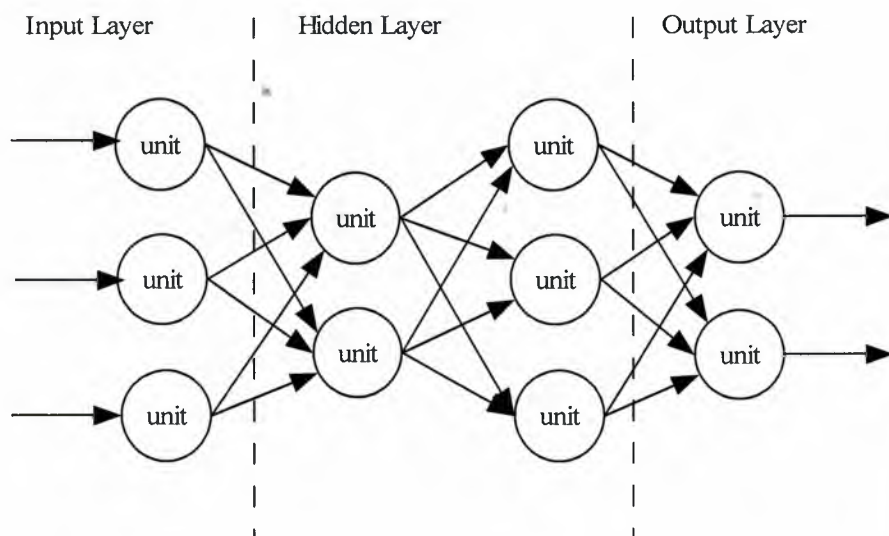


Figure 1.7 Feed Forward Neural Networks

1.7.1.4 Bi-directional

There is another set of connections carrying the output of the neurons of the second layer into the neurons of the first layer.

Feed forward and bi-directional connections could be fully- or partially connected [3].

1.7.1.5 Hierarchical

If a neural network has a hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer (figure 1.8) [3].

1.7.1.6 Resonance

The layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved [3].

1.7.2 Intra-Layer Connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections [3].

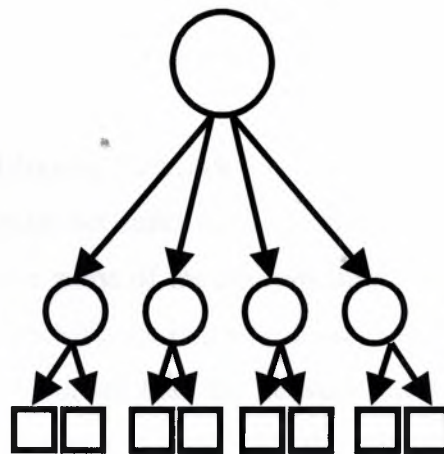


Figure 1.8 Classic Hierarchical Connection

1.7.2.1 Recurrent

The neurons within a layer are fully- or partially connected to one another. After these neurons receive input from another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer [3].

1.7.2.2 On-Center / Off-Surround

A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is allowed to update its and its gang member's weights. (There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.)

1.8 A Simple Artificial Neural Network

Basically, all artificial neural networks have a similar structure or topology as shown in Figure 1.9. In that structure some of the neurons interfaces to the real world to receive its inputs. Other neurons provide the real world with the network's outputs. This output might be the particular character that the network thinks that it has scanned or the particular image it thinks is being viewed. All the rest of the neurons are hidden from view.

But a neural network is more than a bunch of neurons. Some early researchers tried to simply connect neurons in a random manner, without much success. Now, it is known that even the brains of snails are structured devices. One of the easiest ways to design a

structure is to create layers of elements. It is the grouping of these neurons into layers, the connections between these layers, and the summation and transfer functions that comprises a functioning neural network. The general terms used to describe these characteristics are common to all networks.

Although there are useful networks, which contain only one layer, or even one element, most applications require networks that contain at least the three normal types of layers - input, hidden, and output. The layers of input neurons receive the data either from input files or directly from electronic sensors in real-time applications. The output layer sends information directly to the outside world, to a secondary computer process, or to other devices such as a mechanical control system. Between these two layers can be many hidden layers. These internal layers contain many of the neurons in various interconnected structures. The inputs and outputs of each of these hidden neurons simply go to other neurons.

In most networks each neuron in a hidden layer receives the signals from all of the neurons in a layer above it, typically an input layer. After a neuron performs its function it passes its output to all of the neurons in the layer below it, providing a feed forward path to the output.

These lines of communication from one neuron to another are important aspects of neural networks. They are the glue to the system. They are the connections, which provide a variable strength to an input. There are two types of these connections. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. In more human terms one excites while the other inhibits.

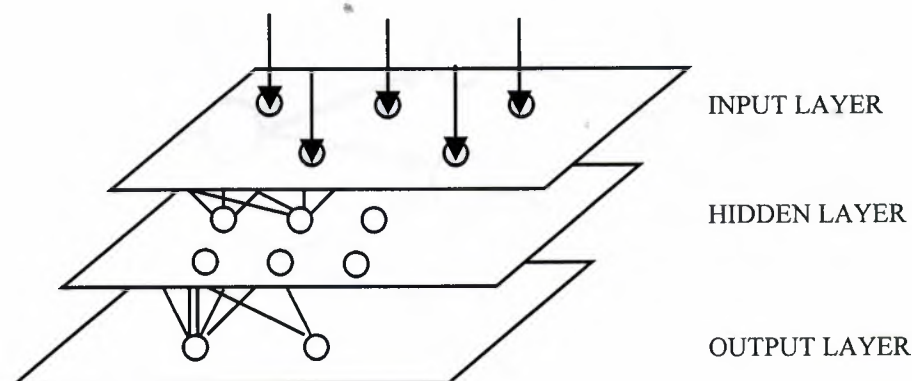


Figure 1.9 Simple Neural Network Diagram

Some networks want a neuron to inhibit the other neurons in the same layer. This is called *lateral inhibition*. The most common use of this is in the output layer. For example in text recognition if the probability of a character being a "P" is .85 and the probability of the character being an "F" is .65, the network wants to choose the highest probability and inhibit all the others. It can do that with lateral inhibition. This concept is also called *competition*.

Another type of connection is *feedback*. This is where the output of one-layer routes back to a previous layer. An example of this is shown in Figure 1.10.

The way that the neurons are connected to each other has a significant impact on the operation of the network. In the larger, more professional software development packages the user is allowed to add, delete, and control these connections at will. By "tweaking" parameters these connections can be made to either excite or inhibit.

1.9 Learning

The brain basically learns from experience. Neural networks are sometimes called machine-learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights. The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.

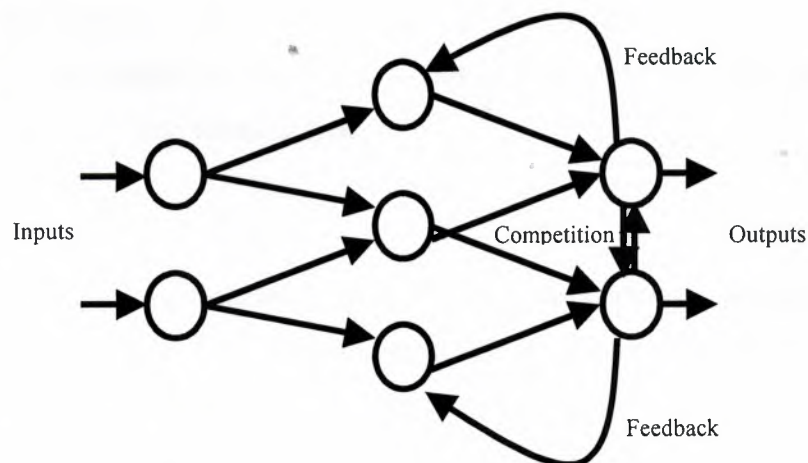


Figure 1.10 Simple Network with Feedback and Competition

The training method usually consists of one of two schemes:

1.9.1 Unsupervised learning

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing.

1.9.2 Supervised Learning

In a supervised learning process, the input data and its corresponding output are presented to the neural network. The neural network, according to defined law, change its weights in order to be able to reproduce the correct output, when an input is applied.

1.10 Off-line and On-line Learning

One can categorize the learning methods into yet another group, off-line or on-line. When the system uses input data to change its weights to learn the domain knowledge, the system could be in training mode or learning mode. When the system is being used as a decision aid to make recommendations, it is in the operation mode; this is also sometimes called recall.

1.10.1 Off-line

In the off-line learning methods, once the systems enters into the operation mode, its weights are fixed and do not change any more. Most of the networks are of the off-line learning type.

1.10.2 On-line

In on-line or real time learning, when the system is in operating mode (recall), it continues to learn while being used as a decision tool. This type of learning has a more complex design structure.

In Chapter 2, Learning Methods will be described in details.

1.11 Learning laws

There is a variety of learning laws, which are in common use [1]. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule. Man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc. A few of the commonly used laws will be described in chapter 2.

1.12 Network Selection

Because all artificial neural networks are based on the concept of neurons, connections, and transfer functions, there is a similarity between the different structures, or architectures, of neural networks. The majority of the variations stems from the various learning rules and how those rules modify a network's typical topology. The following sections outline some of the most common artificial neural networks. They are organized in very rough categories of application. These categories are not meant to be exclusive, they are merely meant to separate out some of the confusion over network architectures and their best matches to specific applications. Basically, most applications of neural networks fall into the following five categories:

- Prediction
- Classification
- Data association
- Data conceptualization
- Data filtering

Table 1.2 shows the differences between these network categories and shows which of the more common network topologies belong to which primary category. This chart is intended as a guide and is not meant to be all-inclusive. Some of these networks, which have been grouped by application, have been used to solve more than one type of problem. Feed forward back-propagation in particular has been used to solve almost all

Table 1.2 Network Selector Table¹

NETWORK TYPE	NETWORKS	USE FOR NETWORK
Prediction	-Back Propagation -Delta Bar Delta -Extended Delta Bar Delta -Directed Random Search -Higher order Neural Networks -Self Organizing Map into Back Propagation	Use input values to predict some output (e.g. pick the best stocks in the stock market, predict the weather, identify people with cancer risk)
Classification	-Learning vector quantization -Counter propagation -Probabilistic Neural Networks	Use input values to determine the classification (e.g. is the input the letter A, is the blob of the video data a plane and what kind of plane is it)
Data Association	-Hopfield -Boltzman Machine -Hamming Network -Bi-directional associative memory -Spatio-temporal pattern recognition	Like classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but also identify when the scanner doesn't work properly)
Data Conceptualization	-Adaptive Resonance Network -Self organizing map	Analyze the input so that grouping relationships can be inferred (e.g. extract from a data base the names of those most likely to buy a particular product)
Data Filtering	-Recirculation	Smooth an input signal (e.g. take the noise out of a telephone signal)

types of problems and indeed is the most popular for the first four categories. The next five subsections describe these five network types.

1.13 Summary

In this chapter, the backgrounds of artificial neural networks and necessary information about them were explained. Now, the parts of neural networks that are used in the application part of this thesis will be focused.

CHAPTER 2

LEARNING METHODS IN NEURAL NETWORKS

2.1 Overview

As it mentioned in Chapter 1, the brain basically learns from experience. The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training. In this chapter, all learning methods and their algorithms will be described in details.

The training method usually consists of one of two schemes:

- Unsupervised learning
- Supervised Learning

2.2 Learning Methods

2.2.1 Unsupervised Learning

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing [1].

2.2.1.1 Unsupervised Learners

a- Kohonen's Learning:

Kohonen suggested that one of the important mechanism in the human brain is placement of neurons in an orderly manner . Kohonen's

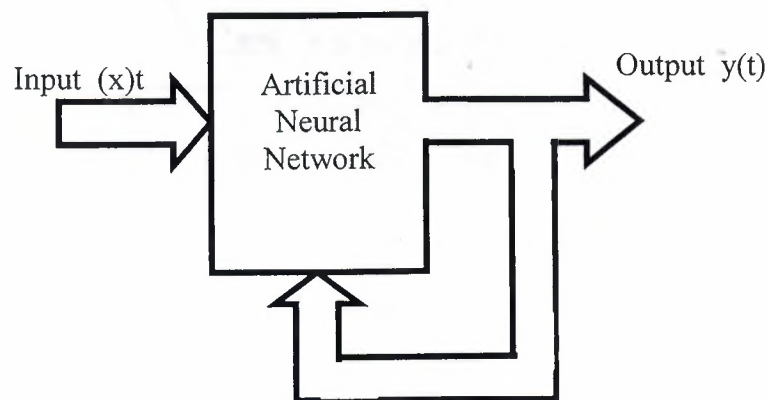


Figure 2.1 - Unsupervised Learning²

learning algorithm creates a *feature map* by adjusting weights from input vectors to output vectors in a *two layer network*. The first layer is input. The second is the *competitive layer*. The two layers are fully interconnected. Input vectors are presented sequentially to layer L1 (input). Each unit computes the dot product of its weight with the input vector. The unit with the highest dot product is declared the winner. This and its neighbors are the only units allowed to learn [5].

b- Competitive Learning

The simplest way to implement competitive learning is where each unit in the *hidden* or output layers receives input from all the units in the preceding layers. Within the layer units are broken down into a set of *inhibitory clusters*. The units within the clusters compete with one another to respond to data appearing at the input layer. The more strongly any particular unit responds to incoming stimulus the more it inhibits other units within the cluster. The unit learns by shifting a fraction of its weights from its inactive lines. The main disadvantage of competitive learning is the loss of previous learnings (Figure 2.3) [5].

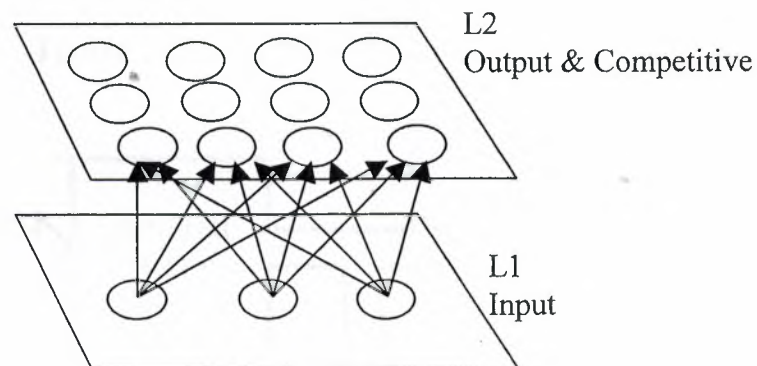


Figure 2.2 Kohonen's Learning⁵

c- Adaptive Resonance Theory (ART)

Divided into 2 methods :

- Accept only binary
- Accept binary & continuous input

2.2.2 Supervised Learning

In a supervised learning process, the input data and its corresponding output are presented to the neural network. The neural network, according to defined law, change its weights in order to be able to reproduce the correct output, when an input is applied [1].

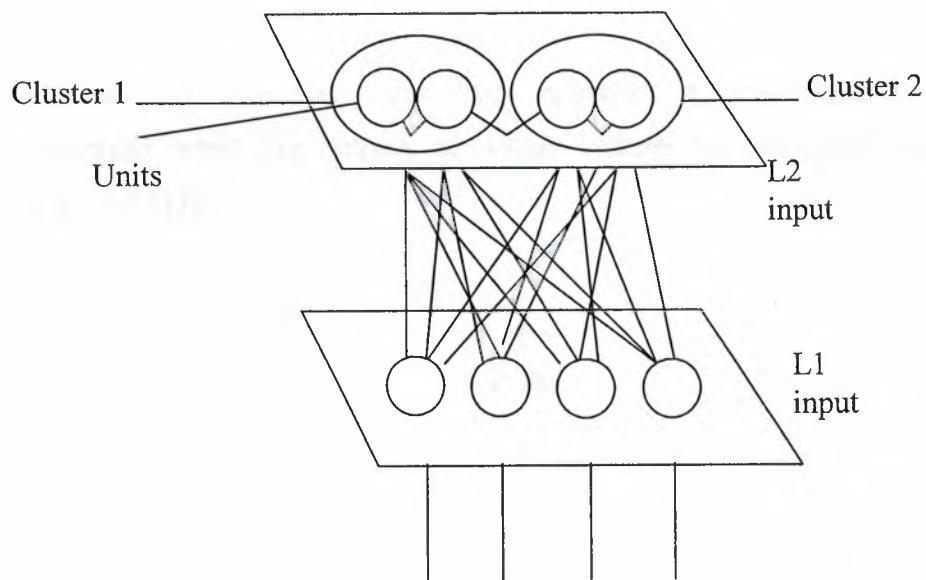


Figure 2.3 Competitive Learning⁵

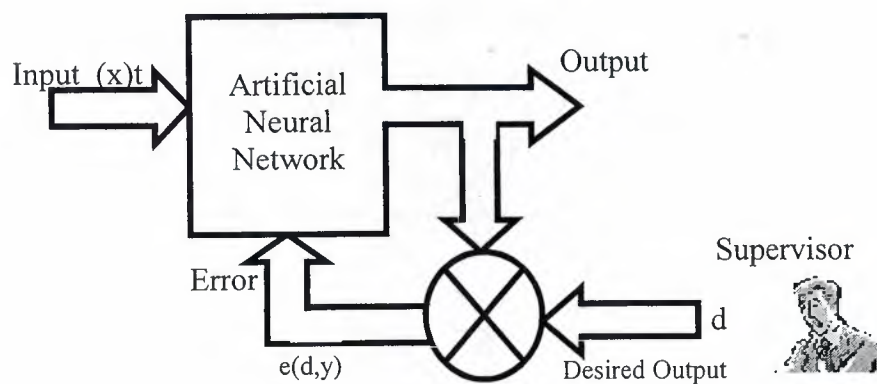


Figure 2.4 - Supervised Learning²

2.2.2.1 Supervised Learners

a- The Perceptron

This can be trained and can make decisions. During the training phase, *pairs* of input & output vectors are used to train the network. With each input vector, the output vector is compared with a *desired* output (*target*) and the error between the *actual* and the *desired* output vectors is used to update the weights [5].

b- Hopfield Network

It is essentially used with binary number. Weights are initialized using training samples. In the decision making phase, the test data is presented to the net at certain time. Following initialization the Hopfield Network *iterates in discrete time* stops using some mathematical function, and the network is considered to have converged when the outputs no longer change on successive iterations (Figure 2.7) [5].

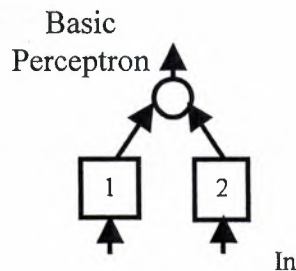


Figure 2.5 Basic Perceptron

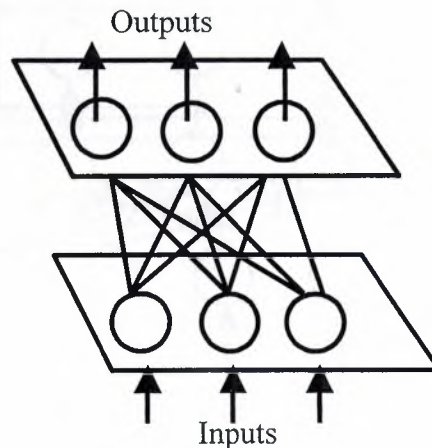


Figure 2.6 The Perceptron Model

c- Hamming Network

It is similar to Hopfield network . As shown in the figure 2.8, it consists of four layer.

L1 : Input layer

L2 : Calculates matching scores

L3 : Feedbacks as in Hopfield

L4 : Output layer

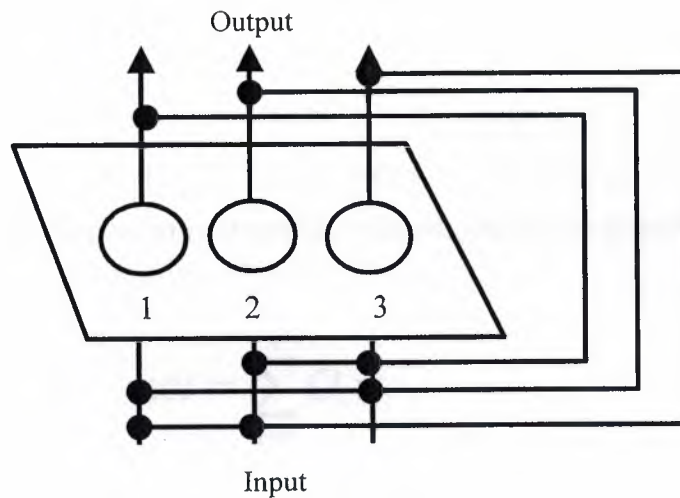


Figure 2.7 Hopfield Network⁵

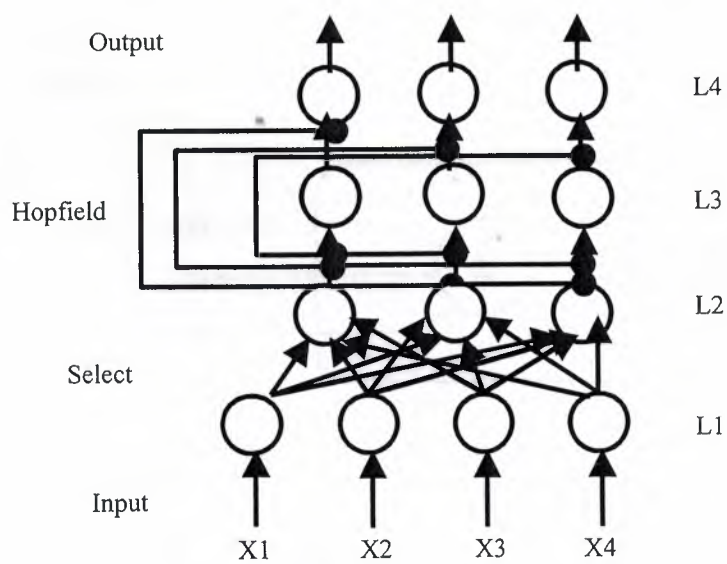


Figure 2.8 Hamming Network⁵

d- Back Propagation Training Algorithm

The equations that describe the network training and operation can be divided into two categories [6]. First, the *feed-forward* calculations. These are used in both training mode in the operation of the trained neural network. Second, the *error back propagation* calculations. These are applied only during training. But before we present the two categories of calculations, we have to describe another important element, *activation function* that the algorithm will be based upon.

i - The Activation Function

An artificial neuron (Figure 2.1), as it was described in chapter 1, is the fundamental building block in a back propagation network. The input to the neuron is obtained as the weighted sum given by equation (2.1).

$$net = \sum_{i=1}^n O_i w_i \quad (2.1)$$

In figure 2.9, F is the activation function, which has a sigmoid form. The simplicity of the derivative of the sigmoid function justifies its popularity and use as an activation function in training algorithms. With a sigmoid activation function the output of the neuron is given by equation (2.2) and (2.3).

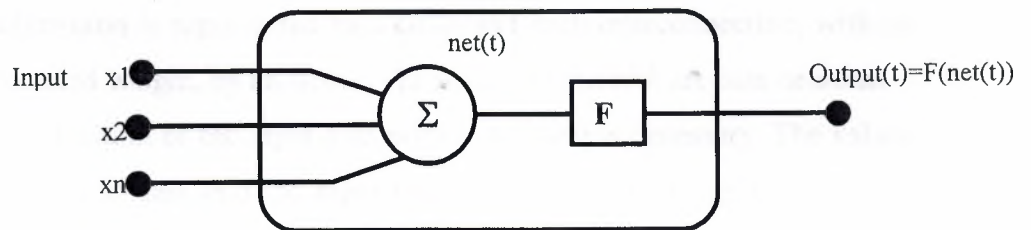


Figure 2.9 Artificial Neuron

$$out = F(net) \quad (2.2)$$

$$F(net) = \frac{1}{(1 + \exp(-net))} \quad (2.3)$$

The derivative of the sigmoid function can be obtained as follows:

$$\begin{aligned} \frac{\partial F(net)}{\partial net} &= \frac{\exp(-net)}{(1 + \exp(-net))^2} \\ &= \left(\frac{1}{1 + \exp(-net)} \right) \left(\frac{\exp(-net)}{1 + \exp(-net)} \right) \\ &= out(1 - out) \\ &= F(net)[1 - F(net)] \end{aligned} \quad (2.4)$$

Any other function that is differentiable everywhere can be used in the back propagation algorithm. For example, linear functions with adjustable gain, relay functions with threshold characteristics, linear threshold characteristic functions and Sigmoid functions for different values of gain, are all common activation functions that can be used.

ii- Feed Forward Calculations

Figure 2.10 shows the most common configuration of a back propagation neural network. This is the simple three layer back propagation model. Each neuron is represented by a circle and each interconnection, with its associated weight, by an arrow. The neurons labeled b are bias neurons. Normalization of the input data prior to training is necessary. The values of the input data into the input layer must be in the range (0 – 1). The stages of the feed forward calculations can be described according to the layers. The suffixes i , h , and j are used for *input*, *hidden* and *output* respectively.

ii.1 Input Layer (i)

Figure 2.11 shows a neuron in the input layer. The output of each input layer neuron is exactly equal to the normalized input.

$$\text{Input - Layer Output} = O_i = I_i \quad (2.5)$$

ii.2 Hidden Layer (h)

Figure 2.12 describes a neuron in the hidden layer. The signal presented to a neuron in the hidden layer is equal to the sum of all outputs of the input layer neurons multiplied by their associated connection weights, as in equation (2.6).

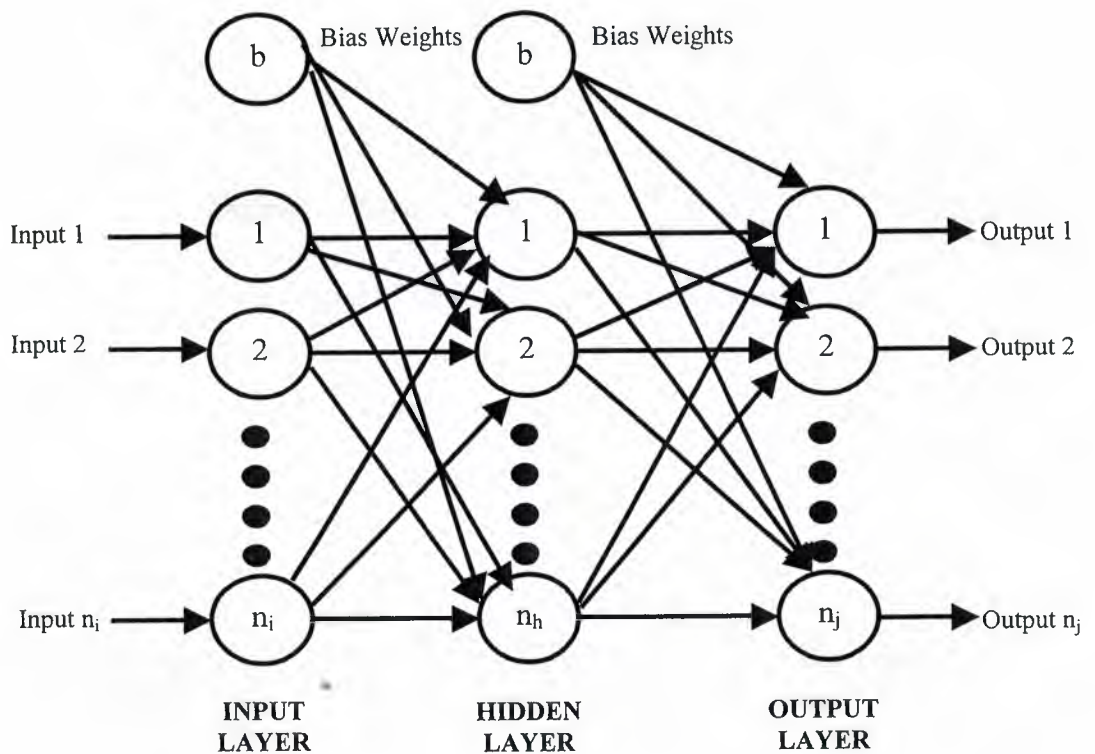


Figure 2.10 Back Propagation Network Structure

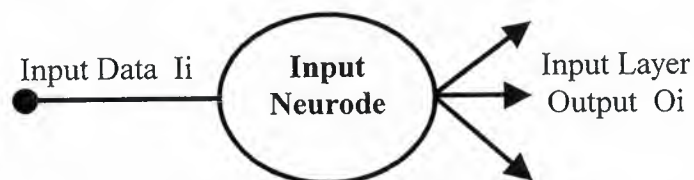


Figure 2.11 An Input Layer Neuron

$$\text{Hidden - Layer Input}_h = I_h = \sum_i W_{hi} O_i \quad (2.6)$$

Each output of a hidden neuron is calculated using the sigmoid function. This is described in equation (2.7).

$$\text{Hidden - Layer Output}_h = O_h = \frac{1}{1 + \exp(-I_h)} \quad (2.7)$$

ii.3 Output Layer (j)

Figure 2.13 describes a neuron in the output layer. The signal presented to a neuron in the output layer is equal to the sum of all outputs of the hidden layer neurons multiplied by their associated weights plus the bias weights at each neuron, as in equation (2.8).

$$\text{Output - Layer Input}_j = I_j = \sum_h W_{jh} O_h \quad (2.8)$$

Each output of an output neuron is calculated using the sigmoid function in a similar manner as in the hidden layer. This is described in equation (2.9).

$$\text{Output - Layer Output}_j = O_j = \frac{1}{1 + \exp(-I_j)} \quad (2.9)$$

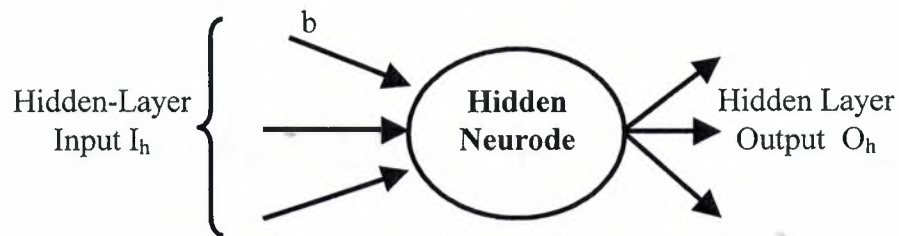


Figure 2.12 A Hidden Layer Neuron

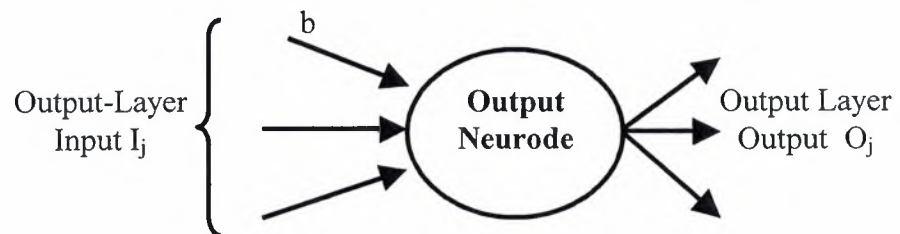


Figure 2.13 An Output Layer Neuron

The set of calculations that has been described so far in the feed forward calculations, can be carried out during the training phase as well as during the testing / running phase.

iii- Error Back Propagation Calculations

The error back propagation calculations are applied only during the training of the neural network. Vital elements in these calculations are described below. These include, the error signal, some essential parameters and weight adjustment.

iii.1 Signal Error

During the network training, the feed forward output state calculation is combined with backward error propagation and weight adjustment calculations that represents the network's learning. Central to the concept of training a neural network is the definition of network *error*. Rumelhart and McClelland define an error term that depends on the difference between the output neuron is supposed to have, called the target value T_j , and the value it actually has as a result of the feed forward calculations, O_j . The error term represents a measure of how well a network is training on a particular training set.

Equation (10) presents the definitions for the error. The subscript p denotes what the value is for a given pattern.

$$E_p = \sum_{j=1}^{n_j} (T_{pj} - O_{pj})^2 \quad (2.10)$$

The aim of the training process is to minimize this error over all training patterns. From equation (2.9), it can be seen that the output of a neuron in the output layer is a function of its input, or $O_j = f(I_j)$. The first derivative of this function, $f'(I_j)$ is an important element in error back propagation. For output layer neurons, a quantity called the error signal

is represented by Δ_j , which is defined in equation (2.11) and thus equation (2.12).

$$\Delta_j = f'(I_j)(T_j - O_j) \quad (2.11)$$

$$= (T_j - O_j)O_j(1 - O_j) \quad (2.12)$$

This error value is propagated back and appropriate weight adjustments are performed. This is done by accumulating the Δ 's for each neuron for the entire training set, add them, and propagate back the error based on the grand total Δ . This called *batch (epoch) training*.

iii.2 Essential Parameters

There are two essential parameters that do affect the learning capability of the neural network. First, the *learning coefficient* η that defines the learning 'power' of a neural network. Second, the *momentum factor* α , which defines the speed at which, the neural network learns. This can be adjusted to a certain value in order to prevent the neural network from getting caught in what is called *local energy minima*. Both rates can have a value between 0 and 1.

iii.3 Weight Adjustment

Each weight has to be set to an initial value. Random initialization is usually performed. Weight adjustment is performed in stages. Starting at the end of the feed forward phase, and going backward to the inputs of the hidden layer.

iii.3.a Output-Layer Weights Update

The weights that feed the output layer (W_{jh}) are updated using equation (2.13). This also includes the bias weights at the output layer neurons. However, in order to avoid the risk of the neural

network getting caught in local minima, the momentum term can be added as in equation (14).

$$W_{jh}(new) = W_{jh}(old) + \eta \Delta_j O_h \quad (2.13)$$

$$W_{jh}(new) = W_{jh}(old) + \eta \Delta_j O_h + \alpha [\delta W_{jh}(old)] \quad (2.14)$$

Where $\delta W_{jh}(old)$ stands for the previous weight change.

iii.3.b Hidden-Layer Weights Update

The error term for an output layer is defined in equation (2.12). For the hidden layer, it is not as simple to figure out a definition for the error term. However, a definition by Rumelhart and McClelland describes the error term for a hidden neuron as in equation (2.15) and, subsequently, in equation (2.16).

$$\Delta_h = f'(I_h) \sum_{j=0}^{n_j} W_{jh} \Delta_j \quad (2.15)$$

$$\Delta_h = O_h (1 - O_h) \sum_{j=0}^{n_j} W_{jh} \Delta_j \quad (2.16)$$

The weight adjustments for the connections feeding the hidden layer from the input layer are now calculated in a similar manner to those feeding the output layer. These adjustments are calculated using equation (2.17).

$$W_{hi}(new) = W_{hi}(old) + \eta \Delta_h O_i + \alpha [\delta W_{hi}(old)] \quad (2.17)$$

The bias weights at the hidden layer neurons are updated, similarly, using equation (2.17).

2.3 Why Back Propagation Neural Networks?

Back Propagation Learning Algorithm is the most popular algorithm of Neural Networks and Supervised Learning. The reasons of choosing Back Propagation Learning Algorithm for real life application are:

- It is the most common type that used in real life applications so it gives us a good opportunity to compare experiments.
- Software Implementation of Back Propagation Learning Algorithm is the easiest and the most efficient way in General Artificial Neural Networks.
- It has lower memory requirements.
- It usually reaches an acceptable error level quite quickly.
- It has variety of program termination conditions such as reaching desired error level or reaching maximum epoch (iterations) number.

2.4 Summary

In this chapter, the learning methods of neural networks and their learners. Supervised and Unsupervised Learning methods were described in details.

In Unsupervised Learning, Kohonen's Learning, Competitive Learning and Adaptive Resonance Theory were described.

In Supervised Learning, A Perceptron, Hopfield, Hamming and Back Propagation Learning were described.

The reasons of choosing Back Propagation Algorithm for real application were explained in section 2.3.

CHAPTER 3

IMAGE PROCESSING FUNDAMENTALS

3.1 Overview

After describing neural networks and reasons of using back propagation learning algorithm, now the image processing fundamentals will be described. After brief introduction about file formats; Image components will be explained.

3.2 What is Image Processing?

Image processing is the science of manipulating a picture [7]. It covers broad scope techniques that are present in numerous applications. These techniques can enhance or distort an image, highlight certain features of an image, create a new image from portions of other images, restore an image that has been degraded during or after the image acquisition.

Image processing is often confused with computer graphics. Computer graphics and image processing are companion technologies. Although there are many common concepts between image processing and computer graphics, they are two separate studies. Computer graphics is the generation of synthetic images. Image processing is the manipulation of images that have already been captured or generated. Computer graphics works with 2-dimensional and 3-dimensional objects. Image processing typically deals with but is not limited to 2-dimensional data.

3.3 Image Processing Applications

3.3.1 Science and Space

In earlier years, solely scientists used image processing. The space programs brought us many image processing techniques. These techniques have been a huge factor in our successful exploration of the solar system.

3.3.2 Movies

Although Hollywood has always experimented with using computers to generate special effects and touch-up frames, the use of computers in filmmaking has a dramatically increased in the last few years. Computers transmute one image into another, remove unwanted objects from a frame, and create new frames by compositing parts other frames.

3.3.3 Medical Industry

The medical industry has long been a user of image processing. There are various well known imaging technologies used including X-rays and ultrasound. Computed tomography, or computer-aided tomography (CT or CAT), wasn't widely used until the 1980's.

3.3.4 Machine Vision

Machine vision is a growing technology that uses both image processing and image analysis. Machine vision is an industrial technology in which acquired image data is processed and control in manufacturing environments. Its early uses were for automated inspection and assembly lines, particularly automobile assembly lines. The very first successful application of image processing in an industrial automation was defect-inspection machine for printed-circuit boards.

3.3.5 Law Enforcement

Law enforcement agencies have been using image processing for quite a while. The FBI used image-processing techniques to enhance and study pertinent features in hundreds of film frames of John F. Kennedy assassination. Police departments are using computers to enhance poor quality fingerprints to make them easier to inspect.

Police departments are also using image-editing software. They can alter mug shots by changing hair color, hair length or adding facial hair. This helps victims identify suspects whose appearances may have changed since the original photography was taken.

3.4 How Your Computer Stores Images?

When you use an endoscopic capture system to make images, or when you scan an X-Ray into your computer, the computer takes the information and translates it into pixels, tiny blocks of information that make up an image. The type of image you get, depends on how much information each pixel can hold. This is commonly referred to as color depth.

- A 1-bit image, for example, can only contain 2 colors, black and white (Figure 3.1). This format is used for line art that needs to be sharply defined.
- A 4-bit image contains 16 colors, usually a default palette (Figure 3.2).

- An 8-bit image contains 256 colors, and is known as indexed color (Figure 3.3). Good-quality grayscale images, and many internet graphics use indexed color because it retains the clarity of the image, albeit in a compressed format.
- 16-bit color images contain literally millions of colors. This format is commonly called high color. Most newer machines, however, can support 24-bit or 32-bit color, or true color. True color is the overall best format you can use for imaging, because it retains the clarity of the original image without the need for compression.



Figure 3.1 A 1-Bit Image



Figure 3.2 A 4-Bit Image



Figure 3.3 An 8-Bit Image



Figure 3.4 A 16-Bit Image

3.5 Image File Formats

There are at least as many image formats as there are graphics programs. Most graphics applications, in addition to having their own proprietary formats, will open a number of common formats. These formats are [8]:

3.5.1 BMP

Bitmaps are the basic Windows image format, and the easiest overall to work with, since just about every graphics application will open and edit them. The advantage of bitmaps is that they save data in full-color, uncompressed format, so you get excellent image reproduction. The disadvantage is that the uncompressed data means a hefty picture size (almost 1 MB for a 640x480 image). BMP is probably the best format to save a permanent copy of your image in.

3.5.2 JPEG

The JPEG format was developed by the Joint Photographic Experts Group (hence the name) as a way of getting true-color images on the Web in a form that wouldn't take forever to download. Jpegs do this by compressing data, using a set of calculations to discard data not essential to the display of the image. Because .JPGs discard data, they are referred to as lossy. While lossy compression doesn't really show up on screen displays, it does make a big difference in the printed image. And images that are .jpg-compressed over and over again (like many web images) degrade rapidly into a blocky distortion. JPG compression is a sliding scale, you can compress file size by anywhere from 5-80%. One of the best uses of .jpg is for e-mailing images, especially since many mail servers put a limit on the size of e-mail attachments you can download.

3.5.3 GIF

Graphics Interchange Format (GIF) is the file format commonly used to display indexed-color (256-color) graphics and images in hypertext markup language (HTML) documents over the World Wide Web and other online services. GIF is

an LZW-compressed format designed to minimize file size and electronic transfer time. GIF format preserves transparency in indexed-color images; however, it does not support alpha channels.

3.5.4 TIF

TIF (tagged image format) is unique in several ways. It's most preferred by print houses because of its versatility and ease of conversion between platforms. First, it allows you to save images in uncompressed or compressed format. The type of compression that .tif uses is called LZW compression. LZW is lossless, meaning you won't lose any data while compressing images. TIFs also let you choose between RGB color and CMYK color.

3.5.5 RAW

Raw format is a flexible file format for transferring images between applications and computer platforms. This format supports CMYK, RGB, and grayscale images with alpha channels, and multichannel and Lab images without alpha channels.

Raw format consists of a stream of bytes describing the color information in the image. Each pixel is described in binary format, with 0 representing black and 255 white (for images with 16-bit channels, the white value is 65535).

3.5.6 PNG

Developed as a patent-free alternative to GIF, Portable Network Graphics (PNG) format is used for lossless compression and for display of images on the World Wide Web. Unlike GIF, PNG supports 24-bit images and produces background transparency without jagged edges; however, some Web browsers do not support PNG images. PNG format supports RGB, indexed-color, grayscale, and Bitmap-mode images without alpha channels. PNG preserves transparency in grayscale and RGB images.

3.6 Image Segmentation

In image segmentation, given a homogeneity criterion, the image must be partitioned into regions within which the criterion is satisfied [9]. The border between two regions corresponds to a discontinuity, i.e., to an edge. However, not all edges are meaningful. Local variations due to noise often can yield significant discontinuities. A trade-off exists between over segmentation, partition into too many regions, and under segmentation, in which case larger regions are obtained at the expense of possible erroneous fusions.

A segmentation based solely on a simply homogeneity criterion, like constant gray levels, cannot provide the decomposition of an image into regions which correspond to parts of an object in the physical world. To obtain a meaningful segmentation the higher level descriptions of the objects and often also of the relations among them must be taken into account. Image segmentations using simple homogeneity criteria can only provide the partition of the image into “*puzzle pieces*” from which the objects have to be assembled. A complete segmentation system is complex and makes use of many heuristics. To reduce over segmentation, in the absence of context dependent information, probabilistic models are required to guide the fusion process.

The difficulty of segmentation is an aspect of the local/global duality problem. A region is declared homogenous by analyzing small local neighborhoods. The larger these neighborhoods, the more reliable are the extracted spatial statistics given that the data in the neighborhood is indeed homogenous. On the other hand, using a larger neighborhood increases the chances of analyzing non-homogenous data under the assumption of homogeneity.

To avoid the problem of local/global duality often edge information (local) and homogeneity information (global) are combined during image segmentation. Haddon and Boyce refined concurrence matrix based segmentation by incorporating boundary information into a relaxation procedure.

In most segmentation methods, the edge (discontinuity) information is based on local analysis and its errors affect the fusion of adjacent homogenous regions.

3.6.1 Image Segmentation Methods

Image segmentation methods can be classified as follows [10]:

3.6.1.1 Edge Base Approaches: use edge detection operators such as Roberts, Prewitt, Sobel, and Laplacian. Resulting regions may not be connected; hence edges need to be joined.

3.6.1.2 Region Base Approaches: based on similarity of regional image data. Some of the more widely used approaches in this category are:

- Thresholding, Clustering
- Region growing
- Splitting and merging

Image segmentation algorithm that used in IBIS, will be described in chapter 4.

3.7 Image Compression

Compressing an image is significantly different than compressing raw binary data. Of course, general-purpose compression programs can be used to compress images, but the result is less than optimal. This is because images have certain statistical properties that can be exploited by encoders specifically designed for them. Also, some of the finer details in the image can be sacrificed for the sake of saving a little more bandwidth or storage space. This also means that lossy compression techniques can be used in this area [11].

Lossless compression involves with compressing data which, when decompressed, will be an exact replica of the original data. This is the case when binary data such as executables, documents etc. are compressed. They need to be exactly reproduced when decompressed. On the other hand, images need not be reproduced 'exactly'. An approximation of the original image is enough for most purposes, as long as the error between the original and the compressed image is tolerable.

Many file formats use compression to reduce the file size of bitmap images. *Lossless* techniques compress the file without removing image detail or color information; *lossy* techniques remove detail. The following are commonly used compression techniques:

3.7.1 Types of Compressions

3.7.1.1 Lossless Compression: Lossless compression uncompresses an image so its quality matches the original source--nothing is lost. Although lossless compression sounds ideal, it doesn't provide much compression and files remain quite large. For this

reason, lossless compression is used mainly where detail is extremely important, as it is when planning to make large prints. Some digital cameras in the form of TIFF offer lossless compression and RAW file formats.

3.7.1.2 Lossy Compression: Because lossless compression isn't practical in many cases, all popular digital cameras offer a lossy compression (rhymes with "*bossy*"). This process degrades images to some degree and the more they're compressed, the more degraded they become. In many situations, such as posting images on the Web or making small to medium sized prints, the image degradation isn't obvious. However, if you enlarge an image enough, it will show.

3.7.2 Compression Formats

3.7.2.1 RLE (Run Length Encoding)

Lossless compression; supported by some common Windows file formats.

3.7.2.2 LZW (Lemple-Zif-Welch)

Lossless compression; supported by TIFF, PDF, GIF, and PostScript language file formats. Most useful for images with large areas of single color.

3.7.2.3 JPEG (Joint Photographic Experts Group)

Lossy compression; supported by JPEG, TIFF, PDF, and PostScript language file formats. Recommended for continuous-tone images, such as photographs.

3.7.2.4 CCITT

A family of lossless compression techniques for black-and-white images; supported by the PDF and PostScript language file formats. (CCITT is an abbreviation for the French spelling of International Telegraph and Telekeyed Consultive Committee.)

3.7.2.5 ZIP

Lossless compression; supported by PDF and TIFF file formats. Like LZW, ZIP compression is most effective for images that contain large areas of single color.

3.7.2.6 Fractal Image Format (FIF)

Fractal compression works by using a variety of methods to identify features within an image and then breaking down the image into a mathematically modeled series of repeating shapes and patterns. Fractal compression is very efficient, achieving compression ratios of up to 250:1; typical fractal compression ratios will range between 20:1 and 100:1. Images can be magnified or reduced, because the compression process allows the modeled images to be resolution-independent. When a fractal-encoded image is converted to a pixel image, it can be enlarged or reduced to any desired size with minimal loss of image quality.

However, published reviews of fractal compression software indicate that there is probably a practical limit to how much a fractal-encoded image can be enlarged before there is a significant loss of image quality; perhaps up to 300% of the original size.

3.7.2.7 Wavelet Image Files (WIF)

Method called "Wavelet Transform" made at the Houston Advanced Research Center (HARC™) by applying the complex wavelet algorithms to a digital image, the "compression engine" software is able to represent the image as a mathematical expression. The result is a WIF files are created using the groundbreaking advances in the mathematical compression compressed WIF file that can be 300 times smaller than the original image file size while still maintaining high image quality.

Image compression algorithm that used in IBIS, will be described in chapter 4.

3.8 Edge Detection

In computer vision, edge detection is a process that attempts to capture significant properties of objects in the image [12]. These properties include discontinuities in the photometrical, geometrical and physical characteristics of objects. Such information gives rise to variations in the gray level image; the most commonly used variations are discontinuities (step edges), local extreme (line edges), and 2D features formed where at least two edges meet (junctions).

The purpose of the edge detection is to localize these variations and to identify the physical phenomena that produce them. Edge detection must be efficient and reliable because the validity, efficiency and possibility of the completion of subsequent processing stages rely on it. To fulfill this requirement, edge detection provides all significant information about the image. For this purpose, image derivatives are computed. However, differentiation of an image is an ill-posed problem; image derivatives are sensitive to various sources of noise, i.e., electronic, semantic, and discretization/quantification effects. To regularize the differentiation, the image must be smoothed. However, there are undesirable effects associated with smoothing, i.e., loss of information and displacement of prominent structures in the image plane. Furthermore, the properties of commonly used differentiation operators are different and therefore they generate different edges. It is difficult to design a general edge detection algorithm, which performs well in many contexts and captures the requirements of subsequent processing stages.

3.8.1 Edge Definition

Physical edges provide important visual information since they correspond to discontinuities in the physical, photometrical and geometrical properties of scene objects. The principal physical edges correspond to significant variations in the reflectance, illumination, orientation, and depth of scene surfaces. Since image intensity is often proportional to scene radiance, physical edges are represented in the image by changes in the intensity function. The most common types of intensity variations are steps, lines and junctions.

Steps are by far the most common type of edge encountered. This type of edge results from various phenomena: for example when one object hides another, or when there is a shadow on a surface. It generally occurs between two regions having almost constant, but different, gray levels.

The step edge is the point at which the gray level discontinuity occurs. In real images, step edges are localized at the inflection points of the image. In fact, the image formation process involves the convolution of the camera point-spread function with the edge profile corrupted by noise that produces a smooth function (Figure 3.5a & 3.5b). Consequently, step edges are localized as positive maxima or negative minima of the first-order derivative (Figure 3.5c) or as zero-

crossings of the second-order derivative (Figure 3.5d). In 2D, the first derivative is defined by the gradient operator and the second derivative is approached by the Laplacian or by the second derivative along the gradient direction. This step edge definition does not include the spatial distribution of edges. It is more realistic to consider a step edge as a combination of several inflection points. The most commonly used edge model is the double step edge (two inflection points in the vicinity of each other). There are two types of double edges: the pulse (Figure 3.5e) and the staircase (Figure 3.5f).

3.8.2 Properties of Edge Detectors

An edge detector accepts discrete, digitized images as input and produces an edge map as output. The edge map of some detectors includes explicit information about the position and strength of edges, their orientation, and the scale. The example in Figure 3.6 includes position information only.

During the history of image processing, a variety of edge detectors have been devised which differ in their purpose (the photometrical and geometrical properties of the edge) and in their mathematical and algorithmical properties. From the point view of integration of an edge detector into a computer vision system there are two classes of detectors. The first includes detectors that do not use a priori knowledge about the scene and the edge detected. This class of “autonomous” detectors is influenced neither by other components of the vision

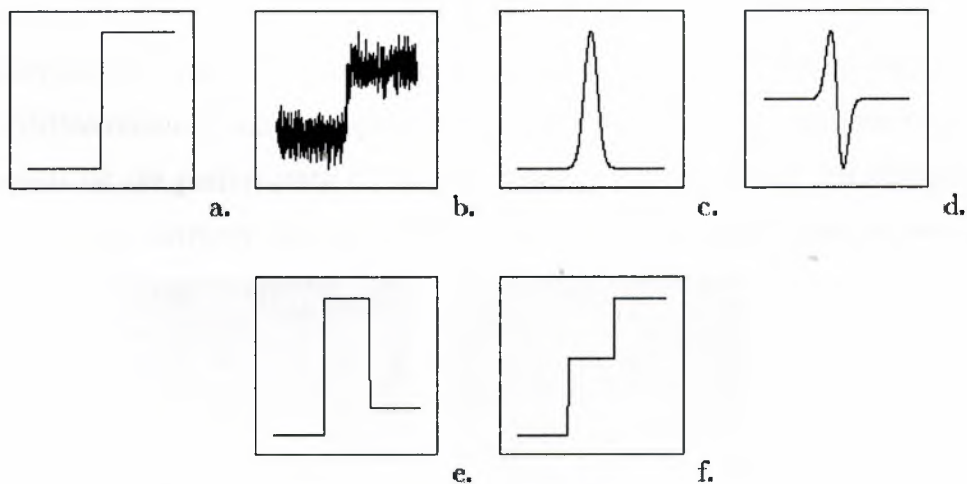


Figure 3.5 Step Edge Profile

a) Ideal Step Edge b) Smoothed step edge corrupted by noise c) and d) First and Second-order derivatives of the smoothed step edge (noise-free) e) Pulse f) Staircase

system nor by the contextual information. These detectors are flexible in the sense that they are not limited to specific images. However, they are based on local processing; the process of labeling an edge is based only on its neighboring pixels. The second class of detectors is contextual, in the sense that they are guided by the results of other components of the system or by priori knowledge about the edge or the structure of the sense. It follows that they perform only in a precise context. Few contextual detectors have been proposed. If we consider the knowledge used by the detectors, it is clear that autonomous detectors are appropriate for general-purpose vision systems. However, contextual detectors are adapted to specific applications where processed images always include the same objects.

Conceptually, the most commonly proposed schemes for edge detection (both autonomous and contextual detectors) include three operations: differentiation, smoothing and labeling. Differentiation consists in evaluating the desired derivatives of the image. Smoothing consists in reducing noise in the image and regularizing the numerical differentiation. Labeling involves localizing edges and increasing the signal-to-noise ratio of the edge image by suppressing false edges. Labeling is the last operation to be run. However, as will be shown below, the order in which differentiation and smoothing are run depends on their properties. Smoothing and differentiation of an image are realized only by filtering the image with the differentiation of the smoothing filter. The performance of these three operations is related. In fact, smoothing regularizes the differentiation and the specification of the false edge suppression step depends on the performance of the two other operations. If the smoothing step reduces noise without loss of information, false edge suppression is easy to



Figure 3.6 a) Original Image

b) Position Information Provided by Edge Detection

accomplish. The specification of an edge detector in terms of three operations is incomplete. In fact, an edge detector includes neither the precise context in which it can be successfully used nor the scale computation. It is necessary to define a methodology of edge detection to make explicit how to choose the scale (multi-scale) and how to select an edge detector (multi-detector) for a target application.

3.9 Image Restoration

Pre-processing methods that aim to suppressing image degradation using knowledge about its nature are called image restoration.

the image restoration problem attempts to recover image that has been degraded by the limited resolution of the sensor as well as by the presence of noise [13]. The resolution of images obtained by the satellite sensors is degraded by sources such as: optical diffraction, detector size and electronic filtering. As a consequence, the effective resolution is, in general, worse than the nominal resolution, that corresponds to the detector projection on the ground and does not take into consideration the sensor imperfections. Through restoration techniques, it is possible to improve image resolution up to a certain level.

3.9.1 Image Restoration Techniques

Image Restoration techniques can be classified into two groups:

3.9.1.1 Deterministic Methods

Deterministic Methods are applicable to images with little noise and a known degradation function.

3.9.1.2 Stochastic Techniques

Stochastic Techniques try to find the best restoration according to particular stochastic criterion, e.g., least squares method.

In most practical cases, there is insufficient knowledge about the degradation, and it must be estimated and modeled. The estimation can be classified into two groups according to the information available *a priori* and *a posteriori*. If degradation type and/or parameters need to be estimated, this step is the most

crucial one, being responsible for image restoration success or failure. It is also the most difficult part of image restoration.

3.9.2 A Priori Knowledge

- A priori knowledge about degradation is either known in advance or can be obtained before restoration.
- E.g., if it is clear in advance that the image was degraded by relative motion of an object with respect to the sensor then the modeling only involves the speed and direction of the motion.
- E.g., parameters of a capturing device such as a TV camera or digitizer, whose degradation remains unchanged over a period of time and can be modeled by studying a known sample image and its degraded version.

3.9.3 A Posteriori Knowledge

- A posteriori knowledge is obtained by analyzing the degraded image.
- A typical example is to find some interest points in the image (e.g. corners, straight lines) and guess how they looked before degradation.
- Another possibility is to use spectral characteristics of the regions in the image that are relatively homogeneous.

3.10 Image Enhancement

Image enhancement techniques are used to emphasize and sharpen image features for display and analysis [14]. *Image enhancement* is the process of applying these techniques to facilitate the development of a solution to a computer-imaging problem. Consequently, the enhancement methods are application specific and are often developed empirically. Figure 3.7 illustrates the importance of the application by the feedback loop from the output image back to the start of the enhancement process and models the experimental nature of the development. The range of applications includes using enhancement techniques as pre-processing steps to ease the next processing step or as post-processing steps to improve the visual perception of a processed image, or image enhancement may be an end in itself. Enhancement methods operate in the spatial domain by manipulating the pixel data or in the frequency domain by modifying the spectral components (Figure 3.8). Some enhancement algorithms use both the spatial and frequency domains.

The type of techniques includes *point operations*, where each pixel is modified according to a particular equation that is not dependent on other pixel values; *mask operations*, where each pixel is modified according to the values of the pixel's neighbors (using convolution masks); or *global operations*, where all the pixel values in the image (or sub-image) are taken into consideration. Spatial domain processing methods include all three types, but frequency domain operations, by nature of the frequency (and sequency) transforms, are global operations. Of course, frequency domain operations can become "mask operations," based only on a local neighborhood, by performing the transform on small image blocks instead of the entire image.

Enhancement is used as a preprocessing step in some computer vision applications to ease the vision task, for example, to enhance the edges of an object to facilitate guidance of a robotic gripper. Enhancement is also used as a preprocessing step in applications where human viewing of an image is required before further processing. For example, in one application, high-speed film images had to be correlated with a computer-simulated model of an aircraft. This process was labor intensive because the high-speed film generated many images per second and difficult because of the fact that the images were all dark. This task was made considerably easier by enhancing the images before correlating them to the model, enabling the technician to process many more images in one session.

Image enhancement is used for post processing to generate a visually desirable image. For instance, we may perform image restoration to eliminate image distortion and find that the output image has lost most of its contrast. Here, we can apply some basic image enhancement methods to restore the image contrast. Alternately, after a compressed image has been restored to its "original" state (decompressed), some post processing enhancement may significantly improve the look of the image. For example, the standard JPEG compression algorithm may generate an image with undesirable "blocky" artifacts, and post processing it with a smoothing filter (low pass or mean) will improve the appearance.

Overall, image enhancement methods are used to make images look better. What works for one application may not be suitable for another application, so the development of

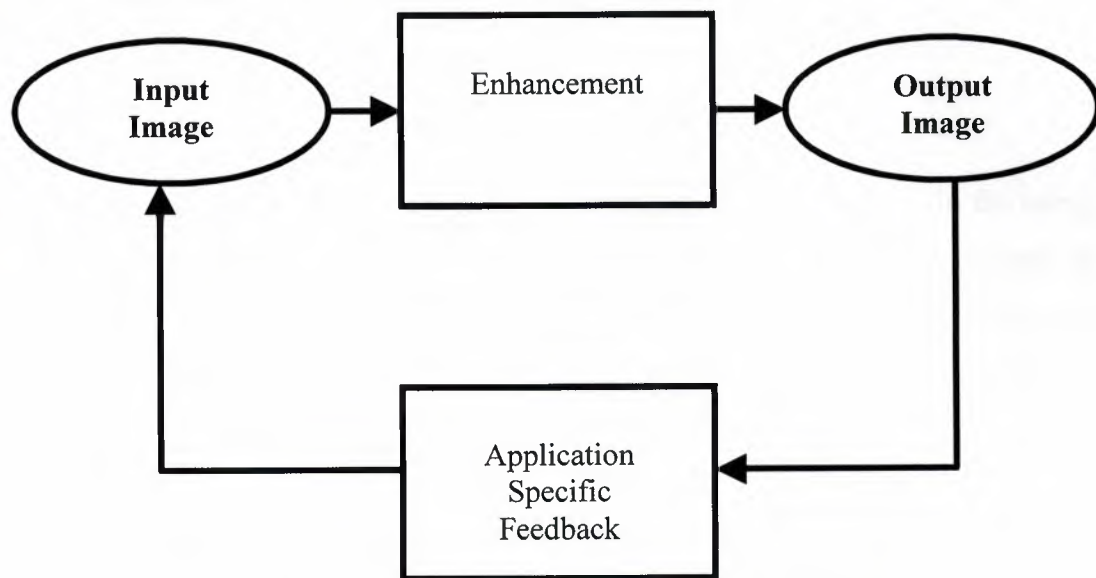


Figure 3.7 The Image Enhancement Process

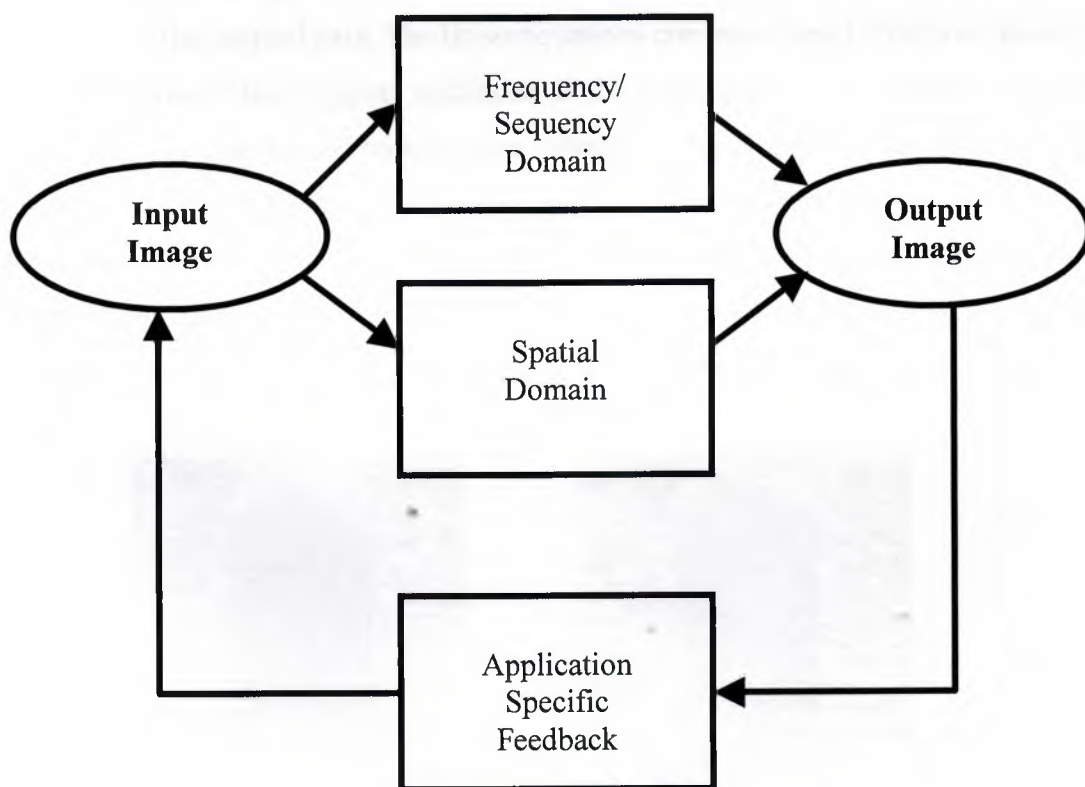


Figure 3.8 Image Enhancement

enhancement methods require problem domain knowledge, as well as image enhancement expertise. Assessment of the success of an image enhancement algorithm

is often "in the eye of the beholder," so image enhancement is as much an art as it is a science.

3.10.1 Gray-Scale Modification

Gray-scale modification (also called gray-level scaling) methods belong in the category of point operations and function by changing the pixel's (gray-level) values by a mapping equation. The *mapping equation* is typically linear (nonlinear equations can be modeled by piecewise linear models) and maps the original gray-level values to other, specified values. Typical applications include contrast enhancement and feature enhancement.

The primary operations applied to the gray scale of an image are to compress or stretch it. We typically compress gray-level ranges that are of little interest to us and stretch the gray-level ranges where we desire more information. This is illustrated in Figure 4.2-1a, where the original image data are shown on the horizontal axis and the modified values are shown on the vertical axis. The linear equations corresponding to the lines shown on the graph represent the mapping equations. If the slope of the line is between zero and one, this is called *gray-level compression*, whereas if the slope is greater than one, it is called *gray-level stretching*. The original and modified images are shown in Figures 3.9a-b and 3.10a-b where we can see that stretching this range exposed previously hidden visual information. In some cases we may want to stretch a specific range of gray levels, while clipping the values at the low and high ends.



Figure 3.9a
Original Image

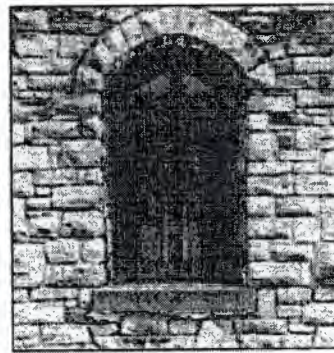


Figure 3.9 b
Image after modification



Figure 3.10a
Original Image



Figure 3.10b
Image after modification



3.10 Summary

In this chapter, fundamentals of image processing and necessary information are described.

First, image processing and its applications were explained. Then image segmentation, compression, restoration, enhancement and edge detection that are the main components of image processing, were explained.

CHAPTER 4

INTELLIGENT BANKNOTE IDENTIFICATION SYSTEM (IBIS)

4.1 Overview

In this chapter, all steps of Intelligent Banknote Identification System (IBIS) will be described. From getting image to the actual outputs of Intelligent Banknote Identification System (IBIS) will be explained in two parts: Image Processing and Neural Networks.

4.2 IBIS

Intelligent Banknote Identification System is a real life application of pattern recognition developed for recognizing Turkish banknotes, using neural networks and image processing. Also IBIS shows that the average pixel / node approach is an efficient approach for object or pattern recognition. Now, let us see the details of methods used in IBIS.

4.3 Image Processing

In IBIS, image processing is applied in two phases of system. Training is the first phase and the running is the second phase. Image processing is used in both phases of system. In training; image processing is used for data preparation for training, and in running; image processing is used for data preparation for testing. Both in training and testing phases, data preparation methods and algorithms are same, only difference is; in training, data preparation program is independent program that prepares inputs of neural network, but in running, data preparation program is dependent, that used to prepare inputs of neural networks in testing program.

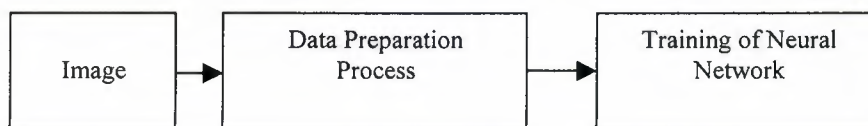


Figure 4.1 - Training Phase of IBIS

4.3.1 Preparing Image for Program

Image Processing part starts with getting image into the computer-using scanner. Firstly, the image saved into the BMP format in its original size 550x256.

Advantages of saving image as BMP format:

- Every graphics application can open and edit this format.
- BMP saves data in full color and in uncompressed format.
- Reproduction of image is lossless.

Disadvantage of saving image as BMP format:

- Image size is too much. (640x480 takes almost 1 MB disk space)

The reason of saving image first in BMP format is to get a lossless backup of original image. After saving image, we have to prepare it to image processing program. As it will be mentioned later again, the program accepts original image with grayscale. After converting image into grayscale, image saved as RAW file.

Advantages and reasons of saving image as RAW format:

- RAW is a flexible file format for transferring images between applications and computer platforms.
- Raw format consists of a stream of bytes describing the color information in the image. Each pixel is described in binary format, with 0 representing black and 255 white.

4.3.2 Data Preparation Program

After preparing image for program, it is now ready to enter processing part. First step of image processing program is trimming the center of original image (550x256) to 256x256.

Calculation of trimming size is as shown below from equation 4.1 – 4.3:

$$X_1 = (Get\ max\ x / 2) - (TargetPx / 2) \quad (4.1)$$

$$X_2 = (Get\ max\ x / 2) + (TargetPy / 2) \quad (4.2)$$

$$T_1[x, y] = [X_1 \rightarrow X_2, 0 \rightarrow \text{Getmax } y] \quad (4.3)$$

where X_1 is the left side, X_2 is the right side, Getmaxx is the pixel size of X-Dimension, Getmaxy is the pixel size of Y-Dimension, TargetPx and TargetPy is the desired pixel sizes of X and Y dimensions, and T is the trimmed (256x256) image.

After trimming original image to 256x256 image, the second step is compressing image into 128x128 pixels. Compression of image is shown below from equation 4.4:

$$C[a, b] = T_1[x, y] \quad \begin{cases} x+ = 2 \\ y+ = 2 \end{cases} \quad (4.4)$$

where C is the compressed image, a and b are the X and Y dimensions of compressed image.

After the compression of image we have 128x128 pixels image. The third step is trimming image again to 100x100 pixels.

$$X_3 = (\text{Getmax } x(C) / 2) - (\text{TargetPx} / 2) \quad (4.5)$$

$$X_4 = (\text{Getmax } x(C) / 2) + (\text{TargetPy} / 2) \quad (4.6)$$

$$T_2[x, y] = [X_3 \rightarrow X_4, 0 \rightarrow \text{Getmax } y(C)] \quad (4.7)$$

Figure 4.3 - Steps of Image Trimming

Neural network part was designed to accept hundred inputs (see p.58) (10x10 image) so now, it is necessary to divide image into 10 segments and apply average pixel per node approach to get hundred inputs for neural network (Figure 4.2). Average Pixel / Node Approach is taking the average of defined segments of image. Average Pixel / Node Approach has been used in IBIS to reduce the time during training.

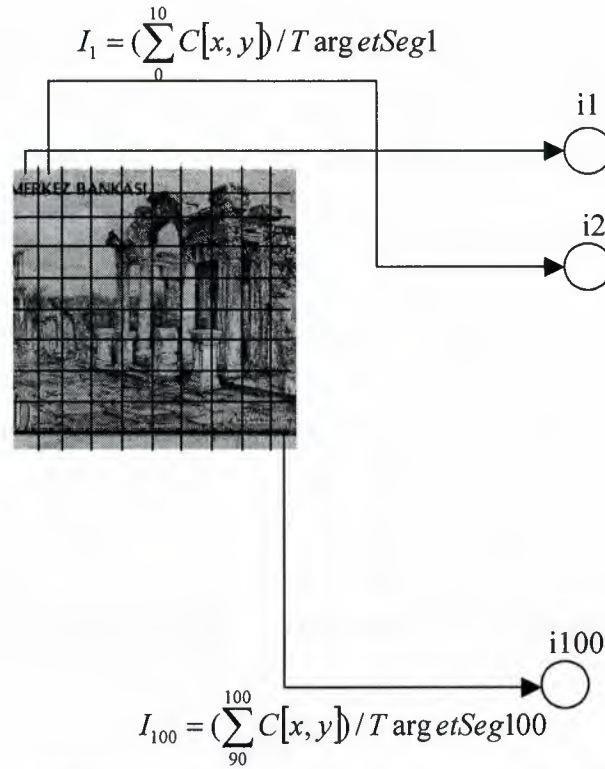


Figure 4.2 Preparing 100 Inputs for Neural Networks
using Average Pixel/Node Approach

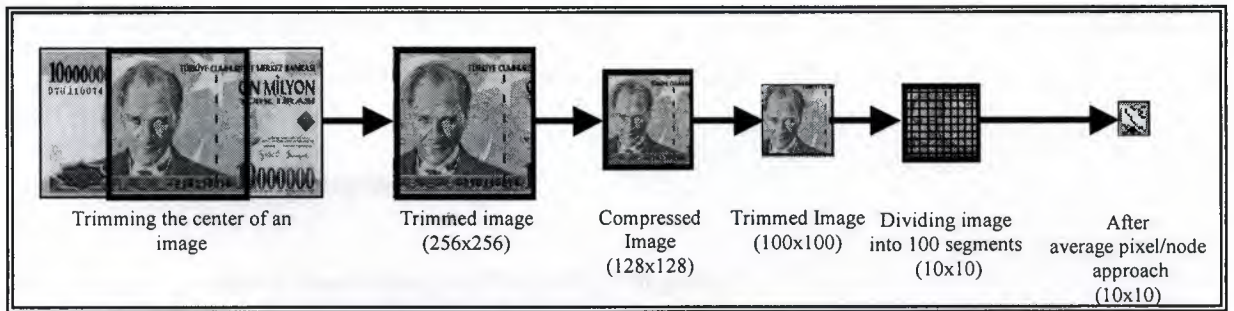


Figure 4.3 – Steps of Image Processing

$$I_1 = \left(\sum_{0}^{10} C[x,y] \right) / TargetSeg1 \quad (4.8)$$

Equation 4.8 shows the averaging first defined segment of image where I is the input of Neural Network, $C[x,y]$ is the pixel value and $TargetSeg1$ is the value of target segmentation.

4.3.2.1 Algorithm for Storing Image into 2-D Array

Storing image into an array is the first step in all image-processing applications. In grayscale images, 2 dimensional array is used to store image. The algorithm of storing image into an array, which is used by IBIS, can be shown in Figure 4.4.

```
void store_image(void)
{
    unsigned char stored_im[x-dim][y-dim]; /**declaration of storage array**/
    FILE *image;
    fopen(image, "image.raw", "rb");          /** Opening Image File **/
    for (i=0; i<x-dim; i++)
    { for (j=0; j<y-dim; j++)
        {
            stored_im[i][j]=fgetc(image);    /** Storing Image into Array **/
        }
    }
}
```

Figure 4.4 Storing Image Algorithm of IBIS

4.3.2.2 Algorithm for Trimming Image

Trimming image is the second step of IBIS in data preparation part. The algorithm that shown in figure 4.5 is used to detect the center of image and to crop it.

```
void trim_image(void)
{
    unsigned char trimed_im[TargetPx][TargetPy];
    for (i=0; i<TargetPx; i++)
    { for (j=X1; j<X2; j++)
        {
            trimed_im[i][j]=stored_im[i][j];    /** Trimming Image **/
        }
    }
}
```

Figure 4.5 Trimming Algorithm of IBIS

4.3.2.3 Algorithm for Compressing Image

Compression image is the third step of IBIS in data preparation part. The algorithm that shown in figure 4.6 is used to compress image.

```
void compress_image(void)
{
    unsigned char comp_im[TargetPx/2][TargetPy/2];
    for (i=0;i<Getmaxx[trimed_im];i+=2)
    {
        for (j=0;j<Getmaxy[trimed_im];j+=2)
        {
            comp_im[i/2][j/2]=trimed_im[i][j];           /** Compressing Image **/
        }
    }
}
```

Figure 4.6 Compression Algorithm of IBIS

4.4 Neural Networks

As shown in the network topology (figure 4.7), neural network has 100 inputs, 1 hidden layer with 30 neurons, and 5 output neurons. In this part, both training and training phases of neural networks will be described.

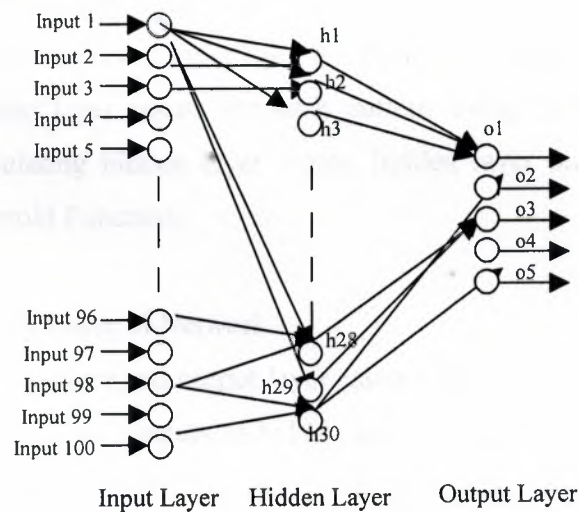


Figure 4.7 – Neural Network Topology

4.4.1 Training of Neural Network

In training part of neural networks; inputs, weights, bias weights, learning rate, momentum, error calculations and weights updates are the main calculations.

4.4.1.1 Inputs of Neural Network

As it mentioned above, neural network gets inputs from image processing program for each pattern. In appendix 1, all inputs can be shown.

As it mentioned in chapter 2, the inputs of input layer is equal to outputs of input layer.

4.4.1.2 First Weights of Network

First weights are assigned randomly from the set of weight numbers from -0.3 to 0.3 . According to the number of neurons in input, hidden and output layers, weights are selected randomly.

There are 100×30 hidden layer weights and 30×5 output layer weights. After each epoch, weights are updated using the equations that were mentioned in chapter 2.

4.4.1.3 Hidden Layer of Network

When outputs of input layer are reached to hidden layer, it starts to calculate hidden layer inputs for each pattern using hidden layer weights. After calculating hidden layer inputs, hidden layer outputs are calculated using Sigmoid Function.

4.4.1.4 Output Layer of Network

Hidden layer and output layer calculations and principles are similar to each other. When outputs of hidden layer are reached to output layer, it starts to calculate output layer inputs for each pattern using output layer weights. After calculating output layer inputs, output layer outputs are calculated using Sigmoid Function.

4.4.1.5 Errors and Back Propagation

After calculating the outputs of output layer for each pattern, it is necessary to compare them with desired outputs.

After comparing output layer output of each pattern with desired output, error for each pattern and Root Mean Square Error is calculated. Error of each pattern is used to update weights and RMS Error is used for stopping condition. RMS Error Level was assigned to 0.005 because this value is sufficient for the required accuracy of the developed system and to keep neural network training time as low as possible. When RMS is reached to desired level, program will save final weights and stop.

These steps will repeat until RMS is less than 0.005 or iterations reach to 10,000. General topology of IBIS can be shown in figure 4.8. All steps of training can be shown in figure 4.9.

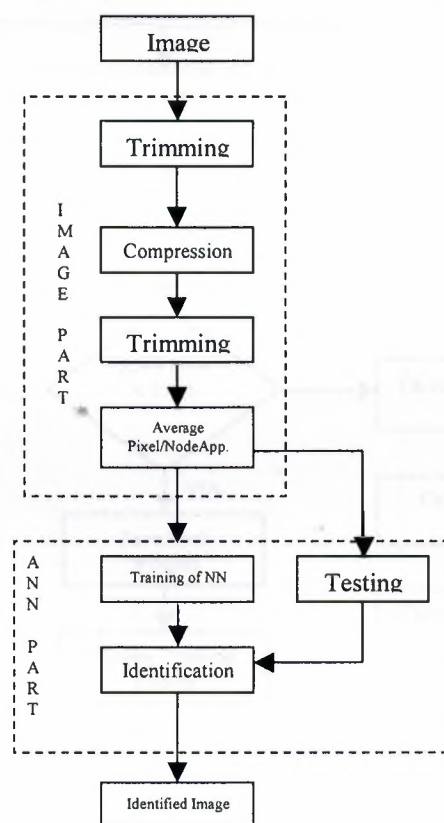
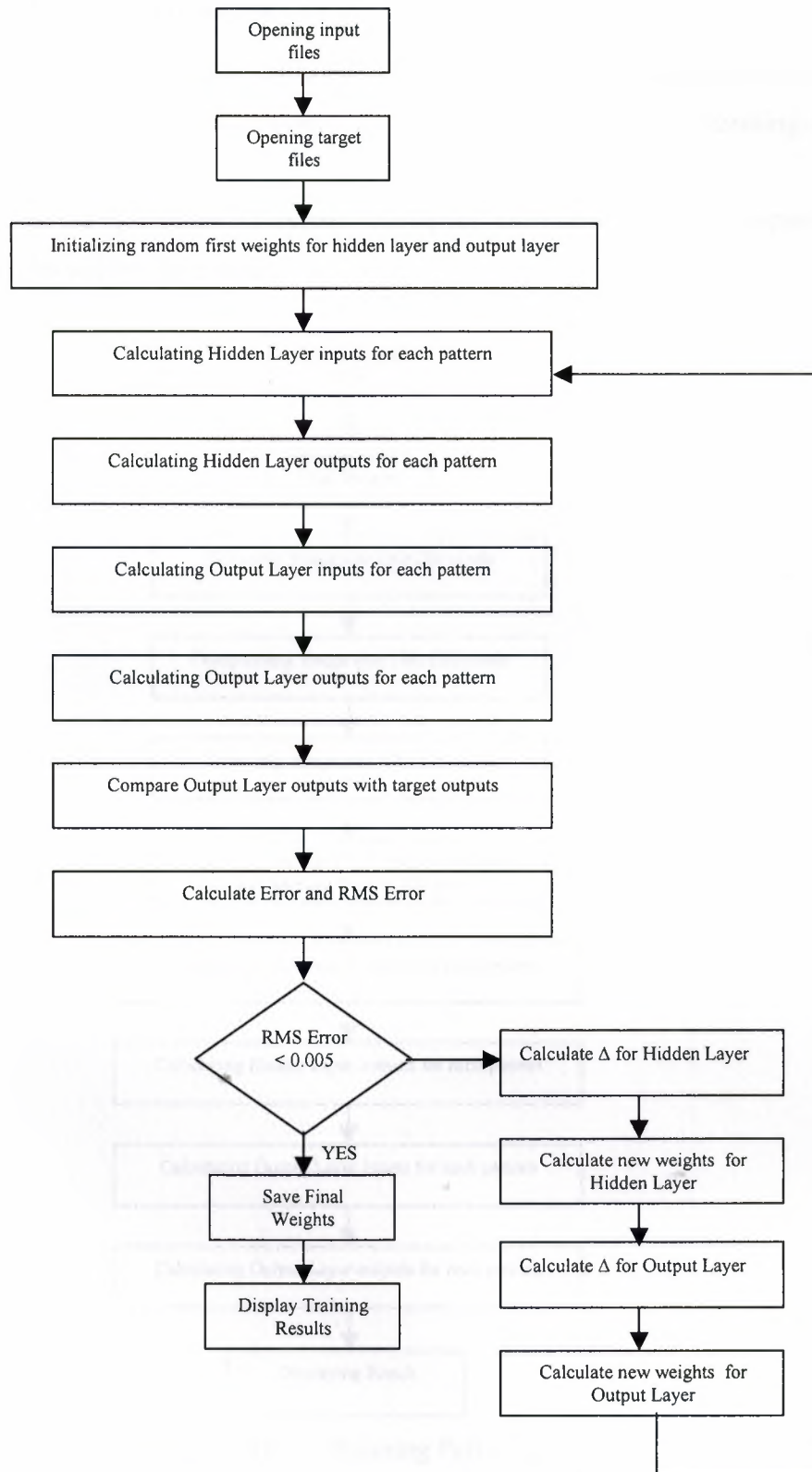


Figure 4.8 General Topology of IBIS

**Figure 4.9 - Flowchart of IBIS Training**

4.4.2 Neural Network Generalization

In running phase, only one iteration could be executed. Running phase uses the saved final weights of training program and it doesn't use any term of momentum, learning rate or error (Figure 4.10).

Running phases is the last phase of system that gives us final results. In chapter 5, all experimental results will be described.

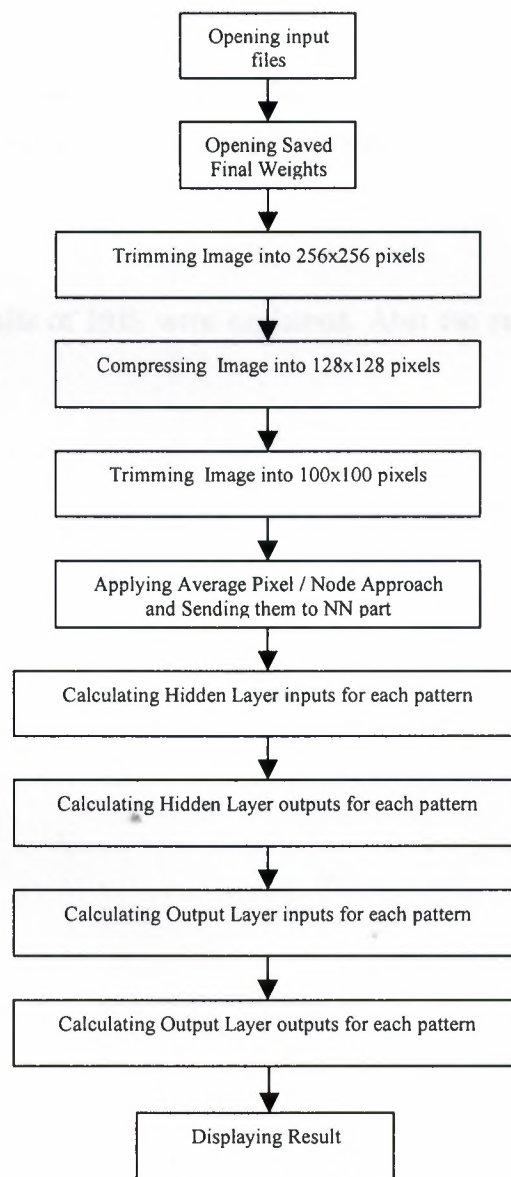


Figure 4.10 - Running Part of IBIS

4.5 Additional Information

IBIS has 100 inputs because of, after average pixel/node approach, we get an image 10x10, it means 100 inputs.

And in hidden layer, we have 30 neurons. As it mentioned before, high number of hidden neurons increases the training time but also increases the learning efficiency. IBIS has programmed to 30 hidden neurons that is the best training time and learning efficiency.

IBIS uses bias weights and momentum. They are not necessary but using bias weights and momentum rate reduces the training time.

IBIS crops the center of image to recognize. But if we train it with any part of image, it can recognize images successfully (see Chapter 5 for results).

4.6 Summary

In this chapter, all details of IBIS were explained. Also the reasons of using methods or approaches were listed.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Overview

In this chapter, experimental results will be described in details. Also the efficiency and success of IBIS will be shown.

5.2 Training Sets

IBIS was trained with 3 training sets and tested with 2 testing sets separately. In table 5.1 – 5.3, you can see the training sets.

Table 5.1 Training Set I

Banknote	Pattern No.
500,000 TL Front Side	1
500,000 TL Back Side	2
1,000,000 TL Front Side	3
1,000,000 TL Back Side	4
5,000,000 TL Front Side	5
5,000,000 TL Back Side	6
10,000,000 TL Front Side	7
10,000,000 TL Back Side	8
20,000,000 TL Front Side	9
20,000,000 TL Back Side	10

Table 5.2 Training Set II

Banknote	Pattern No.
500,000 TL Front Side	1
500,000 TL Back Side	2
500,000 TL (R)Front Side	3
500,000 TL (R)Back Side	4
1,000,000 TL Front Side	5
1,000,000 TL Back Side	6
1,000,000 TL (R)Front Side	7
1,000,000 TL (R)Back Side	8
5,000,000 TL Front Side	9
5,000,000 TL Back Side	10
5,000,000 TL (R)Front Side	11
5,000,000 TL (R)Back Side	12
10,000,000 TL Front Side	13
10,000,000 TL Back Side	14
10,000,000 TL (R)Front Side	15
10,000,000 TL (R)Back Side	16
20,000,000 TL Front Side	17
20,000,000 TL Back Side	18
20,000,000 TL (R)Front Side	19
20,000,000 TL (R)Back Side	20

5.3 Target & Actual Outputs

In IBIS, each banknote has different target outputs. In all systems, it is impossible to reach the actual 1, or actual 0. The actual outputs always approximately 1 or 0. In table 5.4, you can see the target outputs for banknotes and in table 5.5; you can see the example of actual outputs.

Table 5.3 Training Set III

Banknote	Pattern No.
500,000 TL Front Side	1
500,000 TL Back Side	2
1,000,000 TL Front Side	3
1,000,000 TL Back Side	4
5,000,000 TL Front Side	5
5,000,000 TL Back Side	6
10,000,000 TL Front Side	7
10,000,000 TL Back Side	8
20,000,000 TL Front Side	9
20,000,000 TL Back Side	10
1 CYP Front Side	11
1 CYP Back Side	12
5 CYP Front Side	13
5 CYP Back Side	14
10 CYP Front Side	15
10 CYP Back Side	16
20 CYP Front Side	17
20 CYP Back Side	18

Table 5.4 Target Outputs

Banknote	Target Output
500,000 TL Front Side	10000
500,000 TL Back Side	10000
1,000,000 TL Front Side	01000
1,000,000 TL Back Side	01000
5,000,000 TL Front Side	00100
5,000,000 TL Back Side	00100
10,000,000 TL Front Side	00010
10,000,000 TL Back Side	00010
20,000,000 TL Front Side	00001
20,000,000 TL Back Side	00001

5.4 Tolerance

Tolerance degree is an important factor in neural networks. As it mentioned above, the actual outputs never reach to actual 1 or 0. Thus, it became necessary to add tolerance term to system. To get the best result, tolerance can assign for *low*, *medium* and *high* values. You can see the tolerance values of IBIS in table 5.6. During the experiments, *medium tolerance* was applied.

5.5 Testing Sets

As it mentioned above, in experiments, for each training set, 2 testing sets (Table 5.7 & 5.8) was used to prove the success of IBIS.

Table 5.5 Example of Actual Outputs

Banknote	Actual Output
500,000 TL Front Side	0.900 , 0.070 , 0.007 , 0.050 , 0.058
500,000 TL Back Side	0.870 , 0.070 , 0.100 , 0.066 , 0.053
1,000,000 TL Front Side	0.036 , 0.850 , 0.064 , 0.073 , 0.040
1,000,000 TL Back Side	0.020 , 0.910 , 0.069 , 0.067 , 0.041
5,000,000 TL Front Side	0.051 , 0.050 , 0.886 , 0.024 , 0.067
5,000,000 TL Back Side	0.051 , 0.072 , 0.876 , 0.028 , 0.057
10,000,000 TL Front Side	0.097 , 0.044 , 0.010 , 0.881 , 0.063
10,000,000 TL Back Side	0.024 , 0.086 , 0.042 , 0.905 , 0.028
20,000,000 TL Front Side	0.056 , 0.047 , 0.066 , 0.043 , 0.871
20,000,000 TL Back Side	0.058 , 0.029 , 0.049 , 0.046 , 0.919

Table 5.6 Tolerance Values of IBIS

Name	Value
Low	0.9
Medium	0.8
High	0.7

Table 5.7 - Testing Set I

500,000 TL FRONT OLD & DIRTY BANKNOTE	500,000 TL BACK OLD & DIRTY BANKNOTE
1,000,000 TL FRONT OLD & DIRTY BANKNOTE	1,000,000 TL BACK OLD & DIRTY BANKNOTE
5,000,000 TL FRONT OLD & DIRTY BANKNOTE	5,000,000 TL BACK OLD & DIRTY BANKNOTE
10,000,000 TL FRONT OLD & DIRTY BANKNOTE	10,000,000 TL BACK OLD & DIRTY BANKNOTE
20,000,000 TL FRONT OLD & DIRTY BANKNOTE	20,000,000 TL BACK OLD & DIRTY BANKNOTE

Table 5.8 - Testing Set II

5 AUS DOLLAR FRONT	5 AUS DOLLAR BACK
5 CAN. DOLLAR FRONT	5 CAN. DOLLAR BACK
5 PAK. RUPI FRONT	5 PAK. RUPI BACK
5 GB. POUND FRONT	5 GB. POUND BACK
5 CYP. POUND FRONT	5 CYP. POUND BACK
10 CAN. DOLLAR FRONT	10 CAN. DOLLAR BACK
10 CYP. POUND FRONT	10 CYP. POUND BACK
10 GB. POUND FRONT	10 GB. POUND BACK
20 GB. POUND FRONT	20 GB. POUND BACK
1 CYP. POUND FRONT	1 CYP. POUND BACK
50 AUS. DOLLAR FRONT	50 AUS. DOLLAR BACK
50,000 ROM. FL. FRONT	50,000 ROM. FL. BACK
100 AUS. DOLLAR FRONT	100 AUS. DOLLAR BACK
100,000 ROM. FL. FRONT	100,000 ROM. FL. BACK
500,000 ROM. FL. FRONT	500,000 ROM. FL. BACK
10,000 RUS. RUBLE FRONT	10,000 RUS. RUBLE BACK
5,000 RUS. RUBLE FRONT	5,000 RUS. RUBLE BACK
1,000 RUS. RUBLE FRONT	1,000 RUS. RUBLE BACK

5.6 Results

Four experiments had been done to prove the success of IBIS. In the first experiment, Training Set I; in second, Training Set II; and in third one, Training Set III was used. Learning rate, number of hidden layers, Momentum and RMS¹ Error Level is constant for all experiments.

- No. of Hidden Layers : 30
- Learning Rate : 0.0099
- Momentum : 0.50
- RMS Error Level : 0.0050

5.6.1 Results of First Experiment

In this experiment, Training Set I was used (table 5.1). You can see the target outputs for this experiment in table 5.4 and the training results in table 5.9.

- Total Iterations: 3544
- Total CPU Time: 45 sec.

As shown, in table 5.9, training result of Experiment I is %100 in 45 seconds. As in shown table 5.10-11, for this experiment, results are very successful. Error Level Graph can be shown in Figure 5.1. Error Level Graph shows the internal changes of error during the training of neural network. At the beginning of the training, the RMS

¹ : RMS: Root Mean Square

error starts with a high value (usually the result of using random initial weights), on the second iteration the error level drops marginally as network adjusts its weights using the target outputs. The error level continues to decrease with each iteration while adjusting the weights until it reaches the required minimize error level (0.005) or until the maximum number of allowed iterations is achieved.

Table 5.9 - Training Results of Experiment I

Pattern No.	Output 1	Output 2	Output 3	Output 4	Output 5	Result
1	0.904691	0.077821	0.007122	0.057777	0.058100	500,000TL
2	0.879342	0.007407	0.101733	0.066007	0.053872	500,000TL
3	0.036647	0.858724	0.064047	0.073812	0.040664	1,000,000TL
4	0.020259	0.910921	0.069694	0.067841	0.041626	1,000,000TL
5	0.051055	0.050514	0.886891	0.024355	0.067438	5,000,000TL
6	0.055319	0.072543	0.876667	0.028656	0.057137	5,000,000TL
7	0.097204	0.044311	0.010465	0.881030	0.063807	10,000,000TL
8	0.024325	0.086656	0.042492	0.905730	0.028076	10,000,000TL
9	0.056710	0.047427	0.066518	0.043224	0.871287	20,000,000TL
10	0.058901	0.029868	0.049574	0.046052	0.919703	20,000,000TL

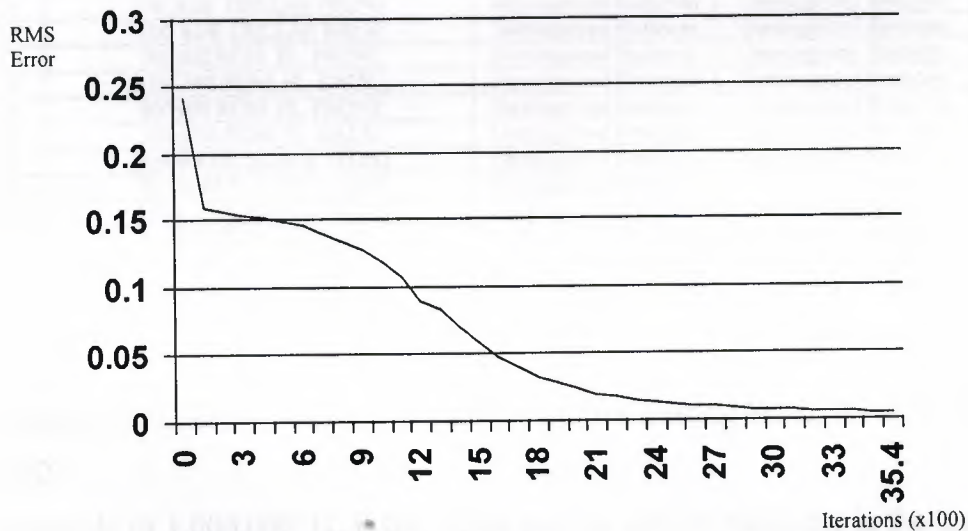


Figure 5.1 – Error Level Graph of Experiment I

Table 5.10 - Results of Experiment I for Testing Set I

NO.	BANKNOTES	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	500,000 TL FRONT OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.05 s.
2	500,000 TL BACK OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.05 s.
3	1,000,000 TL FRONT OLD & DIRTY BANKNOTE	1,000,000 TL	1,000,000 TL	0.06 s.
4	1,000,000 TL BACK OLD & DIRTY BANKNOTE	1,000,000 TL	Unrecognized	0.05 s.
5	5,000,000 TL FRONT OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.05 s.
6	5,000,000 TL BACK OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.06 s.
7	10,000,000TL FRONT OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000TL	0.05 s.
8	10,000,000 TL BACK OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000TL	0.05 s.
9	20,000,000TL FRONT OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000TL	0.05 s.
10	20,000,000 TL BACK OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000TL	0.05 s.

Table 5.11 - Results of Experiment I for Testing Set II

NO.	INPUT BANKNOTE	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	5 AUS DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
2	5 AUS DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
3	5 CAN. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
4	5 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
5	5 PAK. RUPI FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
6	5 PAK. RUPI BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
7	5 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.03 s.
8	5 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
9	5 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
10	5 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.02 s.
11	10 CAN. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
12	10 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
13	10 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
14	10 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
15	10 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
16	10 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
17	20 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
18	20 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
19	1 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
20	1 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
21	50 AUS. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
22	50 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
23	50,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.04 s.
24	50,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
25	100 AUS. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
26	100 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.04 s.
27	100,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
28	100,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
29	500,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
30	500,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
31	10,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
32	10,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
33	5,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
34	5,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
35	1,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
36	1,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.

For Testing Set I, result is 9/10, that is %90, and for Testing Set II, result is 36/36, that is %100.

The backside of 1,000,000 TL is the oldest and the dirtiest banknote in the testing sets. So the color information has big differences. Because of this reason, IBIS couldn't recognize this banknote.

When tolerance value of IBIS set to high tolerance (0.7) during testing set I, IBIS recognized all banknotes and success reached to 100%.

This experiment proves the success of IBIS. To prove the success in any condition, more complex experiment had been done.

5.6.2 Results of Second Experiment

In this experiment, Training Set II was used (table 5.2). You can see the target outputs for this experiment in table 5.4 and the training results in table 5.12.

- Total Iterations: 2921
- Total CPU Time: 75.40 sec.

As shown, in table 5.13, training result of Experiment II is %100 in 74 seconds. And IBIS shows a perfect success also in testing phases, as in shown table 5.14-15. Error Level Graph can be shown in figure 5.2.

Table 5.12 - Training Results of Experiment II

PatternNo.	Output 1	Output 2	Output 3	Output 4	Output 5	Result
1	0.918518	0.043055	0.012473	0.052434	0.046157	500,000TL
2	0.873800	0.015479	0.065651	0.080777	0.032389	500,000TL
3	0.910120	0.042727	0.015185	0.046255	0.061508	500,000TL
4	0.871305	0.013311	0.079872	0.085689	0.033625	500,000TL
5	0.044772	0.895405	0.099167	0.011382	0.049807	1,000,000TL
6	0.013901	0.884839	0.070929	0.056800	0.045753	1,000,000TL
7	0.041575	0.889383	0.086214	0.014311	0.047645	1,000,000TL
8	0.016147	0.891847	0.061419	0.057215	0.041288	1,000,000TL
9	0.049935	0.094782	0.860443	0.068463	0.011315	5,000,000TL
10	0.047982	0.061852	0.889706	0.068885	0.022257	5,000,000TL
11	0.061258	0.095982	0.858208	0.054698	0.015243	5,000,000TL
12	0.041654	0.070564	0.883085	0.058651	0.028559	5,000,000TL
13	0.109570	0.019307	0.007715	0.896261	0.080955	10,000,000TL
14	0.102514	0.022686	0.008996	0.887035	0.069939	10,000,000TL
15	0.109570	0.019307	0.007715	0.896261	0.080955	10,000,000TL
16	0.102514	0.022686	0.008996	0.887035	0.069939	10,000,000TL
17	0.035392	0.068081	0.012111	0.062176	0.901027	20,000,000TL
18	0.042795	0.068252	0.007506	0.083160	0.897761	20,000,000TL
19	0.035392	0.068081	0.012111	0.062176	0.901027	20,000,000TL
20	0.042795	0.068252	0.007506	0.083160	0.897761	20,000,000TL

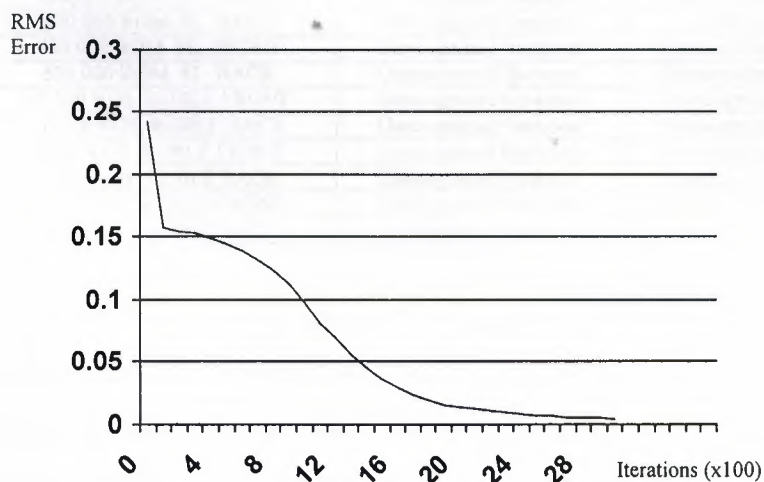


Figure 5.2 - Error Level Graph of Experiment II

Table 5.13 - Results of Experiment II for Testing Set I

NO.	BANKNOTES	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	500,000 TL FRONT OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.05 s.
2	500,000 TL BACK OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.05 s.
3	1,000,000 TL FRONT OLD & DIRTY BANKNOTE	1,000,000 TL	1,000,000 TL	0.06 s.
4	1,000,000 TL BACK OLD & DIRTY BANKNOTE	1,000,000 TL	Unrecognized	0.05 s.
5	5,000,000 TL FRONT OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.05 s.
6	5,000,000 TL BACK OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.05 s.
7	10,000,000 TL FRONT OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000 TL	0.06 s.
8	10,000,000 TL BACK OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000 TL	0.06 s.
9	20,000,000 TL FRONT OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000 TL	0.05 s.
10	20,000,000 TL BACK OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000 TL	0.05 s.

Table 5.14 - Results of Experiment II for Testing Set II

NO.	INPUT BANKNOTE	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	5 AUS DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
2	5 AUS DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
3	5 CAN. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
4	5 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
5	5 PAK. RUPI FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
6	5 PAK. RUPI BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
7	5 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
8	5 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
9	5 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
10	5 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
11	10 CAN. DOLLAR FRONT	Unrecognized Banknote	20,000,000 TL	0.06 s.
12	10 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
13	10 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
14	10 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
15	10 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
16	10 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
17	20 GB. POUND FRONT	Unrecognized Banknote	10,000,000 TL	0.05 s.
18	20 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
19	1 CYP. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05s.
20	1 CYP. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
21	50 AUS. DOLLAR FRONT	Unrecognized Banknote	5,000,000 TL	0.06 s.
22	50 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
23	50,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
24	50,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
25	100 AUS. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
26	100 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
27	100,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
28	100,000 ROM. FL. BACK	Unrecognized Banknote	5,000,000 TL	0.06 s.
29	500,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
30	500,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
31	10,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
32	10,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
33	5,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
34	5,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
35	1,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
36	1,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.

For Testing Set II, result is 9/10, that is %90, and for Testing Set II, result is 32/36, that is % 88.9. And totally IBIS reach to 41/46 that is % 89.13. Here, 4/36 foreign banknotes that IBIS hasn't been trained with, were recognized as Turkish banknotes. This incorrect result is due to the similarities of the values in the averaged input data to the neural network.

When the tolerance value of IBIS set to low tolerance (0.9) for testing set II, IBIS recognized these four banknotes also as “unrecognized banknote” so the success of IBIS in this experiment reached to 100% again.

To prove the success in any condition, more complex experiment had been done.

5.6.3 Results of Third Experiment

In this experiment, Training Set III was used (table 5.3). You can see the target outputs for this experiment in table 5.4 and the training results in table 5.15.

- Total Iterations: 4241
- Total CPU Time: 91 sec.

As shown, in table 5.16, training result of Experiment III is %100 in 91 seconds. And IBIS shows a perfect success also in testing phases, as in shown table 5.17-18. Error Level Graph can be shown in figure 5.3.

Table 5.15 - Training Results of Experiment III

Pat. No.	Out. 1	Out. 2	Out. 3	Out. 4	Out. 5	Out. 6	Out. 7	Out. 8	Result
1	0.91627	0.06006	0.00381	0.06529	0.03380	0.00486	0.04148	0.00744	500,000TL
2	0.887613	0.002234	0.074659	0.010655	0.01596	0.078215	0.032984	0.053531	500,000TL
3	0.029220	0.875848	0.062914	0.091413	0.06291	0.007105	0.030046	0.00398	500,000TL
4	0.014409	0.920122	0.045395	0.054389	0.04241	0.008263	0.066219	0.005589	500,000TL
5	0.056867	0.042954	0.885649	0.008326	0.05421	0.079857	0.029713	0.003040	1,000,000TL
6	0.040261	0.054989	0.864873	0.007431	0.03158	0.073433	0.019431	0.033679	1,000,000TL
7	0.079692	0.048871	0.004880	0.878545	0.05449	0.006920	0.012345	0.074523	1,000,000TL
8	0.012340	0.075666	0.018034	0.825352	0.01502	0.013757	0.039999	0.125806	1,000,000TL
9	0.028713	0.056869	0.064932	0.027219	0.87799	0.033902	0.015469	0.007481	5,000,000TL
10	0.028161	0.029460	0.020892	0.029523	0.91227	0.117728	0.004055	0.021050	5,000,000TL
11	0.026001	0.005763	0.035326	0.016336	0.06510	0.778434	0.137771	0.047633	5,000,000TL
12	0.028003	0.010188	0.109588	0.003767	0.05674	0.840672	0.097418	0.045284	5,000,000TL
13	0.031252	0.025575	0.020570	0.023792	0.00768	0.133442	0.783489	0.097142	10,000,000TL
14	0.043376	0.013329	0.030741	0.008850	0.01537	0.109968	0.865598	0.071349	10,000,000TL
15	0.015969	0.011773	0.019973	0.101872	0.00744	0.048471	0.127676	0.848587	10,000,000TL
16	0.043622	0.005081	0.007959	0.121789	0.02283	0.052785	0.060583	0.846838	10,000,000TL

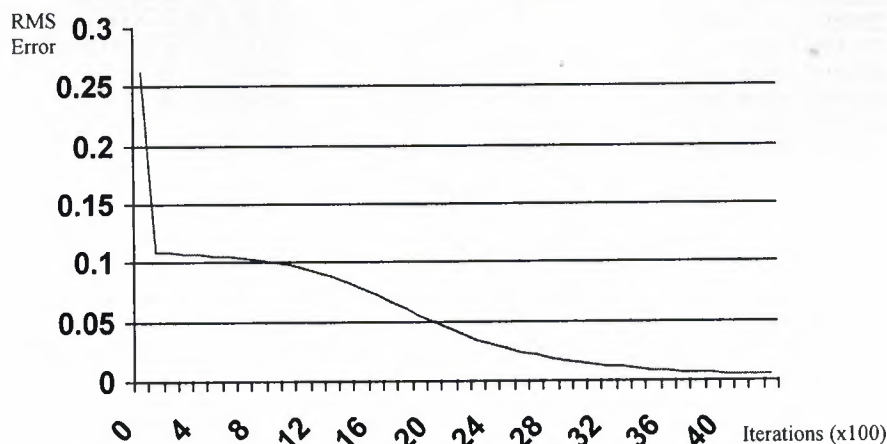


Figure 5.3 - Error Level Graph of Experiment III

Table 5.16 - Results of Experiment III for Testing Set I

NO.	BANKNOTES	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	500,000 TL FRONT OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.06 s.
2	500,000 TL BACK OLD & DIRTY BANKNOTE	500,000 TL	500,000 TL	0.05 s.
3	1,000,000 TL FRONT OLD & DIRTY BANKNOTE	1,000,000 TL	1,000,000 TL	0.06 s.
4	1,000,000 TL BACK OLD & DIRTY BANKNOTE	1,000,000 TL	Unrecognized	0.05 s.
5	5,000,000 TL FRONT OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.05 s.
6	5,000,000 TL BACK OLD & DIRTY BANKNOTE	5,000,000 TL	5,000,000 TL	0.05 s.
7	10,000,000TL FRONT OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000TL	0.06 s.
8	10,000,000 TL BACK OLD & DIRTY BANKNOTE	10,000,000 TL	10,000,000TL	0.06 s.
9	20,000,000TL FRONT OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000TL	0.05 s.
10	20,000,000 TL BACK OLD & DIRTY BANKNOTE	20,000,000 TL	20,000,000TL	0.05 s.

Table 5.17 - Results of Experiment III for Testing Set II

NO.	INPUT BANKNOTE	DESIRED OUTPUT	ACTUAL OUTPUT	CPU TIME
1	5 AUS DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
2	5 AUS DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
3	5 CAN. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
4	5 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
5	5 PAK. RUPI FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
6	5 PAK. RUPI BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
7	5 GB. POUND FRONT	Unrecognized Banknote	1,000,000 TL	0.05 s.
8	5 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
9	5 CYP. POUND FRONT	5 CY. POUND	Unrecognized Banknote	0.06 s.
10	5 CYP. POUND BACK	5 CY. POUND	5 CY. POUND	0.06 s.
11	10 CAN. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
12	10 CAN. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
13	10 CYP. POUND FRONT	10 CY. POUND	10 CY. POUND	0.06 s.
14	10 CYP. POUND BACK	10 CY. POUND	10 CY. POUND	0.06 s.
15	10 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
16	10 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
17	20 GB. POUND FRONT	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
18	20 GB. POUND BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
19	1 CYP. POUND FRONT	1 CY. POUND	Unrecognized Banknote	0.05s.
20	1 CYP. POUND BACK	1 CY. POUND	1 CY. POUND	0.06 s.
21	50 AUS. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
22	50 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
23	50,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
24	50,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
25	100 AUS. DOLLAR FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
26	100 AUS. DOLLAR BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
27	100,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
28	100,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
29	500,000 ROM. FL. FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
30	500,000 ROM. FL. BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
31	10,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
32	10,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
33	5,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
34	5,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.05 s.
35	1,000 RUS. RUBLE FRONT	Unrecognized Banknote	Unrecognized Banknote	0.06 s.
36	1,000 RUS. RUBLE BACK	Unrecognized Banknote	Unrecognized Banknote	0.06 s.

For Testing Set I, result is 9/10, that is %90, and for Testing Set II, result is 33/36, that is % 91.67. And the totally, IBIS reach to 42/46 that is % 91.3. Because of the color value differences, IBIS couldn't recognize some Greek Cyprus Pounds or gave incorrect results for some banknotes.

When low tolerance applied to the IBIS in testing set II, IBIS reached to 94.44% of success and when high tolerance applied to the IBIS in testing set I, IBIS reached to 100% percent of success in testing set I.

This experiment proves the success and flexibility of IBIS.

5.6.4 Performance of IBIS with Corrupted Data

After these three successful experiments, IBIS tested with a corrupted Turkish banknotes that size changed (Figure 5.4) or a counterfeit banknote. During the tests with low tolerance IBIS recognized corrupted image correctly. If tolerance value set to high IBIS recognized more corrupted images upto 396x256 pixels. This shows the success of IBIS in pattern recognition.



(a) Original Image



(b) Recognized Corrupted (496x256) Image
with high and low tolerance



(c) Unrecognized Corrupted (50 pixels from left –
150 pixels from right) Image



(d) Recognized Counterfeit Banknote

Figure 5.4 - Corrupted Images

When corrupted image has symmetric corruptions from left and right corners, IBIS has a good flexibility to recognize the image according to corruption size and tolerance value. But if corrupted data has an asymmetric corruptions from the sides, IBIS has a flexibility maximum about 20 pixels to recognize image, but mostly asymmetric corrupted images couldn't recognized by IBIS.

5.7 Summary

As shown in results, IBIS reaches to desired success in minimum time both in training and generalization.

As it recognizes Turkish banknotes, also it shows that average pixel / node approach is an efficient way to recognize images.

IBIS tested with all tolerance values to get optimum results. In such a conditions, mostly IBIS reached to 95-100% of success. Thus the importance of the tolerance in neural network has been explained with these experiments.

Low tolerance (0.9) gets better results of recognizing foreign banknotes, and high tolerance (0.7) gets better results to recognize same currency banknotes as trained banknotes.

CONCLUSION

This thesis is devoted to one of the actual problems: Intelligent Banknote Identification System. Turkish banknotes have been applied to the NN recognition system. These banknotes were recognized by using the NN with image processes. In this research, average pixel / node approach has been applied to all banknote images. With this approach, the NN performed training and recognition procedure faster and more efficiently.

Moreover, to confirm the recognition ability and flexibility of the system, several banknote images were applied to the system.

From the experimental results, it seems that this recognition system is effective for all world banknotes because all recognition abilities of each banknote patterns, either Turkish Banknotes or Greek Cyprus Banknotes are higher than 97%. Therefore, this system will lead to banknotes recognition machines in real banking system which is useful for world commercial system. However, this system still has a problem that is the variation on the output responses. This problem may affect the recognition ability of the system when applied in real banking machines. A correction of this problem is possible but needs more powerful computers and time.

The investigation of image processing applications using Neural Networks is carried out to provide an intelligent system (Intelligent Banknote Identification System) that identifies banknotes. Simulation of a neural network for banknote identification using C – Language, carried out. The results of experiments confirm the success of IBIS real – life application and show average pixel/node approach can be implemented to represent image.

From now, IBIS started to be a basis of a new research, Sonar Image Classification System (SICS).

REFERENCES

- [1] Anderson D. and McNeill G. , Artificial Neural Networks Technology, A DACS State of Art Report, 1992
- [2] Sağıroğlu Ş. ,Beşdok E. and Erler M. , Mühendislikte Yapay Zeka Uygulamaları – I : Yapay Sinir Ağları, ISBN : 975-95948-5-4 , Ufuk Kitap Kirtasiye – Yayıncılık Tic. Ltd. Şti. - 2003
- [3] Elmas Ç. , Yapay Sinir Ağları (Kuram, Mimari, Eğitim, Uygulama), ISBN : 975 347 612 4 , Seçkin Yayıncılık – 2003
- [4] Russell S. and Norvig P. , Artificial Intelligence : A Modern Approach, ISBN : 0-13-103805-2, Prentice Hall – 1995
- [5] Khashman A. , Near East University – COM 420 Lecture Notes, 2000
- [6] Khashman A. , Near East University - COM 507 Handouts , 2002
- [7] Crane R. , A Simplified Approach to Image Processing , ISBN : 0-13-226416-1 Prentice Hall – 1997
- [8] Adobe Photoshop 7.0 Help Files
- [9] Cho K. and Meer P. , Image Segmentation for Consensus Information , Rutgers University, Piscataway
- [10] Ray S. , Image Segmentation Methods, 2003
- [11] Young I.T. , Gerbrands J.J. and van Villet L. J. , Image Processing Fundamentals, Delft University of Technology - Netherlands
- [12] Ziou D. and Tabbone S. , Edge Detection Techniques – An Overview

- [13] Jiang M. , Digital Image Processing , University of Iowa,
<http://ct.radiology.uiowa.edu/~jiangm/courses/dip/html/node106.html>
- [14] Umbaugh S. E. , Image Enhancement Through Gray-Scale Modification
Computer Vision and Image Processing, Prentice Hall PTR
<http://zone.ni.com/devzone/conceptd.nsf/webmain/709A9F665E431C5986256C3A006B4EB1?OpenDocument>
- [15] Tilton J. C. , Hierarchical Image Segmentation, As applied to Remotely Sensed
Multispectral or Hyperspectral Imagery NASA's Goddard Space Flight Center
<http://code935.gsfc.nasa.gov/code935/tilton/>
- [16] Crevier D. , AI: The Tumultuous History of the Search for Artificial
Intelligence, Basic Books, 1993.
- [17] Kurzweil R. , The Age of Spiritual Machines: When Computers Exceed Human
Intelligence, Viking, 1999.
- [18] Sangalli A. , The Importance of Being Fuzzy and Other Insights from the Border
between Math and Computers , Princeton University Press, 1998.
- [19] Bektaş Ş. , Fakhraddin M. and Khashman A. , Graduate Studies : A Complete
Reference ,ISBN : 8359 – 06 – 01 , NEU Press , 2001
- [20] Takeda, F., and Omatu S., “High Speed Paper Currency Recognition by Neural
Networks”, IEEE Trans. on Neural Networks, Vol.6, No.1, pp.73-77, 1995.
- [21] Widrow, B., Winter, R.G., and Baxter, R.A., “Layered Neural Nets for Pattern
Recognition”, IEEE Transaction Acoustic, Speech & Signal Preprocessing,
Vol.36, No.7, pp.1109-1118, 1988

APPENDIX –I–**Table A1 - Brain Properties**

Brain Properties	Properties
Average Brain Weight	1400 g.
Average Brain Length	167 mm.
Average Brain Height	93 mm.
Average Neurons in Brain	100 Billion
Layer No. of Brain Shell	6
Thickness of Brain Shell	1,5-4,5 mm.
Development Speed of Neurons	250,000 Neurons in minute
Total Area of Synapses over Dendrites	220,000 μm^2
Cerebellum Weight of an Adult	150 g.
New borned weight of Cerebellum	21 g.
Total Neurons in Brain Shell	Man-23 Billion, Woman-19 Billion

Table A2 – Brain Weights of Some Alive

Species	Weight (g)
Adult Human	1300-1400
New Borned Human	350-400
Whale	7800
Elephant	6000
Camel	762
Horse	532
Pole Bear	498
Tiger	263.5
Lion	240
Dog	72
Cat	30
Rabbit	10
Mouse (body weight 400g)	2
Turtle	0.3

Table A3 – Biological Neuron vs. Artificial Neuron

Biological Neuron	Artificial Neuron
Neuron	Processing Element
Dendrites	Summation Function
Cell Body	Transfer Function
Axons	Outputs

APPENDIX –II–

- Training Set I Banknotes



Figure A2.1 – 500,000 TL Front



Figure A2.2 – 500,000 TL Back

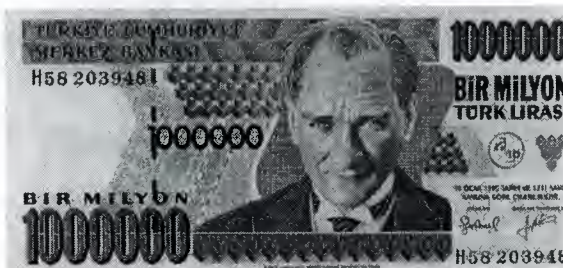


Figure A2.3 – 1,000,000 TL Front



Figure A2.4 – 1,000,000 TL Back



Figure A2.5 – 5,000,000 TL Front



Figure A2.6 – 5,000,000 TL Back

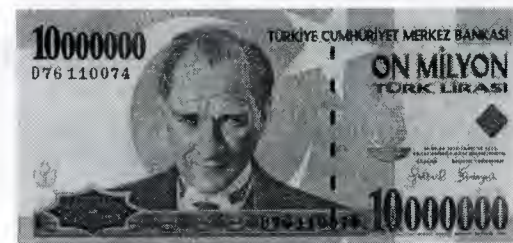


Figure A2.7 – 10,000,000 TL Front



Figure A2.8 – 10,000,000 TL Back



Figure A2.9 – 20,000,000 TL Front



Figure A2.10 – 20,000,000 TL Back

- Added Banknotes to Training Set I – (Training Set II)



Figure A2.11 – 500,000 TL Front Reversed



Figure A2.12 – 500,000 TL Back Reversed



Figure A2.13 – 1,000,000 TL Front Reversed



Figure A2.14 – 1,000,000 TL Back Reversed



Figure A2.15 – 5,000,000 TL Front Reversed

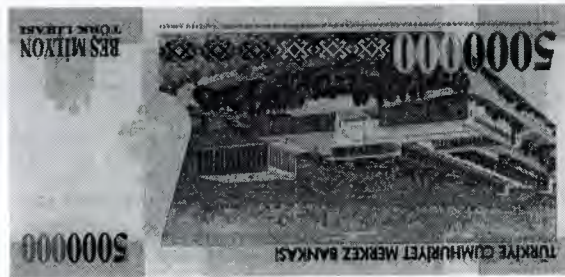


Figure A2.16 – 5,000,000 TL Back Reversed



Figure A2.17 – 10,000,000 TL Front Reversed



Figure A2.18 – 10,000,000 TL Back Reversed



Figure A2.19 – 20,000,000 TL Front Reversed



Figure A2.20 – 20,000,000 TL Back Reversed

APPENDIX –III–

- Testing Set II Banknotes

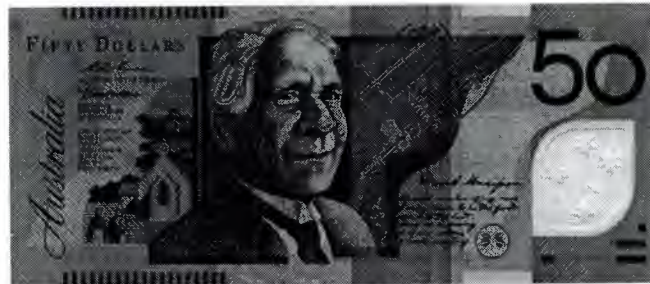


Figure A3.1 – 50 AUS Dollar Front



Figure A3.2 – 50 AUS Dollar Back



Figure A3.3 – 100 AUS Dollar Front

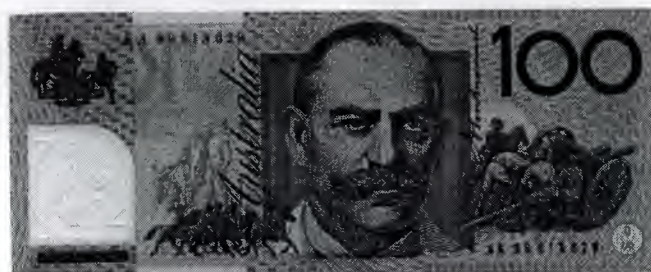


Figure A3.4 – 100 AUS Dollar Back



Figure A3.5 – 5 AUS Dollar Front

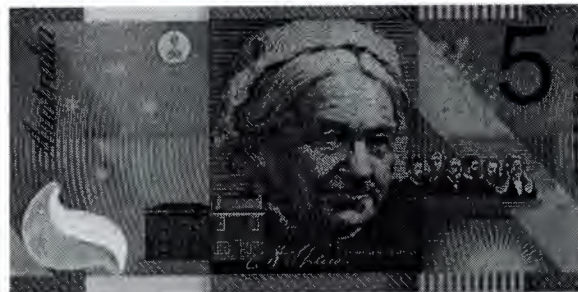


Figure A3.6 – 5 AUS Dollar Back



Figure A3.7 – 100 Russian Ruble Front



Figure A3.8 – 100 Russian Ruble Back



Figure A3.9 – 5000 Russian Ruble Front



Figure A3.10 – 5000 Russian Ruble Back



Figure A3.11 – 10,000 Russian Ruble Front



Figure A3.12 – 10,000 Russian Ruble Back

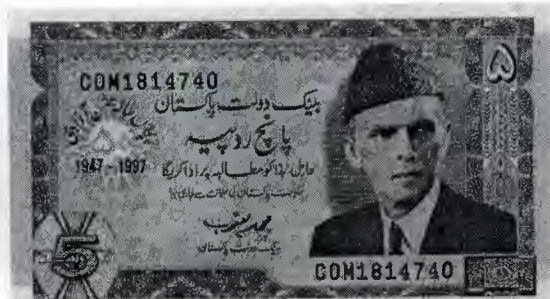


Figure A3.13 – 5 Pakistan Rupee Front



Figure A3.14 – 5 Pakistan Rupee Back

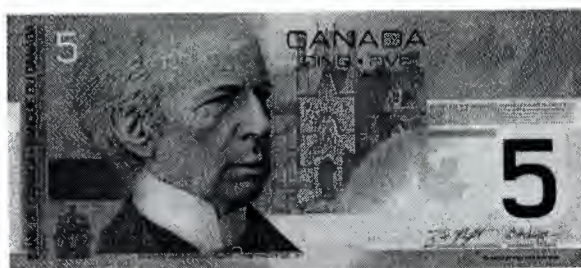


Figure A3.15 – 5 Canada Dollar Front



Figure A3.16 – 5 Canada Dollar Back



Figure A3.17 – 10 Canada Dollar Front



Figure A3.18 – 10 Canada Dollar Back



Figure A3.19 – 500,000 Romania Lei Front



Figure A3.20 – 500,000 Romania Lei Back



Figure A3.21 – 50,000 Romania Lei Front



Figure A3.22 – 50,000 Romania Lei Back



Figure A3.23 – 100,000 Romania Lei Front



Figure A3.24 – 100,000 Romania Lei Back



Figure A3.25 – 5 GP Sterling Front



Figure A3.26 – 5 GP Sterling Back



Figure A3.27 – 10 GP Sterling Front



Figure A3.28 – 10 GP Sterling Back



Figure A3.29 – 20 GP Sterling Front



Figure A3.30 – 20 GP Sterling Back



Figure A3.31 – 1 Cyprus Pound Front



Figure A3.32 – 1 Cyprus Pound Back



Figure A3.33 – 5 Cyprus Pound Front



Figure A3.34 – 5 Cyprus Pound Back



Figure A3.35 – 10 Cyprus Pound Front



Figure A3.36 – 10 Cyprus Pound Back

APPENDIX –IV–

Program Code for Training

```

void randomweights(void)
{
int i,h,o;
float weights[60]=
{
-0.30,-0.29,-0.28,-0.27,-0.26,-0.25,-0.24,-0.23,-0.22,-0.21,
-0.20,-0.19,-0.18,-0.17,-0.16,-0.15,-0.14,-0.13,-0.12,-0.11,
-0.10,-0.09,-0.08,-0.07,-0.06,-0.05,-0.04,-0.03,-0.02,-0.01,
0.30,0.29,0.28,0.27,0.26,0.25,0.24,0.23,0.22,0.21,
0.20,0.19,0.18,0.17,0.16,0.15,0.14,0.13,0.12,0.11,
0.10,0.09,0.08,0.07,0.06,0.05,0.04,0.03,0.02,0.01
};

```

```

FILE *weightsh,*weightso;
weightsh=fopen("c:\\weightsh.dat","w");
weightso=fopen("c:\\weightso.dat","w");
/*ASSIGNING INITIAL RANDOM HIDDEN LAYER WEIGHTS*/
for (i=0;i<inpunit;i++)
{
for (h=0;h<hidunit;h++)
{
weightsofhiddenlayer[i][h]=weights[random(60)];
fprintf(weightsh,"[%d][%d]=%f\n",i,h,weightsofhiddenlayer[i][h]);
}
}

```

```

void iterations(void){
int path,i;          /** i for input layer & path for patterns ***/
int h,o;             /** h for hidden layer & o for output layer ***/
float sumdelo;        /** Sum of delta of ol ***/
float sumdelh;        /** Sum of delta of hl ***/
float root;           /** Root Mean Square ***/
float dw;             /** Difference in Weights***/
float TotalError;     /** Total Error of all patterns ***/
float Error;          /** Error for each pattern ***/
float sum;
int max;

```

```

FILE *nweightsh,*nweightso;          /** Final weights ***/
FILE *nbiash,*nbiaso;

```



```

FILE *oo;
FILE *summary;
FILE *error;
nweightso=fopen("c:\\finalwo.dat","w");
nweightsh=fopen("c:\\finalwh.dat","w");
nbiash=fopen("c:\\finalbh.dat","w");
nbiaso=fopen("c:\\finalbo.dat","w");
oo=fopen("c:\\oo.dat","w");
error=fopen("c:\\error.dat","w");
summary=fopen("c:\\summary.dat","w");

for(iter=0;iter<iteration;iter++)
{
    TotalError=0;

    /*CALCULATION OF HIDDEN LAYER INPUTS FOR EACH PATTERN*/
    for (path=0;path<pattern;path++)
    {
        Read_Patterns(path);

        for (h=0;h<hidunit;h++)
        {
            hiddenin[path][h]=0;
            for (i=0;i<inpunit;i++)
            {
                hiddenin[path][h]+=weightsofhiddenlayer[i][h]*inputofinputlayer[path][i];
            }
            hiddenin[path][h]+=biaswh[h];
        }
        /*CALCULATION OF HIDDEN LAYER OUTPUTS FOR EACH PATTERN*/
        for (h=0;h<hidunit;h++)
        {
            hiddenout[path][h]=1/(1+exp(-hiddenin[path][h])); /*Sigmoid Function*/
        }
        /*CALCULATION OF OUTPUT LAYER INPUTS FOR EACH PATTERN*/
        for (o=0;o<outunit;o++)
        {
            outin[path][o]=0;
            for (h=0;h<hidunit;h++)
            {
                outin[path][o]+=weightsofoutputlayer[h][o]*hiddenout[path][h];
            }
            outin[path][o]+=biaswo[o];
        }
        /*CALCULATION OF OUTPUT LAYER OUTPUTS FOR EACH PATTERN*/
        for (o=0;o<outunit;o++)
        {

```

```

olo[path][o]=1/(1+exp(-outin[path][o])); /*Sigmoid Function*/
}

} /*end of path*/

/*CALCULATION OF ERROR FOR EACH PATTERN*/

for (path=0;path<pattern;path++)
{
for (o=0;o<outunit;o++)
{
Error=tar[path][o]-olo[path][o];
TotalError+=Error*Error;
teta[path][o]=Error*olo[path][o]*(1-olo[path][o]);
}
}

for (path=0;path<pattern;path++)
{
for(h=0;h<hidunit;h++)
{
summation=0;
for(o=0;o<outunit;o++)
{
summation+=(weightsofoutputlayer[h][o]*teta[path][o]);
}
TetaHidden[path][h]=summation*hiddenout[path][h]*(1-hiddenout[path][h]);
}
}

for(o=0;o<outunit;o++)
{
sumdelo=0;
for(path=0;path<pattern;path++)
{
sumdelo+=teta[path][o];
}
dw=(lr*sumdelo)+(alpha*biasodelta[o]);
biaswo[o]+=dw;
biasodelta[o]=dw;

for(h=0;h<hidunit;h++)
{
sum=0;
for(path=0;path<pattern;path++)
{

```

```

sum+=(teta[path][o]*hiddenout[path][h]);
}
dw=(sum*lr)+(alpha*deltaweightso[h][o]);
weightsofoutputlayer[h][o]+=dw;
deltaweightso[h][o]=dw;
}
}

for(h=0;h<hidunit;h++)
{
sumdelh=0;
for(path=0;path<pattern;path++)
{
sumdelh+=TetaHidden[path][h];
}
dw=(lr*sumdelh)+(alpha*biashdelta[h]);
biaswh[h]+=dw;
biashdelta[h]=dw;
for(i=0;i<inpunit;i++)
{
sum=0;
for(path=0;path<pattern;path++)
{
sum+=(TetaHidden[path][h]*inputofinputlayer[path][i]);
}
dw=(lr*sum)+(alpha*deltaweightsh[i][h]);
weightsofhiddenlayer[i][h]+=dw;
deltaweightsh[i][h]=dw;
}
}
}
}

```


Program Code for Generalization

```

void getimage(void);
void forward(void);

void getimage(void){
int t,z,x,y;
float Total[100];
unsigned char image[xoriginal][ypixels];
unsigned char image2[cimagex][cimagey];
unsigned char trimed1[xpixels][ypixels];
unsigned char trimed[100][100];
FILE *in;
if((in=fopen("c:\\500RB.raw","rb"))==NULL)
{
printf("\nCouldn't open image ....");
getch();
exit(1);
}
for (y=0;y<256;y++)
{
for (x=0;x<xoriginal;x++)
{
image[x][y]=fgetc(in);
}}
for (y=0;y<256;y++)
{
for (x=147;x<403;x++)
{
trimed1[x-147][y]=image[x][y];
}}
for (y=0;y<256;y=y+2)
{
for (x=0;x<256;x=x+2)
{
image2[x/2][y/2]=trimed1[x][y];
}
}
for (y=14;y<114;y++)
for (x=14;x<114;x++){
trimed[x-14][y-14]=image2[x][y];
}

for(i=0;i<100;i++)
Total[i]=0;
i=0;
for(y=0;y<100;y=y+10)

```

```

for(x=0;x<100;x=x+10)
{
for(t=y;t<y+10;t++)
for(z=x;z<x+10;z++)
{
Total[i]=(Total[i]+trimed[z][t]);
}
i++;
}
for(i=0;i<100;i++) {
inputofinputlayer[i]=(Total[i]/256)/100;

}

}

void forward(void){
int i,h,o; /*** h for hidden layer & o for output layer ***/
float max;
int no;
/*CALCULATION OF HIDDEN LAYER INPUTS FOR EACH PATTERN*/
for (h=0;h<hidunit;h++)
{
hiddenin[h]=0;
for (i=0;i<inpunit;i++)
{
hiddenin[h]+=weightsofhiddenlayer[i][h]*inputofinputlayer[i];
}
hiddenin[h]+=biaswh[h];
}

/*CALCULATION OF HIDDEN LAYER OUTPUTS FOR EACH PATTERN*/
for (h=0;h<hidunit;h++)
{
hiddenout[h]=0;
hiddenout[h]=1/(1+exp(-hiddenin[h])); /*Sigmoid Function*/
}

/*CALCULATION OF OUTPUT LAYER INPUTS FOR EACH PATTERN*/
for (o=0;o<outunit;o++)
{
outin [o]=0;
for (h=0;h<hidunit;h++)
{
outin[o]+=weightsofoutputlayer[h][o]*hiddenout[h];
}
}

```

```
}
outin[o]+=biaswo[o];

}

for (o=0;o<outunit;o++)
{
    olo[o]=0;
    olo[o]=1/(1+exp(-outin[o])); /*Sigmoid Function*/
}

/**Sensitivity & Tolerance***/
no=0;

for (o=1;o<outunit;o++)
{ if(olo[o]>olo[no])
    {no=o;}
}

}
```