PATTERN RECOGNITION OF SIMPLE SHAPES IN A MATLAB/SIMULINK ENVIRONMENT:

DESIGN AND DEVELOPMENT OF AN EFFICIENT HIGH-SPEED FACE RECOGNITION SYSTEM

By

JAWAD NAGI

Project Supervisor:
MR. SYED KHALEEL AHMED

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF ELECTRICAL AND ELECTRONICS ENGINEERING

COLLEGE OF ENGINEERING
UNIVERSITY TENAGA NASIONAL

2007
LETTER OF APPROVAL

Title of Thesis : Design of Development an Efficient High-speed Face Recognition System in a MATLAB/Simulink Environment

Submitted by : Jawad Nagi

Student ID : EE072771

The above thesis, submitted in partial fulfillment of the requirements for the degree of Bachelors of Electrical and Electronics Engineering, College of Engineering, Universiti Tenaga Nasional, has been accepted.

Supervisor : Mr. Syed Khaleel Ahmed

Signature :

Date : 30 April 2007
DECLARATION

I hereby declare that this thesis, submitted to Universiti Tenaga Nasional as a partial fulfillment of the requirements for the degree of Bachelors of Electrical and Electronics Engineering, has not been submitted as an exercise for a degree at any other university. I also certify that the work described here is entirely my own, except for quotations and summaries whose sources have been appropriately cited in the references.

This thesis may be made available within the university library and may be photocopied or loaned to other libraries for the purpose of consultation.

30 April 2007
Jawad Nagi
EE072771
ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude and respect to my project supervisor, Mr. Syed Khaleel Ahmed for his advice, guidance, supervision and patience. Besides that, I would like to thank Dr. Farrukh Nagi for his kind support.

I wish to express my gratitude to all subjects that have participated to the experiments in this project. Last but not least, my family for their understanding, patience, encouragement and support.
Automatic recognition of people is a challenging problem which has received much attention during the recent years due to its many applications in different fields. Face recognition is one of those challenging problems and up to date, there is no technique that provides a robust solution to all situations and different applications that face recognition may encounter.

Given that recognizing faces is critical for humans in their everyday activities, automating this process is very useful in a wide range of applications including human-machine interfaces and automatic access control systems such as security, surveillance, criminal identification, video compression, video indexing of large databases and numerous other multimedia applications.

This document describes the work completed for the final year project, design of an efficient high-speed face recognition system. This project includes a broad literature survey about some of the existing methods for face recognition. The developed algorithm for the face recognition system formulates an image-based approach, using the Two-Dimensional Discrete Cosine Transform (2D-DCT) for image compression and the Self-Organizing Map (SOM) Neural Network for recognition purpose, simulated in MATLAB.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LETTER OF APPROVAL</td>
<td>i</td>
</tr>
<tr>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xv</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xx</td>
</tr>
</tbody>
</table>

CHAPTER I  INTRODUCTION

1.1 Overview

1.1.1 Pattern Recognition

1.1.2 Face Recognition

1.2 Project Overview

1.3 Problem Statement

1.4 Project Objective

1.5 Scope of Thesis

1.6 Scope of Project

1.7 Project Methodology

1.8 Document Overview
CHAPTER II LITERATURE SURVEY

2.1 Introduction 10
2.2 Face Detection – A General Background 10
2.3 Face Detection Approaches 12
  2.3.1 Feature-based Approach 13
    2.3.1.1 Low-level Analysis 13
    2.3.1.2 Feature Analysis 14
    2.3.1.3 Active Shape Model 15
  2.3.2 Image-based Approach 16
    2.3.2.1 Linear Subspace Methods 17
    2.3.2.2 Neural Networks 18
    2.3.2.3 Statistical Approaches 19
2.4 Examples of Various Face Detection Systems 20
  2.4.1 Neural Network Approach 21
  2.4.2 Example Based Learning 22
  2.4.3 Decision Trees 24
  2.4.4 Model/Distance Based 25
  2.4.5 Rejection Based Classification 27
  2.4.6 Support Vector Machines 28
2.5 Conclusion 30

CHAPTER III IMAGE COMPRESSION

3.1 Introduction 32
3.2 Image Compression 33
3.2.1 Types of Image Compression

3.2.1.1 Lossy Image Compression

3.2.1.2 Lossless Image Compression

3.2.1.3 Lossy vs. Lossless Image Compression

3.2.2 Typical Image Encoder

3.2.2.1 Source Encoder

3.2.2.2 Quantizer

3.2.2.3 Entropy Encoder

3.2.3 JPEG Image Coding Standard

3.2.3.1 JPEG Image Compression Steps

3.2.3.1.1 Block Preparation

3.2.3.1.2 Discrete Cosine Transform

3.2.3.1.3 Quantization

3.2.3.1.4 Encoding

3.2.3.2 Baseline JPEG Coder

3.3 Discrete Cosine Transform

3.3.1 Overview

3.3.2 Background

3.3.3 Definition

3.3.3.1 One-Dimensional DCT

3.3.3.2 Two-Dimensional DCT

3.3.4 DCT Image Compression

3.3.5 Applications of DCT
CHAPTER IV ARTIFICIAL NEURAL NETWORKS

4.1 Introduction

4.1.1 Historical Background

4.1.1.1 Initial Attempts
4.1.1.2 Promising and Emerging Technology
4.1.1.3 Period of Frustration and Dispute
4.1.1.4 Innovation
4.1.1.5 Re-emergence
4.1.1.6 Today

4.2 Human and Artificial Neurons

4.2.1 Biological Inspiration
4.2.2 Human Neurons
4.2.3 Artificial Neurons – An Engineering Approach
4.2.4 Electronic Implementation of Artificial Neurons

4.3 Neural Networks

4.3.1 Definition
4.3.2 Neural Network Models
4.3.2.1 The Network in Artificial Neural Network
4.3.3 Neural Network Training
4.3.3.1 Definition
4.3.3.2 Choosing a Cost Function
4.3.3.3 Training Process
4.3.3.3.1  Associative Mapping  77
   4.3.3.3.1.1  Auto-association  78
   4.3.3.3.1.2  Hetero-association  79
      4.3.3.3.1.2.1  Nearest-neighbor  78
      4.3.3.3.1.2.2  Interpolative  78

4.3.3.3.2  Regularity Detection  79

4.3.3.3.3  Learning as Determination of Weights  79
   4.3.3.3.3.1  Fixed Networks  79
   4.3.3.3.3.2  Adaptive Networks  79

4.3.3.4  Learning Paradigms  79
   4.3.3.4.1  Supervised Learning  80
   4.3.3.4.2  Unsupervised Learning  80
   4.3.3.4.3  Reinforcement Learning  81

4.3.3.5  Training Function  82

4.3.3.6  Learning Algorithms  83

4.3.4  Types of Neural Networks  83
   4.3.4.1  Feed-forward Neural Networks  83
      4.3.4.1.1  Single-layer Perceptron  84
      4.3.4.1.2  Multi-layer Perceptron  87
      4.3.4.1.3  ADALINE  89
      4.3.4.1.4  Radial Basis Function Networks  90
      4.3.4.1.5  Kohonen Self-Organizing Map  97
      4.3.4.1.5.1  Overview  97
4.3.4.1.5.2 Network Architecture

4.3.4.2 Recurrent Neural Networks
  4.3.4.2.1 Simple Recurrent Network
  4.3.4.2.2 Simple Hopfield Network
  4.3.4.2.3 Echo State Network
  4.3.4.2.4 Long Short Term Memory Network

4.3.4.3 Stochastic Neural Networks
  4.3.4.3.1 Boltzmann Machine

4.3.4.3 Modular Neural Networks

4.3.5 Employing Artificial Neural Networks
  4.3.5.1 Choice of Model
  4.3.5.2 Learning Algorithm
  4.3.5.3 Robustness

4.3.6 Applications of Artificial Neural Networks

4.4 Conclusion

CHAPTER V MATLAB AND SIMULINK

5.1 Introduction
  5.1.1 History of MATLAB

5.2 What Is MATLAB

5.3 The MATLAB System
  5.3.1 Desktop and Tools Development Environment
  5.3.2 MATLAB Mathematical Function Library
  5.3.3 MATLAB Language
5.3.2 MATLAB Graphics 110
5.3.3 MATLAB External Interfaces/API 110

5.4 What Is Simulink 111
5.4.1 Tool for Model-based Design 111
5.4.2 Tool for Simulation 112
5.4.3 Tool for Analysis 112

5.5 Conclusion 112

CHAPTER VI DESIGN AND IMPLEMENTATION

6.1 Introduction 113

6.2 Face Image Processing 113
6.2.1 Data Gathering 114
6.2.2 Image Pre-processing in Adobe Photoshop 116
   6.2.2.1 Auto Adjusting Image Levels, Contrast and Brightness 116
   6.2.2.2 Image Conversion from RGB to Grayscale 118
   6.2.2.3 Image Resizing 119
6.2.3 Import Face Images to MATLAB 122
6.2.4 Image Resize in MATLAB 122
6.2.5 Face Image DCT 124
6.2.6 Image Data Reshape 129

6.3 Neural Network Design 130
6.3.1 Radial Basis Function Neural Network 131
6.3.2 Fuzzy C-Means Clustering 131
6.3.3 Self-Organizing Maps 132
### CHAPTER VII  TESTING AND EXPERIMENTAL WORK

7.1 Neural Network Validation 146
   7.1.1 Trained Face Images 146
   7.1.2 Untrained Face Images 147

7.2 Neural Network Efficiency 152
   7.2.1 Addition of Random Noise to Face Images 152
   7.2.2 Determining Optimal Number of Epochs for Training 154

### CHAPTER VIII  DISCUSSION AND ANALYSIS OF RESULTS

8.1 Discussion and Analysis 157

### CHAPTER IX  GUI IMPLEMENTATION OF PROGRAM

9.1 Introduction 158

9.2 GUI Program Design 158

9.3 GUI Program Demonstration 164

### CHAPTER X  RECOMMENDATIONS AND CONCLUSION

10.1 Recommendations for Improvement 171

10.2 Project Conclusion 172

### REFERENCES

173

### APPENDICES

A DCT Image Compression Source Code 178
B  Loading Images Source Code  179
C  SOM Neural Network Simulation Source Code  181
D  Face Recognition GUI Program Source Code  183
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table No.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2.1</td>
<td>Comparison of Number of Epochs vs. Network Training</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>Time and Network Accuracy for 10 Trials</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>Generic Representation of a Facial Recognition System</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1.1</td>
<td>Functional Block Diagram of a Pattern Recognition System</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2.1</td>
<td>Outline of a Typical Face Recognition System</td>
<td>5</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Flowchart of Designed Face Recognition Algorithm</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Steps for Face Recognition</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Different Approaches for Face Detection</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2.1</td>
<td>Face Recognition Techniques on the Image-based Approach</td>
<td>17</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>Lossy Image Compression</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>Typical Lossy Image Encoder</td>
<td>37</td>
</tr>
<tr>
<td>3.2.3.1.1.1</td>
<td>Image Divided into Block of 8x8 pixels</td>
<td>40</td>
</tr>
<tr>
<td>3.2.3.2.1</td>
<td>JPEG Encoder Block Diagram</td>
<td>41</td>
</tr>
<tr>
<td>3.2.3.2.2</td>
<td>JPEG Decoder Block Diagram</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3.2.3</td>
<td>Quantization DCT Coefficients with Quantization Table</td>
<td>43</td>
</tr>
<tr>
<td>3.2.3.2.4</td>
<td>Zig-Zag sequence</td>
<td>43</td>
</tr>
<tr>
<td>3.3.2.1</td>
<td>DCT Compared to DFT of an Input Signal</td>
<td>45</td>
</tr>
<tr>
<td>3.3.3.1</td>
<td>Two-dimensional representation of the DCT</td>
<td>48</td>
</tr>
<tr>
<td>3.3.3.2</td>
<td>Spatial to Frequency Domain Conversion of an Image</td>
<td>49</td>
</tr>
<tr>
<td>3.3.3.2.1</td>
<td>Generic 2D-DCT Architecture of an N x M Input Image</td>
<td>51</td>
</tr>
<tr>
<td>3.3.3.2.2</td>
<td>The 64 DCT Basis Functions for an 8 x 8 Block</td>
<td>52</td>
</tr>
<tr>
<td>3.3.3.2.3</td>
<td>Comparison between a Picture Matrix and DCT Coefficients</td>
<td>53</td>
</tr>
</tbody>
</table>
3.3.3.2.4  Illustration of 2D Blocked DCT of Face Image with 8x8 Blocks  54
3.3.5.1  2D-DCT and 2D-IDCT Computed on a Sample Face Image  56
4.2.2.1  A Simple Neuron  63
4.2.2.2  General Structure of a Neuron Cell  64
4.2.3.1  Artificial Neuron with Respect to a Human Neuron  65
4.2.3.2  Artificial Neuron or a Single Neural Processor  66
4.2.4.1  Detailed Schematic of Simplistic Artificial Neuron  68
4.2.4.2  Sigmoid Transfer Function  69
4.3.1.1  General Neural Network Architecture  72
4.3.1.2  Ideal Design Flowchart of an Artificial Neural Network  72
4.3.2.1.1  Feed-forward Artificial Neural Network Dependency Graph  74
4.3.2.1.2  Recurrent Artificial Neural Network Dependency Graph  75
4.3.4.1.1.1  Single-layered Perceptron Neural Network Architecture  85
4.3.4.1.2.1  Multi-layered Perceptron Neural Network Architecture  88
4.3.4.1.4.1  Architecture of Radial Basis Function Neural Network  91
4.3.4.1.4.2  Approximation by Radial Basis Functions in One Dimension  92
4.3.4.1.4.3  A Gauss Function  93
4.3.4.1.5.1.1  A Self-Organizing Map  98
4.3.4.1.5.2.1  Self-Organizing Map Architecture  99
4.3.4.2.1.1  Simple Recurrent Network  102
6.2.1  Face Image Process Structure  115
6.2.1.1  Face Images of Individual Subjects  116
6.2.1.2  Face Images of a Subject with Different Facial Expressions  116
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2.1.1</td>
<td>Adjusting Brightness, Contrast and Colors in Adobe Photoshop</td>
</tr>
<tr>
<td>6.2.2.1.2</td>
<td>Comparison of Original Image and Adjusted Image</td>
</tr>
<tr>
<td>6.2.2.2.1</td>
<td>Converting RGB Face Image into 8-bit Grayscale</td>
</tr>
<tr>
<td>6.2.2.2.2</td>
<td>Comparison of RGB color Image and Grayscale Image</td>
</tr>
<tr>
<td>6.2.2.3.1</td>
<td>Reducing Image Size – Setting Width to 512 pixels</td>
</tr>
<tr>
<td>6.2.2.3.2</td>
<td>Reducing Canvas Size – Setting Height to 512 pixels</td>
</tr>
<tr>
<td>6.2.2.3.3</td>
<td>Comparison of Original Image and Resized Image</td>
</tr>
<tr>
<td>6.2.3.1</td>
<td>Importing Face Images into MATLAB</td>
</tr>
<tr>
<td>6.2.4.1</td>
<td>Resizing Face Image from 512 x 512 pixels to 8 x 8 pixels</td>
</tr>
<tr>
<td>6.2.4.2</td>
<td>Saving all Resized Face Images under Different Filenames</td>
</tr>
<tr>
<td>6.2.5.1</td>
<td>DCT Image Compression Program Found in MATLAB Help</td>
</tr>
<tr>
<td>6.2.5.2</td>
<td>DCT Masking Matrix – 8 Coefficients Chosen</td>
</tr>
<tr>
<td>6.2.5.3</td>
<td>DCT Program for Face Image Compression</td>
</tr>
<tr>
<td>6.2.5.4</td>
<td>Comparison of Original and Compressed Un-resized Face Image</td>
</tr>
<tr>
<td>6.2.5.5</td>
<td>Comparison of Original and Compressed Image in 8 x 8 pixels</td>
</tr>
<tr>
<td>6.2.5.6</td>
<td>Image Data for Resized Image in Figure 6.2.5.5</td>
</tr>
<tr>
<td>6.2.6.1</td>
<td>Image Data Reshaped from an 8 x 8 Matrix to a 64 x 1 Array</td>
</tr>
<tr>
<td>6.2.6.2</td>
<td>Original and Compressed Image data reshaped in a 64 x 1 Array</td>
</tr>
<tr>
<td>6.3.3.1</td>
<td>25 Image Data Loaded into MATLAB in Vector ‘P’</td>
</tr>
<tr>
<td>6.3.3.2</td>
<td>Minimum and Maximum Points for SOM in a 64 x 2 Array</td>
</tr>
<tr>
<td>6.3.3.3</td>
<td>Creation of a SOM Network with Minimum and Maximum Points</td>
</tr>
<tr>
<td>6.3.3.4</td>
<td>SOM Neural Network Training for 1000 Epochs</td>
</tr>
<tr>
<td>6.3.3.5</td>
<td>SOM Layer Weights for 25 face Images Training Database</td>
</tr>
</tbody>
</table>
6.3.3.6 SOM Weight Vectors 138
6.3.3.7 SOM Network Simulation for 25 Face Image Training Database 139
6.3.3.8 Loading Input Image into MATLAB and Simulating SOM 140
6.3.3.9 Finding Picture in Database Closely Matched with Input Image 141
6.3.3.10 Classification of Subject Found in Training Database for Input 142
6.3.3.1.1 Simulink Block Model of SOM Neural Network 143
6.3.3.1.2 Simulink Block model of SOM Neural Network Layers 143
6.3.3.1.3 Simulink Subsystem Model of SOM Neural Network Layer 1 143
6.3.3.1.4 Simulink Block Model of SOM Neural Network TDL 144
6.3.3.1.5 Simulink Block Model of SOM Neural Network Layer Weights 145
7.1.1.1 Training Database of 6 Subjects 147
7.1.1.2 Trained Input Face Images 148
7.1.2.1 Training Database of 8 Subjects 149
7.1.2.2 Untrained Input Face Images 149
7.1.2.3 Training Database of 8 Subjects 150
7.1.2.4 Untrained Input Face Images with Different Facial Expressions 150
7.1.2.5 Architecture for Untrained Face Image SOM Network Validation 151
7.1.2.6 Training Database of 25 Subjects with Facial Expressions 152
7.1.2.7 Untrained Input Face Image with Different Facial Expression 152
7.2.1.1 Original Input Signal vs. Input Signal with $\alpha <= 0.30$ 154
7.2.1.2 Original Input Signal vs. Input Signal with $\alpha <= 0.35$ 154
7.2.2.1 Number of Epochs vs. Neural Network Accuracy for 10 Trials 157
9.2.1 Startup Screen for MATLAB GUIDE 160
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.2</td>
<td>Main MATLAB GUIDE window with a Blank Figure</td>
<td>161</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Face Recognition Program Development in MATLAB GUIDE</td>
<td>162</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Completed Face Recognition Program in MATLAB GUIDE</td>
<td>163</td>
</tr>
<tr>
<td>9.2.5</td>
<td>Setting callbacks for the GUI push buttons in M-file editor</td>
<td>164</td>
</tr>
<tr>
<td>9.2.6</td>
<td>Face Recognition GUI Demonstration Program</td>
<td>165</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Output Generated after pressing Push Button: DCT Sample</td>
<td>166</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Output Generated after pressing Push Button: Input Face Image</td>
<td>167</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Output Generated after pressing Push Button: Training Database</td>
<td>168</td>
</tr>
<tr>
<td>9.3.4</td>
<td>Comparison of the Input Face Image and Training Database</td>
<td>169</td>
</tr>
<tr>
<td>9.3.5</td>
<td>Face recognition GUI program simulation results</td>
<td>170</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1D-DCT</td>
<td>One Dimensional Discrete Cosine Transform</td>
<td></td>
</tr>
<tr>
<td>2D-DCT</td>
<td>Two Dimensional Discrete Cosine Transform</td>
<td></td>
</tr>
<tr>
<td>ADALINE</td>
<td>Adaptive Linear Element</td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
<td></td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
<td></td>
</tr>
<tr>
<td>ART</td>
<td>Adaptive Resonance Theory</td>
<td></td>
</tr>
<tr>
<td>CGM</td>
<td>Constrained Generative Model</td>
<td></td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
<td></td>
</tr>
<tr>
<td>DFFS</td>
<td>Distance from Face Space</td>
<td></td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
<td></td>
</tr>
<tr>
<td>DST</td>
<td>Discrete Sine Transform</td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>Decision Tree</td>
<td></td>
</tr>
<tr>
<td>DV</td>
<td>Digital Video</td>
<td></td>
</tr>
<tr>
<td>DWT</td>
<td>Discrete Wavelet Transform</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>Entropy Coding</td>
<td></td>
</tr>
<tr>
<td>ESN</td>
<td>Echo State Network</td>
<td></td>
</tr>
<tr>
<td>FA</td>
<td>Factory Analysis</td>
<td></td>
</tr>
<tr>
<td>FDCT</td>
<td>Forward Discrete Cosine Transform</td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
<td></td>
</tr>
<tr>
<td>FCM</td>
<td>Fuzzy C-Means</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
<td></td>
</tr>
<tr>
<td>GUIDE</td>
<td>Graphical User Interface Design Environment</td>
<td></td>
</tr>
<tr>
<td>HVS</td>
<td>Human Visual System</td>
<td></td>
</tr>
<tr>
<td>IEC</td>
<td>International Electro-Technical Commission</td>
<td></td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
<td></td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
<td></td>
</tr>
<tr>
<td>KDD</td>
<td>Knowledge Discovery in Databases</td>
<td></td>
</tr>
<tr>
<td>KLT</td>
<td>Karhunen-Loève Transform</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>Linear Discriminant Axis</td>
<td></td>
</tr>
<tr>
<td>LMS</td>
<td>Least Mean Square</td>
<td></td>
</tr>
<tr>
<td>MADALINE</td>
<td>Multiple Adaptive Linear Element</td>
<td></td>
</tr>
<tr>
<td>MC</td>
<td>Markov Chain</td>
<td></td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
<td></td>
</tr>
<tr>
<td>MLP</td>
<td>Multiple Layer Perceptron</td>
<td></td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Picture Experts Group</td>
<td></td>
</tr>
<tr>
<td>MRC</td>
<td>Maximal Rejection Classifier</td>
<td></td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
<td></td>
</tr>
<tr>
<td>MUX</td>
<td>Multiplexer</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
<td></td>
</tr>
<tr>
<td>PCA</td>
<td>Principle Component Axis</td>
<td></td>
</tr>
<tr>
<td>PDBNN</td>
<td>Probabilistic Decision-Based Neural Network</td>
<td></td>
</tr>
<tr>
<td>PDM</td>
<td>Point Distributed Models</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>QT</td>
<td>Quantization Table</td>
<td></td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
<td></td>
</tr>
<tr>
<td>RBFNN</td>
<td>Radial Basis Function Neural Network</td>
<td></td>
</tr>
<tr>
<td>RLE</td>
<td>Run Length Encoding</td>
<td></td>
</tr>
<tr>
<td>SGA</td>
<td>Simple Genetic Algorithm</td>
<td></td>
</tr>
<tr>
<td>SLP</td>
<td>Single Layer Perceptron</td>
<td></td>
</tr>
<tr>
<td>SOM</td>
<td>Self-Organizing Map</td>
<td></td>
</tr>
<tr>
<td>SQ</td>
<td>Scalar Quantization</td>
<td></td>
</tr>
<tr>
<td>SRN</td>
<td>Simple Recurrent Network</td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
<td></td>
</tr>
<tr>
<td>TDL</td>
<td>Tapped Delay Line</td>
<td></td>
</tr>
<tr>
<td>VLC</td>
<td>Variable Length Coding</td>
<td></td>
</tr>
<tr>
<td>VQ</td>
<td>Vector Quantization</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 Overview

Human communication has two main aspects: verbal (auditory) and non-verbal (visual), examples of the latter being facial expressions, body movements and physiological reactions. All of these provide significant information regarding the state of the person:

- Affective state, including both emotions such as fear, surprise, anger, disgust, sadness, and more enduring moods such as euphoria, dysphoria or irritability.
- Cognitive activity, such as perplexity, boredom, or concentration; temperament and personality, including such traits as hostility, sociability or shyness.
- Truthfulness, including the leakage of concealed emotions, and clues as to when the information provided in words about plans or actions is false.
- Psychopathology, including not only diagnostic information relevant to depression, mania, schizophrenia, and other less severe disorders, but also information relevant to monitoring response to treatment.

Considerable research in social psychology has shown that facial expressions help coordinate conversation and have considerably more effect on whether a listener feels liked or disliked than the speaker's spoken words. Mehrabian indicated that the verbal part of a message contributes only for 7 percent to the effect of the message as a whole. The vocal part contributes for 38 percent, while facial expression of the speaker contributes for 55 percent to the effect of the spoken message. Therefore face recognition is an important addition to computer vision research. [8]
Computer vision aims to duplicate the effect of human vision by electronically perceiving and understanding an image. Computer vision techniques use the results of mathematics, pattern recognition, artificial intelligence, psycho-physiology, computer science, electronics, and other scientific disciplines.

Recent advances in image analysis and pattern recognition open up the possibility of automatic detection and classification of emotional and conversational facial signals. Automating facial expression analysis could bring facial expressions into man-machine interaction as a new modality and make interaction tighter and more efficient. Such a system could also make classification of facial expressions widely accessible as a tool for research in behavioral science and medicine. In Figure 1.1.1, the outline of a typical face recognition system is given. This outline carries the characteristics of a typical pattern recognition system. [29]

![FIGURE 1.1.1 Generic representation of a facial recognition system](image-url)

### 1.1.1 Pattern Recognition

Pattern recognition is a modern day machine intelligence problem with numerous applications in a wide field, including face recognition, character recognition, speech recognition as well as other types of object recognition. Its ability to read text and shapes plays an important role in medical diagnostic science, machine conditioning...
and monitoring and sorting. The speed of processing is important for its real-time application in banks, stores and voice recognition. A basic pattern classification system consists of sensor inputs, feature extraction, selection and classification. Sensors could be microphone, digital camera or graphic, features are extracted with statistical, geometrical or descriptor means.

The field of pattern recognition is still very much in its infancy, although in recent years some of the barriers that hampered such automated pattern recognition systems have been lifted due to advances in computer hardware providing machines capable of faster and more complex computation. [38]

Pattern recognition can be classified as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail. The functional block diagram of an adaptive pattern recognition system is shown in Figure 1.1.1.1.

![Functional block diagram of a pattern recognition system](image)

**FIGURE 1.1.1.1** Functional block diagram of a pattern recognition system
1.1.2 Face Recognition

Face recognition is a pattern recognition task performed specifically on faces. It can be described as classifying a face either "known" or "unknown", after comparing it with stored known individuals. It is desirable to have a system that has the ability of learning to recognize unknown faces.

Face recognition involves comparing an image with a database of stored faces in order to identify the individual in the input image. The related task of face detection has direct relevance to recognition because images must be analyzed and faces identified, before they can be recognized.

Face recognition, although a trivial task for the human brain has proved to be extremely difficult to imitate artificially, because although commonalities exist between faces, they can vary considerably in terms of age, skin color, orientation, facial expression and presence of facial furniture such as glasses or facial hair. The problem is further complicated by differing light conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. Hence, a robust face recognition system should be capable of classifying a face image as "known" under the following conditions: [29]

- Scale variance
- Orientation variance
- Illumination variance
- Background variance
- Emotional expression and detail variance
- Noise variance
- Presence of details

For basic face recognition systems, some of these effects can be avoided by assuming and ensuring a uniform background and uniform light conditions. This assumption is
acceptable for some applications such as automated separation of nuts from screws on a production line, where light conditions can be controlled and the image background will be uniform. For many applications however, this is unsuitable, and systems must be designed to accurately classify images subjected to a variety of unpredictable conditions. Figure 1.1.2.1 outlines the block diagram of a typical face recognition system.

1.2 Project Overview

Face recognition focuses on still images, which can be broadly grouped into image-based and feature-based approaches. Face recognition is commonly used in applications such as human-machine interfaces, automatic access control systems,
which involve comparing an image with a database of stored faces in order to identify the subject in the image.

This project involves the design and development of an efficient high-speed facial recognition system. The pattern recognition algorithm designed for this project is based on general architecture of facial recognition systems. Program source code and simulation is executed in MATLAB and Simulink.

1.3 Problem Statement

Face recognition has become a popular area of research in computer vision, mainly due to increasing security demands and its potential, commercial and law enforcement applications. It is a very challenging problem and up to date and there is no technique that provides a robust solution to all situations and different applications that face recognition may encounter. Hence, this project focuses on developing a technique that provides a solution for an efficient high-speed face recognition system in different applications.

1.4 Project Objective

The aim of this project is to study and design an efficient high-speed facial recognition system in MATLAB. Project objectives are listed as follows:

- To study and understand pattern recognition of simple shapes using faces.
- To design a model for an ideal facial recognition system.
- To enhance the model for a high-speed facial recognition system.
- To develop a program in MATLAB based on the designed model.
- To create a database set of face images.
- To validate and test the facial recognition system.
- To perform tests for program optimization and accuracy.
• To debug the program code for errors.
• To demonstrate an effective and high-speed face recognition system.

1.5 Scope of Thesis

This thesis is written to report the design and development of a face recognition system which consists of two phases. The first phase is to study and research different face recognition approaches and to develop a methodology for the face recognition system. The second phase of this project is to implement the design based on the methodology by developing software for the face recognition program. This thesis consists of a brief introduction, background study on pattern recognition and face recognition systems, literature surveys, methodologies used for the development of the face recognition system and work done.

1.6 Scope of Project

This scope of this project is targeted to electrical and electronic engineering undergraduate students. The face recognition program developed can be used as an education tool to study an efficient face recognition technique. The program can also be used by researchers to learn how to design high-speed face recognition systems.

1.7 Project Methodology

The methodology of this project is based upon information collected and processed the study and research phase. The technique to be applied for the design and implementation of the face recognition system is as follows:

• Data gathering of face images of subjects from a digital camera.
• Pre-processing of face images in Adobe Photoshop.
• Importing face images into MATLAB.
- Discrete Cosine Transform (DCT) image compression of face images.
- Design of a Self-Organizing Map (SOM) Neural Network in MATLAB.
- Input face images into SOM Artificial Neural Network (ANN).
- Training the neural network and simulating it for different input images.
- Testing and validation of the program and technique.
- Creating a user-friendly program in MATLAB from the source code.

Figure 1.4.1 illustrates the flowchart proposed for the design of the face recognition system, based on the methodology.

![Flowchart of designed face recognition algorithm](image)

**FIGURE 1.4.1 Flowchart of designed face recognition algorithm**

### 1.8 Document Overview

The second chapter of this report, chapter 2 covers the Literature survey on face recognition techniques. The survey analyzes and compares some of the approaches to automatic facial expression analysis that have been done in the period between the mid 90s and beginning of 2000 conducted by Hjelmås and Low.
In chapter 3, the theory and background study of the Discrete Cosine Transform technique, which is to be applied for the design and implementation of this project is explained. Chapter 4 covers the background study and research over Artificial Neural Networks which are the heart of the face recognition system. Chapter 5 gives information regarding MATLAB and Simulink software used for the design of this project.

In chapter 6, design and implementation of the project is explained in an orderly manner step by step. Chapter 7 focuses on the testing and experimental work carried out for neural network validation and efficiency improvement. Chapter 8 discusses the results obtained from Chapter 7 from the various validation tests performed on the face recognition system.

Chapter 9 involves the development of a face recognition GUI program in MATLAB. And finally, Chapter 10 gives future recommendations for improvement of the project and the project conclusion.
CHAPTER II

LITERATURE SURVEY

2.1 Introduction

This chapter contains a general overview of the existing methods for face detection. This survey analyzes and compares some of the approaches to automatic facial expression analysis of static images that have been done in the period between the mid 90s and beginning of 2000 conducted by Hjelmås and Low.

2.2 Face Detection – A General Background

Face detection is a necessary first-step in face recognition systems, with the purpose of localizing and extracting the face region from the background.

Human visual systems are characterized by the fact that the face is perceived as a whole, not as a collection of the facial features. The presence of the features and the geometrical relationship with each other appear to be more important than the details of the features. The other characteristic is that humans detect faces easily in a wide range of conditions, under bad lighting conditions or from a great distance. It is believed that two-grey-levels images of 100 to 200 pixels form a lower limit for detection of a face by a human observer. [29]

However in the case of computer vision, face detection is not an easy task. Detecting a face in a digital image involves segmentation, extraction, and verification of faces and possibly facial features from an uncontrolled background. Therefore this is a very important topic in the computer vision field where a considerable amount of research has been done.
All the existing methods for automated face recognition are mainly based on three steps: face detection, facial feature extraction and expression classification from the observed facial image as shown in Figure 2.2.1.

In the case of static images, the process of extracting the facial expression information is referred to as ‘localizing’ the face and its features in the scene. In the case of facial image sequences, this process is referred to as ‘tracking’ the face and its features in the scene. It is important to make a distinction between facial features and face model features. The first ones are the prominent features of the face: eyebrows, eyes, nose, mouth, and chin. The face model features are used to represent the face. The face can be represented in a range of ways: as a whole unit, Holistic representation; as a set of features, Analytic representation; and finally as a set of combinations of those, Hybrid approach.
2.3 Face Detection Approaches – An Overview

Hjelmås and Low conducted a survey on face detection techniques, and identified two broad categories that separate the various approaches, namely Feature-based and Image-based approaches. Each category will be explained, providing a brief yet thorough overview of the various face detection techniques. Figure 2.3.1 illustrates the different approaches for face detection.

FIGURE 2.3.1 Different approaches for face detection
2.3.1 Feature-based Approach

Hjelmås and Low divided the group of feature-based system into three sub-categories: *Low-level Analysis, Feature Analysis* and *Active Shape Models*.

2.3.1.1 Low-level Analysis

Low-level analysis deals with the segmentation of visual features using various properties of pixels, predominantly gray-scale or color.

Edge representation (detecting changes in pixel properties) was first implemented by Sakai et al for detecting facial features in line drawings. Craw et al developed this further to trace a human head outline, allowing feature analysis to be constrained to within the head outline. Various operators are used to detect the presence of an edge, including the Sobel operator, the Marr-Hildreth operator, and a variety of first and second derivatives of Gaussians. All edge-based techniques rely on labeled edges which are matched to a face model for verification. Govindaraju labeled edges as left side, hairline, and right side, developing a system capable of detecting 76% of faces in a set of 60 images with complex backgrounds, with an average of two false alarms per image.

Gray information can be used to identify various facial features. Generally eyebrows, pupils and lips appear darker than surrounding regions, and this extraction algorithm can search for local minima. In contrast, local maxima can be used to indicate the bright facial sports such as nose tips. Detection is then performed using low-level gray-scale thresholding. Yang and Huang explore grayscale using an image pyramid where an image is resized. Using the hypothesis that at low resolutions face regions will become uniform, face candidates are established which are verified by the existence of local minima at high resolution, representing facial features.
Color contains extra dimensions which can help differentiate two regions which may contain similar gray information but appear very different in color space. It was found that different skin color gives rise to a tight cluster in color space, thus color composition of human skin differs little across individuals, regardless of race. The most widely used color model is RGB, although there are many others that exist and have been used.

Motion information (where available) can be used to assist in the detection of human faces, using the principle that, if using a fixed-camera, the "background clutter" will remain somewhat static, relative any "moving object". A straightforward way to achieve motion segmentation is by frame difference analysis. Thresholding accumulated frame differences is used to detect faces or facial features. Another way to measure motion is thought the estimation of moving image contours, a technique that has proven to be more reliable, particularly when motion is insignificant.

2.3.1.2 Feature Analysis

Low-level analysis introduces ambiguity which can be solved by high-level feature analysis, often through the use of some additional knowledge about the face. There are two approaches for the application of this additional knowledge (commonly face geometry).

The first involves sequential feature searching strategies based on the relative positioning of individual facial features. Initially prominent facial features are determined which allow less prominent features to be hypothesized (for example a pair of dark regions found in the face area increases the confidence of facial existence). The facial feature extraction algorithm is a good example of feature searching, achieving 82% accuracy with invariance to gray and color information, failing to detect faces with glasses and hair covering the forehead. A similar system proposed by Jeng et al reported an 86% detection rate.
The second technique, constellation analysis, is less rigid and is more capable of locating faces of various poses in complex backgrounds. It groups facial features in face-like constellations, using robust modeling methods such as statistical analysis. Burl et al used statistical shape theory on features detected from a multi-scale Gaussian derivative filter, capable of detecting 84% of faces, with some invariance to missing features, translation, rotation and scale. Probabilistic face models based on multiple face appearance have also been used in many systems including Yow and Cipolla's model reporting a 92% detection rate. [29]

### 2.3.1.3 Active Shape Model

Active shape model represents the actual physical and hence higher-level appearance of features. These models are released near to a feature, such that they interact with the local image, deforming to take the shape of the feature. There are three types of active shape models that have been used through the literature: snakes, deformable templates and smart snakes.

Snakes or active contours are commonly used to create a head boundary. Created nearby, they lock on to nearby edges, eventually assuming the shape of the head. The evolution of a snake is achieved by minimizing an energy function, which consists of the sum of an internal energy function, defining its natural evolution (typically shrinking or expanding), and an external energy function, which counteracts the internal energy enabling the contours to deviate from the natural evolution. Energy minimization can be obtained by optimization techniques such as the steepest gradient descent although the additional computational demands have encouraged others to use faster iteration methods, and Lam and Yan. Snakes are prone to becoming trapped on false image features and are not efficient in extracting non-convex features due to their tendency to attain minimum curvature. Gunn and Nixon, introduced a parameterized snake model to overcome these limitations by using dual integrated snakes.
Deformable temples can be used as an extension to the snake models. Yuille et al incorporated global information of the eye to improve the extraction process, using a deformable eye template. Once established near an eye feature, would deform itself, toward optimal feature boundaries using steepest gradient descent minimization as the deformation mechanism. One limitation of such techniques is that they are sensitive to initial placement. Yuille et al showed promising results when the template was placed below the eye, but noted that if place above the eye, the template may be attracted towards the eyebrow instead. A further limitation is the processing time attempts have been made to improve this. Besides eye templates, the use of mouth templates has also been introduced.

Smart snakes or Point Distributed Models (PDMs) are compact parameterized descriptions of a shape based upon statistics. They use Principle Component Analysis (PCA) to construct a linear flexible model from variations of the features in a training set. Face PDM was first developed by Lantis et al as a flexible model with promising results (95% detection rate). Multiple faces can be detected tests have shown that partial occlusion is not a problem as other features are still available to contribute to a global optimal solution. [8]

2.3.2 Image-based Approach

Face detection by explicit modeling of facial features is a very rigid approach which has been shown to be troubled by the unpredictability of faces and environmental conditions. There is a need for more robust techniques, capable of performing in hostile environments, such as detecting multiple faces with clutter-intensive backgrounds. This has inspired a new research area in which face detection is treated as a general pattern recognition problem. Whereas face recognition deals with recognizing the face, face detectors must recognize an object as a face, from examples. This eliminates the problem of potentially inaccurate models based on the erroneous or incomplete face knowledge and instead places the emphasis on the training examples from which the system leans to distinguish a face. Most image-
based approaches apply a window scanning technique for detecting faces, which due to its exhaustive nature, increases computational demand.

Hjelmås and Low divided the group of image-based system into three sub-categories: Linear Subspace Methods, Neural Networks and Statistical Approaches. Figure 2.3.2.1 illustrates different face recognition techniques using on the image-based approach.

![FIGURE 2.3.2.1 Face recognition techniques on the image-based approach](image)

### 2.3.2.1 Linear Subspace Methods

Images of human faces lie in a subspace of overall image space which can be represented by methods closely related to standard multivariate statistical analysis, including Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Factory Analysis (FA).

In the late 1980's Sirovich and Kirby developed a technique using PCA to represent human faces. The technique first finds the principal components of the distribution of faces (expressed in terms of eigenvectors). Each face in the set can then be approximated by a linear combination of the largest eigenvectors, more commonly
referred to as eigenfaces. Moghaddam and Pentland later proposed a facial feature detection using 'Distance from Face Space' (DFFS – a measure of faceness), generated from eigenfeatures (eigeneyes, eigennose, eigenmouth) obtained from various facial feature templates in a training set. The performance of the eye locations was reported to be 94% with out 6% false positive (uniform background). [29]

Various improvements to this simple system were made, notably that of Sung and Pogio which represented the facespace by diving into subclasses. Their detector involves four steps: image pre-processing, distribution-based model construction, image to model measurements taken and classification. One issue arising from the application of such systems is the problem associated with the collection of a representable set of training samples, particularly for non-face images. Sung and Pogio suggest the use of a bootstrap algorithm, now widely implemented, which involves adding false detections to the training set as non-face examples.

Yang et al proposed a method for face detection based on Factor Analysis (FA), which is similar to PCA but assumes the observed data samples come from a well defined model. Using a mixture of factor analyzers, a set of training images is used to estimate the parameters in the mixture model. This model is then applied to sub-windows in the input image and the probability of a face being present is returned. Yang et al also proposed a system using Kohonen's Self Organizing Maps, and an LDA. Whereas PCA is aimed at representation, LDA aims for discrimination and is therefore well suited for face detection when the class of faces and non-faces is divided into subclasses.

2.3.2.2 Neural Networks

Early approaches based on the simple Multiple Layer Perceptrons (MLP) gave encouraging results on fairly simple datasets. The first advanced neural approach which reported performance statistics on a large, visually complex dataset was by Rowley et al. Their system incorporates face knowledge in the retinally connected
neural network architecture, with specialized window sizes designed to best capture facial information (e.g. horizontal strips to identify the mouth). Images are pre-processed before being classified by the network, the output from which is post-processed to remove overlapping detections, resulting in one detection per face and a reduction in false positives. Multiple networks were trained independently and their outputs combined using various arbitration methods to further improve performance. Their detector produced very impressive results, and has since been incorporated in many other systems including several commercial applications.

A different neural approach was suggested by Feraud et al based on a Constrained Generative Model (CGM), an auto associative fully connected MLP with three layers. Lin et al, proposed a fully automatic face recognition system based on probabilistic decision-based neural networks (PDBNN). For face detection the network has only one subnet representing the face class, and training is unsupervised. [29]

### 2.3.2.3 Statistical Approaches

Systems based on information theory, support vector machines and Bayes' decision rule are examples of image-based approaches that do not fit into either of the other categories.

Colomenarez and Huang developed a system based on Kullback relative information. This divergence is a non-negative measure of the difference between the two probability density functions. During training, for each pair of pixels in the training set, a joint-histogram is used to create probability functions for the classes of faces and non-faces. The training procedure uses a large quantity of 11x11 pixel images and results in a set of look-up tables of likelihood ratios. Pixels of pixels which contribute poorly to detection is completely removed from the look-up tables to reduce computational requirements. The system was further improved by incorporating a bootstrap training algorithm.
In Osuna et al a Support vector Machine (SVM) is applied to face detection. A SVM with a second degree polynomial as a kernel function is trained with a decomposition algorithm which guarantees global optimality. Images are pre-processed and trained with a bootstrap learning algorithm. Other research into SVMs has attempted to improve the Osuna detector with good results. [29]

Schneiderman and Kanade propose two face detectors based on Bayes' decision rule:

\[
\frac{P(\text{image} | \text{object})}{P(\text{image} | \text{non-object})} > \frac{P(\text{non-object})}{P(\text{object})}
\]

A face exists at the current location if the above condition is true. The Bayes decision rule is proven to given the optimal solution providing \( P(\text{image}|\text{object}) \) and \( P(\text{image}|\text{non-object}) \) are accurate. Schneiderman and Kanade implement a wavelet transform in their second system which decomposes the image into 10 sub-bands, from which 17 visual attributes are extracted, and treated as statistically independent random variables. The view based detector is capable of detecting frontal and profile views as it is trained with frontal and right profile images (which are mirrored to detect left profile faces). Promising results are reported from these systems.

### 2.4 Examples of Various Face Detection Systems

This section details six examples of the various different approaches to the face detection problem, with examples of both feature-based and image-based approaches. Performance statistics have been presented where available, although due to the lack of a common test set and procedure, comparison of the detectors is not provided.
2.4.1 Neural Network Approach

Henry A. Rowley, Shumeet Baluja, and Takeo Kanade co-wrote an article presenting their Neural Network-Based Face Detection system. Their system arbitrates between multiple networks to improve performance over a single neural network. Capable of correctly locating upright frontal faces in gray-level images, the detector proved to perform well with varying lighting conditions, and relatively complex backgrounds. Presented here is a summary of the detector, its unique features, and an evaluation of its performance.

Images presented to the network go through a pre-processing step to improve the classification of the system. An attempt is made to equalize the intensity values of the pixels present in the examination windows and histogram equalization is performed to expand the range of intensities, in order to compensate for the differences in lighting and camera input gains. The image is passed through a neural network, consisting of a layer of hidden units: four looking at a 10x10 pixels sub region, sixteen looking at a 5x5 pixel sub region (for detecting the eyes and nose), and six looking at overlapping 20x5 horizontal strips of the pixels (for detecting the mouth). The output of the network is a single real value, indicating the presence of a face. The network is trained with both faces and non-faces example images, such that it can learn to classify images in the relevant class correctly. A bootstrap method is used to simplify the training process, by adding false detections into the set as the training progresses. To give the classifier invariance to small degrees of scaling, translation and rotation, the training set is extended through the addition of rotated (up to 10 degrees), scaled (90-110%) and translated (up to half a pixel) versions of the training images. The network is initialized with random weights, and is trained to output '1' for face examples, and '-1' for non-face examples. [8]

Experiments with a single network gives good detection rates (90.3-92.3%), but suffer large numbers of false positives. To improve the performance both merging overlapping detections from a single network and arbitrating among multiple networks can be implemented. Merging overlapping detections preserves those
locations in an image with a high number of detections (within a neighborhood), and eliminates locations with fewer detections. There are relatively few cases in which this heuristic fails, except in examples where one face partially occludes another. When using multiple networks, arbitrating between the networks in order to produce a signal output is necessary. With 'ANDing', the detection rate of the faces is reduced (77.9%) as both networks had to detect a face for the overall output to be '1', but similarly very few false positives occur. When replaced with 'ORing', the detection rate improves (90.3%), but at the expense of a large number of false positives. A third more complex method of arbitration is to use further stage of computation in the form of another Neural Network. This method produces good detection rates (83.4-84.4%), with an acceptable number of false positives.

This approach to face detection is highly comparable to other studies in the field, able to detect between 77.9% and 90.03% of faces in a set of 130 test images, with an acceptable number of false detections. The system does however, have some limitations. It can only successfully detect upright frontal faces (looking directly at the camera), although this could be corrected by training a set of arbitrated multiple networks with a different training set.

2.4.2 Example Based Learning

Kah-Kay Sung and Tomaso Pogio published a paper in 1998 entitled 'Example-based Learning for View-based Human Face Detection'. It presents an example-based learning approach for locating unoccluded vertical frontal views of human faces at various scales in complex scenes, under a relatively wide range of light conditions. The performance of this detection system relies heavily on the face model. An image set of approximately 1000 mugshots is used to construct a distribution based generic face model, thus creating a face model thought to be more accurate and more descriptive, than manually synthesized models which rely on prior knowledge of facial structures. Once a model has been created, a decision procedure is then trained on a large set of "face" and "non-face" examples, to empirically discover a set of
operating parameters and thresholds that separate the classes. Most similar approaches arrive at these conditions by manually examining a few cases, whereas this automatic approach presented here is thought to provide statistically more reliable values, and is likely to be capably of learning any more complex high-dimensional or non-linear relationships for distinguishing between "face" and "non-face" patterns.

The system divides an image into multiple, possibly overlapping sub-images (19x19 pixels in size) and attempts to classify each image window as either "a face" or "not a face". Pertinent information is extracted by taking measurements on each image window, which a pre-defined decision procedure uses to determine whether the enclose pattern resembles the canonical face structure represented by the model. Multiple scales are handled by examining and classifying fixed sized window patterns on scaled versions of the image. Whenever a window "matches" the model, the system reports a face has been found and returns the location and scale of that face. The most critical part of this system is the algorithm for classifying window patterns. Input windows are masked to eliminate near boundary pixels which are likely to fall outside the spatial boundary of a face. A best fit brightness plane is subtracted from the image, and histogram equalization is performed to correct for variations in illumination brightness and camera response curves. The model with which image windows are compared consists of a set of 12 prototypes, constructed by performing a modified version of the k-means clustering algorithm on a representative set of face images and non-face images independently. When a new image is presented to the detector, a set of measurements such that each set of measurements is a vector of 12 distances from the new window pattern to each of the window pattern prototypes. Rather than using simple Euclidean distances, a new "Mahalanobis-like" distance metric was defined, whose tests have proved very well performance. The computed distances are then fed into a Multi-layer Perceptron (MLP) Neural Network with 12 pairs of input terminals, 24 hidden units and one output, and uses sigmoidal thresholding. The network is trained using a standard back propagation algorithm to output a '1' for face pattern and '0' otherwise. The training set consists of 47316 "face"
and "non-face" window patterns artificially enlarged to include slightly rotated and mirrored versions of the original face patterns. [29]

The detector was shown to correctly identify 96.3% of all face patterns with 3 false detects with a database thought to represent the systems "best case" performance (high quality images with a fair amount in lighting variation). Those images that were missed had either strong illumination shadows or large rotations. The average case performance was determined using a database consisting of varying quality images with complex backgrounds. The detector could correctly identify 79.9% of all face patterns with 5 false detects, with low quality images or hand drawn faces being missed, thought to be acceptable. Experiments were also done to evaluate the various parts of the system, which showed that the chosen distance metric outperforms other comparable metrics although the chosen MLP is often outperformed by a single perceptron network suggesting the two classes are linearly separable. As with all model based approaches, the limitations lie primarily with the quality of the model. As models are fairly rigid, it has been suggested that this system could be duplicated, having several identical detectors running in parallel, each with distinctly different models for identifying different poses.

### 2.4.3 Decision Trees

Jeffery Huang, Srinivas Gutta, and Harry Wechsler proposed a novel algorithm for face detection using decision trees and showed its generality and feasibility using a database consisting of 2,340 face images from the FERET database.

The basic aim of all these face detection systems is to separate the foreground face (object), from the background (clutter). This is done by constructing a set of rules for classifying objects given a training set of objects whose target class labels are known. The rules used for this detector form a decision tree (DT). The decision tree iteratively determines the facial features most discriminatory, and splits the data into classes categorized by this feature. The next significant feature is then used to further
partition each subset, and so on until each subset contains only one feature. This
decision tree contains nodes which represent feature discrimination tests and
associated exit branches representing those subclasses that satisfy the test. Once a tree
has been constructed, a tree pruning mechanism is often invoked, discarding some of
the unimportant sub trees whilst retaining those covering the largest number of
examples, thus providing a more general description of the learned concept. Prior to
images being presented to the classifier, they are pre-processed to focus
computational resources and thus provide speed improvements. This initial stage
comprises of three functions: histogram equalization to correct illumination
differences, edge detection to find transitions in pixel intensity and analysis of
projection profiles ultimately yielding a boundary box.

A cropping stage then takes each 8x8 window identified, and labels it "face" or "non-
face" using decision trees induced using both positive (face) and negative (non-face)
examples. This training set contains 12 images whose resolution is 256x384 and
consists of 2759 windows tagged CORRECT (positive examples), and 15673 tagged
INCORRECT (negative examples). Finally the labeled output is post-processed to
decide if a face is present by counting the number of face labels. If this value exceeds
a threshold, this indicates the presence of a face. If found the face boxes are then
normalized to yield a uniform size of output image. [29]

The accuracy achieved with this detector is quoted to be 96%, where accuracy
information is gathered manually, based on visual observation that the face box
includes both the eyes, nose, and mouth, and that the top of the face box is below the
hairline. Experiments showed that the detector was capable of rejecting 23 out of 25
complex background scenery images (92%).

2.4.4 Model/Distance Based

Klaus J. Kirchberg, Oliver Jesorsky and Robert W. Frischolz summarized their
findings in a paper published in 2002 entitled "Genetic Model Optimization for
Hausdorff Distance-Based Face Localization". They presented a model-based approach to perform robust, high-speed face localization based on the Hausdorff distance. An optimization approach is proposed to create and improve an appropriate representative edge model by means of genetic algorithms. With model-based approaches, it is important to ensure that the chosen model accurately represents all face variations that the face detector is likely to be asked to classify. In the simplest form, this model could be an ellipse, loosely expressing this feature of the face, but this simple model is far from adequate for most applications. The model is optimized using a large database of sample images.

For simplicity this approach focuses on finding a single face (although extension to multiple faces). An edge magnitude image is first calculated with the Sobel operator, where the relevant edge feature points are extracted by a locally adaptive threshold filter to compensate for variable illumination. It is assumed that this will produce a characteristic arrangement of segmentation points which allows the gray-level image to be presented in binary form. The associated face model used itself consists of a set of feature points and can be represented as a binary image, with the chosen feature points being similar to the typically observed patterns in the images' face area. The model is optimized using a Simple Genetic Algorithm (SGA), which requires genotype coding of the face model, a fitness function, and some additional parameters. The genotype coding assumes the face model is symmetric along the vertical axis, and thus only half the model is coded in a two-dimensional binary genome. The fitness function assigns a real-valued number to the model that reflects the performance of the face localization algorithm with that model. The model is tested with a set of sample images, and the ratio of found faces to the overall number of faces in the set determines the fitness value. A face is said to be found if some distance measure between the true position and the found position is below a certain threshold. To detect a face the model is superimposed over the image (at several discrete positions) and the similarity between the model and the covered part of the image is calculated. A face is considered to exist at the location yielding the highest
similarity. In order to determine the "similarity" between the two binary images, a modified Hausdorff distance metric is used.

As the model can be scaled to detect faces of different sizes, it is important for the model to be sufficiently detailed to be able to do so, whilst maintaining an accurate generic representation of the face. A 45x47 model grid proved to be a good trade-off. Tests were carried out to analyze the performance on the Genetic Algorithms used. The GA was shown to perform better when developing a face model from scratch, with localization performance of over 90% (compared to roughly 60% for hand drawn models). Face localization could be further improved by using a multi-step detection approach using more than one model in different grades of detail. Each model would be optimized separately producing more exact face coordinates. [29]

2.4.5 Rejection-Based Classification

In 2002, Michael Elad, Yacov Hel-Or and Tena Keshet co-wrote a paper presenting their Rejection-based classifier for face detection. Their classification algorithm exploits the property that a greater proportion of image windows will be classified as clutter (background of non-face elements), than as targets (face elements). This yields an effective yet low complexity target (face) detection algorithm, entitled the Maximal Rejection Classifier (MRC), which is based on linear successive rejection operations.

The face detection can be characterized, in practice, as designing a classifier C(z), which, given an input vector 'z' has to decide whether z belongs to the target class X or the clutter class Y, i.e. decide whether the image contains a face or not. As the classifier lies at the heart of the face detection system, and is applied many times, simplifying it implies an efficient detection algorithm. The MRC maintains the simplicity of linear classifiers, but is powerful enough to deal with classes that are not linearly separable providing the clutter class and the convex hull of the target class to be disjoint (a condition known as convex separability). The classifier exploits the
property of the high clutter probability ensuring clutter labeling is very fast, even at the expense of slower target labeling.

The MRC is an iterative algorithm that attempts to maximize the number of rejecter clutter samples. Man samples are labeled as clutter in the early iterations, and thus are exempt from processing in successive iterations. When only a few non-clutter samples remain, these are labeled targets. The algorithm was demonstrated in this system, designed to detect frontal and vertical faces. A comprehensive face set was constructed containing 400 frontal and vertical face images 15x15 pixels in size. The non-face set was required to be larger and was constructed by decomposing 54 arbitrary images into a Gaussian pyramid creating 1290 images. Each 15x15 block in these images was treated as a separate non-face example, effectively forming over 40 million non-face examples.

In order to improve the detection algorithm, various pre-processing techniques were presented; removal of the mean, shaping the frequency representation, masking out the pixels. The performance was evaluated with the mean removal and masking pre-processing took place. All of six examples tested with the system required an effective number of close to two convolutions per each pixel in each resolution layer, with few false alarms. The results started that the MRC face detection algorithm had performance accuracy comparable to the Rowley et al. detector but with computation speed an order of magnitude faster. [29]

2.4.6 Support Vector Machines

Edgar Osuna, Robert Freund and Federico Girosi investigated the application of Support Vector Machines (SVMs) in computer vision and presented their findings in an article entitled 'Training Support Vector Machines an Application to Face Detection'. SVM is a learning technique developed by V. Vapnik and his team (AT&T Bell Labs), and is a relatively new method for training polynomial, neural network, or Radial Basis Function (RBF) classifiers. This short summary contains a
brief overview of this technique and provides an indication of its worth in the field in terms of performance statistics.

The proposed face detector is capable of detecting vertically oriented, unoccluded, frontal views of human faces in gray level images. Although scale variations and lighting conditions are dealt with to an extent by the detection system, there is no mention of the background complexity that can be effectively classified. This system required no prior knowledge of faces, unlike both the Rowley et al approach which uses various feature specific receptors and the Sung and Pogio approach which uses a complex face model. Decision surfaces are found through training equivalent to solving a linearly constrained quadratic programming problem which is challenging as the memory requirements grow as the square of the number of data points (the main limitation of this approach). A complex decision boundary is calculated to separate the two classes (face and non-face) with the largest possible margin, and fewest misclassifications. Those data points that play a vital role in this task usually lie close to the border and are called support vectors. An initial estimate of the decision boundary is made and then improved iteratively by computing the error (the number of support vectors that violate optimality). The face detection system is trained using a set of 19x19 images containing faces and non-face patterns with associated expected outputs of '1' and '-1' respectively. Bootstrapping is used to strengthen the system; that is to say misclassified non-face images are stored for use as negative examples. Once trained, the system can classify new examples presented to it. Images are first rescaled several times and cut into 19x19 windows. Each of these 19x19 windows is then pre-processed and classified by the SVM. Windows identified as containing faces are outlines with a rectangle to indicate its location.

The results of the SVM are comparable with those of the systems proposed by Rowley et al, and Sung and Pogio. Two tests set were used to analyze the performance of the system. The first, 'Set A' gave an impressive detection rate of 97.1% with only 4 false alarms. 'Set B', with more complex backgrounds and varying lighting conditions gave a 74.2% detection rate with 20 false alarms. This approach is
though to extend to other object detection problems, through with the current limitation of memory requirements. [29]

2.5 Conclusion

The literature survey describes some of the different approaches to face detection, and should give the reader an insight into some of the techniques being applied in the field. Feature-based and image-based approaches are discussed and both have a great deal to offer the face detection problem. Feature-based systems, are better suited to real-time application where the limitations of the Image-based detectors restrict their use. Image-based systems have been shown to be more robust and more powerful, and have many uses. Despite reports of immense computational expense, recent attempts to optimize algorithms, coupled with advances in computer hardware, have reduced processing time dramatically.

Face detection is still most commonly applied as a pre-processing stage in face recognition systems. It has many potential applications in a wide range of fields, some of which are described here. Accompanied by voice data and fingerprints, face recognitions is often an integral part of Biometric Identification systems, with model-based, template-based and eigenface methods all in development. Popularity with video conferencing, a technology used extensively for worldwide corporate communication, has driven research for improvements in the area. The integration of face detection, allows a single camera to track a speaker, or a multiple camera set-up, to choose which camera to use based upon the orientation of the speaker's face, improving the experience for those participating. The increasing popularity of digital cameras and internet improvements, have spurred a growth in the amount and accessibility of digital information. Efforts have been made to catalogue this digital content search and retrieval purposes, including 'Name-it' and the 'Informedia project', both of which utilize the face detection system of Rowley et al, as does Webseer, as part of an image based search engine.
The survey details many different approaches, although, a comparative analysis of the techniques is not possible, due to the absence of a standard test procedure. The report results in each case are from tests carried out on different test sets under different conditions, making performance comparison an impossible task.
CHAPTER III

IMAGE COMPRESSION

3.1 Introduction

Compression is a process by which the description of computerized information is modified so that the capacity required to store or the bit-rate required to transmit it is reduced. Compression is carried out for the following reasons:

- Reduce storage requirement
- Reduce processing time
- Reduce transmission duration

Uncompressed multimedia (graphics, audio and video) data requires considerable storage capacity and transmission bandwidth. Despite rapid progress in mass-storage density, processor speeds, and digital communication system performance, demand for data storage capacity and data-transmission bandwidth continues to outstrip the capabilities of available technologies. The recent growth of data intensive multimedia-based web applications have not only sustained the need for more efficient ways to encode signals and images but have made compression of such signals central to storage and communication technology.

For still image compression, the ‘Joint Photographic Experts Group’ or JPEG standard has been established by ISO (International Standards Organization) and IEC (International Electro-Technical Commission). The performance of these coders generally degrades at low bit-rates mainly because of the underlying block-based Discrete Cosine Transform (DCT) scheme. [35]
The goal of this chapter is to introduce new readers to compression and review some basic concepts on image compression and present an overview of the DCT-based JPEG standard image coding scheme.

### 3.2 Image Compression

Image compression is the application of data compression on digital images. In effect, the objective is to reduce redundancy of the image data in order to be able to store or transmit data in an efficient form.

A common characteristic of most images is that their neighboring pixels are correlated and therefore contain redundant information. The foremost task is to find less correlated representation of the image. Two fundamental components of compression are *redundancy* and *irrelevancy reduction*. Redundancy reduction aims at removing duplication from the signal source (image/video). Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System (HVS). In general, three types of redundancy can be identified:

- Spatial Redundancy or correlation between neighboring pixel values.
- Spectral Redundancy or correlation between different color planes or spectral bands.
- Temporal Redundancy or correlation between adjacent frames in a sequence of images.

Image compression research aims at reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible. [2]

#### 3.2.1 Types of Image Compression

There are two types of image file compression algorithms: *Lossy* and *Lossless*.
3.2.1.1 Lossy Image Compression

A lossy data compression method is one where compressing data and then decompressing it retrieves data that may well be different from the original, but is "close enough" to be useful in some way. Lossy data compression is most commonly used to compress multimedia data (audio, video, still images) especially in applications, such as streaming media and internet telephony.

In lossy compression the decompressed information is different from the original uncompressed information. Lossless compression is also known as irreversible compression. Lossy image compression is mostly suitable for continuous media such as sound and video as well as for many images.

Most lossy data compression formats suffer from generation loss: repeatedly compressing and decompressing the file will cause it to progressively lose quality. There are two basic lossy compression schemes:

- In lossy transform codecs, samples of picture or sound are taken, chopped into small segments, transformed into a new basis space, and quantized. The resulting quantized values are then entropy coded.
- In lossy predictive codecs, previous and/or subsequent decoded data is used to predict the current sound sample or image frame. The error between the predicted data and the real data, together with any extra information needed to reproduce the prediction, is then quantized and coded.

In some systems the two techniques are combined, with transform codecs being used to compress the error signals generated by the predictive stage. Figure 3.2.1.1 shows the usage of lossy compression to reduce the file size of the image.
3.2.1.2 Lossless Image Compression

Lossless data compression is a class of data compression algorithms that allows the exact original data to be reconstructed from the compressed data. This can be contrasted to lossy data compression, which does not allow the exact original data to be reconstructed from the compressed data.

Lossless compression is used when it is important that the original and the decompressed data be identical, or when no assumption can be made on whether certain deviation is uncritical. The information is recovered without any alteration after the decompression process. The information contained in the compression bit stream is identical to the information contained in the original source bit stream. Lossless compression is also called reversible compression or bit-preserving compression.

Lossless compression methods may be categorized according to the type of data they are designed to compress. The three main types of targets for compression algorithms are text, images, and sound. Whilst, in principle, any general-purpose lossless compression algorithm (general-purpose means that they can handle all binary input)
can be used on any type of data, many are unable to achieve significant compression on data that is not of the form that they are designed to deal with. Sound data, for instance, cannot be compressed well with conventional text compression algorithms.

Most lossless compression programs use two different kinds of algorithms: one which generates a statistical model for the input data, and another which maps the input data to bit strings using this model in such a way that "probable" (e.g. frequently encountered) data will produce shorter output than "improbable" data. Often, only the former algorithm is named, while the latter is implied (through common use, standardization etc.) or unspecified. Lossless image compression coding techniques include: Huffman coding, fano coding and run length coding.

Lossless data compression is used in many applications. For example, it is used in the popular ZIP file format. It is also often used as a component within lossy data compression technologies. Typical examples are executable programs and source code. Image file formats, like PNG or GIF, use only lossless compression.

3.2.1.3 Lossy vs. Lossless Image Compression

Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely, but nevertheless perfectly. Lossless compression is possible because most real-world data has statistical redundancy. For example, in English text, the letter 'e' is much more common than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is very small.

Lossy data compression is possible if some loss of fidelity is acceptable. For example, a person viewing a picture or television video scene might not notice if some of its finest details are removed or not represented perfectly (i.e. may not even notice compression artifacts). Lossy data compression algorithms introduce relatively minor differences and represent the picture, video, or audio using fewer bits.
Lossless compression schemes are reversible so that the original data can be reconstructed, while lossy schemes accept some loss of data in order to achieve higher compression.

However, lossless data compression algorithms will always fail to compress some files; indeed, any compression algorithm will necessarily fail to compress any data containing no discernible patterns. Attempts to compress data that has been compressed already will therefore usually result in an expansion, as will attempts to compress encrypted data.

In practice, lossy data compression will also come to a point where compressing again does not work, although an extremely lossy algorithm, which for example always removes the last byte of a file, will always compress a file up to the point where it is empty.

### 3.2.2 Typical Image Encoder

A typical lossy image compression system is shown in Figure 3.2.2.1. It consists of three closely connected components namely: Source Encoder, Quantizer and Entropy Encoder. Compression is accomplished by applying a linear transform to decorrelate the image data, quantizing the resulting transform coefficients, and entropy coding the quantized values.

![Diagram](image_url)

**FIGURE 3.2.2.1 Typical lossy image encoder**
3.2.2.1 Source Encoder

The source encoder includes linear transforms such as the Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT) and many more, each with their own advantages and disadvantages.

3.2.2.2 Quantizer

A quantizer simply reduces the number of bits needed to store the transformed coefficients by reducing the precision of those values. Since this is a many-to-one mapping, it is a lossy process and is the main source of compression in an encoder. Quantization can be formed in two ways, namely:

- Scalar Quantization (SQ), where quantization is performed in each individual coefficient
- Vector Quantization (VQ), where quantization is performed on a group of coefficients together

Both uniform and non-uniform quantizers can be used depending on the problem at hand.

3.2.2.3 Entropy Encoder

An entropy encoder further compresses the quantized values losslessly to give better overall compression. It uses a model to accurately determine the probabilities for each quantized value and produces an appropriate code based on these probabilities so that the resultant output code stream will be smaller than the input stream. The most commonly used entropy encoders are the Huffman encoder and the Arithmetic encoder, although for applications requiring fast execution, simple Run-length encoding (RLE) has proven very effective.
3.2.3 JPEG: Image Coding Standard

In 1992 the Joint Photographic Experts Group (JPEG) established the first international standard for still image compression where the encoders and decoders were Discrete Cosine Transform (DCT) based.

JPEG provides lossy image compression (although there are variations on the standard baseline JPEG which are lossless). JPEG specifies both the codec, defining how an image is transformed into a stream of bytes, and the file format used to contain that stream. The file format is known as 'JPEG Interchange Format' and is specified in Annex B of the standard.

The JPEG standard specifies three modes namely: sequential, progressive, and hierarchical for lossy encoding, and one mode of lossless encoding. There are four basic steps involved in the JPEG image compression:

1) Block Preparation
2) Discrete Cosine Transform
3) Quantization
4) Encoding

3.2.3.1 JPEG Image Compression Steps

3.2.3.1.1 Block Preparation

For the block preparation an image is divided into individual blocks. A block consists of 8x8 pixels. Images are divided into blocks because each block is treated individually (compression steps are applied onto individual blocks, not onto the image as a whole). Figure 3.2.3.1.1.1 illustrates block preparation by dividing an image into a block of 8x8 pixels.
3.2.3.1.2 Discrete Cosine Transform

The block of 8x8 sampled values in space domain is transformed into another block of 8x8 coefficient values in the spectral frequency domain by applying the DCT, since it is easier to compress data in the frequency domain. Compression is based on the assumption that samples values in individual blocks of an image usually contain similar information (i.e. high coefficients that have low frequency) – spatial correlation.

3.2.3.1.3 Quantization

The block of 8x8 DCT coefficients are divided by an 8x8 quantization table. In quantization the low DCT coefficients of the high frequencies are discarded. Thus, quantization is applied to allow further compression of entropy encoding by neglecting insignificant low coefficients.
3.2.3.1.4 Encoding

To achieve higher level of compression by getting rid of long successive identical coefficient values, encoding is carried out in a zig-zag manner. For normal image compression encoding techniques such as the Run Length Coding is applied. To achieve greater level of compression Variable Length Coding can be applied.

3.2.3.2 Baseline JPEG Coder

The ‘baseline JPEG coder’ is the sequential encoding in its simplest form. Figure 3.2.3.1 and 3.2.3.2 show the key processing steps in a baseline JPEG encoder and decoder for grayscale images. Color image compression can be approximately regarded as compression of multiple grayscale images, which are either compressed entirely one at a time, or are compressed by alternately interleaving 8x8 sample blocks from each in turn.

![FIGURE 3.2.3.2.1 JPEG Encoder Block Diagram](image)
The DCT-based encoder compresses a stream of 8x8 blocks of image samples. Each 8x8 block makes its way through each processing step, and yields output in compressed form into the data stream. Because adjacent image pixels are highly correlated, the ‘Forward DCT’ (FDCT) processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. For a typical 8x8 sample block from a typical source image, most of the spatial frequencies have zero or near-zero amplitude and need not be encoded. In principle, the DCT introduces no loss to the source image samples; it merely transforms them to a domain in which they can be more efficiently encoded.

After output from the FDCT, each of the 64 DCT coefficients are uniformly quantized in conjunction with a carefully designed 64-element Quantization Table (QT). At the decoder, the quantized values are multiplied by the corresponding QT elements to recover the original unquantized values. Figure 3.2.3.3 illustrates quantization of 64 DCT coefficients with the quantization table Q(i,j). [36]
After quantization, all of the quantized coefficients are ordered into a ‘zig-zag’ sequence as shown in Figure 3.2.3.4. This ordering helps to facilitate entropy encoding by placing low-frequency non-zero coefficients before high-frequency coefficients. The DC coefficient, which contains a significant fraction of the total image energy, is differentially encoded.
Entropy Coding (EC) achieves additional compression losslessly by encoding the quantized DCT coefficients more compactly based on their statistical characteristics. The JPEG proposal specifies both Huffman coding and Arithmetic coding. The baseline sequential codec uses Huffman coding, but codecs with both methods are specified for all modes of operation. Arithmetic coding, though more complex, normally achieves 5-10% better compression than Huffman coding. [38]

3.3 Discrete Cosine Transform

3.3.1 Overview

The discovery of the Discrete Cosine Transform (DCT) in 1974 is an important achievement for the research community working on image compression. The Discrete Cosine Transform possess some fine properties, i.e., de-correlation, energy compaction, separability, symmetry and orthogonality, due to which it is virtually used in every image/video processing standard such as signal and image processing and especially for lossy data compression because it has a strong "energy compaction" property: most of the signal information tends to be concentrated in a few low-frequency components of the DCT and high-frequency components are eliminated, approaching the Karhunen-Loève Transform (KLT) for signals based on certain limits of Markov processes.

Compression standards like JPEG, for compression of still images, MPEG, for compression of motion video, MP3, for compression of audio streams and the H.263, for compression of video telephony and teleconferencing all employ the basic technique of the DCT. [36]
3.3.2 Background

The DCT is an algorithm that is widely used for data compression. Similar to Fast Fourier Transform, DCT converts data (pixels, waveforms, etc.) into sets of frequencies. The first frequencies in the set are the most meaningful; the latter, the least. To compress data, the least meaningful frequencies are stripped away based on allowable resolution loss.

Like any Fourier-related transform, discrete cosine transforms (DCTs) express a function or a signal in terms of a sum of sinusoids with different frequencies and amplitudes. Like the Discrete Fourier Transform (DFT), a DCT operates on a function at a finite number of discrete data points. The obvious distinction between a DCT and a DFT is that the former uses only cosine functions, while the latter uses both cosines and sines (in the form of complex exponentials). However, this visible difference is merely a consequence of a deeper distinction: a DCT implies different boundary conditions than the DFT or other related transforms.

The Fourier-related transforms that operate on a function over a finite domain, such as the DFT or DCT or a Fourier series, can be thought of as implicitly defining an extension of that function outside the domain. That is, once you write a function \( f(x) \) as a sum of sinusoids, you can evaluate that sum at any \( x \), even for \( x \) where the original \( f(x) \) was not specified. The DFT, like the Fourier series, implies a periodic extension of the original function. A DCT, like a cosine transform, implies an even extension of the original function. [2]

FIGURE 3.3.2.1 DCT (bottom) compared to DFT (middle) of an input signal (top)
(a) A generic sampled signal

(b) The modulus of its DFT

(c) Its DCT

However, because DCTs operate on finite, discrete sequences, two issues arise that do not for the continuous cosine transform. First, one has to specify whether the function is even or odd at both the left and right boundaries of the domain (i.e. the min-n and max-n boundaries in the definitions below, respectively). Second, one has to specify around what point the function is even or odd. In particular, consider a sequence ‘abcd’ of four equally spaced data points, and say that we specify an even left boundary. There are two sensible possibilities: either the data is even about the sample a, in which case the even extension is dcbaabcd, or the data is even about the point halfway between a and the previous point, in which case the even extension is dcbaabdc (a is repeated).

These choices lead to all the standard variations of DCTs and also discrete sine transforms (DSTs). Each boundary can be either even or odd (2 choices per boundary)
and can be symmetric about a data point or the point halfway between two data points (2 choices per boundary), for a total of $2 \times 2 \times 2 \times 2 = 16$ possibilities. Half of these possibilities, those where the left boundary is even, correspond to the 8 types of DCT; the other half are the 8 types of DST.

These different boundary conditions strongly affect the applications of the transform, and lead to uniquely useful properties for the various DCT types. Most directly, when using Fourier-related transforms to solve partial differential equations by spectral methods, the boundary conditions are directly specified as a part of the problem being solved. In a more subtle fashion, the boundary conditions are responsible for the "energy compaction" properties that make DCTs useful for image and audio compression, because the boundaries affect the rate of convergence of any Fourier-like series.

In particular, it is well known that any discontinuities in a function reduce the rate of convergence of the Fourier series, so that more sinusoids are needed to represent the function with a given accuracy. The same principle governs the usefulness of the DFT and other transforms for signal compression: the more smooth a function is, the fewer terms in its DFT or DCT are required to represent it accurately, and the more it can be compressed. However, the implicit periodicity of the DFT means that discontinuities usually occur at the boundaries: any random segment of a signal is unlikely to have the same value at both the left and right boundaries. In contrast, a DCT where both boundaries are even always yields a continuous extension at the boundaries (although the slope is generally discontinuous). This is why DCTs generally perform better signal compression than DFTs. In practice, a type-II (two-dimensional) DCT is usually preferred for such applications, in part for reasons of computational convenience. [23]
3.3.3 Definition

The DCT is regarded as a discrete-time version of the Fourier-cosine series. Hence, it is considered as a Fourier-related transform similar to the Discrete Fourier Transform (DFT), using only real numbers. Since DCT is real-valued, it provides a better approximation of a signal with fewer coefficients. Figure 3.3.3.1 illustrates the two-dimensional representation of the DCT.

![Two-dimensional representation of the DCT](image)

The DCT is a transform which transforms a signal or image from the spatial domain to the elementary frequency domain. Lower frequencies are more obvious in an image than higher frequencies an image is transferred into its frequency components and higher frequency coefficients are discarded, the amount of data needed to describe the image without sacrificing too much image quality will reduce. Thus, DCT can be computed with a Fast Fourier Transform (FFT) like algorithm. Hence it, can be concluded that:

“The DCT decorrelates image data, after which each transform coefficient is encoded independently without losing compression efficiency.”
FIGURE 3.3.3.2 Spatial (left) to frequency domain (right) conversion of an image

3.3.3.1 One-dimensional DCT

The discrete cosine transform is a linear invertible function $F: \mathbb{R}^N \rightarrow \mathbb{R}^N$ (where $\mathbb{R}$ denotes the set of real numbers), or equivalently an $N \times N$ square matrix. Mathematically, the 1D discrete cosine transform (1D DCT) $X[k]$ of a sequence $x[n]$ of length $N$ is defined as:

$$X[k] = \alpha[k] \sum_{n=0}^{N-1} x[n] \cos \left( \frac{\pi(2n+1)k}{2N} \right), \quad k = 0, 1, ..., N-1 \quad (3.1)$$

Also, the inverse 1D DCT is defined as:

$$X[k] = \sum_{n=0}^{N-1} \alpha[k] X[k] \cos \left( \frac{\pi(2n+1)k}{2N} \right), \quad n = 0, 1, ..., N-1 \quad (3.2)$$
where in both Equations 3.1 and 3.2, \( \alpha[k] \) is defined as:

\[
\alpha[k] = \begin{cases} 
\frac{1}{\sqrt{N}}, & \text{for } k = 0 \\
\frac{2}{\sqrt{N}}, & \text{for } k = 1, 2, ..., N-1 
\end{cases}
\]  

(3.3)

The basis sequences of the 1D DCT are real, discrete-time sinusoids are defined by:

\[
c_n[k] = \cos\left(\frac{\pi(2n+1)k}{2N}\right) 
\]  

(3.4)

Each element of the transformed list \( X[k] \) in equation 3.1 is the inner dot product of the input list \( x[n] \) and a basis vector. Constant factors are chosen so the basis vectors are orthogonal and normalized. The DCT can be written as the product of a vector (the input list) and the \( N \times N \) orthogonal matrix whose rows are the basis vectors.

### 3.3.3.2 Two-dimensional DCT

The two-dimensional discrete cosine transform (2D-DCT) is used for processing signals such as images. The 2D DCT resembles the 1D DCT transform since it is a separable linear transformation; that is if the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension. For e.g, in an \( n \times m \) matrix, \( S \), the 2D DCT is computed by applying it to each row of \( S \) and then to each column of the result. Figure 3.3.3.2.1 illustrates the generic 2D-DCT architecture of an \( N \times M \) input image.
Since the 2D DCT can be computed by applying 1D transforms separately to the rows and columns, hence the 2D DCT is separable in the two dimensions.

The 2-D DCT is similar to a Fourier transform but uses purely real math. It has purely real transform domain coefficients and incorporates strictly positive frequencies. The 2D DCT is equivalent to a DFT of roughly twice the length, operating on real data with even symmetry, where in some variants the input and/or output data are shifted by half a sample. As the 2D DCT is simpler to evaluate than the Fourier transform, it has become the transform of choice in image compression standards such as JPEG.

The 2D DCT represents an image as a sum of sinusoids of varying magnitudes and frequencies. It has the property that, for a typical image, most of the visually significant information about the image is concentrated in just a few coefficients of the DCT. The series form of the 2D discrete cosine transform (2D DCT) is defined as:

\[
X[k_1,k_2] = \alpha[k_1] \alpha[k_2] \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1,n_2] \cos \left( \frac{\pi(2n_1+1)k_1}{2N_1} \right) \cos \left( \frac{\pi(2n_2+1)k_2}{2N_2} \right)
\]

(3.5)

for \( k_1 = 0, 1, ..., N_1-1 \) and \( k_2 = 0, 1, ..., N_2-1 \) and,
\[ x[n_1, n_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \alpha[k_1] \alpha[k_2] X[k_1, k_2] \cos\left(\frac{\pi(2n_1 + 1)k_1}{2N_1}\right) \cos\left(\frac{\pi(2n_2 + 1)k_2}{2N_2}\right) \]

(3.6)

for \( n_1 = 0, 1, \ldots, N_1 - 1 \) and \( n_2 = 0, 1, \ldots, N_2 - 1 \), with \( \alpha[k] \) defined in equation 3.3.

Equation 3.5 is called the *analysis formula* or the ‘forward transform’, while equation 3.6 is called the *synthesis formula* or ‘inverse transform’. Mathematically, the DCT is perfectly reversible and there is no loss of image definition until coefficients are quantized.

The 2D-DCT can be efficiently regarded as a set of basis functions which given a known input array size \((n \times m)\) can be pre-computed and stored. The 2D basis matrices are the outer product of the 2D basis vectors. Each basis matrix can be thought of as an image and is characterized by a horizontal and vertical spatial frequency. This involves computing values for a convolution mask (8x8 window) that gets applied (sum values x pixels in the window which overlap with image apply a window across all rows/columns of image). The 64 (8 x 8) DCT basis functions are shown in Figure 3.3.3.2.2. [35]

\[ \text{FIGURE 3.3.3.2.2 The 64 DCT basis functions for an 8 x 8 block} \]
The pixels in the DCT image describe the proportion of each two-dimensional basis function present in the image. Each basis matrix is characterized by a horizontal and vertical spatial frequency. The matrices shown in Figure 3.3.3.2.2 are arranged left to right and top to bottom in order of decreasing frequencies. The top-left function (brightest pixel) is the basis function of the "DC" coefficient, with frequency \{0,0\} and represents zero spatial frequency. It is the average of the pixels in the input, and is typically the largest coefficient in the DCT of "natural" images. Along the top row the basis functions have increasing horizontal spatial frequency content. Down the left column the functions have increasing vertical spatial frequency content. Figure 3.3.3.2.3 illustrates the comparison between a normal picture matrix and DCT coefficients.

![Figure 3.3.3.2.3 Comparison between (a) Picture Matrix and (b) DCT Coefficients](image)

For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT as shown in Figure 3.3.3.2.2. Compression is achieved since the lower right values represent higher frequencies, and are often very small enough to be neglected with little visible distortion.
The 2D DCT however, is partly restricted to compute to a limited size when it is used for image compression. Rather than taking the transform of the image as a whole, the DCT is applied separately to 8 x 8 blocks of the image. To compute the 2D blocked DCT, the image is not actually divided into blocks, since the 2D-DCT is separable, a partition of each row into lists of length 8 is performed, and the DCT is applied to them and the resulting lists are rejoined and then the whole image is transposed by repeating the process.

For an $N \times M$ input image, the DCT input is an 8 x 8 array of integers. This array contains each pixel's gray scale level, where the 8 bit pixels have levels from 0 to 255. When the two-dimensional blocked DCT of 8 x 8 blocks is computed, the coefficients are quantized, the entropy is coded and transmitted. Generally, $N = 8$ and the DCT formula is applied to each row and column of the block. Figure 3.3.3.2.4 illustrates a 2D blocked DCT by dividing the face image into 8x8 blocks.

![FIGURE 3.3.3.2.4 Illustration of 2D blocked DCT of face image with 8x8 blocks](image-url)
In Figure 3.3.3.2.4 the upper left corner illustrates the face image divided into 8x8 blocks. Each block further consists of 8x8 pixels, 64 pixels altogether in each separate block. The 64 pixels of 3 random blocks are hence represented and shown in the upper right and lower region of Figure 3.3.4.4.

### 3.3.4 DCT Image Compression

Since the DCT algorithm is used for JPEG image compression, the input image is firstly divided into 8 x 8 blocks and the 2D-DCT for each block is separately computed. The DCT coefficients for each block are quantized separately by discarding redundant information, high-coefficients are discarded. The receiver decodes the quantized DCT coefficients of each block separately and computes the 2D-IDCT of each block and then puts the blocks back together into a single image.

The image file size is also reduced by dividing the coefficients by a quantization matrix in order to get long runs of zeros. Quantized coefficients are encoded separately using a lossless method (RLC, Huffman, Arithmetic coding etc). Although there is some loss of quality in the reconstructed image, it is recognizable as an approximation of the original image. [14]

An 8-bit grayscale face image of a subject in JPEG format with dimension 150 x 177 pixels is chosen to be compressed. The image is divided into 8 x 8 blocks and the 2D-DCT for each block is computed. Each block consists of 8 x 8 pixels. After DCT computation a lattice of bright dots is formed by DC coefficients from each block as shown in Figure 3.3.4.1

The DCT transformed image is masked. Only 8 DCT coefficients are selected out of the 64 for masking. Lesser the number of DCT coefficients selected for masking, lesser will be the compression and quality deterioration of the compressed image.
The masked image is generated which is concentrated on few low-frequency components as high frequency components are discarded during quantization. The masked image is reconstructed using the 2D-IDCT transform. The resultant image is compressed image, which is blurred due to the loss of quality, evidently showing the block structure. Figure 3.3.4.1 illustrates this 2D-DCT compression example. [14]

FIGURE 3.3.5.1 2D-DCT and 2D-IDCT computed on a sample face image
3.3.5 Applications of DCT

The DCT has a number of applications in information theory. DCT is widely used in JPEG image compression, MJPEG, MPEG, and DV video compression. DCTs are widely employed in solving partial differential equations by spectral methods, where the different variants of the DCT correspond to slightly different even/odd boundary conditions at the two ends of the array.

DCT’s are also closely related to Chebyshev polynomials, and fast DCT algorithms are used in Chebyshev approximation of arbitrary functions by series of Chebyshev polynomials, for example in Clenshaw-Curtis quadrature.

3.4 Conclusion

Since high information redundancy and correlation in face images results in inefficiencies when such images are used face recognition, DCTs can be used to reduce image information redundancy because only a subset of the transform coefficients are necessary to preserve the most important facial features such as hair outline, eyes and mouth. It will be demonstrated experimentally further in this report that when DCT coefficients are fed into a neural network, an efficient high-speed face recognition rate can be achieved by using a very small proportion of transform coefficients. This makes DCT-based face recognition much faster than other approaches.
CHAPTER IV

ARTIFICIAL NEURAL NETWORKS

As discussed in the Literature Survey, there are many different approaches to face detection, each with their own relative merits and limitations. One such approach is that of Neural Networks. This chapter gives an introduction to Artificial Neural Networks. The various types of neural networks are explained and demonstrated, applications of neural networks are described, and a detailed historical background is provided. This section presents the subtype neural network-based to be used for the face recognition system in this project.

4.1 Introduction

Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain. The brain basically learns from experience. It is natural proof that some problems that are beyond the scope of current computers are indeed solvable by small energy efficient packages. This brain modeling also promises a less technical way to develop machine solutions. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts.

These biologically inspired methods of computing are thought to be the next major advancement in the computing industry. Even simple animal brains are capable of functions that are currently impossible for computers. Computers do rote things well, like keeping ledgers or performing complex math. But computers have trouble recognizing even simple patterns much less generalizing those patterns of the past into actions of the future. [3]
Now, advances in biological research promise an initial understanding of the natural thinking mechanism. This research shows that brains store information as patterns. Some of these patterns are very complicated and allow us the ability to recognize individual faces from many different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing. This field, as mentioned before, does not utilize traditional programming but involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing, words like behave, react, self-organize, learn, generalize, and forget.

4.1.1 Historical Background

The history of neural networks that was described above can be divided into several periods.

4.1.1.1 Initial Attempts

There were some initial simulations using formal logic. McCulloch and Pitts in 1943 developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "a or b" and "a and b". Another attempt was by using computer simulations by two groups: Farley and Clark, 1954 and Rochester, Holland, Haibit and Duda, 1956. The first group (IBM researchers) maintained closed contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a multidisciplinary trend which continues to the present day. [3]
4.1.1.2 Promising and Emerging Technology

Not only was neuroscience influential in the development of neural networks, but psychologists and engineers also contributed to the progress of neural network simulations. Rosenblatt in 1958 stirred considerable interest and activity in the field when he designed and developed the Perceptron. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. Another system was the ADALINE (ADAptive LIinear Element) which was developed in 1960 by Widrow and Hoff, of Stanford University. The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron, it employed the Least-Mean-Squares (LMS) learning rule.

4.1.1.3 Period of Frustration and Disrepute

In 1969 Minsky and Papert wrote a book in which they generalized the limitations of single layer Perceptrons to multilayered systems. In the book they said: "Our intuitive judgment that the extension (to multilayer systems) is sterile". The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.

4.1.1.4 Innovation

Although public interest and available funding were minimal, several researchers continued working to develop neuromorphically based computational methods for problems such as pattern recognition. During this period several paradigms were generated which modern work continues to enhance. In 1988, Steve Grossberg and Gail Carpenter were influenced and founded a school of thought which explored resonating algorithms. They developed the ART (Adaptive Resonance Theory) networks based on biologically plausible models. Anderson and Kohonen developed
associative techniques independent of each other. Henry Klopf in 1972, developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis. In 1974, Paul Werbos developed and used the back-propagation learning method, however several years passed before this approach was popularized. Back-propagation nets are probably the most well known and widely applied of the neural networks today. In essence, the back-propagation network is a Perceptron with multiple layers, a different threshold function in the artificial neuron, and a more robust and capable learning rule. In 1967, Amari Shun-Ichi was involved with theoretical developments. He published a paper which established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification. While Fukushima Kunihiko developed a step wise trained multilayered neural network for interpretation of handwritten characters. The original network was published in 1975 and was called the Cognitron. [27]

4.1.1.5 Re-emergence

Progress during the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology. Academic programs appeared and courses were introduced at most major Universities. Attention is now focused on funding levels throughout the world, and as this funding becomes available, several new commercial with applications in industry and financial institutions are emerging.
4.1.1.6 Today

Significant progress has been made in the field of neural networks—enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neurally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

4.2 Human and Artificial Neurons

4.2.1 Biological Inspiration

The exact workings of the human brain are still a mystery. Yet, some aspects of this amazing processor are known. In particular, the most basic element of the human brain is a specific type of cell which, unlike the rest of the body, doesn't appear to regenerate. Because this type of cell is the only part of the body that isn't slowly replaced, it is assumed that these cells provide us with our abilities to remember, think, and apply previous experiences to our every action. These cells, all 100 billion of them, are known as neurons. Each of these neurons can connect with up to 200,000 other neurons.

The power of the human mind comes from the sheer numbers of these basic components and the multiple connections between them. It also comes from genetic programming and learning.

The individual neurons are complicated. They have a myriad of parts, sub-systems, and control mechanisms. They convey information via a host of electrochemical pathways. There are over one hundred different classes of neurons, depending on the classification method used. Together these neurons and their connections form a
process which is not binary, not stable, and not synchronous. In short, it is nothing like the currently available electronic computers, or even artificial neural networks.

These artificial neural networks try to replicate only the most basic elements of this complicated, versatile, and powerful organism. They do it in a primitive way. But for the software engineer who is trying to solve problems, neural computing was never about replicating human brains. It is about machines and a new way to solve problems. [26]

4.2.2 Human Neurons

The fundamental processing element of a neural network is a neuron. This building block of human awareness encompasses a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then outputs the final result. A simple neuron consists of four parts: dendrites, soma, axon and synapses, as shown in Figure 4.2.2.1.

![Diagram of a simple neuron](image)

FIGURE 4.2.2.1 A simple neuron
Within humans there are many variations on this basic type of neuron, further complicating man's attempts at electrically replicating the process of thinking. Yet, all natural neurons have these same four basic components, which are known by their biological names.

A typical neuron collects signals from others through a host of fine structures called *dendrites*. Dendrites are hair-like extensions of the soma which act like input channels. The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches. At the end of each branch, a structure called *synapses* converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. The *soma* then processes these incoming input signals over time. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity to other neurons through the axon and synapses. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. Figure 4.2.2.2 shows the general structure of a neuron cell. [25]

![General structure of a neuron cell](image)

Recent experimental data has provided further evidence that biological neurons are structurally more complex than the simplistic explanation above. They are significantly more complex than the existing artificial neurons that are built into
today's artificial neural networks. As biology provides a better understanding of neurons, and as technology advances, network designers can continue to improve their systems by building upon man's understanding of the biological brain.

The current goal of artificial neural networks is not the grandiose recreation of the brain. On the contrary, neural network researchers are seeking an understanding of nature's capabilities for which people can engineer solutions to problems that have not been solved by traditional computing.

4.2.3 Artificial Neurons – An Engineering Approach

Artificial neurons to be used neural networks are first deduced by obtaining the essential features of human neurons and their interconnections. Typically a computer program is used to simulate these features. However, because our knowledge of neurons is incomplete and our computing power is limited, and our models are gross idealizations of real networks of neurons. Figure 4.2.3.1 illustrates a typical artificial neuron with respect to Figure 4.2.2.1.
Since artificial neurons simulate the four basic functions of natural neurons: dendrites, soma, axon and synapses, Figure 4.2.3.2 shows a fundamental representation of an artificial neuron with respect to an engineering approach.

**FIGURE 4.2.3.2** Artificial neuron or a single neural processor

In Figure 4.2.3.2, various inputs to the network are represented by the mathematical symbol, i(n). Each of these inputs is multiplied by a connection weight. These weights are represented by w(n). In the simplest case, these products are simply summed together, fed through a transfer function to generate a result, and then output. This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures which utilize different summing functions as well as different transfer functions.

Some applications require “black and white,” or binary, answers. These applications include the recognition of text, the identification of speech, and the image deciphering of scenes. These applications are required to turn real-world inputs into discrete values. These potential values are limited to some known set, like the ASCII characters or the most common 50,000 English words. Because of this limitation of
output options, these applications don’t always utilize networks composed of neurons that simply sum up, and thereby smooth, inputs. These networks may utilize the binary properties of ORing and ANDing of inputs. These functions, and many others, can be built into the summation and transfer functions of a network. [40]

Other networks work on problems where the resolutions are not just one of several known values. These networks need to be capable of an infinite number of responses. Applications of this type include the “intelligence” behind robotic movements. This “intelligence” processes inputs and then creates outputs which actually cause some device to move. That movement can span an infinite number of very precise motions. These networks do indeed want to smooth their inputs which, due to limitations of sensors come in non-continuous bursts, say thirty times a second. To do that, they might accept these inputs, sum that data, and then produce an output by, for example, applying a hyperbolic tangent as a transfer function. In this manner, output values from the network are continuous and satisfy more real world interfaces.

Other applications might simply sum and compare to a threshold, thereby producing one of two possible outputs, a zero or a one. Other functions scale the outputs to match the application, such as the values minus one and one. Some functions even integrate the input data over time, creating time-dependent networks.

### 4.2.4 Electronic Implementation of Artificial Neurons

In currently available software packages such as MATLAB, artificial neurons are called "processing elements" and have many more capabilities than the simple artificial neuron described above. Figure 4.2.4.1 is a more detailed schematic of a simplistic artificial neuron.
In Figure 4.2.4.1, inputs enter into the processing element from the upper left side. The first step is for each of these inputs to be multiplied by their respective weighting factor \( w(n) \). Then these modified inputs are fed into the summing function, which usually just sums these products. Yet, many different types of operations can be selected. These operations could produce a number of different values which are then propagated forward; values such as the average, the largest, the smallest, the ORed values, the ANDed values, etc. Furthermore, most commercial development products allow software engineers to create their own summing functions via routines coded in a higher level language. Sometimes the summing function is further complicated by the addition of an activation function which enables the summing function to operate in a time sensitive way.
Either way, the output of the summing function is then sent into a transfer function. This function then turns this number into a real output via some algorithm. It is this algorithm that takes the input and turns it into a zero or a one, a minus one or a one, or some other number. The transfer functions that are commonly supported are sigmoid, sine, hyperbolic tangent, etc. This transfer function also can scale the output or control its value via thresholds. The result of the transfer function is usually the direct output of the processing element. An example of a sigmoid transfer function is shown in Figure 4.2.4.2. [25]

![Sigmoid transfer function](image)

**FIGURE 4.2.4.2 Sigmoid transfer function**

This sigmoid transfer function takes the value from the summation function, called sum in Figure 4.2.4.2, and turns it into a value between zero and one.

Finally, the processing element is ready to output the result of its transfer function. This output is then input into other processing elements, or to an outside connection, as dictated by the structure of the network.

All artificial neural networks are constructed from this basic building block - the processing element or the artificial neuron. It is variety and the fundamental differences in these building blocks which partially cause the implementing of neural networks to be an "art."
4.3 Neural Networks

4.3.1 Definition

An Artificial Neural Network (ANN) or commonly known as Neural Network (NN) is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation. In most cases an artificial neural network is an adaptive system that changes its structure based on external or internal information that flows through the network.

Artificial neural networks are a type of artificial intelligence that attempt to imitate the way a human brain works. However, the human brain is a highly complex, nonlinear and parallel information processor. The speed of a neuron is approximately $10^{-3}$ seconds versus the speed of silicon, approximately $10^{-9}$ seconds. The fast response is due to the number of neurons and connectivity. In rough analogy, neural networks are built of a densely set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produce a single real-valued output (which may become the input of many other units). [3]

These networks, similar to biological neural networks perform functions collectively and parallelly by units, rather than performing a clear delineation of subtasks to which various units are assigned. Currently, the term Artificial Neural Network (ANN) tends to refer mostly to neural network models employed in statistics, cognitive psychology and artificial intelligence.

Rather than using a digital model, in which all computations manipulate zeros and ones, a neural network works by creating connections between processing elements, the computer equivalent of neurons. The organization and weights of the connections determine the output.
The original inspiration for the artificial neural network technique is from examining of the human central nervous system and its neurons (and their axons, somas, dendrites and synapses) which constitute one of the most significant information processing elements. In a neural network model, simple nodes \textit{(neurons)} and PEs \textit{(processing elements)} are connected together to form a network of nodes, hence the term "neural network".

In practical terms neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. An artificial neural network involves a network of simple processing elements \textit{(neurons)} which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. They are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. Figure 4.3.1.1 shows a general neural network architecture; interconnected group of nodes.

![General neural network architecture](image)

Neural networks are powerful pattern classifiers and have many similarities with statistical pattern recognition approaches. For this reason neural networks are
increasingly being used in pattern recognition systems, since they have a better performance in non-linear applications. Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output as illustrated in Figure 4.3.1.2. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are needed to train a network.

![Image of Artificial Neural Network](image)

**FIGURE 4.3.1.2 Ideal design flowchart of an Artificial Neural Network**

The neural network design part consists of two processes, one for training and one for application. Training requires several hours more time than does the application. The training of the neural network continues until the mean square error of the weights diminishes below a certain threshold or until the maximum number of iterations is reached. Neural networks can been trained to perform complex functions in various fields, including pattern recognition, identification, classification, speech, vision, and control systems.

Neural networks can be *single layer* or *multi-layered*. A single layer neural network usually consists of only inputs and outputs, where the inputs are fed to the outputs through a series of weights. A multi-layered neural network consists of inputs, hidden layers and outputs. [28]

However, in modern software implementations of artificial neural networks the approach inspired by biology has more or less been abandoned for a more practical
approach based on statistics and signal processing. In some of these systems neural networks, or parts of neural networks (such as artificial neurons) are used as components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such adaptive systems is more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models. What they do however have in common is the principle of nonlinear, distributed, parallel and local processing and adaptation.

4.3.2 Neural Network Models

Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs). These are essentially simple mathematical models defining a function, $f: X \rightarrow Y$. Each type of ANN model corresponds to a class of such functions.

4.3.2.1 The Network in Artificial Neural Network

The word network in the term 'artificial neural network' arises because the function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where:

$$f(x) = K\left(\sum w_i g_i(x)\right) \quad (4.1)\$$

where, $K$ is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions $g_i$ as simply a vector:

$$g = (g_1, g_2, \ldots, g_n) \quad (4.2)$$
Figure 4.3.2.1.1 depicts such a decomposition of $f$, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

![FIGURE 4.3.2.1.1 Feedforward artificial neural network dependency graph](image)

In Figure 4.3.2.1.1, the first view is the functional view: the input $x$ is transformed into a 3-dimensional vector $h$, which is then transformed into a 2-dimensional vector $g$, which is finally transformed into $f$. This view is most commonly encountered in the context of optimization.

The second view in Figure 4.3.2.1.1, is the probabilistic view: the random variable $F = f(G)$ depends upon the random variable $G = g(H)$, which depends upon $H = h(X)$, which depends upon the random variable $X$. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of $g$ are independent of each other given their input $h$). This naturally enables a degree of parallelism in the implementation.

Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called
Recurrent. Figure 4.3.2.1.2 shows the dependency graph of a recurrent artificial neural network. [28]

Recurrent networks as shown in Figure 4.3.2.1.2, has \( f \) as dependent upon itself. However, there is an implied temporal dependence which is not shown. What this actually means in practice is that the value of \( f \) at some point in time \( t \) depends upon the values of \( f \) at zero or at one or more other points in time. The graphical model at the bottom of Figure 4.3.2.1.2 illustrates the case: the value of \( f \) at time \( t \) only depends upon its last value. Models such as these, which have no dependencies in the future, are called causal models.

### 4.3.3 Neural Network Training

Once a network has been structured for a particular application, that network is ready to be trained. To start this process the initial weights are chosen randomly. Then, the training, or learning, begins.
4.3.3.1 Definition

In neural networks is the possibility of learning in practice means the following:

Given a specific task to solve, and a class of functions \( F \), learning means using a set of observations, in order to find: \( f^* \in F \), which solves the task in an optimal sense.

This entails defining a cost function \( C: F \rightarrow \mathbb{R} \), such that, for the optimal solution \( f^* \), \( C(f^*) \leq C(f) \ \forall \ f \in F \) (no solution has a cost less than the cost of the optimal solution).

The cost function \( C \) is an important concept in learning, as it is a measure of how far away we are from an optimal solution to the problem that we want to solve. Learning algorithms search through the solution space in order to find a function that has the smallest possible cost.

For applications where the solution is dependent on some data, the cost must necessarily be a function of the observations; otherwise we would not be modeling anything related to the data. It is frequently defined as a statistic to which only approximations can be made. As a simple example, consider the problem of finding the model \( f \) which minimizes: [26]

\[
C = E \left[ f(x) - y \right]^2
\]

(4.3)

For data pairs \((x,y)\) drawn from some distribution \( D \). In practical situations we would only have \( N \) samples from \( D \) and thus, for the above example, we would only minimize:

\[
\hat{C} = \frac{1}{N} \sum_{i=1}^{N} \left| f(x_i) - y_i \right|^2
\]

(4.4)
Thus, the cost is minimized over a sample of the data rather than the true data distribution. When $N \rightarrow \infty$, some form of online learning must be used, where the cost is partially minimized as each new example is seen. While online learning is often used when $D$ is fixed, it is most useful in the case where the distribution changes slowly over time. In neural network methods, some form of online learning is frequently also used for finite datasets.

### 4.3.3.2 Choosing a Cost Function

While it is possible to arbitrarily define some ad hoc cost function, frequently a particular cost will be used either because it has desirable properties (such as convexity) or because it arises naturally from a particular formulation of the problem (i.e., in a probabilistic formulation the posterior probability of the model can be used as an inverse cost). Ultimately, the cost function will depend on the task we wish to perform. The three main categories of learning tasks are overviewed below.

### 4.3.3.3 Training Process

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms:

1) Associative Mapping
2) Regularity Detection

#### 4.3.3.3.1 Associative Mapping

Associative mapping is one in which the network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. Associative mapping can generally be broken down into two mechanisms:
1) Auto-association  
2) Hetero-association

4.3.3.3.1 Auto-association

In auto-association, an input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern completion, i.e. to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns.

4.3.3.3.1.2 Hetero-association

Hetero-association is further related to two recall mechanisms:

1) Nearest-neighbor  
2) Interpolative

4.3.3.3.1.2.1 Nearest-neighbor

Nearest-neighbor recall is where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented.

4.3.3.3.1.2.2 Interpolative

Interpolative recall is where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping, is classification, i.e. when there is a fixed set of categories into which the input patterns are to be classified.
4.3.3.3.2 Regularity Detection

In regularity detection units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular 'meaning'. This type of learning mechanism is essential for feature discovery and knowledge representation.

4.3.3.3 Learning as Determination of Weights

Information is stored in the weight matrix $W$ of a neural network. Learning is the determination of the weights. Following the way learning is performed, we can distinguish two major categories of neural networks:

4.3.3.3.1 Fixed Networks

Fixed networks are networks in which the weights cannot be changed, i.e. $dW/dt = 0$. In such networks, the weights are fixed a priori according to the problem to solve.

4.3.3.3.2 Adaptive Networks

Adaptive networks are networks which are able to change their weights, i.e. $dW/dt \neq 0$.

4.3.3.4 Learning Paradigms

There are three major learning paradigms, each corresponding to a particular abstract learning task. These are: supervised learning, unsupervised learning and reinforcement learning. Usually any given type of network architecture can be employed in any of those tasks.
4.3.3.4.1 Supervised Learning

Supervised learning constitutes of a given a set of example pairs \((x,y), x \in X, y \in Y\) and the aim is to find a function \(f\) in the allowed class of functions that matches the examples. In other words, we wish to infer the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.

A commonly used cost is the mean-squared error which tries to minimize the average error between the network's output, \(f(x)\), and the target value \(y\) over all the example pairs. When one tries to minimize this cost using gradient descent for the class of neural networks called Multi-Layer Perceptrons, one obtains the well-known back-propagation algorithm for training neural networks.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and gesture recognition). This can be thought of as learning with a "teacher," in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

4.3.3.4.2 Unsupervised Learning

In unsupervised learning we are given some data \(x\), and the cost function to be minimized can be any function of the data \(x\) and the network's output, \(f\).

The cost function is dependent on the task (what we are trying to model) and our a priori assumptions (the implicit properties of our model, its parameters and the observed variables).
As a trivial example, consider the model \( f(x) = a \), where \( a \) is a constant and the cost, \( C = (E[x] - f(x))^2 \). Minimizing this cost will give us a value of \( a \), that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: For example in compression it could be related to the mutual information between \( x \) and \( y \). In statistical modeling, it could be related to the posterior probability of the model given the data. (Note that in both of those examples those quantities would be maximized rather than minimized)

Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

### 4.3.3.4.3 Reinforcement Learning

In reinforcement learning, data \( x \) is usually not given, but generated by an agent's interactions with the environment. At each point in time \( t \), the agent performs an action \( y_t \) and the environment generates an observation \( x_t \) and an instantaneous cost \( c_t \), according to some (usually unknown) dynamics. The aim is to discover a policy for selecting actions that minimises some measure of a long-term cost, i.e. the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated.

More formally, the environment is modeled as a Markov Decision Process (MDP) with states \( s_1, \ldots, s_n \) and actions \( a_1, \ldots, a_m \in A \) with the following probability distributions: the instantaneous cost distribution \( P(c_t|s_t) \), the observation distribution \( P(x_t|s_t) \), and the transition \( P(s_{t+1}|s_t, a_t) \), while a policy is defined as conditional distribution over actions given the observations. Taken together, the two define a Markov Chain (MC). The aim is to discover the policy that minimises the cost, i.e. the MC for which the cost is minimal.
Artificial neural networks are frequently used in reinforcement learning as part of the overall algorithm. Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks.

4.3.3.5 Training Function

The behavior of an artificial neural network depends on both the weights and the input-output function, *transfer function* that is specified for the units. This function typically falls into one of three categories:

1) Linear (or ramp)
2) Threshold
3) Sigmoid

For *linear units*, the output activity is proportional to the total weighted output.

For *threshold units*, the output is a set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

For *sigmoid units*, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

To make a neural network that performs some specific task, the method of connecting units to one another should be chosen, and the weights on the connections must be set properly. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.
4.3.3.6 Learning Algorithms

Training a neural network model essentially means selecting one model from the set of allowed models (or, in a Bayesian framework, determining a distribution over the set of allowed models) that minimises the cost criterion. There are numerous algorithms available for training neural network models; most of them can be viewed as a straightforward application of optimization theory and statistical estimation.

Most of the algorithms used in training artificial neural networks are employing some form of gradient descent. This is done by simply taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient-related direction.

Evolutionary methods, simulated annealing, and Expectation-maximization and non-parametric methods are among other commonly used methods for training neural networks.

4.3.4 Types of Neural Networks

There are four broad categories of neural networks, mainly: [3]

1) Feed-forward
2) Recurrent
3) Stochastic
4) Modular

4.3.4.1 Feed-forward Neural Networks

Feedforward neural networks are the first and arguably simplest type of artificial neural networks devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the
output nodes. There are no cycles or loops in the network. Feed-forward networks can be further divided into subtypes of neural networks, mainly:

1) Single-layer Perceptron
2) Multi-layer Perceptron
3) ADALINE
4) Radial Basis Function (RBF) network
5) Kohonen Self-Organizing Map (SOM)

4.3.4.1.1 Single-layer Perceptron

The earliest kind of neural network is a Single-layer Perceptron (SLP) network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called McCulloch-Pitts neurons or threshold neurons. In the literature the term perceptron often refers to networks consisting of just one of these units. They were described by Warren McCulloch and Walter Pitts in the 1940s. Figure 4.3.4.1.1.1 shows a single-layer feedforward perceptron neural network. [3]
Figure 4.3.4.1.1 depicts a single-layered neural network architecture consisting of input nodes and output nodes. Although there are two layers of nodes in the neural network structure, it is a single-layered neural network because only the nodes in the output layer are neurons. The nodes in the input layer do not perform any computation. Their only purpose is to register the values of the input neurons that are applied to the neural network architecture. The input layer is frequently referred to as Layer 0, while the output layer is referred as Layer 1.

As it is noticeable from Figure 4.3.4.1.1 weight connections are shown emanating from Layer 0 and converging to Layer 1. In other words, weights are shown from a layer of certain index to a layer of higher index only. No weight connections are allowed amongst the nodes belonging to the same layer. These types of connections are allowed amongst the nodes belonging to the same layer. These types of connection characteristic of the layered structures are they are denoted feed-forward connections. That is also the reason why these architectures are referred to as single-layered feedforward neural networks. The idea is that signals (patterns) propagate through this structure forward from a layer of certain index to a layer of higher index.
A perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two. Most perceptrons have outputs of 1 or -1 with a threshold of 0 and there is some evidence that such networks can be trained more quickly than networks created from nodes with different activation and deactivation values.

Perceptrons can be trained by a simple learning algorithm that is usually called the delta rule. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

Single-unit perceptrons are only capable of learning linearly separable patterns; in 1969 in a famous monograph entitled Perceptrons Marvin Minsky and Seymour Papert showed that it was impossible for a single-layer perceptron network to learn an XOR function. They conjectured (incorrectly) that a similar result would hold for a multi-layer perceptron network. Although a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1].

A single-layer neural network can compute a continuous output instead of a step function. A common choice is the so-called logistic function:

\[ y = \frac{1}{1 + e^{-x}} \]  

With this choice, the single-layer network is identical to the logistic regression model, widely used in statistical modelling. The logistic function is also known as the sigmoid function. It has a continuous derivative, which allows it to be used in backpropagation. This function is also preferred because its derivative is easily calculated:
$$y' = y(1 - y)$$ (4.6)

4.3.4.1.2 Multi-layer Perceptron

A Multi-layer Perceptron (MLP) network consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions.

Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Here the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles the network will usually converge to some state where the error of the calculations is small. In this case one says that the network has learned a certain target function. To adjust weights properly one applies a general method for non-linear optimization that is called gradient descent. For this, the derivative of the error function with respect to the network weights is calculated and the weights are then changed such that the error decreases. For this reason back-propagation can only be applied on networks with differentiable activation functions. Figure 4.3.4.1.2.1 shows a single-layer feedforward perceptron neural network. [3]
In general the problem of teaching a network to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The danger is that the network over fits the training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping, often ensures that the network will generalize well to examples not in the training set.

Other typical problems of the back-propagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function. Today there are practical solutions that make back-propagation in multi-layer perceptrons the solution of choice for many machine learning tasks.
4.3.4.1.3 ADALINE

Adaptive Linear Neuron or later called *Adaptive Linear Element*. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It's based on the McCulloch-Pitts model. It consists of a weight, a bias and a summation function.

\[ y_i = w x_i + b \]

(4.7)

Its adaptation is defined through a cost function (error metric) of the residual, \( e = d_i - (b + w x_i) \), where \( d_i \) is the desired input. With the MSE error metric,

\[ E = \frac{1}{2N} \sum_i^N e_i^2 \]

(4.8)

the adapted weight and bias becomes:

\[ b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i d_i}{N(\sum_i (x_i - \bar{x})^2)} \quad \text{and} \quad \]

\[ w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2} \]

(4.9)

(4.10)

While the Adaline is through this capable of simple linear regression, it has limited practical use. There is an extension of the Adaline, called the Multiple Adaline (MADALINE) that consists of two or more adalines serially connected.
4.3.4.1.4 Radial Basis Function Networks

Radial basis function (RBF) networks were introduced into the neural network literature by Broomhead/Lowe and Poggio/Girosi in the late 1980s. Radial basis function network models are motivated by the locally tuned response observed in biologic neurons, e.g. in the visual or in the auditory system. RBFs have been studied in multivariate approximation theory, particularly in the field of function interpolation. RBF neural network models are an alternative to multilayer perceptron which is perhaps the most commonly used form of neural network architecture.

A radial basis function neural network is basically a structure that represents the idea show in Figure 4.3.4.1.4.1. A RBF neural network can be seen as another type of feed-forward artificial neural network. Additionally, it uses the technology of ANN training to identify the values of the mentioned parameters, and a clustering algorithm to identify positions of centers. [15]

Typically in a RBF network, there are three layers: one input, one hidden and one output layer as shown in Figure 4.3.4.1.4.1. The hidden layer uses the Gaussian transfer function instead of the sigmoid function. In RBF networks, one major advantage is that if the number of input variables is not too high, the leaning is much faster than other types of networks. However, the required number of hidden units increases geometrically with the number of the input variables. It becomes practically impossible to use this network for a large number of input variables. Compared to the feed-forward network, the RBF network is the next-most-used network model. As the name implies, this network makes use of radial functions.
Radial basis functions represent a special class of functions. They could be seen as a sensible alternative to the mentioned attempts to use complex polynomials for function fitting. Consider a function $z = f(x)$, where, $x$ is a vector $\{x_1, ..., x_I\}$ in an $I$-dimensional space. The idea is approximate a function $z = f(x)$ by another function $F(x)$ in a proximity to some “representative” locations (centers) $w_j$, $j=1, ..., J$. Figure 4.3.4.1.4.2 shows the approximation by RBFs in one dimension.
Such basis functions $F(x)$ depend only on the distance (radius) from these centers, are identical and are defined everywhere but quickly drop to zero as the distance from the location $w_i$ increases. For some point $x$ the approximating function is then always expressed as the sum of the basis functions:

$$z(x) = \sum_{j=1}^{J} F\left(\|x - w_j\| b_j \right)$$  \hspace{1cm} (4.11)

where, $b_j$ are coefficients associated with the $j$-th center $w_j$. If the centers are fixed and the centers are entered linearly, then finding an RBF becomes a linear problem:

$$z(x) = \sum_{j=1}^{J} b_j F\left(\|x - w_j\| \right)$$  \hspace{1cm} (4.12)

Common choice for these $F$ is the fixed-width Gaussian ($f(r) = \exp(-r^2)$). Other functions that can be used include linear ($f(r) = r$) and cubic ($f(r) = r^3$). In order to allow for higher accuracy, we can introduce the width parameter ‘$\sigma$’ into Gaussian functions for each center: $f(r) = \exp\left(-\frac{r^2}{\sigma^2}\right)$. The distance $|x - w_j|$ is usually understood in Euclidean sense and denoted as $\|x - w_j\|$: [15]

$$\delta_j = \sum_{i=1}^{J} \left(x_i - w_{ij}\right)^2$$  \hspace{1cm} (4.13)

In this case the approximation becomes:

$$z(x) = \sum_{j=1}^{J} b_j \exp\left(-\frac{\delta_j^2}{\sigma_j^2}\right)$$  \hspace{1cm} (4.14)

where, ‘$\sigma$’ controls the Gaussian’s width

The problem of approximation requires:
• The placement of the localized Gaussians to cover the space (positions of centers: \( w_i \))
• The control of the width of each Gaussian (parameter: \( \sigma \))
• The setting of the amplitude of each Gaussian (parameter: \( b_i \))

The radial basis function’s response monotonically decreases (increases) as the distance from the central point grows. A typical example is Gauss function, which has the following analytical representation for a scalar argument:

\[
h(x) = \exp\left( -\frac{(x - c)^2}{2 \sigma^2} \right)
\]  

(4.15)

For the case when \( c=0 \) and \( \sigma=1 \), the function graph is shown in Figure 4.3.4.1.4.3.

![A Gauss function](image)

FIGURE 4.3.4.1.4.3 A Gauss function

Other functions can also serve as examples of radial basis functions:

\[
h(x) = \left( x^2 + \sigma^2 \right)^{-a}, \quad a \geq 0 \quad \text{and}
\]

(4.16)

\[
h(x) = \sqrt{x^2 + \sigma^2}
\]

(4.17)

In case of vector variable \( x \) the formulae are only a little bit more complicated. For example, for Gauss function the formula is:
\[ h(\overline{x}) = \exp \left( -\frac{\|\overline{x} - \overline{c}\|^2}{r^2} \right) \quad (4.18) \]

RBF architecture can be of two forms: normalized and un-normalized. These forms can be further expanded into a superposition of local linear models. RBFs are a means for interpolation in a stream of data. They differ from statistical approaches, approximations must be performed on streams of data rather than on complete data sets. RBFs use supervised learning and sometimes unsupervised learning to minimize approximation error in a stream of data. They are used in function approximation, time series prediction and control.

RBFs are powerful techniques for interpolation in a multidimensional space. A RBF is a function which has built into a distance criterion with respect to a center. RBF networks have 2 layers of processing: in the first the layer the input is mapped onto each RBF in the hidden layer. The RBF chosen is usually a Gaussian. In regression problems the output layer is the linear combination of the hidden layer values representing the mean predicted output. The interpretation of this output layer value is the same as a regression model in statistics. In classification problems the output layer is typically a sigmoid function of a linear combination of hidden layer values representing a posterior probability. Performance in both cases is often improved by shrinkage techniques known as ridge regression in classical statistics and known to correspond to a prior belief in small parameter values in a Bayesian framework. [15]

The hidden layer in RBF networks consists of an array of nodes that contains a parameter vector called a *radial center* vector. The hidden layer performs a fixed non-linear transformation with non-adjustable parameters. The approximation of the input-output relation is derived by obtaining a suitable number of nodes in the hidden layer and by positioning them in the input space where the data is mostly clustered. At every iteration, the position of the radial centers, their width and their linear weights to each output node are modified. The learning is completed when each radial center
is brought up as close as possible to each discrete cluster center formed from the input space, and the error of the networks output is within the desired limit. The centers and widths of the Gaussian are set by the unsupervised learning rules and supervised learning is applied to the output layer. For this reason RBF networks are called *hybrid networks*.

RBF networks have a static Gaussian function as the nonlinearity for the hidden layer processing elements. The Gaussian function responds only to a small region of the input space where the Gaussian is centered. The key to a successful implementation of these networks is to find suitable centers for the Gaussian functions. This can be done with supervised learning, but an unsupervised approach usually produces better results. [3]

The problem solved by RBFs is the development of an analytic approximation for the input and output mappings described by a deterministic, noisy or stochastic data stream:

\[
\{x(t), y(t) : \left[ R^n, R \right] \}_{t=1}^\infty
\]  

(4.19)

where,

- \(x(t)\) is the input vector at time \(t\),
- \(y(t)\) is the output at time \(t\), and
- \(n\) is the dimension of the input space

In the deterministic case the data is drawn from the set:

\[
\left\{ x(t), y(t) = f(x(t)) \right\}_{t=1}^\infty
\]  

(4.20)

In the noisy case data is drawn from the set:

\[
\left\{ x(t), y(t) = f(x(t)) + \epsilon(t) \right\}_{t=1}^\infty
\]  

(4.21)
where, \( \varepsilon(t) \) is a partially known random process.

In the stochastic case, data is drawn from the joint probability distribution:

\[
P(x \wedge y)
\]  

(4.22)

RBF networks can be trained by:

- Deciding on how many hidden units there should be
- Deciding on their centers and the sharpness (standard deviation) of their Gaussians
- Training up the output layer

Generally, the centers and the standard deviations are decided on first by examining the vectors in the training data. The output layer weights are then trained using the Delta rule. Back-propagation is the most widely applied neural network technique.

RBF networks can be trained on:

- Classification data (each output represents one class), and then they can be used directly as classifiers of new data
- \((x, f(x))\) points of an unknown function \(f\), and then can be used to interpolate

Advantages of RBF networks include finding the input to output map using local approximators. Usually the supervised segment is simply a linear combination of the approximators. Since linear combiners have few eights, these networks train extremely fast and require fewer training samples. RBFs also have the advantage the one can add extra units with centers near parts of the input which are difficult to classify. RBFs can also be used for processing time-varying data.

RBF networks can require more neurons than standard feed-forward back-propagation networks, but often they can be designed in a fraction of the time it takes to train
standard feed-forward networks. They work best when many training vectors are available.

### 4.3.4.1.5 Kohonen Self-Organizing Maps

Self-Organizing Maps (SOM) were introduced by a Finnish Professor, Teuvo Kohonen in 1982, thus SOM’s are also sometimes referred to as Kohonen Maps. Self-Organizing Maps are a subtype of Artificial Neural Network. They are trained using unsupervised learning to produce low dimensional representation of the training samples while preserving the topological properties of the input space. Thus, SOM are reasonable for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling.

The principal goal of self-organizing maps is to transform an incoming signal pattern of arbitrary dimension into a one or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion. Self-organizing maps learn to recognize groups of similar input vectors in such a way that neurons physically near each other in the neuron layer respond to similar input vectors. They provide a quantization of the image samples into a topological space where inputs that are nearby in the original space are also nearby in the output space, thereby providing dimensionality reduction and invariance to minor changes in the image sample. [39]

### 4.3.4.1.5.1 Overview

Self-organizing maps are a single layer feed forward network where the output syntaxes are arranged in low dimensional (usually 2D or 3D) grid. Each input is connected to all output neurons. Attached to every neuron there is a weight vector with the same dimensionality as the input vectors. The number of input dimensions is usually a lot higher than the output grid dimension. A Self-organizing map is shown in Figure 4.3.4.1.5.1.1
4.3.4.1.5.2 Network Architecture

Self-organizing maps are single layer feedforward networks where the output syntaxes are arranged in low dimensional (usually 2D or 3D) grid. Each input is connected to all output neurons. Attached to every neuron there is a weight vector with the same dimensionality as the input vectors. The number of input dimensions is usually a lot higher than the output grid dimension. SOMs are mainly used for dimensionality reduction rather than expansion. The architecture for a simple self-organizing map is shown in Figure 4.3.4.1.5.2.1. [31]
The input vector $p$ is the row of pixels of the image. The $||\text{ndis}||$ box in the Figure 4.3.4.1.5.1 accepts the input vector $p$ and the input weight matrix $IW_{1,1}$ produces a vector having $S_1$ elements. The elements are the negative of the distances between the input vector and vectors $IW_{1,1}$ formed from the rows of the input weight matrix. The competitive transfer function accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the winner, the neuron associated with the most positive element of net input $n^1$. The winner’s output is 1. The neuron whose weight vector is closest to the input vector has the least negative net input and, therefore, wins the competition to output a 1. Thus the competitive transfer function produces a 1 for output element $a^1_i$ corresponding to $i^*$, the winning neuron. All other output elements in $a^1$ are 0.

A self-organizing feature map network identifies a winning neuron using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons $i \in N_i(d)$ as follows: [39]

$$i, w(q) = i, w(q-1) + \alpha(p(q) - i, w(q-1)) \quad \text{or} \quad (4.23)$$
\[ w(q) = (1 - \alpha)_i w(q-1) + \alpha p(q) \]  \hspace{1cm} (4.24)

Here the neighborhood \( N_i(d) \) contains the indices for all of the neurons that lie within a radius \( d \) of the winning neuron \( i^* \).

\[ N_i(d) = \{ j, d_{ij} \leq d \} \]  \hspace{1cm} (4.25)

Thus, when a vector \( p \) is presented, the weights of the winning neuron and its close neighbors move toward \( p \). Consequently, after many presentations, neighboring neurons learn vectors similar to each other.

Self-organizing maps are commonly used as visualization aids for classification purposes. They make it easier to see relationships between vast amounts of data. Typical applications of self-organizing maps include visualization of process states or financial results by representing the central dependencies within the data on the map. Self-organizing maps have been used in many practical applications such as: [31]

- State diagrams for processes and machines
- Bibliographic classification
- Medical diagnosis
- Speech recognition
- Face recognition
- Environmental modeling
- Data mining applications
- Image browsing systems
- Interpreting seismic activity
- Data compression
- Separating sound sources
- Handwriting recognition

### 4.3.4.2 Recurrent Neural Networks

Recurrent neural networks (RNs) are models with bi-directional data flow. While a feed-forward network propagates data linearly from input to output, recurrent
networks also propagate data from later processing stages to earlier stages. Recurrent networks can be mainly divided into four subtypes of neural networks, mainly:

1) Simple Recurrent Network  
2) Hopfield Network  
3) Echo State Network  
4) Long Short Term Memory Network

4.3.4.2.1 Simple Recurrent Network

A simple recurrent network (SRN) is a variation on the multi-layer perceptron, sometimes called an Elman network due to its invention by Jeff Elman. A three-layer network is used, with the addition of a set of "context units" in the input layer. There are connections from the middle (hidden) layer to these context units fixed with a weight of one. At each time step, the input is propagated in a standard feed-forward fashion, and then a learning rule (usually back-propagation) is applied. The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction that are beyond the power of a standard multi-layer perceptron. [3]

In a fully recurrent network, every neuron receives inputs from every other neuron in the network. These networks are not arranged in layers. Usually only a subset of the neurons receive external inputs in addition to the inputs from all the other neurons, and another disjunct subset of neurons report their output externally as well as sending it to all the neurons. These distinctive inputs and outputs perform the function of the input and output layers of a feed-forward or simple recurrent network, and also join all the other neurons in the recurrent processing. Figure 4.3.4.2.1.1 shows a simple recurrent network.
4.3.4.2.2 Hopfield Network

The Hopfield network is a recurrent neural network in which all connections are symmetric. Invented by John Hopfield in 1982, the Hopfield network guarantees that its dynamics will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration.

4.3.4.2.3 Echo State Network

The Echo State Network (ESN) is a recurrent neural network with a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change and be learned. ESN are good to (re)produce temporal patterns.
4.3.4.2.4 Long Short Term Memory Network

The Long short term memory is an artificial neural net structure that unlike traditional recurrent networks doesn't have the problem of vanishing gradients. It can therefore use long delays and can handle signals that have a mix of low and high frequency components.

4.3.4.3 Stochastic Neural Networks

Stochastic neural networks differ from regular neural networks in the fact that they introduce random variations into the network. In a probabilistic view of neural networks, such random variations can be viewed as a form of statistical sampling, such as Monte Carlo sampling. The Boltzmann Machine is classified as a stochastic neural network.

4.3.4.3.1 Boltzmann Machine

The Boltzmann machine can be thought of as a noisy Hopfield network. Invented by Geoff Hinton and Terry Sejnowski in 1985, the Boltzmann machine is important because it is one of the first neural networks to demonstrate learning of latent variables (hidden units). Boltzmann machine learning was at first slow to simulate, but the contrastive divergence algorithm of Geoff Hinton allows models such as Boltzmann machines and products of experts to be trained much faster. [3]

4.3.4.4 Modular Neural Networks

Biological studies showed that the human brain functions not as a single massive network, but as a collection of small networks. This realization gave birth to the concept of modular neural networks, in which several small networks cooperate or compete to solve problems.
4.3.5 Employing Artificial Neural Networks

The greatest advantage of artificial neural networks is their ability to be used as an arbitrary function approximation mechanism which learns from observed data.

4.3.5.1 Choice of model

This will depend on the data representation and the application. Overly complex models tend to lead to problems with learning.

4.3.5.2 Learning algorithm

Numerous tradeoffs exist between learning algorithms. Almost any algorithm will work well with the correct hyper-parameters for training on a particular fixed dataset. However selecting and tuning an algorithm for training on unseen data requires a significant amount of experimentation.

4.3.5.3 Robustness

If the model, cost function and learning algorithm are selected appropriately the resulting artificial neural network can be extremely robust.

With the correct implementation artificial neural networks can be used naturally in learning and large dataset applications. Their simple implementation and the existence of mostly local dependencies exhibited in the structure allows for fast, parallel implementations in hardware.
4.3.6 Applications of Artificial Neural Networks

Artificial neural network models can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical. [3]

For real life applications the tasks to which artificial neural networks are applied tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.

Neural networks are particularly effective for predicting events when the networks have a large database of prior examples to draw on. Application areas include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face recognition, voice recognition, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, industrial robotics, financial applications, data mining (or knowledge discovery in databases, "KDD"), visualization, e-mail spam filtering and aerospace industry. [3]

4.4 Conclusion

The computing world has gained a lot from artificial neural networks. Their ability to learn by example makes them very flexible and powerful. They are also very well
suited for real time systems because of their fast response and computational times which are due to their parallel architecture.

Neural networks contribute to areas of research such as neurology and psychology. They are regularly used to model parts of living organisms and to investigate the internal mechanisms of the brain.

Even though neural networks have a huge potential our current focus in this project is to integrate neural networks with AI, fuzzy logic and related subjects. The reason for choosing artificial neural networks in this project is because they are well suited pattern recognition problems for developing artificial intelligence systems such as face recognition and identification systems as proved by the literature survey, since neural network training data corresponds to noisy, complex sensor data, such as inputs from cameras. In this project the input data and training database are digital images.
CHAPTER V

MATLAB AND SIMULINK

5.1 Introduction

MATLAB is a high-performance language for technical computing created by The MathWorks in 1984. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis. [22]

5.1.1 History of MATLAB

MATLAB was invented in the late 1970s by Cleve Moler, then chairman of the computer science department at the University of New Mexico. He designed it to give his student’s access to LINPACK and EISPACK without having to learn Fortran. It
soon spread to other universities and found a strong audience within the applied mathematics community.

Jack Little, an engineer, was exposed to it during a visit Moler made to Stanford University in 1983. Recognizing its commercial potential, he joined with Moler and Steve Bangert. They rewrote MATLAB in C and founded The MathWorks in 1984 to continue its development. These rewritten libraries were known as JACKPAC.

MATLAB was first adopted by control design engineers, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is popular amongst scientists involved with image processing.

5.2 What Is MATLAB

MATLAB is a numerical computing environment and programming language. It allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Although it specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allows it to be part of a full computer algebra system. Besides dealing with explicit matrices in linear algebra, it can handle differential equations, polynomials, signal processing, and other applications. Results can be made available both numerically and as excellent graphics. [22]

MATLAB solves many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for Matrix Laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK
projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB features a family of add-on application-specific solutions called Toolboxes. Toolboxes allow learning and applying specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

As of 2004, MATLAB was reported to be used by more than one million people in industry and academia.

5.3 The MATLAB System

The MATLAB system consists of the following five parts:

1) Desktop Tools and Development Environment
2) MATLAB Mathematical Function Library
3) MATLAB Language
4) MATLAB Graphics
5) MATLAB External Interfaces/API

5.3.1 Desktop Tools and Development Environment

This is the set of tools and facilities that help to use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, a code analyzer and other reports, and browsers for viewing help, the workspace, files, and the search path.
5.3.2 MATLAB Mathematical Function Library

The MATLAB mathematical function library is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

5.3.3 MATLAB Language

The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both programming in the small to rapidly create quick and dirty throw-away programs, and programming in the large to create large and complex application programs.

5.3.4 MATLAB Graphics

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

5.3.5 MATLAB External Interfaces/API

The MATLAB External Interface is a library that allows writing C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.
5.4 What Is Simulink

Simulink is a software bundled with MATLAB for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multi-rate, i.e., have different parts that are sampled or updated at different rates.

Simulink enables users to pose a question about a system, model it, and see what happens. With Simulink, models can be built easily from scratch, existing models can be taken and be added to it. Thousands of engineers around the world use Simulink to model and solve real problems in a variety of industries. [22]

5.4.1 Tool for Model-Based Design

Simulink can move beyond idealized linear models to explore more realistic nonlinear models, factoring in friction, air resistance, gear slippage, hard stops, and the other things that describe real-world phenomena. Simulink turns a computer into a lab for modelling and analyzing systems that simply wouldn't be possible or practical otherwise, whether the behaviour of an automotive clutch system, the flutter of an airplane wing, the dynamics of a predator-prey model, or the effect of the monetary supply on the economy. Simulink provides numerous demos that model a wide variety of such real-world phenomena.

For modeling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations. With this interface, users can draw models just as with pencil and paper. This interactive graphical environment is a far cry from simulation packages that require formulating differential equations and difference equations in a language or program. Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors.
Models are hierarchical, so models can be built using both top-down and bottom-up approaches. The system can be viewed at a high level, then double-click the blocks to go down through the levels to see increasing levels of model detail. This approach provides insight into how a model is organized and how its parts interact.

5.4.2 Tool for Simulation

After a model is defined, it can be simulated, using choices of mathematical integration methods, either from the Simulink menus or by entering commands in the MATLAB Command Window. Simulink menus are convenient for interactive work, while the command line is useful for running a batch of simulations. Scopes and other display blocks allow viewing simulation results while the simulation runs. In addition, many parameters can be changed and see what happens for "what if" exploration. Simulation results can also be put in the MATLAB workspace for post-processing and visualization.

5.4.3 Tool for Analysis

Model analysis tools include linearization and trimming tools, which can be accessed from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes. Because MATLAB and Simulink are integrated, users can simulate, analyze, and revise your models in either environment at any point.

5.5 Conclusion

The reason for using MATLAB for the design and development of this project is its toolboxes and superior processing capabilities. The MATLAB Neural Network toolbox and Image Processing toolbox played an important role for the development of this project.
CHAPTER VI

DESIGN AND IMPLEMENTATION

This chapter gives a detailed explanation of the design technique used for the implementation of this project. The initial design technique proposed by my project supervisor did not work to build an efficient face recognition system. This chapter explains how I managed to devise a technique for the design and implementation of a high-speed efficient face recognition system in MATLAB and the work done in the design phase.

6.1 Introduction

The programming language used to design and implement the face recognition system recognition code is MATLAB. The reason for using MATLAB in this project is due to its Neural Network and Image Processing toolbox that helped to obtain an efficient code.

6.2 Face Image Processing

Face image processing consists of the following six steps:

1) Data Gathering
2) Image Preprocessing in Adobe Photoshop
3) Import Faces Images to MATLAB
4) Image Resize in MATLAB
5) Face Image DCT
6) Image Data Reshape
All face images taken from a digital camera need to be processed before being input into the neural network. The face image process structure is shown in Figure 6.2.1

![Face image process structure diagram](image)

**FIGURE 6.2.1** Face image process structure

### 6.2.1 Data Gathering

Face images of different subjects were taken under uniform light conditions and light backgrounds with a 2.0 megapixel digital camera for the training database. The face images were then transferred from the digital camera to the computer. Each face image size was equivalent to 1200 x 1600 pixels. Figure 6.2.1.1 shows the faces images taken.
Similarly, five additional pictures of each individual subject with different facial expressions and facial furniture were taken. The five face expressions comprise of: neutral, happy, sad, angry and smiling. Facial furniture comprises of face glasses. Figure 6.2.1.2 shows face images with different facial expressions of an individual subject.
6.2.2 Image Pre-processing in Adobe Photoshop

Face images were then preprocessed in Adobe Photoshop. Image preprocessing in Adobe Photoshop has three phases:

1) Auto adjusting image levels, contrast, brightness and colors
2) Image conversion from RGB color to 8-bit grayscale
3) Image resizing from 1200 x 1600 pixels to 512 x 512 pixels

6.2.2.1 Auto Adjusting Image Levels, Contrast and Brightness

Auto adjusting image levels, contrast, brightness and colors in *Adobe Photoshop CS* was achieved by going to:

- Image → Adjustments → Auto Levels
- Image → Adjustments → Auto Contrast
- Image → Adjustments → Auto Color
- File → Save
FIGURE 6.2.2.1.1 Adjusting brightness, contrast and colors in Adobe Photoshop

FIGURE 6.2.2.1.2 (a) Original face image from digital camera (b) Face image auto adjusted in Adobe Photoshop for brightness, contrast and color
6.2.2.2 Image Conversion from RGB to Grayscale

Face images were converted from RGB color to 8-bit grayscale, in *Adobe Photoshop* CS by going to:

- Image → Mode → Grayscale
  - Discard color information? → OK
- File → Save

![Converting RGB face image into 8-bit grayscale](image)

**FIGURE 6.2.2.2.1** Converting RGB face image into 8-bit grayscale
FIGURE 6.2.2.2.2  (a) Auto adjusted face image in RGB color (b) Grayscale face image

6.2.2.3 Image Resizing

Face images were resized from 1200 x 1600 pixels to 512 x 512 pixels by reducing the image size and then by reducing the canvas size. This was done by going to:

- **Image → Image Size**
  - Reduce width to 512 pixels; height automatically adjusts itself
- **Image → Canvas Size**
  - Select pixels as the measuring unit
  - Reduce height to 512 pixels
  - Select best anchor setting; redundant portion of face image is cropped
- **File → Save**
FIGURE 6.2.2.3.1 Reducing image size – setting width to 512 pixels
FIGURE 6.2.2.3.2 Reducing canvas size – setting height to 512 pixels

(a) Original image – 1200 x 1600 pixels  (b) Resized image – 512 x 512 pixels

FIGURE 6.2.2.3.3 (a) Original image – 1200 x 1600 pixels (b) Resized image – 512 x 512 pixels
6.2.3 Import Faces Images to MATLAB

After all face images were preprocessed in Adobe Photoshop, then they were imported into MATLAB. The MATLAB command `imread` was used to load pictures into the workspace. Figure 6.2.3.1 shows importing pictures into MATLAB workspace.

![Figure 6.2.3.1 Importing face images into MATLAB](image)

**FIGURE 6.2.3.1 Importing face images into MATLAB**

6.2.4 Image Resize in MATLAB

After all face images were imported into MATLAB, they were resized further from 512 x 512 pixels to 8 x 8 pixels. For this purpose the MATLAB command `imresize` was used to resize the imported pictures. Figure 6.2.4.1 shows the resizing of a face image from 512 x 512 pixels to 8 x 8 pixels.
After all face images were resized, they were saved under different filenames, using the MATLAB `imwrite` command. Figure 6.2.4.2 shows the saving of a face image under a different filename.
6.2.5 Face Image DCT

After all face images were resized to 8 x 8 pixels and saved, the next step was to compress them by applying the 2D blocked DCT. When the 2D DCT is applied with a mask, high-coefficients are in the image discarded. Then the 2D IDCT is applied to regenerate the compressed image, which is blurred due to loss of quality and also smaller in size.

To find a technique to apply the 2D DCT to a face image, the MATLAB help was searched in the Image Processing Toolbox. A program for 2D DCT image compression in MATLAB help was found with its source code. You can find this program by going to:
• Help  →  MATLAB Help (opens help window)
• In help window: Search (tab)
  ▪ Type “DCT image compression”
  ▪ Press “Go”
• Resulting page will be found and displayed

The source code found from the MATLAB image processing toolbox help was used to DCT all face images, after few modifications. Before DCT compression, the image data of the resized images needed to be converted into the double format. This was achieved by using the MATLAB `double` command. The mask used for the 2D-DCT was of 8 coefficients out of 64. Figure 6.2.5.2 shows the DCT masking matrix.
After the DCT program found in MATLAB help was modified according to the DCT options and the masking host was set to 8 coefficients and all face images were compressed using the DCT. The newly compressed face images were saved under a different filename. Figure 6.2.5.3 shows the program used for DCT face image compression.

```
mask = [1 1 1 1 0 0 0 0
       1 1 1 0 0 0 0 0
       1 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0];
```

FIGURE 6.2.5.2 DCT masking matrix – 8 coefficients chosen

FIGURE 6.2.5.3 DCT program for face image compression
All face images were run through the program in Figure 6.2.5.3 and were compressed. Each input image and the image to written (saved) in Figure 6.2.5.3 was changed and the program was run one by one, for all face images which were to be compressed. Figure 6.2.5.4 shows example un-resized compressed and uncompressed face image in 512 x 512 pixels. Figure 6.2.5.5 shows a resized compressed and uncompressed face image in 8 x 8 pixels.

FIGURE 6.2.5.4 Example of un-resized face image in 512 x 512 pixels
(a) Original face image (b) DCT compressed face image
FIGURE 6.2.5.5 Actual resized face image in 8 x 8 pixels – zoomed 2600%
(a) Original face image (b) DCT compressed face image

FIGURE 6.2.5.6 Image data for resized image in Figure 6.2.5.5
6.2.6 Image Data Reshape

For the image data to be input into the neural network, it should follow the form of *only one column*, despite the number of rows. Currently, all the resized and DCT compressed face images are in the form of 8 x 8 pixels as shown in Figure 6.2.5.6.

Hence the image data needed to be reshaped from an 8 x 8 matrix to a *64 x 1 array* for it to be used both for the input and training database of the neural network. For reshaping the MATLAB *reshape* command is used. Figure 6.2.6.1 shows the reshaping of compressed image data from an 8 x 8 matrix to a 64 x 1 array.

![MATLAB Command Window](image)

**FIGURE 6.2.6.1** Image data reshaped from an 8 x 8 matrix to a 64 x 1 array
All face image data was then reshaped to a 64 x 1 array. Figure 6.2.6.2 shows the actual image data in Figure 6.2.5.5 reshaped in a 64 x 1 array.

Figure 6.2.6.2 (b) is the actual data which will be input into the neural network as an input face image or a set of more data combined together, similar to Figure 6.2.6.2 (b) for the training database.

6.3 Neural Network Design

The initially proposed design technique using the Radial Basis Function (RBF) Neural Network failed to give accurate results for an untrained input image with different facial expressions. Hence, Fuzzy C-Means (FCM) Clustering was applied for proper
image data management, but FCM also failed to give accurate output results for the face recognition system. After research, Self-Organizing Maps (SOM’s) were found and chosen to be the technique used for the face recognition system based on their improved data management and neural network accuracy.

### 6.3.1 Radial Basis Function Neural Network

The RBF neural network was tested in MATLAB for a training database of subjects with different facial expressions for an untrained input image of the subject with a different facial expression than the one in the trained database. All outputs after continuous program simulations and change in input face images resulted in incorrect outputs; classifying wrong images from the input with the trained database and generating incorrect answers. Hence, the idea of using RBF for the design of the face recognition system was dropped.

### 6.3.2 Fuzzy C-Means Clustering

Fuzzy C-Means (FCM) Clustering was chosen due to the failure of RBF networks. FCM was integrated with the current face images and implemented in MATLAB. FCM proved to be efficient in managing image data, but on every trial it calculated cluster centers at different positions. Cluster centers were calculated based on iterations. The position difference of cluster centers for every trial made it difficult to correctly match the input image with the trained database for every trial. Since the output result on some trials used to be correct and on some trials used to be incorrect, due to the output un-certainty of this technique the idea was dropped for the face recognition system.
6.3.3 Self-Organizing Maps

Self-Organizing Maps (SOM’s) were chosen due to the failure of FCM. After research and study, SOM’s were found to be efficient for image data management and proved to be an accurate closest matching technique of untrained input images with trained database of images.

For the design of SOM, a set of 25 image data, 5 different subjects with 5 different facial expressions for the training database was loaded into MATLAB. Figure 6.3.3.1 shows 25 image data loaded into MATLAB into vector $P$.

![FIGURE 6.3.3.1 25 Image data loaded into MATLAB in vector ‘P’](image-url)
A SOM was then created with the MATLAB command *newsom*. Parameters for the SOM network were selected to be a minimum and maximum point for each row on vector $P$; training database. There were 64 minimum and 64 maximum points selected altogether. Figure 6.3.3.2 shows the SOM minimum and maximum points for the training database in a 64 x 2 array.

![MATLAB window showing SOM creation](image)

**FIGURE 6.3.3.2** Minimum and maximum points for SOM in a 64 x 2 array

Figure 6.3.3.3 shows the creation of a SOM neural network with minimum and maximum points.
FIGURE 6.3.3.3 Creation of a SOM network with minimum and maximum points

After the SOM neural network was created, it was trained for 1000 epochs. Figure 6.3.3.4 shows the training of the SOM neural network for 1000 epochs.
FIGURE 6.3.3.4 SOM neural network training for 1000 epochs

Figure 6.3.3.5 shows the SOM layer weights for the 25 face images in the training database. Figure 6.3.3.6 shows the SOM weight vectors.
FIGURE 6.3.3.5 SOM layer weights for the 25 face images in the training database
FIGURE 6.3.3.6 SOM weight vectors

After SOM neural network training the, network was simulated for the 25 trained images. Figure 6.3.3.7 shows the simulation of the SOM neural network for the 25 images in the training database.
After the SOM neural network was trained and simulated for the 25 images in the training database, the SOM neural network was then simulated for the single input face image. Figure 6.3.3.8 shows loading the input image into MATLAB and simulating it for the SOM neural network.
After the SOM neural network is simulated for the input face image, the image in the training database which is the closest match by the SOM neural network for the input face image is found by finding the minimum absolute deviation. Figure 6.3.3.9 shows how the closest matched training database image with respect to the input face image is found.
As show in Figure 6.3.3.9, picture number 16, 17 and 20 in the 25 image training database contain face images with different facial expressions. All three pictures contain the same subject as in the input image and match very closely to the input face image.

After the closest matched training database images are found, they are then classified. Classification of the subject is the answer of the face recognition system. Classification in MATLAB was done through simple if and else statements. So, if the number of training database images change, the number of if and else statements would also change. Classification of the subject found in the 25 image training database with respect to the input image is shown in Figure 6.3.3.10.
FIGURE 6.3.3.10 Classification of the subject found in the 25 image training database with respect to the input image

As show in Figure 6.3.3.10, pictures from 1 to 5 contain a subject, pictures from 6 to 10 contain a different subject, and pictures from 11 to 15 contain another different subject and so on. Hence, pictures 16, 17 and 20 as shown in Figure 6.3.3.9 are classified in Figure 6.3.3.10 as subject 4, which is the correct answer for the input face image.

6.3.3.1 Simulink Block Model

The SOM neural network model for the 25 image training database was converted from MATLAB into a Simulink block model. The MATLAB command `gensim`
generated a Simulink block model of the SOM neural network. Figure 6.3.3.1.1 shows the Simulink block model of the SOM neural network.

![Simulink block model of SOM neural network](image)

**FIGURE 6.3.3.1.1 Simulink block model of SOM neural network**

As shown in Figure 6.3.3.1.1 the input is vector $p\{I\}$. The processing stage is the Neural Network block and the output result is $y\{I\}$, shown in the scope. The neural network block consists of only one layer which connects the input and the output. The neural network layers for the simulink block model are shown in Figure 6.3.3.1.2.

![Simulink block model of SOM neural network layers](image)

**FIGURE 6.3.3.1.2 Simulink block model of SOM neural network layers**

As shown in Figure 6.3.3.1.2 the SOM neural network only consists of one layer, which is Layer 1. The neural network layer 1 block consists of a subsystem of blocks as shown in Figure 6.3.3.1.3.

![Simulink subsystem model of SOM neural network layer 1](image)

**FIGURE 6.3.3.1.3 Simulink subsystem model of SOM neural network layer 1**
The input \( p[I] \) is delayed using a Tapped Delay Line (TDL). The input signal enters from the left and passes through \( N-1 \) delays. The output of the tapped delay line is an \( N \)-dimensional vector made up of the input signal at the current time, the previous signal etc. The TDL system in simplicity Multiplexes (MUX) vector is shown in Figure 6.3.3.1.4.

![Simulink block model of SOM neural network TDL](image)

From the output of the TDL, the \( N \)-dimensional vector is calculated and adjusted for weights. For the case for the 25 images for the training database, the 64 minimum and 64 maximum points in the SOM neural network comprise together as the layer weights. So the total number of layer weights is 128. These weights are calculated and adjusted automatically with the negative of the euclidean distance weight function after which its scalars are Multiplexed (MUX). The SOM neural network layer weights block model is shown in Figure 6.3.3.1.5.
FIGURE 6.3.3.1.5 Simulink block model of SOM neural network layer weights

After the weights are calculated and adjusted, the layers net input is calculated by combining its weighted inputs and biases. After the layers net input is calculated its
competitive neural transfer function as shown in Figure 6.3.1.3 calculates the layers output from the net input.

6.4 Conclusion

As a conclusion for the design and implementation stage, the 2D-DCT in MATLAB was successful. It compressed face images and reduced image size and quality. The initially proposed design technique of RBF networks failed to recognize untrained face images with different facial expressions. The FCM technique also proved to be inefficient and uncertain by choosing different positions for cluster centers on every new trial. The SOM proved to be highly accurate by recognizing face images under a test of a training database of 25 face images, trained with 1000 epochs for both trained and untrained input face images with different facial expressions. The classification method chosen was simple and faster than fuzzy logic.
CHAPTER VII

TESTING AND EXPERIMENTAL WORK

This chapter gives a detailed explanation of the SOM neural network testing for a variety of trained and untrained inputs and the experimental work carried out for optimal efficiency and to attain a high-speed design.

7.1 Neural Network Validation

A variety of tests were carried out with both trained and untrained input face images for different subjects with different facial expressions leading to the validation of the SOM neural network.

7.1.1 Trained face images

An initial test was carried out on the SOM neural network to test its validity. The training database, consisting of 6 subjects is shown in Figure 7.1.1.1. The trained input face images are shown in Figure 7.1.1.2.

FIGURE 7.1.1.1 Training database of 6 subjects
The SOM neural network generated correct answers for all trained input images in Figure 7.1.1.2, except for Figure 7.1.1.2 (c), which is not present in the training database set. For this case the classification section generated an error. This bug was corrected by adding an extra `else` statement displaying, “subject not found in database”.

### 7.1.2 Untrained face images

A number were tests were carried out for untrained face images. Firstly a modified training database similar to the trained face images in Figure 7.1.1.1 was trained. Inputs
FIGURE 7.1.2.1 Training database of 8 subjects

FIGURE 7.1.2.2 Untrained input face images

The SOM neural network generated correct answers for all untrained input images in Figure 7.1.2.2, except for Figure 7.1.2.2 (d), which is not present in the training database set. This validated the accuracy of the SOM neural network for face images with different subjects.

For further testing and SOM neural network validation untrained face images with different facial expressions were used as shown in Figure 7.1.2.4, while the training database remained the same as shown in Figure 7.1.2.3.
FIGURE 7.1.2.3 Training database of 8 subjects

FIGURE 7.1.2.4 Untrained input face images with different facial expressions

The SOM neural network generated correct answers for all untrained input images with different facial expressions in Figure 7.1.2.4. The subject in the untrained input images matched correctly with the same subject training database under a different facial expression.
For advanced SOM neural network validation a final training database of 25 face images with different facial expressions was constructed. A single untrained input image was input into the SOM neural network and the SOM network was simulated. The architecture for the untrained face image validation is shown in Figure 7.1.2.5.

FIGURE 7.1.2.5 Architecture for untrained face image SOM network validation

The training database consists of 25 face images, including 5 subjects, each subject having 5 individual face images with different facial expressions as shown in Figure 7.1.2.6. The untrained input face is shown in Figure 7.1.2.7.
FIGURE 7.1.2.6 Training database of 25 subjects with different facial expressions

FIGURE 7.1.2.7 Untrained input face image with different facial expression
The untrained input face image in Figure 7.1.2.7 was simulated in the SOM network for the trained database in Figure 7.1.2.6 for 1000 epochs. The untrained input matched with the closest image of the same subject in the training database, generating a correct answer. Input face images were then changed and the network was simulated repeatedly and the SOM neural network generated correct answers for all trials.

Thus, through the tests mentioned above and other validation tests performed the SOM neural network validation proves the face recognition system to be accurate for untrained face images with different facial expressions.

### 7.2 Neural Network Efficiency

Since this project requires the design of a *high-speed efficient* face recognition system, the least amount of time required for program execution, while the accuracy is at its highest peak contributes to neural network efficiency. Some factors affecting the efficiency of the neural network are discussed below.

#### 7.2.1 Addition of Random Noise to Face Images

The SOM neural network was tested for face images with random noise. Random noise with uniform distribution was added through MATLAB to the input face images after DCT computation. It was observed that with a random noise scale of $\alpha \leq 0.3$, the SOM neural network generated a correct output for the desired input image, but, for a random noise scale $\alpha > 0.3$, the output generated by the SOM neural network was incorrect with respect to the given input face image. A comparison of both situations is given below in Figure 7.2.1.1 and Figure 7.2.1.2.
FIGURE 7.2.1.1 SOM neural network input signal for single image data - Original input signal vs. input signal with $\alpha \leq 0.3$

FIGURE 7.2.1.2 SOM neural network input signal for single image data - Original input signal vs. input signal with $\alpha = 0.35$
7.2.2 Determining Optimal Number of Epochs for Training

Epochs are neural network training parameters. They are defined as one complete cycle through the neural network for all cases, which present the entire training set to the neural network. Each time a record goes through the net, it is one trial, one sweep of all records is termed as an Epoch.

During each epoch, the order in which the compounds are presented are randomized. This procedure improves the overall performance of the neural network by reducing the training errors. Less number of epochs used for training leads to less training time for the training data set. The goal is to find the optimal number of epochs for training which will produce accurate neural network results and at the same require the least amount of time for program execution.

The SOM neural network was tested to determine the optimal number of epochs to be used for neural network training. This test was performed by varying the number of epochs on every trial to find the optimal number of epochs required for the best SOM network output accuracy.

Table 7.2.2.1 shows the tabulated results for the test. The training database for this test consists 25 face images of 5 subjects, each subject having 5 individual images with different facial expressions. For this test 8 DCT coefficients for image masking and 64 input neurons for each face image were selected.
Table 7.2.2.1 shows the tabulated results generated by varying the number of epochs. The most efficient number of epochs for training and with respect to fastest training time and accurate SOM neural network output results was found to be 1000. Figure 7.2.2.1 illustrates the tabulated results from Table 7.2.2.1 on a graph.
FIGURE 7.2.2.1 Number of epochs vs. neural network accuracy for 10 trials
CHAPTER VIII

DISCUSSION AND ANALYSIS OF RESULTS

8.1 Discussion and Analysis

The tests performed in Chapter 7 on trained face images in section 7.1.1 and on untrained face images in section 7.1.2 verify the accuracy of the SOM neural network and the overall face recognition system. These test results confirm that the face recognition system designed has optimum accuracy for face images with different facial expression under uniform light conditions and light backgrounds.

The neural network efficiency test performed in section 7.2.1 determines the tolerance of noise in the face images the neural network can handle. In section 7.2.2 the neural network efficiency was optimized to obtain a high-speed program by determining the optimal number epochs to be used for neural network training. These test results optimize the neural network to obtain a high-speed efficient face recognition system and also determine the tolerance of noise in the neural network input which the program can handle before giving an incorrect output.

From the test results mentioned above and other experimental work and analysis carried out, it can be analyzed that the design and implementation of an optimal high-speed efficient face recognition system has been successfully achieved as the primary objective of this project.
CHAPTER IX

GUI IMPLEMENTATION OF PROGRAM

9.1 Introduction

The GUI implementation of the face recognition system is not a part of the project objective. Due to additional GUI of the source code was implemented, since a user-friendly GUI program is considered a good demonstration of the project work completed.

9.2 GUI Program Design

The final source code of the face recognition system was written MATLAB M-files. Hence MATLAB was chosen to implement the GUI program of the face recognition system.

The MATLAB GUI Design Environment tool GUIDE was used to implement the face recognition program source code into a GUI program. GUIDE was run from the MATLAB workspace by typing `guide`. Figure 9.2.1 shows the startup screen for the MATLAB GUIDE.
The MATLAB GUIDE was opened and a simple GUI demonstration program was constructed using push buttons and static text from the GUIDE toolbar. Figure 9.2.2 shows the main MATLAB GUIDE window with a blank figure.
FIGURE 9.2.2 Main MATLAB GUIDE window with a blank figure
FIGURE 9.2.3 Face recognition program being constructed in MATLAB GUIDE
After the design of the GUI face recognition demonstration program in MATLAB GUIDE was completed, the callbacks on all push buttons on the GUI program were set by adding separate M-files from the original face recognition program source code developed in the MATLAB workspace. Figure 9.2.5 shows setting of the callbacks on the push buttons of the GUI program in the MATLAB M-file editor.
After all callbacks on all GUI push buttons were set, they were re-tested for errors.
The face recognition GUI demonstration program was saved in MATLAB as a figure.
Figure 9.2.6 shows the face recognition GUI demonstration program.
9.3 GUI Program Demonstration

The face recognition GUI program demonstration displays the results of the program in the MATLAB workspace.

Firstly all MATLAB workspace variables and all generated image and data files were deleted from the program directory by pressing the push button labeled *Delete All.*
Then the DCT image compression example was run by pressing the push button labeled *DCT Sample*. This example was included in the GUI program to show the difference between an original picture and a DCT compressed picture. Figure 9.3.1 shows the output generated in MATLAB after pressing the push button ‘DCT Sample’.

![Figure 9.3.1 Output generated in MATLAB after pressing push button: DCT Sample](image)

In Figure 9.3.1 the face image on the left is the original image taken from the digital camera, while the face image on the right is the DCT compressed image. Next the untrained input face image into the face recognition system was displayed by pressing the push button labeled *Input Face Image*. Figure 9.3.2 shows the output generated in MATLAB after pressing the push button ‘Input Face Image’.
FIGURE 9.3.2 Output generated in MATLAB after pressing push button: Input Face Image

Then the training database set was displayed by pressing the push button labeled *Training Database*. Figure 9.3.3 shows the output generated in MATLAB after pressing the push button ‘Training Database’.
FIGURE 9.3.3 Output generated in MATLAB after pressing push button: Training Database
In Figure 9.3.4 the face image of the subject in the left is not included in the training database with the same facial expression, but the training database does include pictures of the same subject with other facial expressions other than the one in the input face image. Hence the face recognition program needs to match the subject in the input image with any of the closest pictures of that subject found in the training database.

The face recognition program can be executed by two different ways in the GUI. The first is through a 3-step push button method in the panel Run Program Step-by-Step or the other through a single-step push button method in the panel Run Program In One Go. Both ways will give the same output result.
The ‘Run Program Step-by-Step’ GUI design was implemented for users who wish to run the program step by step and check the MATLAB workspace, DCT images generated and data files generated in the program directory from time to time and continue to the next step. It was basically created for users who wish to monitor the program. The ‘Run Program In One Go’ GUI design was implemented for users who wish to simulate the program by the click of one single button, which is faster.

For the case of this demonstration, the ‘Run Program In One Go’ is selected and the push button labeled *Execute Program* is pressed. After the ‘Execute Program’ push button is pressed the face recognition program is simulated and the results are shown in Figure 9.3.5.

![Face recognition GUI program simulation results](image)

**FIGURE 9.3.5** Face recognition GUI program simulation results
In Figure 9.3.5 the image on the left is the untrained face image as shown in Figure 9.3.2, while the image on the right is the image in the training database with which the input image closely matched with. As seen in Figure 9.3.5 both input and matched training database face images are of the same person, hence the face recognition program accurately matched the subject.

After pressing the ‘Execute Program’ push button the SOM neural network is simulated and the program is run by displaying Figure 9.3.5. The SOM layer weights are also displayed as shown in Figure 6.3.3.5, the SOM weight vectors are also displayed as shown in Figure 6.3.3.6 and a Simulink block model of the SOM neural network is also generated as shown in Figure 6.3.3.1.1.
CHAPTER X

RECOMMENDATIONS AND CONCLUSION

10.1 Recommendations for Improvement

Upon extensive study and research, recommendations for improvement and enhancement of the face recognition system program are concluded as follows:

- Replacing the DCT with the Discrete Wavelet Transform (DWT) as it is a superior and improved algorithm for image compression which requires less processing time than the DCT with better compression capability.

- Further SOM neural network efficiency testing based on the following factors:
  - Selecting the optimal number of DCT coefficients to use for face image compression, which will lead to less DCT processing time and increase the program execution time.
  - Selecting the optimal number of input neurons for neural network simulation, which is the size of the input image and training database images in pixels. The smallest amount of image pixels giving the most accurate SOM neural network output would be the optimal number of input neurons to be used for input images and training database images for the SOM neural network.

All these above mentioned factors will result to achieve the shortest amount of time for program execution to produce an efficient high-speed face recognition system with maximum output accuracy.
10.2 Project Conclusion

This project met all objectives by the successful design of an efficient high-speed face recognition system. The 2D-DCT in MATLAB was successful and all face images were successfully compressed to the desired size and quality. The DCT proved its high-speed efficiency by discarding high-coefficients and redundant image data. The initially proposed design technique utilizing the Radial Basis Function (RBF) Neural Network failed, since untrained face images with different facial expressions were not recognized by the RBF network. Hence, RBF networks were replaced by the Fuzzy C-Means (FCM) Clustering technique. FCM proved to be efficient for managing image data, but it chose different positions for cluster centers on every new simulation, making it very uncertain and impossible to be used for this project. FCM was then replaced by introducing Self-Organizing Maps (SOM’s) which proved to be highly accurate for recognizing a variety of face images with different facial expressions under uniform light conditions with light backgrounds.

Through the continuous usage of MATLAB during the course of this project, I was able to familiarize myself with MATLAB’s commands and processing capabilities. Thanks to this project I also have got good awareness with Artificial Neural Networks and Fuzzy Logic.

Hence as a conclusion, the 2D-DCT and the SOM neural network are the heart for the design and implementation, which are the final algorithms used for the design of an efficient high-speed face recognition system.
REFERENCES


[37] Xiao Yun Jing and David Zhang. A Face and Palmprint Recognition Approach Based on Discriminant DCT Feature Extraction. Nanjing University of Science and Technology.

[38] Yeuhui Chen, Shuan Jiang and Ajit Abraham. Face Recognition using Discrete Cosine Transform and Hybrid Flexible Neural Tree. School of Information Science and Engineering, Jinan University, China.


[40] Yuan Wang, Yunde Jia, Changbo Hu and Matthew Turk. Face Recognition Based on Kernel Radial Basis Function Networks. Computer Science Department, Beijing Institute of Technology, Beijing, China.

[42] Z. Pan, H. Bolouri. *High Speed Face Recognition based on Discrete Cosine Transform and Neural Networks*. University of Hertfordshire, Herts, United Kingdom.
APPENDICES
APPENDIX A: DCT Image Compression Source Code

dctcomp.m

% dctcomp.m
% Written by Jawad Nagi
% Date: 13th July 2006
% Last modified: 3rd August 2006
%

clear all
clc

% Reading image
I = imread('hayyan3.jpg');

% Resizing original image
%(512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

% Converting image data to double
I = im2double(I);

% Generating random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T);

% Applying the mask
%(8 DCT coefficients selected)
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

% Regenerating the image with mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],''P1*x*P2','T',T);

% Displaying original and compressed images
figure, imshow(I)
figure, imshow(I2)

% Writing compressed image to file
imwrite(I2,'h3.jpg');
APPENDIX B: Loading Images Source Code

loaddata.m

%loaddata.m
%
%Written by Jawad Nagi
%Date: 7th October 2006
%Last modified: 25 October 2006
%
%Loading all DCT images
a1=imread('a1.jpg');
a2=imread('a2.jpg');
a3=imread('a3.jpg');
a4=imread('a4.jpg');
a5=imread('a5.jpg');

f1=imread('f1.jpg');
f2=imread('f2.jpg');
f3=imread('f3.jpg');
f4=imread('f4.jpg');
f5=imread('f5.jpg');

h1=imread('h1.jpg');
h2=imread('h2.jpg');
h3=imread('h3.jpg');
h4=imread('h4.jpg');
h5=imread('h5.jpg');

n1=imread('n1.jpg');
n2=imread('n2.jpg');
n3=imread('n3.jpg');
n4=imread('n4.jpg');
n5=imread('n5.jpg');

r1=imread('r1.jpg');
r2=imread('r2.jpg');
r3=imread('r3.jpg');
r4=imread('r4.jpg');
r5=imread('r5.jpg');

%Reshaping data into a 64x1 array
a1=double(reshape(a1,64,1));
a2=double(reshape(a2,64,1));
a3=double(reshape(a3,64,1));
a4=double(reshape(a4,64,1));
a5=double(reshape(a5,64,1));

f1=double(reshape(f1,64,1));
f2=double(reshape(f2,64,1));
f3=double(reshape(f3,64,1));
f4=double(reshape(f4,64,1));  
f5=double(reshape(f5,64,1));  
h1=double(reshape(h1,64,1));  
h2=double(reshape(h2,64,1));  
h3=double(reshape(h3,64,1));  
h4=double(reshape(h4,64,1));  
h5=double(reshape(h5,64,1));  
n1=double(reshape(n1,64,1));  
n2=double(reshape(n2,64,1));  
n3=double(reshape(n3,64,1));  
n4=double(reshape(n4,64,1));  
n5=double(reshape(n5,64,1));  
r1=double(reshape(r1,64,1));  
r2=double(reshape(r2,64,1));  
r3=double(reshape(r3,64,1));  
r4=double(reshape(r4,64,1));  
r5=double(reshape(r5,64,1));  

%Adding training data into vectors  
a=[a1 a2 a3 a4 a5];  
f=[f1 f2 f3 f4 f5];  
h=[h1 h2 h3 h4 h5];  
n=[n1 n2 n3 n4 n5];  
r=[r1 r2 r3 r4 r5];  

%Saving training data into file  
save('data.mat','a','f','h','n','r');
APPENDIX C: SOM Neural Network Simulation Source Code

simsom.m

% Written by Jawad Naqi
% Date: 8th January 2006
% Last modified: 26th February 2006

clc
clear all

% Load saved training data into workspace
load data.mat

% Declare training data
P=[a f h n r];

% Create SOM neural network
%(64 minimum points and 64 maximum points)
net=newsom(minmax(P),[64 2]);

% Train SOM neural network for 1000 epochs
net.trainParam.epochs=100;
net=train(net,P);

% Plot SOM layer weights
x=net.iw{1,1}.net.layers{1}.distances;
figure, plot(dist(x),'o');
title('SOM Layer Weights');
xlabel('Layer Weights');
ylabel('Magnitude');

% Plot SOM weight vectors
figure, plotsom(net.iw{1,1},net.layers{1}.distances);

sx=sim(net,P);
[l,m]=size(sx);

for u=1:l:m
    sm(u)=find(sx(:,u));
end

% Load input face image
Q=imread('n6.jpg');
Q=double(reshape(Q,64,1));

% Simulate SOM neural network for input face image
msx=sim(net,Q);
msxm=find(msx);
```matlab
% Classification of subject and displaying output
ans = find((sm - msxm)==min(abs((sm-msxm))));

if (ans > 0) & (ans <= 5)
    disp('Subject is identified to be: Asif');
else if (ans > 5) & (ans <= 10)
    disp('Subject is identified to be: Fahad');
else if (ans > 10) & (ans <= 15)
    disp('Subject is identified to be: Hayyan');
else if (ans > 15) & (ans <= 20)
    disp('Subject is identified to be: Naseem');
else if (ans > 20) & (ans <= 25)
    disp('Subject is identified to be: Rafay');
else
    disp('Subject is not identified in the training database.');
end
end
end
```
facerecog.m

function varargout = facerecog_demo(varargin)
% FACERECOG DEMO M-file for facerecog_demo.fig
% Last Modified by GUIDE v2.5 16-Apr-2007 17:33:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ... 
    'gui_Singleton',  gui_Singleton, ... 
    'gui_OpeningFcn', @facerecog_demo_OpeningFcn, ... 
    'gui_OutputFcn',  @facerecog_demo_OutputFcn, ... 
    'gui_LayoutFcn',  [] , ... 
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before facerecog_demo is made visible.
function facerecog_demo_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to facerecog_demo (see VARARGIN)

% Choose default command line output for facerecog_demo
handles.output = hObject;

% Update handles structure
guida(hObject, handles);

% UIWAIT makes facerecog_demo wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = facerecog_demo_OutputFcn(hObject, eventdata, handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject      handle to figure
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton2.
function pushbutton1_Callback(hObject, eventdata, handles)
dellallfiles

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
dctexample

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton2.
function pushbutton3_Callback(hObject, eventdata, handles)
inpimgdisp

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton4_Callback(hObject, eventdata, handles)
traindbdisp

% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
dctallimg

% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton5.
function pushbutton6_Callback(hObject, eventdata, handles)
loadimgdata

% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton6.
function pushbutton7_Callback(hObject, eventdata, handles)
trainsimsom
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
dctallimg
loadimgdata
trainimsom

% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clear all
clc

delete dctexample2.jpg

delete a1.jpg
delete a2.jpg
delete a3.jpg
delete a4.jpg
delete a5.jpg

delete f1.jpg
delete f2.jpg
delete f3.jpg
delete f4.jpg
delete f5.jpg

delete h1.jpg
delete h2.jpg
delete h3.jpg
delete h4.jpg
delete h5.jpg

delete n1.jpg
delete n2.jpg
delete n3.jpg
delete n4.jpg
delete n5.jpg
delete n6.jpg

delete r1.jpg
delete r2.jpg
delete r3.jpg
delete r4.jpg
delete r5.jpg

delete imgdata.mat

clc

disp(' ');
disp('Cleared all variables and deleted all generated files.');
disp(' ');
delete dctexample2.jpg

clear all
clc

%Reading image
I = imread('dctexample1.jpg');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'
P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'
P1.*x',mask);
I2 = blkproc(B2,[8 8],'
P1*x*P2',T',T);

%Displaying images
figure
subplot(1,2,1), subimage(I)
subplot(1,2,2), subimage(I2)

%Writing the compressed image to file
imwrite(I2,'dctexample2.jpg');

clear all
clc

disp('');
disp('DCT Example: Image compression successful');
disp('');
inpimgdisp.m

```
inppic = imread('naseem6.jpg');
figure, imshow(inppic)

disp(' ');
disp('Displaying input face image.');
disp(' ');
```
traindbdisp.m

inppic=imread('naseem6.jpg');
traindbpic=imread('traindb.jpg');

figure, imshow(traindbpic)

figure
subplot(1,2,1), subimage(inppic)
subplot(1,2,2), subimage(traindbpic)

disp(' ');
disp('Displaying training database set.');
disp(' ');}
delete a1.jpg
delete a2.jpg
delete a3.jpg
delete a4.jpg
delete a5.jpg

clear all
clc

%Reading image
I = imread('asif1.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'a1.jpg');

clear all
clc

%Reading image
I = imread('asif2.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');
%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'a2.jpg');

clear all
clc

%Reading image
I = imread('asif3.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
% Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

% Writing the compressed image to file
imwrite(I2,'a3.jpg');

clear all
clc

% Reading image
I = imread('asif4.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T);

% Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

% Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

% Writing the compressed image to file
imwrite(I2,'a4.jpg');

clear all
clc

% Reading image
I = imread('asif5.jpg');
% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I, [8 8], 'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I, [8 8], 'P1*x*P2', T, T);

% Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

% Regenerating the image with the mask
B2 = blkproc(B, [8 8], 'P1.*x', mask);
I2 = blkproc(B2, [8 8], 'P1*x*P2', T', T);

% Writing the compressed image to file
imwrite(I2, 'a5.jpg');

delete f1.jpg
delete f2.jpg
delete f3.jpg
delete f4.jpg
delete f5.jpg
clear all
clc

% Reading image
I = imread('fahad1.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I, [8 8], 'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I, [8 8], 'P1*x*P2', T, T);
%Applying the mask
mask = [1 1 1 1 0 0 0 0 
1 1 1 0 0 0 0 0 
1 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 ];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T);

%Writing the compressed image to file
imwrite(I2,'f1.jpg');

clear all
clc

%Reading image
I = imread('fahad2.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0 
1 1 1 0 0 0 0 0 
1 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 
0 0 0 0 0 0 0 0 ];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T);

%Writing the compressed image to file
imwrite(I2,'f2.jpg');
clear all
clc

%Reading image
I = imread('fahad3.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x',mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'f3.jpg');

clear all
clc

%Reading image
I = imread('fahad4.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2',T,T');
%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],P1.*x*,mask);
I2 = blkproc(B2,[8 8],P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'f4.jpg');

clear all
clc

%Reading image
I = imread('fahad5.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],P1.*x*,mask);
I2 = blkproc(B2,[8 8],P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'f5.jpg');
clear all
clc

%Reading image
I = imread('hayyan1.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T);

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T);

%Writing the compressed image to file
imwrite(I2,'h1.jpg');

clear all
clc

%Reading image
I = imread('hayyan2.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);
%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'h2.jpg');

clear all
clc

%Reading image
I = imread('hayyan3.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'h3.jpg');

clear all
clc

%Reading image
I = imread('hayyan4.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'h4.jpg');

clear all
clc

%Reading image
I = imread('hayyan5.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8],'nearest');
% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I, [8 8], 'P1*x*P2', T, T);

% Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

% Regenerating the image with the mask
B2 = blkproc(B, [8 8], 'P1.*x', mask);
I2 = blkproc(B2, [8 8], 'P1*x*P2', T', T);

% Writing the compressed image to file
imwrite(I2, 'h5.jpg');

% Deleting original images
delete n1.jpg
delete n2.jpg
delete n3.jpg
delete n4.jpg
delete n5.jpg
delete n6.jpg

clear all
clc

% Reading image
I = imread('naseem1.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I, [8 8], 'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I, [8 8], 'P1*x*P2', T, T);
%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'n1.jpg');

clear all
clc

%Reading image
I = imread('naseem2.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I=imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T);

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'n2.jpg');
clear all
clc

%Reading image
I = imread('naseem3.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
         1 1 1 0 0 0 0 0
         1 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'n3.jpg');

clear all
clc

%Reading image
I = imread('naseem4.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');
%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],"P1.*x",mask);
I2 = blkproc(B2,[8 8],"P1*x*P2",T',T);

%Writing the compressed image to file
imwrite(I2,'n4.jpg');

clear all
clc

%Reading image
I = imread('naseem5.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],"nearest");

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],"P1*x*P2",T,T);

%Applying the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],"P1.*x",mask);
I2 = blkproc(B2,[8 8],"P1*x*P2",T',T);

%Writing the compressed image to file
imwrite(I2,'n5.jpg');
clear all
clc

%Reading image
I = imread('naseem6.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T);

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

%Writing the compressed image to file
imwrite(I2,'n6.jpg');
delete r1.jpg
delete r2.jpg
delete r3.jpg
delete r4.jpg
delete r5.jpg
clear all
clc

%Reading image
I = imread('rafay1.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

%Convert image to double
I = im2double(I);
%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

%Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T');

%Writing the compressed image to file
imwrite(I2,'r1.jpg');

clear all
c_SC

%Reading image
I = imread('rafay2.jpg');

%Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8], 'nearest');

%Convert image to double
I = im2double(I);

%Random 8x8 DCT matrix
T = dctmtx(8);

%Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T');

%Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
% Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

% Writing the compressed image to file
imwrite(I2,'r2.jpg');

clear all
clc

% Reading image
I = imread('rafay3.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8],'P1*x*P2',T,T');

% Appyling the mask
mask = [1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

% Regenerating the image with the mask
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

% Writing the compressed image to file
imwrite(I2,'r3.jpg');

clear all
clc

% Reading image
I = imread('rafay4.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8],'nearest');
% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T');

% Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

% Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T);

% Writing the compressed image to file
imwrite(I2,'r4.jpg');

clear all
clc

% Reading image
I = imread('rafay5.jpg');

% Resizing original image (512x512 pixels => 8x8 pixels)
I = imresize(I,[8 8], 'nearest');

% Convert image to double
I = im2double(I);

% Random 8x8 DCT matrix
T = dctmtx(8);

% Taking 8x8 DCT of each block of the image
B = blkproc(I,[8 8], 'P1*x*P2', T, T');

% Applying the mask
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
Regenerating the image with the mask
B2 = blkproc(B,[8 8], 'P1.*x', mask);
I2 = blkproc(B2,[8 8], 'P1*x*P2', T', T);

Writing the compressed image to file
imwrite(I2, 'r5.jpg');

disp(' ');
disp('All face images have been compressed with the 2-D DCT successfully. ');
disp(' ');}
%load all images
a1=imread('a1.jpg');
a2=imread('a2.jpg');
a3=imread('a3.jpg');
a4=imread('a4.jpg');
a5=imread('a5.jpg');

f1=imread('f1.jpg');
f2=imread('f2.jpg');
f3=imread('f3.jpg');
f4=imread('f4.jpg');
f5=imread('f5.jpg');

h1=imread('h1.jpg');
h2=imread('h2.jpg');
h3=imread('h3.jpg');
h4=imread('h4.jpg');
h5=imread('h5.jpg');

n1=imread('n1.jpg');
n2=imread('n2.jpg');
n3=imread('n3.jpg');
n4=imread('n4.jpg');
n5=imread('n5.jpg');

r1=imread('r1.jpg');
r2=imread('r2.jpg');
r3=imread('r3.jpg');
r4=imread('r4.jpg');
r5=imread('r5.jpg');

%reshape data matrices
a1=double(reshape(a1,64,1));
a2=double(reshape(a2,64,1));
a3=double(reshape(a3,64,1));
a4=double(reshape(a4,64,1));
a5=double(reshape(a5,64,1));

f1=double(reshape(f1,64,1));
f2=double(reshape(f2,64,1));
f3=double(reshape(f3,64,1));
f4=double(reshape(f4,64,1));
f5=double(reshape(f5,64,1));

h1=double(reshape(h1,64,1));
h2=double(reshape(h2,64,1));
h3=double(reshape(h3,64,1));
h4=double(reshape(h4,64,1));
h5=double(reshape(h5,64,1));

n1=double(reshape(n1,64,1));
n2=double(reshape(n2,64,1));
n3=double(reshape(n3,64,1));
n4=double(reshape(n4,64,1));
n5=double(reshape(n5,64,1));

r1=double(reshape(r1,64,1));
r2=double(reshape(r2,64,1));
r3=double(reshape(r3,64,1));
r4=double(reshape(r4,64,1));
r5=double(reshape(r5,64,1));

a=[a1 a2 a3 a4 a5];
f=[f1 f2 f3 f4 f5];
h=[h1 h2 h3 h4 h5];
n=[n1 n2 n3 n4 n5];
r=[r1 r2 r3 r4 r5];

%save data into file
save('imgdata.mat','a','f','h','n','r');

disp('');
disp('All image data has been loaded into MATLAB workspace.');
disp('');
tic;

t1=toc;
%loading image data
load imgdata.mat;

%add data into vectors
P=[a f h n r];

%training SOM neural network
disp ('Creating SOM neural network with minimum maximum arguments...');
disp ('No. of input neuron for each image => 62 max pts + 64 minimum pts => 128');
disp (' ');
net=newsom(minmax(P),[64 2])
t2=toc;
disp (' ');
disp ('Training SOM neural network for: 1000 epochs');
disp (' ');
net.trainParam.epochs=500;
net=train(net,P)
t3=toc;

%plot SOM weight vectors
figure, plotsom(net.iw{1,1},net.layers{1}.distances);

%plot SOM layer weights
x=net.iw{1,1},net.layers{1}.distances;
figure, plot(dist(x),'o');
title('SOM Layer Weights');
xlabel('Layer Weights');
ylabel('Magnitude');

%simulate SOM neural network
disp (' ');
disp ('Simulating SOM neural network for the training database...');
disp (' ');

sx=sim(net,P)
[l,m]=size(sx);

for u=1:l:m
    sm(u)=find(sx(:,u));
end

%read input image
Q=imread('n6.jpg');
imginp=double(reshape(Q,64,1));
disp ('');
disp ('Simulating SOM neural network for the input image...');
disp ('');
msx = sim(net, imginp)
msxm = find(msx);

% classification and displaying output
disp ('');
disp ('Classifying the subject in input image with the matched image in the training database...');
disp ('');
ans = find((sm - msxm) == min(abs((sm - msxm))));

if (ans > 0) & (ans <= 5)
disp('Subject is identified to be: Asif');
else if (ans > 5) & (ans <= 10)
disp('Subject is identified to be: Fahad');
else if (ans > 10) & (ans <= 15)
disp('Subject is identified to be: Hayyan');
else if (ans > 15) & (ans <= 20)
disp('Subject is identified to be: Naseem');
else if (ans > 20) & (ans <= 25)
disp('Subject is identified to be: Rafay');
else
disp('Subject is not identified in the training database.');
end
end
end

if (ans == 16)
    pic = imread('naseem1.jpg');
else if (ans == 17)
    pic = imread('naseem2.jpg');
else if (ans == 18)
    pic = imread('naseem3.jpg');
else if (ans == 19)
    pic = imread('naseem4.jpg');
else if (ans == 20)
    pic = imread('naseem5.jpg');
end
end
end
end

inporigimg = imread('naseem6.jpg');
if (ans > 15) & (ans <= 20)
    disp ('');
    disp('--------------------------------------------------------------------- ');
    disp('Subject has been identified correctly from the input image.');
    disp('Displaying input and training database match results...');
    disp('--------------------------------------------------------------------- ')
    disp (' ');
    figure
    subplot(1,2,1), subimage(inporigimg)
    subplot(1,2,2), subimage(pic)
else
    disp ('');
    disp('--------------------------------------------------------------------- ');
    disp('A possible error has occurred!');
    disp('Please increase the number of training epochs and run the program again. ');
    disp (' ');
end

 t4=toc;

%generate simulink model
 disp('Generating Simulink model of SOM neural network...');
 disp (' ');
 gensim(net);

%displaying training time
 disp('--------------------------------------------------------------------- ');
 disp('Displaying training time and program execute time. ');
 disp (' ');

 training_time=t3-t2
 program_execution_time=t4-t1

%end of program