NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

# IMAGE PROCESSING TECHNIQUES AND NEURAL NETWOKS

Graduation Project
COM - 400

Student: Adeeb Al-Husseini

Supervisor: Asst. Prof. Dr Firudin Muradov

Nicosia - 2004

## ACKNOWLEDGMENT

*Praise be to GOD Most Gracious most Merciful*

First of all, I would like to extend my utmost and deepest thanks to my advisor, Asst. Prof. Dr. Firudin, for the intellectual support, encouragement, and enthusiasm, which made this project possible.

I also pay tribute to my dearest parents, my friends and all my computer engineering dept. staff members specially Assoc. Prof. Dr. Adnan Kashman, Assoc. Prof. Dr. Rahib Abiyev , Mr. Umit ilhan, whom had taught me that no dream is impossible.

I am so happy and excited to complete the task which I had been given with the blessing of God and also I am grateful to all the people in my life whom had supported me, advised me, taught me and whom had always encouraged me to follow my dreams and ambitions.

Also, my sincerest thanks must go to my friends, Dalia Hamdan , Samah Shoman, Izz Al Deen Hanoun, Yazan Kassawne ,Qais Albene, Ahmad gharaibeh, Ahmad taweel, Tayeb alatawne, magda, gintare, and specially my brother, Mahmoud Al-husseini, whom shared their suggestions and evaluations throughout the completion of my project. The comments from these friends enabled me to present this project successfully.

Finally, I thank God for giving me the will and courage to achieve my objectives.

# ABSTRACT

Image processing is the process which can we apply over the image to manipulate some features, such as removing the noise , edge detection , filtration, segmentation and so on

A neural network can be regarded as a machine that is designed to model, the way in which the brain performs a task or a function, the neural network is usually implemented using electronic components or simulated software.

The novel idea is based on combining neural network arbitration and image processing to automatically identify the object using edge detection techniques within the image and manipulate some furthers within the image. An edge detection technique is the process of doing extracting discontinuities in the gray level. The aim of edge detection is that the objects can be recognized from only simple outlines

The project fulfills the request of combining image processing with neural network to recognize the objects within the image. In this system user can teach the computer to identify simple objects within the image, manipulate the image such as changing it to gray level, make a filtration, object segmentation, edge detection.

# TABLE OF CONTENTS

# LIST OF ABBREVIATIONS

| | |
|---|---|
| RGB | Red, Green, Blue |
| A.I.S | Artificial Intelligent System |
| I.B.M | International Business Machines |
| N.N | Neural Network |
| E.S | Expert System |
| UNIX | UNiplexed Information and Computing system |
| PCPL | Basic Combined Programming Language |
| ANSI | American National Standards Institute |
| SQL | Structured Query Language |

# INTRODUCTION

Image processing is the process which can we apply over the image to manipulate some features, such as removing the noise , edge detection , filtration, segmentation and so on.

Neural networks are computational constructs loosely modeled on the structure of the human and animal brain. They are comprised of neurons that are the information processor of a brain, and synapses, which are spaces between neurons that can be thought of as weighted buses that connect these processors.

Chapter one presents a detail of image processing techniques and some applications of it through topics covered in this chapter. It also describes the most fundamental parts of image processing.

Chapter two describes neural computing, neural networks and the types of intelligent computing, and the classification of neural network. it also show's the process of designing and learning a neural network, in addition of some application for the neural networks and where they are currently being used. Beside we are going to use the back propagation algorithm to train the neural network.

Chapter three is intended to review a quick brief on the language which used to implement the Project, and that is *C++*, and quick introduction to the database that I used in my project by using Microsoft Access.

Chapter four describes details of software implementation and the source code for each function of the software. It covers the most fundamental parts of image processing algorithms and how can we use it to implement it in the real world.

# CHAPTER 1

# IMAGE PROCESSING

## 1.1 Overview

In this chapter I will briefly give a detail view of image processing techniques and some applications of it through topics covered in this chapter. It also describes the most fundamental parts of image processing.

## 1.2 Image processing techniques

Image processing techniques can be categorized into four main areas:

1) Image compression.
2) Image restoration.
3) Image enhancement.
4) Image recognition.

### 1.2.1 Image compression

This is process where the amount of data necessary to store the image is reduced much as possible. There are two forms of image compression:

**a)** Information preserving compression: this retains all the original information within the image. This compression yields good quality decompressed images but offers small incompressins ratio.

**b)** Lossy compression: this dose not keeps all original information but has better compression ratio.

### 1.2.2 Image restoration

This involves reconstructing an original image using knowledge of the nature of degradation of the image. The image could have been corrupted through:

**1.** Loss of data.

**2.** Addition of unwanted data such as (noise).

Image restoration is often applied prior to implementing other image processing techniques in order to improve their success rate.

### 1.2.3 Image enhancement

This involves emphasizing some features within an image which are not clear to the human or a recognition system. The aim of image enhancement is to make an image more acceptable to certain applications. Its often applied prior recognition.

### 1.2.4 Image recognition

This involves classifying an object within an image or whole image into a set of known classes. This can be done in two processes:

1. Low level process:

This uses little or no prior knowledge they are used as pre-processing stages for high-level process e.g. (edge detection).

2. High-level process:

This uses prior knowledge, where it takes the output of low-level process and further process it e.g. (template matching).

### 1.2.5 Problems with image processing techniques

There are five main problems with image processing techniques:

a) Speed.

b) Computational expenses.

c) Noise sensitivity and scale dependency.

d) Low quality edge detection.

e) Large amount of data.

#### 1.2.5.1 Speed

Speed could be a problem if the size of image is large.

#### 1.2.5.2 Computational expenses

This also related to image size a compromise is needed between image size and quality.

#### 1.2.5.3 Noise sensitivity and scale dependency

Images can have noise and effects of noise is mostly shown when changing scale.

#### 1.2.5.4 Low quality edge detection

Edge detection is low-level process that is applied prior to higher process. Usually results are of low quality.

#### 1.2.5.5 Large amount of data

Because of large amount of data more powerful computers are required. And for basic processing parallel processing is needed like NN

## 1.3 Sequence of Image Processing

- Image processing.
- Image segmentation.
- Nural network.

### 1.3.1 Image processing

Image Bitmaps are defined as a regular rectangular mesh of cells called Pixels. Each pixel containing a color value. Bitmaps are used to represent images on the computer.

#### 8-bit image

In this case each pixel takes 1 byte (8 bits) of storage resulting in 256 different states_256 different color .By convention 0 is normally black and 256 is white. All other colors lie in between, for example, 136 represent the color brown, and the bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by 1 byte index into the color palette

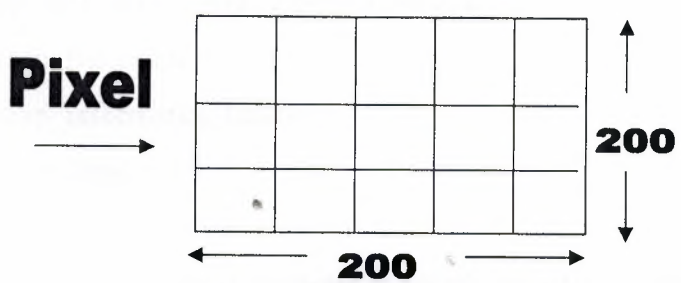


Figure 1.1 8-bit image

**24-bit image**

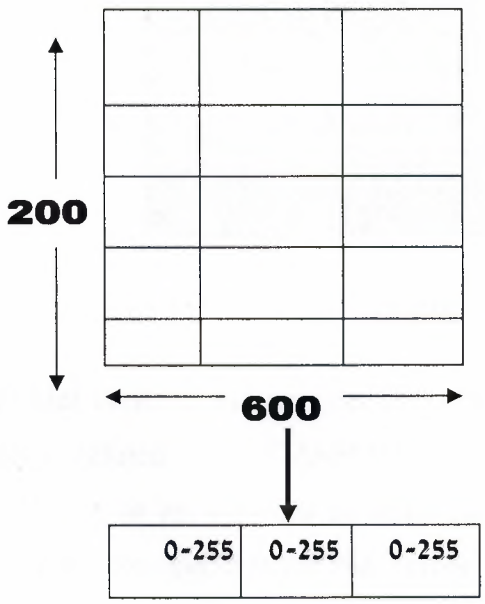


Figure 1.2 24-bit image

This consists of three different types (each 8 bits): red, green, and blue each green, blue, and red byte has its 256 combination.

### 1.3.1.1 File organization

Each Bitmap file Contains four different sections:

- Bitmap_file header.
- Bitmap_information header.
- Color palette.
- Bitmap data.

The BMP file header is 14 bytes in length, it is followed by a second header (the BMP header is 40 bytes in length), a variable sized palette, and finally, the BMP data which from one image to another.

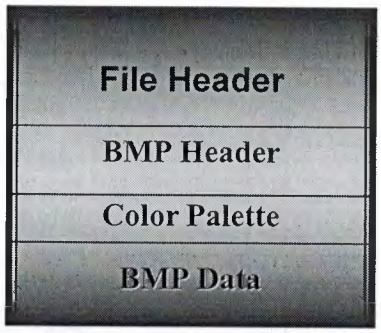


Figure 1.3 File Organization

The bitmap header contains information about the type, size, and layout of a Bitmap file, the header is defined as a BITMAPFILEHEADER structure.

The color palette is defined as an, contains as many colors as there are colors in the bitmap. The array of structures specifies the red , green , and blue intensity values of each color in a display devices color palette .each pixel in the bitmap data stores a single value used an index into the color palette .

The color information stored in the element at that index specifies the color of that pixel. The color table is not present in the 24_bit bitmaps because each pixel is represented by 24 bit red-green-blue (RGB) values in the actual bitmap data area, and having a color palette in this case could be very impractical.

The last part immediately followed by the color table, consist of an array of? BYTE values representing consecutive rows of the bitmap.

Table 1.1 Bitmap File Headers

| BitmapFile Header consists of: | Example |
|---|---|
| Type | 19778 |
| Size | 3118 |
| Reserved 1 | 0 |
| Reserved 2 | 0 |
| OffsetBits | 118 |

Table 1.2 Bitmap info Header

| | |
|---|---|
| Size | 40 |
| Width | 80 |
| Height | 75 |
| Planes | 1 |
| BitCount | 4 |
| Compression | 0 (always valued to zero in the images) |
| SizeImage | 3000 |

### 1.3.1.2 Pixel

A pixel or element is the smallest unit possible in an images, the physical size of a pixel is determinated by the display or output device that expresses it, while the computational size is limited only by the memory of the machine processing the picture, the sampling Soft an images is an expression of pixel size, the depth of the pixel, expresses the level of quantization.

**- Histogram**

Brightness histogram provides the frequency of the brightness value z in the image.

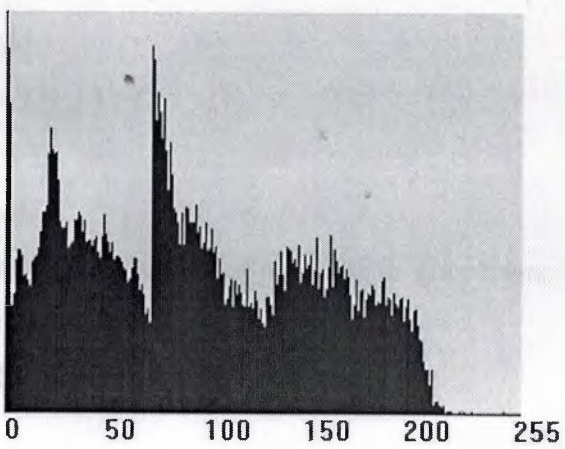


Figure 1.4 Histogram

### 1.3.1.3 Edge

Locate sharp changes in the intensity function. Edges are pixels where brightness changes abruptly.

Calculus describes change of continuous function using derivatives; an image function depends on two variables – partial derivatives. A change of the image function can be described by a gradient that points in the direction of the largest growth of the image function. An edge is property attached to an individual pixel and is calculated from the image function behavior in a neighborhood of the pixel. It is a vector variable:

- Magnitude of the gradient.
- Direction Ø.

The gradient direction gives the direction of maximal growth of the function, from black (f (i,j)=0) to white (f ( i,j)=255) .

This is illustrated below; closed lines of the same brightness. The orientation 0 points east. Edges are often used in image analysis for finding region boundaries. Boundary and its parts (edges) are perpendicular to the direction of the gradient.

### 1.3.1.4 Threshold

- Gray level threshold is the simplest segmentation process.
- Many objects or image region are characterized by constant reflectivity or light absorption their surface.
- Thresholding is computationally inexpensive and fast.
- Thresholding can easily be done in real time using specialized hardware.
- Complete segmentation can result from threshold in simple scenes.
- Normal threshold:

  g (i ,j) = 1 for f (i ,j ) >T

  = 0 for f (i, j) < T.
- Binary threshold.

Segmentation an image into region of pixels with gray levels from a set D and into background otherwise

g (i, j) = 1 for f (i ,j ) Ø D

= 0 otherwise.

- Can also serve border detection.

### 1.3.2 Image Segmentation

One of the most important steps leading to the analysis of processed image data. Its main goal is to divide an image into parts that have a strong correlation with objects or areas of the real world contain. Complete segmentation –set of disjoint regions uniquely corresponding with objects in the input image. Cooperation with higher processing level which uses specific knowledge of the problem domain is necessary. Partial segmentation – regions do not correspond directly with image objects. Image is divided into separate regions that are homogeneous with respect to a chosen property such as brightness, color, reflectivity, texture, etc.

In a complex scene, a set of possibly overlapping homogeneous regions may result. The partially segmented image must then be subjected to further processing, and the final image segmentation may be found with the help of higher level information. Simple segmentation problems:

Contrasted objects on a uniform background.

Simple assembly tasks, blood cell, printed characters, ect.

Totally correct and complete segmentation of complex scenes usually cannot be achieved in this processing phase. A reasonable aim is to use partial segmentation as an input to higher level processing.

#### 1.3.2.1 Segmentation problems

- Image data ambiguity.
- Information noise.

#### 1.3.2.2 Segmentation method

- Global approaches, e.g. using histogram of image features.
- Edge-based segmentations.
- Region-based segmentations.
- Characteristics used in edge detection or region growing:

  Brightness.

  Texture.

  Velocity field.
- Edge-based and region-bases segmentation approaches solve a dual problem border x region.

- Because of different natures of the various edge-and region–and algorithms, the may be expected to give somewhat different results and consequently different information.
- The segmentation results of these two approaches can therefore be combined in a single description structure.

# CHAPTER 2

# NEURAL NETWORKS

## 2.1 Overview

This chapter describes neural computing, neural networks and the types of intelligent computing, and the classification of neural network. it also show's the process of designing and learning a neural network, in addition of some application for the neural networks and where they are currently being used. Beside we are going to use the back propagation algorithm to train the neural network.

## 2.2 Neural Computing

It is a concept of processing data based on the way neurons in the brain process information of communication with each other .neural c omputing i s p erformed u sing artificial neural network.

In order to know how dose it works we have to look how the brain works. The brain consists of neurons it is high complex, on-linear and parallel computers.

It has capability to organize the neurons so as to perform certain computations many times faster, than the fastest digital computer existence now days.

## 2.3 Neural Networks

A neural network is massively parallel distributed processors that has a neutral ability for storing experimental knowledge and make it available for late use.

The neural network resembles the brain in two respects:

1. Knowledge is acquired by a network thought a learning process.
2. interconnection between neurons; known as synaptic weights and used to store the knowledge.

## 2.4 Artificial Neural Networks

Artificial Neural Networks is a system loosely modeled on the human brain. The field goes by many, such as connectionism, parallel distributed processing.

Neuro–computing, neural intelligent systems, machine learning algorithms, and artificial neural networks, it is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neuron, Each neuron is linked to cretin of its neighbors with varying coefficient of connectivity that represent the strengths of these connections Learning is accomplished by adjusting these strengths to cause the overall network to output appropriate result.

### 2.4.1 The Analogy to the Brain

The most Basic component of neural networks are modeled after the structure of the brain. Some neural network structures are not closely to the brain and some does not have a biological counterpart in the brain.

However, neural networks have a strong similarity to the biological brain and therefore great deal of the terminology is borrowed from neuroscience.

The most basic of the human brain is a specific type of cell, which provides us with the abilities to remember, think, and apply previous experience to our every action.

These cells are known as neurons, each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them.

Even though all artificial neural networks are constructed form this building block the fundamentals may very in these building blocks and these are differences.

### 2.4.2 Intelligent Computing

There are three kind of intelligent computing:

1. Artificial intelligent system (A.I.S).
2. Experts system (E.S).
3. Neural Networks (N.N).

### 2.4.2.1 Artificial Intelligent System (A.I.S)

Artificial intelligent systems used for computation and recognition .they follow this general algorithm:

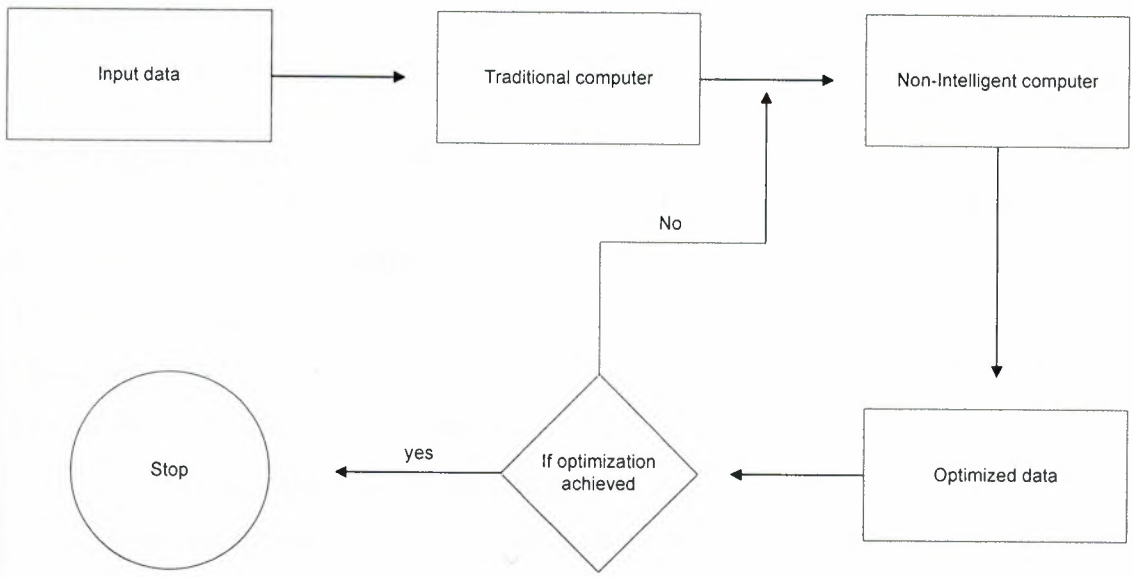


Figure 2.1 Artificial intelligent Algorithm

#### 2.4.2.1.1 Characteristics of Artificial Intelligent System (A.I.S)

1. Imitation of human reasoning process.
2. Sequential information processing.
3. Explicit knowledge representation.
4. Use of deductive reasoning.
5. Learning outside system.

### 2.4.2.2 Experts system (E.S)

Experts system is a data base system which stores detailed information given by experts to be used by non-experts .such as system is usually found in medical systems. Users see a user-friendly interface and only learn to apply using certain parameters.

### 2.4.2.3 Neural Networks (NN)

Neural networks are the data related to the brain. In a 3-pound brain, there are:

- 100 billion neuron (cells).
- 10 billion computational cells.
- 60 trillion connections.

#### 2.4.2.3.1 Characteristics of Neural Networks (NN)

1. Imitation of the structure and function of the brain.
2. Parallel information processing.
3. Implicit knowledge representation.
4. Application of inductive reasoning.
5. Learning is within the system.

## 2.5 Design Neural Network

The developer must go through a period of trial and error in the design decisions before coming up with a satisfactory design.

The design issues in neural networks are complex and are the major concerns of the system developers.

Designing a neural network consists of:

- Arranging neurons in various layers.
- Deciding the type of connections among neuron for different layers, as Well as among the neurons within a layer.
- Deciding the way a neuron receives input and produces output.
- Determining the strength of connection weights by using a training data set.

The process of designing a neural network is an iterative process.

### 2.5.1 Layers

Biologically, neural networks are constructed in three dimensional ways from microscopic component, these neuron seem capable of nearly unrestricted interconnections. This is not true in any man-made network. Artificial neural networks are the simple clustering of the primitive artificial neurons.

This clustering occurs by creating layers, which are then connected to one another .How these layers connect may also vary .basically, all artificial neural networks have a similar structure of topology .Some of the neurons interface the real world to receive its input and other neurons provide the real world with the network's output .All the rest of the neurons are hidden from view.

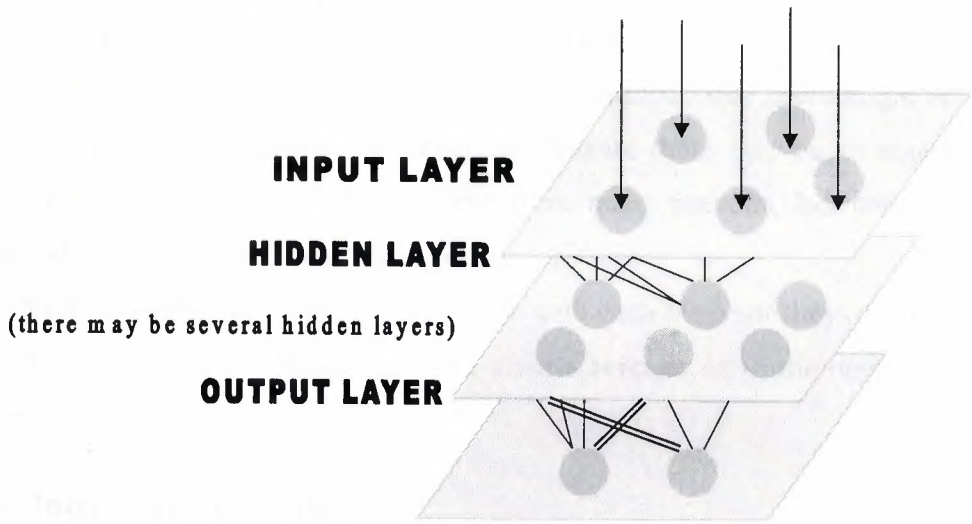


Figure 2.2. Architectural graph of Neural Network Layer

As the figure2.2 shows, the neurons are grouped into layers the input layer consists of neurons that receive input from the external environment.

The output layer consists of neurons that communicate the output of the system to the user or external environment. There are usually a number of hidden layers between these two layers.

When the input layer receives the input its neurons produce output, which becomes input to the other layers of the system. The process continues until a certain condition is satisfied or until the output layer is invoked and fires their output to the external environment.

To determine the number of hidden neurons the network should have to perform its best one are often left out to the method trial and errors, if you increase the hidden number of neurons too much you will get an over fit, that is the net will have problem to generalize. The training set of data will be memorizes, making the network useless on new data sets.

### 2.5.2 Communication and types of connections

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

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

#### 2.5.2.1 Inter-layer Connections

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

**1. Fully connected:** Each neuron on the first layer is connected to every neuron on the second layer.

**2. Partially connected:** A neuron of the first layer does not have to be connected to all neurons on the second layer.

**3. Feed forward:** The neurons on the first layers send their output to the neurons on the second layer, but they do not receive any input back from the neurons on the second layer.

**4. Bi-directional:** There is another set of connections could be fully – or partially connected.

**5. Hierarchical:** If a neural network has hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer.

**6. Resonance:** The layer has bi-directions, and they can continue sending messages across the connections a number of times until a condition is achieved.

#### 2.5.2.2 Inter-layer Connections

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

**1. Recurrent**

The neurons within a layer are fully – or partially connected to one another .After these neurons receive input form another layer, they communicate their output with one another a number of times before they are allowed to send their outputs to another layer.

Generally some conditions among the neurons of the layer should be achieved before they communicate their output to another layer.

**2. on-center / off surround**

Neuron within a layer has excitatory connections to itself and its immediate Neighbors, and has inhibitory connections to other neurons.

One can imagine this type of connection as a competitive gang of neurons each Gang excites itself and its gang members and inhibits all member of other gang, After a few round of signal interchange, the neurons with an active output value Will win, and is allowed to update its and its gang member's weights. (There are two types of connections between two neurons, excitatory or inhibitory. in the excitatory connections the output of one neurons increase the action potential of the neurons to which it is connected. When the connection type between two Neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or the action potential of the receiving neuron. One causes the Summing mechanism of the next neuron to add while the other causes it to Subtract, one excites while the other inhibits).

**2.5.3 Learning Process**

The brain basically learn form experience .Neural networks are sometimes called machine learning algorithms, because changing of its connection weight (training) Causes the network to learn the solution to a problem.

The strength of connection between the neurons is r\stored as weight-value for the specific connection, the system learn new knowledge by adjusting these connection weights.

The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training, the training method usually consists three schemes:

**Unsupervised learning**

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

**Reinforcement learning**

This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged, then reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher the teacher may be a training set of data or an observer who grades the performance of the network result. Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the paper connection weights.

**Back propagation**

This method is proven highly successful in training of multiplayer neural nets. The networks is not just given reinforcement for how it is doing on task .Information about errors is also filtered back through the system and is used to adjust the connections between the layers ,thus improving performance .A form of supervised learning.

**2.5.3.1 Off-line or On-line**

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

**1. Off-line**

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

**2. On-line**

In on-line or real time learning, when the system is not in operation mode (recall), it continues to learn while being used as decision tool. This type of learning has a more complex design structure.

### 2.5.3.2 Learning laws

The main idea in the Neural networks is to emulate the human thinking process, how would a person recognize an object a pen or a car or a friend.

An important consideration in ANN is the use of appropriate learning algorithm (training algorithms) there are a hundreds of methods, the learning algorithms can be classified as supervised and unsupervised.

There are a variety of learning laws which are in common use. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule .Man;s understanding of how neural processing actually works is very limited .

**1. Hebb's Rule**

The first and the best known learning rule was introduced by Donald Hebb .the description appeared in his book the organization of Behavior in 1949 .this basic rule is:" If a neuron receive an input form another neuron, and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened".

**2. Partially connected**

A neuron of the first layer does not have to be connected to all neurons on the second layer.

**3. Hopfield Laws**

This laws is similar to Hebb's Rule with exception that it specifies the magnitude of the strengthening or weakening .It states ,"if the desired output and the input are both active or both inactive ,increment the connection weight by the learning rate , otherwise decrement the weight by the learning rate ". (Most learning functions have some provision for a learning rate, or a learning constant. usually this term is positive and between zero and one).

**4. The Delta Rule**

Delta Rule is further variation of Hebb's Rule , and it is one of the most commonly used .this rule is based on the idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) Between the desired output value and the and the actual output of a neuron.

This rule changes the connection weight in the way that minimizes the mean squared error of the network. The error is back propagated into pervious layers one at a time.

The process of the error is back propagating the network errors continues until the first layer is reached the network type called Feed forward ,back-propagation derives its name from this method of computing the error term.

This rule is also referred to as the Windrow-Hoff Learning Rule and the Least Mean Square Learning Rule.

**5. Kohonen's Learning Law**

This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the neurons compete for the opportunity to learn, or to update their weights.

The processing neuron with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors are allowed to update their connection weights.

The kohonen rule does not require desired output . therefore it is implemented in the unsupervised methods of learning. Kohonen has used this rule combined with the on – center/off-surround intra-layer connection to create the self-organizing neural network, which has unsupervised learning. Only communicate with neurons on the next level of layer

## 2.6 Classification of Neural Networks (N.N)

A neural network can be classified to three man section according to:

1. Flow of information.
2. Their way of learning.
3. There objective function.

### 2.6.1 Classification of (N.N) according to flow of information

**1. Feed forward N.N**

Input data flows from (bottom to top) or (left to right) of the network.

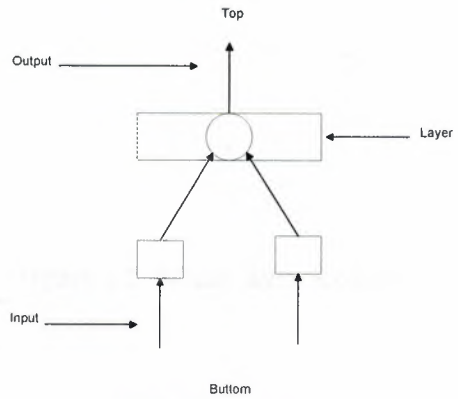


Figure 2.3 Single Layer N.N (information flow from bottom to top)

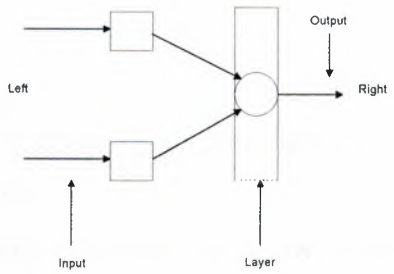


Figure 2.4 Single Layer N.N (information flow from bottom to top)

There are three main parts of a N.N:

**1. Input object:** non-processing element and normally contain a code for input data they acts as gates

**2. Hidden object:** these contain processing elements (neurons).and can be as many layers as the N.N design needs.

**3. Output object:** processing element (neurons) and provide the N.N output.

**2. Recurrent N.N (feedback)**

Here data is feedback from output layers to previous layer.

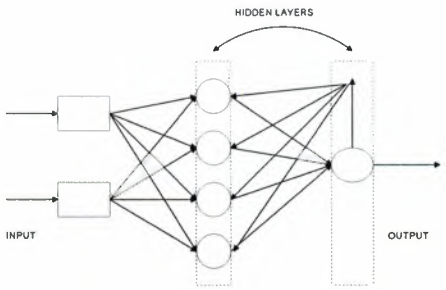


Figure 2.5 Multi layer feedback N.N

### 2.6.2 Classification of (N.N) according to there way of learning

Neural Network (N.N) can be classified according to the way they learn:

a- Supervised basis.

b- Unsupervised basis.

### 2.6.2.1 Supervised Learning

In a supervised learning process the input data and its corresponding output are presented to the neural network.

These networks will, according to defined law, change its weight in order to be able to reproduce the correct output (target).

Example of supervised learners:

1. Perceptron.
2. Back propagation algorithm.
3. Hopfield algorithm.
4. Hamming algorithm.

#### 2.6.2.1.1 Perceptron

A perceptron model is the building block of neural networks .its the smallest network with two input nodes and one processing output neuron. It can make decision and learn.When teaching a percepton an input and output are provided with each example.

The difference between the target and output is called error .this used to update and change the weight, thus giving the perceptron new memory.

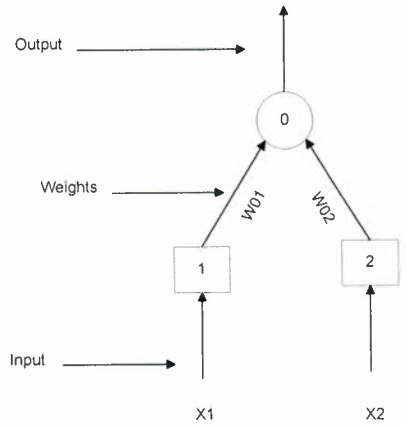


Figure 2.6 Architectural graph of Perceptron

**2.6.2.1.2 Back Propagation Algorithm**

*Back propagation* is short for back *error propagation* .the word PROPAGATION is synonymous to broadcast , dissemination promulgation ,circulation , and if we take the whole name it means disseminate the error of a result back to the input in order to rectify the result . Back propagation (Bp) method is the *Generalized Delta Rule*. There are two major steps in the Back Propagation (Bp):

**Forward Pass:**

The forward pass the network works as usual with each input multiplied by it own random weight, then a transfer function is applied to the result (depending on the number of layers) then armed with actual required and error value we start the backward pass .

**Backward Pass:**

The backward pass the process is only to enhance the weight so that new weights will be calculated to be used again in the forward pass, the Network usually has one more hidden layers

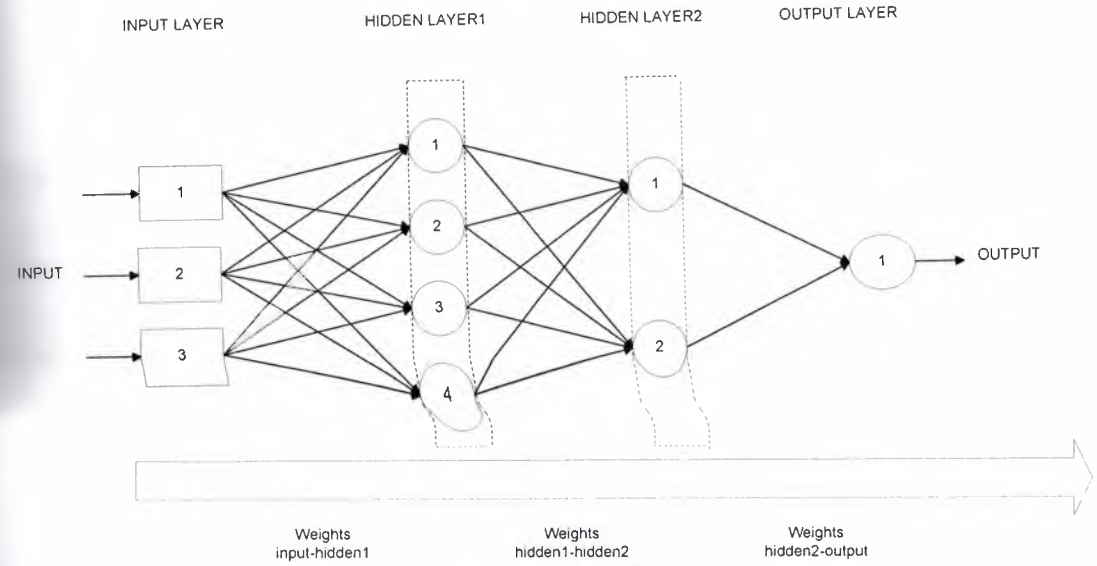


Figure 2.7 Architectural graph of Back Propagation algorithm

In general the Bp works as follows:

- Initialize weight (W ij) with random values and set parameters
- Read in the inputs (Xi) vector and the desired output
- Compute the error =Desired output – Actual output or (delta) = Zj-Yj.
- Change the weights by working backwards from the output layer through the hidden layers.

In the figure (2.8) I will show the network that I used in my project:

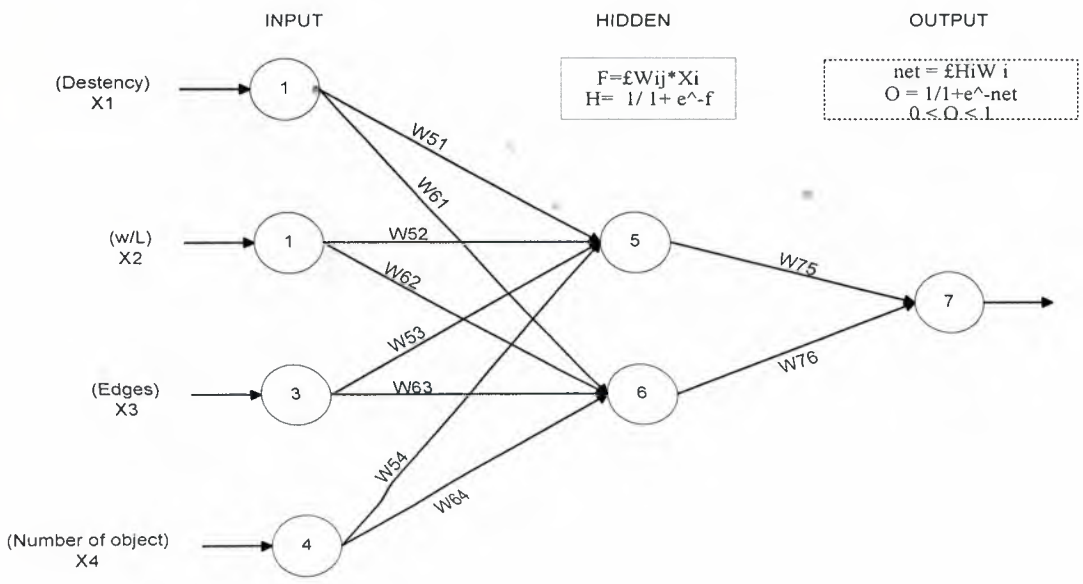

Figure 2.8 Back Propagation algorithm for software implementation

**2.6.2.1.3 Hopfield Network**

This network accepts binary inputs. Inputs are represented to the network in discrete times it used feedback.

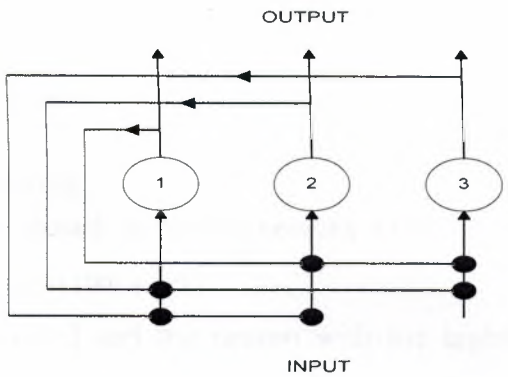


Figure 2.8 Architectural graph of Hopfield Network

**2.6.2.1.4 Hamming Network**

This network based on Hopfield network and it has four layers:

- L1: Input layer.
- L2: Score matching layer.
- L3: Hopfield network.
- L4: Output layer.

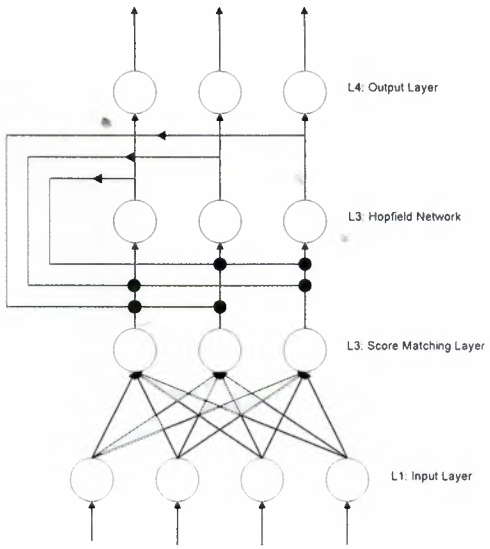


Figure 2.9 Architectural graph of Hamming Network

### 2.6.2.2 Unsupervised Learning

In this kind of learning only input vectors are presented to the network once this input data is given to the network the weights are adjusted in an ordered way according to some figure of merit.

Example of supervised learners:

1. Kohonen's self-organizing maps.
2. Competitive learning.
3. Adaptive Resonance Theory.

#### 2.6.2.2.1 Kohonen's Learning

This type of learning based on placing neurons within layers in an organized way. Assuming two layers arranged like a grid.

The dot product is calculated and the neuron with the highest value is allowed to process, together with its neighbors.

#### 2.6.2.2.2 Competitive Learning

In this type of learning the neurons are organized into inhibitory clusters. The cluster with the highest total-dot product is allowed to learn and the other switches off.

#### 2.6.2.2.3 Adaptive Learning

In this type of learning it can process both binary and continuous input data.

### 2.6.3 Classification of (N.N) according to there objective functions

Neural networks can be classified according to there objective function into four parts:

1. Classification.
2. Association.
3. Optimization.
4. Self-organization.

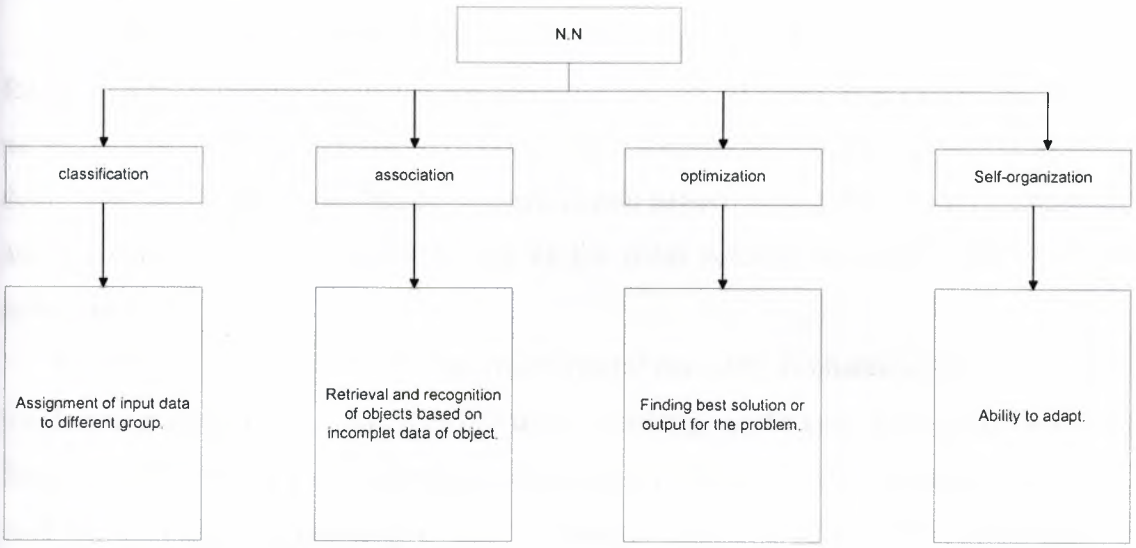


Figure 2.1 Architectural graph of N.N according to there objective function

## 2.7 Where are Neural Networks being used

Neural networks are performing successfully where other methods do not, recognizing and matching complicated, vague or incomplete patterns.

Neural networks have been applied in solving a wide variety of problems. The most common use for neural networks is to project what will most likely happen. There are many areas where prediction can help in setting priorities .for example, the emergency room at a hospital can be hectic place, to know who the most critical needs help can enable a more successful operation.

Basically, all organization must establish priorities, which govern the allocation of their resources .Neural networks have been used as a mechanism of knowledge acquisition for expert system in stock marker forecasting with astonishingly accurate results. Neural networks have also been used for bankruptcy.

Although one may apply neural network systems for interpretation, prediction, diagnosis planning, monitoring

debugging , repair , instruction , and c ontrol , the m ost s uccessful a pplications o f neural networks are in categorization and pattern recognition .such a system classifies the object under investigation (e.g. an illness , pattern , a picture ,a chemical compound ,a word ,the financial profile of a customer ) as one of numbers possible categories that ,in return , may trigger the recommendation of an action (such as a treatment plan or financial plan).

A company called Nestor, have used neural networks for financial risk assessment for mortgage insurance decisions, categorizing the risk of loans as good or bad .Neural networks has also been applied to convert text to speech, NET talk is one of the systems developed for this purpose .Image processing and pattern recognition from an important area of neural networks, probably one of the most actively research areas of neural networks.

An other of research for applications of neural networks is character recognition and handwriting recognition. This area has use in banking, credit card processing and other financial services. the pattern recognition capability of neural networks has been used to read handwriting in processing checks, the amount must normally be entered into the system by a human .A system that could automate this would expedite check processing and reduce errors. One such system has been developed by HNC (hecht-Nielsen Co.) for Bank Tec.

One of the best known applications is the bomb detector installed in some U.S. airports. This device called SNOOPE, determine the presence of cretin compounds from the chemical configuration of their components.

In a document form International Joint conference, one can find reports on using neural networks in areas ranging from robotics, speech, signal prospecting, vision, character recognition to musical composition, detection of heart malfunction and epilepsy, fish detection and classifications, optimization, and scheduling. One may take under consideration that most of the reported applications are still in research stage.

Basically, most application of neural networks falls into the following five categories:

**1. Prediction**

Uses input values to predict some output. E.g. pick the best stocks in the market, predict weather, and identify people with cancer risk.

**2. Classification**

Uses input values to determine the classification. E.g. is the input the letter A, is the blob of the video data a plane and what kind of plane is it .

**3. Data association**

Like classification but it also recognizes data that contains errors. E.g. not only identify the characters that were scanned but identify when the scanner is not working properly.

**4. Data conceptualization**

Analyze the inputs so that grouping relationships can be inferred. E.g. extract from a database the names of those most likely to by a particular product.

**5. Data Filtering**

Smooth an input signal. E.g. take the noise out of a telephone signal

# CHAPTER 3

# SYSTEM DEVELOPER

## 3.1 Overview

This chapter is intended to review a quick brief on the language which used to implement the Project, and that is *C++*, and quick introduction to the database that I used in my project by using Microsoft Access.

## 3.2 Program Implementation

The program is divided in tow two parts: the database part and the application part. For the database, to meet the flexibility, I have chosen MS ACCESS database which is one of the main topic of my overall project as well.

For the application part, C++ is my choice due to its object-oriented functionality and flexibility.

## 3.3 A Brief History of C++

The C++ Programming Language is basically an extension of the C Programming Language. The C Programming language was developed from 1969-1973 at Bell labs, at the same time the UNIX operating system was being developed there. C was a direct descendant of the language B, which was developed by Ken Thompson as a systems programming language for the fledgling UNIX operating system. B, in turn, descended from the language BCPL which was designed in the 1960s by Martin Richards while at MIT.

In 1971 Dennis Ritchie at Bell Labs extended the B language (by adding types) into what he called NB, for "New B". Ritchie credits some of his changes to language construct found in Algol68, although he states "although it [the type scheme], perhaps, did not emerge in a form that Algol's adherents would approve of" After restructuring the language and rewriting the compiler for B, Ritchie gave his new language a name: "C".

In 1983, with various versions of C floating around the computer world, ANSI established a committee that eventually published a standard for C in 1989.

In 1983 Bjarne Stroustrup at Bell Labs created C++. C++ was designed for the UNIX system environment, it represents an enhancement of the C programming language and enables programmers to improve the quality of code produced, thus making reusable code easier to write.

### 3.3.1 C in C++

To a large extent, C++ is a superset of C, and most carefully written ANSI C will compile as C++. There are a few major caveats though:

- All functions must be declared before they are used, rather than defaulting to Type int.
- All function declarations and definition headers must use new-style declarations, e.g., extern int foo(int a, char* b);

The form extern int foo(); means that foo takes no arguments, rather than arguments of an unspecified type and number. In fact, some advise using a C++ Compiler even on normal C code, because it will catch errors like misused Functions that a normal C compiler will let slide.

- If you need to link C object files together with C++, when you declare the C Functions for the C++ files, they must be done like this:

  Extern "C" int foo(int a, char* b);

  Otherwise the C++ compiler will alter the name in a strange manner.
- There are a number of new keywords, which you may not use as identifiers some common ones are new, delete, const, and class.

### 3.3.2 Basic Concepts

Before giving examples of C++ features, I will go over some of the basic concepts of Object-oriented languages.

**Classes and objects:** A class is similar to a C structure, except that the definition of the data structure, and all of the functions that operate on the data structure are Grouped together in one place. An object is an instance of a class (an instance of the data structure); objects share the same functions with other objects of the same Class, but each object (each instance) has its own copy of the data structure. A Class thus defines two aspects of the objects: the data they contain, and the Behavior they have.

**Member functions:** These are functions which are considered part of the object and are declared in the class dentition. They are often referred to as methods of the class. In addition to member functions, a class's behavior is also defined by:

(a) What to do when you create a new object (the constructor for that object) in other words initialize the object's data.

(b) What to do when you delete an object (the destructor for that object).

**Private vs. public members:** A public member of a class is one that can be read or written by anybody, in the case of a data member, or called by anybody, in the Case of a member function. A private member can only be read, written, or called by a member function of that class.

Classes are used for two main reasons:

(1) It makes it much easier to organize your Programs if you can group together data with the functions that manipulate that Data.

(2) The use of private members makes it possible to do information hiding, so that you can be more confident about the way information flows in your programs

### 3.3.3 Classes

C++ classes are similar to C structures in many ways. In fact, a C++ *struct* is really a class that has only public data members. In the following explanation of how classes work:

**Member functions:**

This class has two data members, top and stack, and one member function, Push the notation *class::function* denotes the function member of the class class. (In the Style we use, most function names are capitalized.) The function is *de_ned* beneath it. As an aside, the definition of class Stack would typically go in the file *stack.h* and the definitions of the member functions, like *Stack::Push*, would go in the file *stack.cc*. The purpose of member functions is to encapsulate the functionality of atype of object along with the data that the object contains. A member function does not take up space in an object of the class.

**Private members:**

One can declare some members of a class to be private, which are hidden to all But the member functions of that class, and some to be public, which are visible and accessible to everybody. Both data and function members can be either public or private. Fined how the data members are stored without changing how you Access them.

**Constructors and the operator new:**

In order to create a new object of type in C++ is as follows:

*Stack *s = new Stack (17);* The new function takes the place of *malloc()*. To specify how the object should be Initialized, one declares a constructor function as a member of the class, with the Name of the function being the same as the class name: Note there are two ways of providing arguments to constructors: with new, you Put the argument list after the class name, and with automatic or global variables, You put them after the variable name.

It is crucial that you always define a constructor for every class you define, and that the constructor initializes every data member of the class. If you don't define your own constructor, the compiler will automatically define one for you, and believe me, it won't do what you want (\the unhelpful compiler"). The data members will be initialized to random, unrepeatable values, and while your program may work anyway, it might not the next time you recompile (or vice versa!). The new operator can also be used to allocate arrays, illustrated above in allocating an array of int, of dimension size:

*Stack = new int[size];*

**Destructors and the operator delete:**

The destructor has the job of de allocating the data the constructor allocated. Many classes won't need destructors, and some will use them to close files and otherwise clean up after themselves.

The destructor for an object is called when the object is de allocated. If the object was created with new, then you must call delete on the object, or else the object will continue to occupy space until the program is over this is called "*a memory leak.*"

### 3.3.4 Why Programming in C++

So what is so special about C++? Why should you use C++ to develop your applications? First, C++ is not the best language to use in every instance. C++ is a great choice in most instances, but some special circumstances would be better suited to another language.

There are a few major advantages to using C++:

**1- C++ allows expression of abstract ideas.**

C++ is a third generation language that allows a programmer to express their Ideas at a high level as compared to assembly languages.

**2- C++ still allows a programmer to keep low-level control.**

Even though C++ is a third generation language, it has some of the "feel" of an Assembly language. It allows a programmer to get down into the low-level Workings and tune as necessary. C++ allows programmers strict control over Memory management.

**3- C++ has national standards (ANSI)**

C++ is a language with national standards. This is good for many reasons. Code Written in C++ that conforms to the national standards can be easily integrated with preexisting code. Also, this allows programmers to reuse certain common Libraries, so certain common functions do not need to be written more than once, and these functions behave the same anywhere they are used.

**4- C++ is reusable and object-oriented**

C++ is an object-oriented language. This makes programming conceptually easier (once the object paradigm has been learned) and allows easy reuse of code, or Parts of code through inheritance.

**5- C++ is widely used and taught**

C++ is a very widely used programming language. Because of this, there are many tools available for C++ programming, and there is a broad base of Programmers contributing to the C++ "community".

## 3.4 History of Ms Access

The Microsoft Access lines of products are not all created equal. Access 1.0 came out in 1992, which was quickly followed by version 1.1. This initial offering was a cheap ($99) desktop database with a competitive feature set that enjoyed instant popularity amongst Windows users. Access 2.0 was a major update released in May 1994. For 16 bit Windows development, Access 2.0 with the service pack to update the JET engine to v2.5 is now the industry standard. No serious development should be attempted with Access version 1.x.

The first 32 bit Windows version was Access 95 (a.k.a. Access 7.0). Released in November 1995, this is a program to be avoided at all costs. There are difficulties with the release that were never completely resolved. Access 97 (a.k.a. Access 8.0) has been around since January 1997 and is a great improvement on the previous versions. Although it has a few bugs of its own, Access 97 is quite a stable product overall with a greatly enhanced development environment. Access 2000 is a worthy successor with a greatly improved database architecture. Unfortunately, forms and other parts of the Windows interface are a bit bloated. For Windows application development, Access97 is still a preferred choice among many developers. However, if you are writing for the Web or in Visual Basic you owe it to yourself to use the 2000 database engine. Coming soon is Office XP showing Microsoft's commitment to continuously updating its big money making products.

## 3.5 Brief overview of Relational Databases and Database Applications

The first databases implemented during the 1960s and 1970s were based upon either flat data files or the hierarchical or networked data models. These methods of storing data were relatively inflexible due to their rigid structure and heavy reliance on applications programs to perform even the most routine processing.

In the late 1970s, the *relational database model* which originated in the academic research community became available in commercial implementations such as IBM DB2 and Oracle. The relational data model specifies data stored in *relations* that have some *relationships* among them (hence the name *relational*).

In relational databases such as Sybase, Oracle, IBM DB2, MS SQL Server and MS Access, data is stored in *tables* made up of one or more *columns* (Access calls a column a *field*). The data stored in each column must be of a single *data type* such as Character,

Number or Date. A collection of values from each column of a table is called a *record* or a *row* in the table.

Different tables can have the same column in common. This feature is used to explicitly specify a relationship between two tables. Values appearing in column a in one table are shared with another table.

Below are two examples of tables in a relational database for a local bank:

Table 3.1 Customer

| CustomerID | Name | Address | City | State | Zip |
|---|---|---|---|---|---|
| *Number* | *Character* | *Character* | *Character* | *Character* | *Character* |
| 1001 | Mr. Smith | 123 Lexington | Smithville | KY | 91232 |
| 1002 | Mrs. Jones | 12 Davis Ave. | Smithville | KY | 91232 |
| 1003 | Mr. Axe | 443 Grinder Ln. | Broadville | GA | 81992 |
| 1004 | Mr. & Mrs. Builder | 661 Parker Rd. | Streetville | GA | 81990 |

Table 3.2 Accounts

| CustomerID | AccountNumber | AccountType | DateOpened | Balance |
|---|---|---|---|---|
| *Number* | *Number* | *Character* | *Date* | *Number* |
| 1001 | 9987 | Checking | 10/12/1989 | 4000.00 |
| 1001 | 9980 | Savings | 10/12/1989 | 2000.00 |
| 1002 | 8811 | Savings | 01/05/1992 | 1000.00 |
| 1003 | 4422 | Checking | 12/01/1994 | 6000.00 |
| 1003 | 4433 | Savings | 12/01/1994 | 9000.00 |
| 1004 | 3322 | Savings | 08/22/1994 | 500.00 |
| 1004 | 1122 | Checking | 11/13/1988 | 800.00 |

The Customer table has 6 columns (CustomerID, Name, Address, City, State and Zip) and 4 rows (or records) of data. The Accounts table has 5 columns (CustomerID, AccountNumber, AccountType, DateOpened and Balance) with 7 rows of data.

Each of the columns conforms to one of three basic *data types*: Character, Number or Date. The data type for a column indicates the type of data values that may be stored in that column.

- Number - may only store numbers, possibly with a decimal point.
- Character - may store numbers, letters and punctuation. Access calls this data type *Text*.
- Date - may only store date and time data.

In some database implementations other data types exist such as Images (for pictures or other data). However, the above three data types are most commonly used.

Notice that the two tables share the column CustomerID and that the values of the CustomerID column in the Customer table are the same the values in the CustomerID column in the Accounts table. This *relationship* allows us to specify that the Customer *Mr. Axe* has both a Checking and a Savings account that were both opened on the same day: December 1, 1994.

Another name given to such a relationship is *Master/Detail*. In a master/detail relationship, a single master record (such as Customer 1003, Mr. Axe) can have many details records (the two accounts) associated with it.

In a Master/Detail relationship, it is possible for a Master record to exist without any Details. However, it is impossible to have a Detail record without a matching Master record. For example, a Customer may not necessarily have any account information at all. However, any account information *must* be associated with a single Customer.

Each table also must have a special column called the **Key** that is used to uniquely identify rows or records in the table. Values in a key column (or columns) may never be duplicated. In the above tables, the CustomerID is the key for the Customer table while the AccountNumber is the key for the Accounts table.

## 3.6 MS Access advantage

1. Multi- user sharing.
2. Can eliminate repeated data.
3. Data entry available.
4. Simple debugging.

5. No upper row limit.
6. Tabular or other display formats.
7. User level Security.
8. Data validation & checking.
9. Search, retrieval, & sub-sets.

## 3.7 MS Access disadvantage

1. Harder to setup.
2. Graphics harder to setup.
3. Calculations are not cell based.

In this chapter I will show the data base I used to implement my software by storing the weights (memory) which are very important for the training process to identify the objects that I will present to my neural network as follows:

Table 3.3 Objects

| IDNO | name | w11 | w12 | w21 | w22 | w31 | w32 | w41 | w42 | wh1 | wh2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | s | 1 | 3 | 43 | 2 | 3 | 2 | 1 | 2 | 5 | 6 |
| 0.01 | line | 0.91823164848 | 1.2615498129 | 0.71823164848 | 1.0615498129 | 0.51823164848 | 0.8615498129 | 0.31823164848 | 0.5615498129 | -3.4845631146 | -2.5437602101 |
| 0.02 | line | 1.42463359243 | 0.80187964618 | 1.22463359243 | 0.60187964618 | 1.02463359243 | 0.40187964618 | 0.82463359243 | 0.10187964618 | -2.8578740236 | -1.7452019861 |
| 0.03 | Full circle | 1.11915460997 | 0.90060903387 | 0.91915460997 | 0.70060903387 | 0.71915460997 | 0.50060903387 | 0.51915460997 | 0.20060903387 | -2.4179448673 | -1.4396861872 |
| 0.04 | elieps | 1.21428632603 | 0.84317195563 | 1.01428632603 | 0.64317195563 | 0.81428632603 | 0.44317195563 | 0.61428632603 | 0.14317195563 | -2.3851923418 | -1.3698646941 |
| 0.06 | line | 0.91063682090 | 0.92057420694 | 0.71063682090 | 0.72057420694 | 0.51063682090 | 0.52057420694 | 0.31063682090 | 0.22057420694 | -2.4112147896 | -1.4031616631 |
| 0.07 | rde | 1.13622555893 | 0.81351934735 | 0.93622555893 | 0.61351934735 | 0.73622555893 | 0.41351934735 | 0.53622555893 | 0.11351934735 | -1.9654020296 | -1.0264486995 |
| 0.1 | Full Circle | 1.04064543073 | 0.78465403938 | 0.84064543073 | 0.58465403938 | 0.64064543073 | 0.38465403938 | 0.44064543073 | 0.08465403938 | -1.7384762544 | -0.8124173486 |
| 0.2 | Full Circle | 0.97048032739 | 0.73199657096 | 0.77048032739 | 0.53199657096 | 0.57048032739 | 0.33199657096 | 0.37048032739 | 0.03199657096 | -1.23063839 | -0.3770832319 |
| 0.3 | Full Circle | 0.93723500512 | 0.71874779282 | 0.73723500512 | 0.51874779282 | 0.53723500512 | 0.31874779282 | 0.33723500512 | 0.01874779282 | -0.8912033533 | -0.0819987538 |
| 0.4 | Full Circle | 0.9179287678 | 0.72237807924 | 0.7179287678 | 0.52237807924 | 0.5179287678 | 0.32237807924 | 0.3179287678 | 0.02237807924 | -0.6125943198 | 0.16366894365 |
| 0.5 | Circle | 0.90042536471 | 0.70032745499 | 0.70042536471 | 0.50032745499 | 0.50042536471 | 0.30032745499 | 0.30042536471 | 0.00032745499 | -0.3519009222 | 0.41527368207 |
| 0.6 | Circle | 0.89983530684 | 0.74569440933 | 0.69983530684 | 0.54569440933 | 0.49983530684 | 0.34569440933 | 0.29983530684 | 0.04569440933 | -0.0616327127 | 0.66547481139 |
| 0.7 | Circle | 0.90010017304 | 0.80991798268 | 0.70010017304 | 0.60991798268 | 0.50010017304 | 0.40991798268 | 0.30010017304 | 0.10991798268 | 0.24277777968 | 0.93924757046 |
| 0.8 | line | 0.90034132453 | 0.91212236876 | 0.70034132453 | 0.71212236876 | 0.50034132453 | 0.51212236876 | 0.30034132453 | 0.21212236876 | 0.5983996666 | 1.27706001774 |
| 0.9 | line | 1.02108242458 | 0.80198200121 | 0.82108242458 | 0.60198200121 | 0.62108242458 | 0.40198200121 | 0.42108242458 | 0.10198200121 | 1.13149869488 | 1.74427601034 |
| 1 | Full Circle | 1.7069368954 | 0.81046528172 | 1.5069368954 | 0.61046528172 | 1.3069368954 | 0.41046528172 | 1.1069368954 | 0.11046528172 | 3.88444472852 | 3.956293536 |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In the table (3.3) I showed some simple objects that I will present them to my neural network (N.N) in order to identify them.

# CHAPTER 4

# SOFTWARE IMPLEMENTATION

## 4.1 Overview

This chapter describes details of software implementation and the source code for each function of the software. It covers the most fundamental parts of image processing algorithms and how can we use it to implement it in the real world.

## 4.2 Entering the Software program

After opening the Software, the user will see a white screen with three drop down menus from the tool bar File**, Tool,** and **Help**.

- **File**

Choosing File will display a drop down menu with the following option:

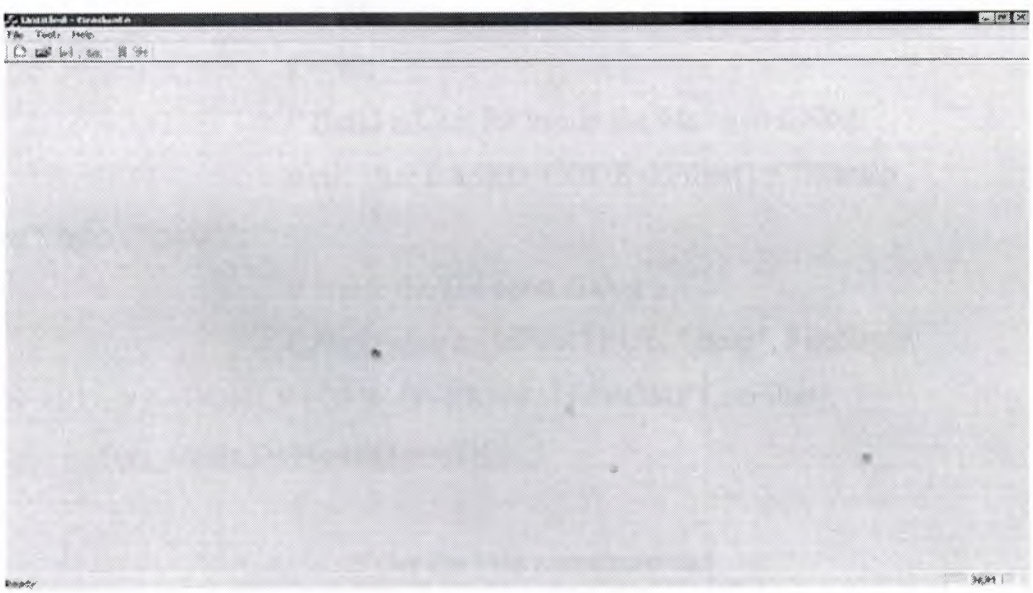

Figure 4.1. Program Interface

- Open

By clocking on Open, will automatically opens a dialog box where we can choose any an existing Bitmap image whether it is an 8 bitmap image or 24 bitmap image. The selected image will have the extension of ".bmp", another file is also created automatically that has a ".txt:" extension. in the previously created text file : file header information ,bitmap header information are called and displayed Note that in order to do so, the image manipulated has to be either 8_bit image or 24_bit image, otherwise a message will alert the user that a wrong file format file has been chosen. The biBitCount member of the BITMAPINFOHEADER structure determines the number of bits that define each pixel and the maximum number of colors in the Bitmap

```
void CGraduateView::OnFileOpen()
    {
                    Invalidate(true);
                    his = false;
                    threesho = false;
                    gray = false;
                    segment = false;
                    x = 90;
                    y = 90;
                    // Build a filter for use in the file open dialog
                    static char BASED_CODE szFilter[] = "Bitmap
Files(*.bmp)|*.bmp||";
                    // create the file open dialog
                    CFileDialog m_ldFile(TRUE, ".bmp", FileName,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,szFilter);
        if (m_ldFile.DoModal() == IDOK )
                    {
                        // Get the File name selected
                        FileName = m_ldFile.GetPathName() ;
                        p = fopen (FileName,"rb");
                        q = fopen(FileOut,"w");
                        FNameS = m_ldFile.GetFileTitle ();
                        Matrix();
                    }}
```

If an 8_bit image is selected, its color table and image data are being extracted and added to the text file, also the image is opened and displayed through the Software.

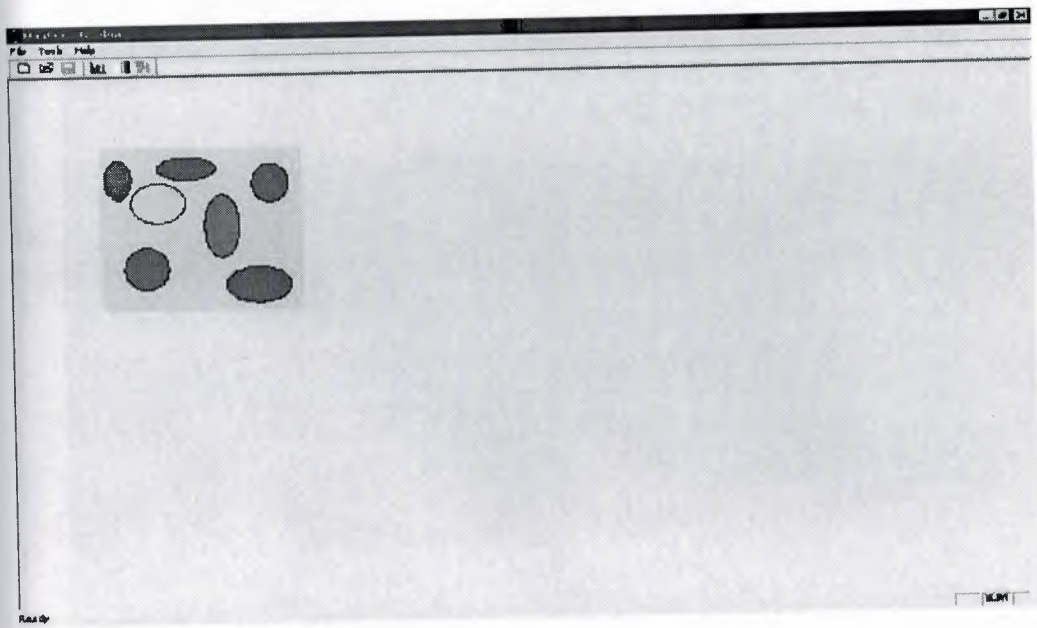

Figure 4.2. 2 8-bit image

```
void CGraduateView::Display(UCHAR image[200][200],UCHAR
Color[256][4])
{
 CClientDC cd(this);
 for (i = 0 ; i < BH.biWidth; i++)
        for (k = 0 ; k < BH.biHeight; k++)
        {
   j = image[k][i];
        cd.SetPixel(i+x, k+y,RGB(Color[j][2],Color[j][1],Color[j][0]));
        }
}
```

If a 24_bit image is selected, the image data are being extracted and added to the text file .also the image is opened and displayed through the Software.

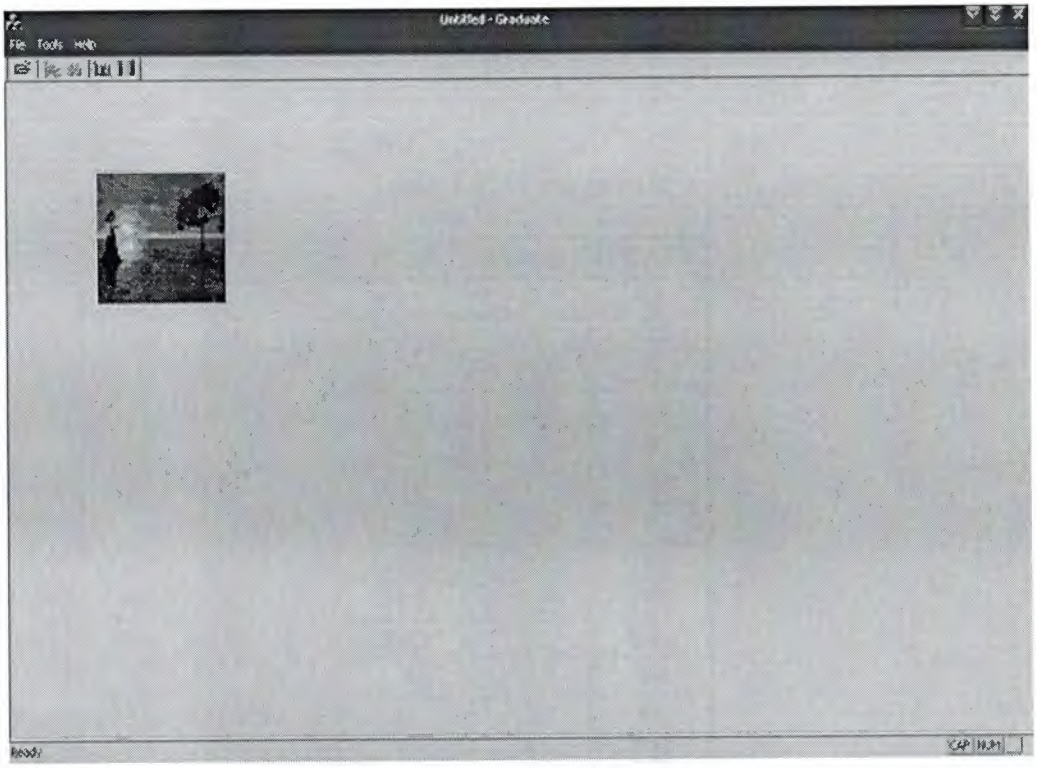

Figure 4.3. 24-bit image

```
void CGraduateView::Displaym()
{
CClientDC cd(this);
UCHAR R,G,B;
int h = 0;
 for (i = 0 ; i < BH.biHeight ; i++)
 {
 j = 0;
 for (k = 0 ; k < BH.biWidth*3 -3  ; k++)
 {
B = mimage[i][k];
        G = mimage[i][k+1];
        R = mimage[i][k+2];
        cd.SetPixel(j+x, h+y,RGB(R,G,B));
        k = k+2;
```

```
        j++;
}
        h++;
}}
```

Table 4.1. Color palette

| Blue | Green | Red |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 128 |
| 0 | 128 | 0 |
| 0 | 128 | 128 |
| 128 | 0 | 0 |
| 128 | 0 | 128 |
| 128 | 128 | 0 |
| 192 | 192 | 192 |
| 192 | 220 | 192 |
| 240 | 202 | 166 |
| 0 | 32 | 64 |
| 0 | 32 | 96 |
| 0 | 32 | 128 |
| 0 | 32 | 160 |
| 0 | 32 | 192 |
| 0 | 32 | 224 |
| 0 | 64 | 0 |
| 0 | 64 | 32 |
| 0 | 64 | 64 |
| 0 | 64 | 96 |
| 0 | 64 | 128 |
| 0 | 64 | 160 |
| 0 | 64 | 192 |
| 0 | 64 | 224 |
| 0 | 96 | 0 |
| 0 | 96 | 32 |
| 0 | 96 | 64 |
| 0 | 96 | 96 |
| 0 | 96 | 128 |
| 0 | 96 | 160 |

| Blue | Green | Red |
|---|---|---|
| 0 | 96 | 194 |
| 0 | 96 | 224 |
| 0 | 128 | 0 |
| 0 | 128 | 32 |
| 0 | 128 | 64 |
| 0 | 128 | 96 |
| 0 | 128 | 128 |
| 0 | 128 | 160 |
| 0 | 128 | 192 |
| 0 | 128 | 224 |
| 0 | 160 | 0 |
| 0 | 160 | 32 |
| 0 | 160 | 64 |
| 0 | 160 | 96 |
| 0 | 160 | 128 |
| 0 | 160 | 160 |
| 0 | 160 | 192 |
| 0 | 160 | 224 |
| 0 | 192 | 0 |
| 0 | 192 | 32 |
| 0 | 192 | 64 |
| 0 | 192 | 96 |
| 0 | 192 | 128 |
| 0 | 192 | 160 |
| 0 | 192 | 192 |
| 0 | 192 | 224 |
| 0 | 224 | 0 |
| 0 | 224 | 32 |
| 0 | 224 | 64 |
| 0 | 224 | 96 |
| 0 | 224 | 128 |
| 0 | 224 | 160 |

| Blue | Green | Red |
|---|---|---|
| 0 | 224 | 193 |
| 0 | 224 | 224 |
| 64 | 0 | 0 |
| 64 | 0 | 32 |
| 64 | 0 | 64 |
| 64 | 0 | 96 |
| 64 | 0 | 128 |
| 64 | 0 | 160 |
| 64 | 0 | 192 |
| 64 | 0 | 224 |
| 64 | 32 | 0 |
| 64 | 32 | 32 |
| 64 | 32 | 64 |
| 64 | 32 | 96 |
| 64 | 32 | 128 |
| 64 | 32 | 160 |
| 64 | 32 | 192 |
| 64 | 32 | 224 |
| 64 | 64 | 0 |
| 64 | 64 | 32 |
| 64 | 64 | 64 |
| 64 | 64 | 96 |
| 64 | 64 | 128 |
| 64 | 64 | 160 |
| 64 | 64 | 192 |
| 64 | 64 | 224 |
| 64 | 96 | 0 |
| 64 | 96 | 32 |
| 64 | 96 | 64 |
| 64 | 96 | 96 |
| 64 | 96 | 128 |
| 64 | 96 | 160 |

| Blue | Green | Red |
|---|---|---|
| 64 | 96 | 193 |
| 64 | 96 | 224 |
| 64 | 128 | 0 |
| 64 | 128 | 32 |
| 64 | 128 | 64 |
| 64 | 128 | 96 |
| 64 | 128 | 128 |
| 64 | 128 | 160 |
| 64 | 128 | 192 |
| 64 | 128 | 224 |
| 64 | 160 | 0 |
| 64 | 160 | 32 |
| 64 | 160 | 64 |
| 64 | 160 | 96 |
| 64 | 160 | 128 |
| 64 | 160 | 160 |
| 64 | 160 | 192 |
| 64 | 160 | 224 |
| 64 | 192 | 0 |
| 64 | 192 | 32 |
| 64 | 192 | 64 |
| 64 | 192 | 96 |
| 64 | 192 | 128 |
| 64 | 192 | 160 |
| 64 | 192 | 192 |
| 64 | 192 | 224 |
| 64 | 224 | 0 |
| 64 | 224 | 32 |
| 64 | 224 | 64 |
| 64 | 224 | 96 |
| 64 | 224 | 128 |

| Blue | Green | Red |
|---|---|---|
| 64 | 224 | 160 |
| 64 | 224 | 192 |
| 64 | 224 | 224 |
| 128 | 0 | 0 |
| 128 | 0 | 32 |
| 128 | 0 | 64 |
| 128 | 0 | 96 |
| 128 | 0 | 128 |
| 128 | 0 | 160 |
| 128 | 0 | 192 |
| 128 | 0 | 224 |
| 128 | 32 | 0 |
| 128 | 32 | 32 |
| 128 | 32 | 64 |
| 128 | 32 | 96 |
| 128 | 32 | 128 |
| 128 | 32 | 160 |
| 128 | 32 | 192 |
| 128 | 32 | 224 |
| 128 | 64 | 0 |
| 128 | 64 | 32 |
| 128 | 64 | 64 |
| 128 | 64 | 96 |
| 128 | 64 | 128 |
| 128 | 64 | 160 |
| 128 | 64 | 192 |
| 128 | 64 | 224 |
| 128 | 96 | 0 |
| 128 | 96 | 32 |
| 128 | 96 | 64 |
| 128 | 96 | 96 |
| 128 | 96 | 128 |

| Blue | Green | Red |
|---|---|---|
| 128 | 96 | 160 |
| 128 | 96 | 192 |
| 128 | 96 | 224 |
| 128 | 128 | 0 |
| 128 | 128 | 32 |
| 128 | 128 | 64 |
| 128 | 128 | 96 |
| 128 | 128 | 128 |
| 128 | 128 | 160 |
| 128 | 128 | 192 |
| 128 | 128 | 224 |
| 128 | 160 | 0 |
| 128 | 160 | 32 |
| 128 | 160 | 64 |
| 128 | 160 | 96 |
| 128 | 160 | 128 |
| 128 | 160 | 160 |
| 128 | 160 | 192 |
| 128 | 160 | 224 |
| 128 | 192 | 0 |
| 128 | 192 | 32 |
| 128 | 192 | 64 |
| 128 | 192 | 96 |
| 128 | 192 | 128 |
| 128 | 192 | 160 |
| 128 | 192 | 192 |
| 128 | 192 | 224 |
| 128 | 224 | 0 |
| 128 | 224 | 32 |
| 128 | 224 | 64 |
| 128 | 224 | 96 |
| 128 | 224 | 128 |

| Blue | Green | Red |
|---|---|---|
| 128 | 224 | 160 |
| 128 | 224 | 192 |
| 128 | 224 | 224 |
| 192 | 0 | 0 |
| 192 | 0 | 32 |
| 192 | 0 | 64 |
| 192 | 0 | 96 |
| 192 | 0 | 128 |
| 192 | 0 | 160 |
| 192 | 0 | 192 |
| 192 | 0 | 224 |
| 192 | 32 | 0 |
| 192 | 32 | 32 |
| 192 | 32 | 64 |
| 192 | 32 | 96 |
| 192 | 32 | 128 |
| 192 | 32 | 160 |
| 192 | 32 | 192 |
| 192 | 32 | 224 |
| 192 | 64 | 0 |
| 192 | 64 | 32 |
| 192 | 64 | 64 |
| 192 | 64 | 96 |
| 192 | 64 | 128 |
| 192 | 64 | 160 |
| 192 | 64 | 192 |
| 192 | 64 | 224 |
| 192 | 96 | 0 |
| 192 | 96 | 32 |
| 192 | 96 | 64 |
| 192 | 96 | 96 |
| 192 | 96 | 128 |

| Blue | Green | Red |
|---|---|---|
| 192 | 96 | 160 |
| 192 | 96 | 192 |
| 192 | 96 | 224 |
| 192 | 128 | 0 |
| 192 | 128 | 32 |
| 192 | 128 | 64 |
| 192 | 128 | 96 |
| 192 | 128 | 128 |
| 192 | 128 | 160 |
| 192 | 128 | 192 |
| 192 | 128 | 224 |
| 192 | 160 | 0 |
| 192 | 160 | 32 |
| 192 | 160 | 64 |
| 192 | 160 | 96 |
| 192 | 160 | 128 |
| 192 | 160 | 160 |
| 192 | 160 | 192 |
| 192 | 160 | 224 |
| 192 | 192 | 0 |
| 192 | 192 | 32 |
| 192 | 192 | 64 |
| 192 | 192 | 96 |
| 192 | 192 | 128 |
| 192 | 192 | 160 |
| 240 | 251 | 255 |
| 164 | 160 | 160 |
| 128 | 128 | 128 |
| 0 | 0 | 255 |
| 0 | 255 | 0 |
| 0 | 255 | 255 |
| 255 | 0 | 0 |

| Blue | Green | Red |
|---|---|---|
| 255 | 0 | 255 |
| 255 | 255 | 0 |
| 255 | 255 | 255 |

## 4.3 Program Tools

- Tool

By choosing Tool, will display a drop menu with the following options:

***Histogram, Gray level, Threshold, Filter, Edges, Segmentation, Learning, and Search***

### 1- Histogram

That will branch into 2 main options: one for the 8_bit Bitmap image, and the other for the 24_bit Bitmap image.

Histogram 8_bits: the following function is established to relatively calculate the repetition of each color in each pixel in the image. Bearing in mind, that we have her 256 different colors in the color palette. To do so, we created a special canvas form the paint dialog to perform the drawing on, Note that each pixel her is 1 byte long.

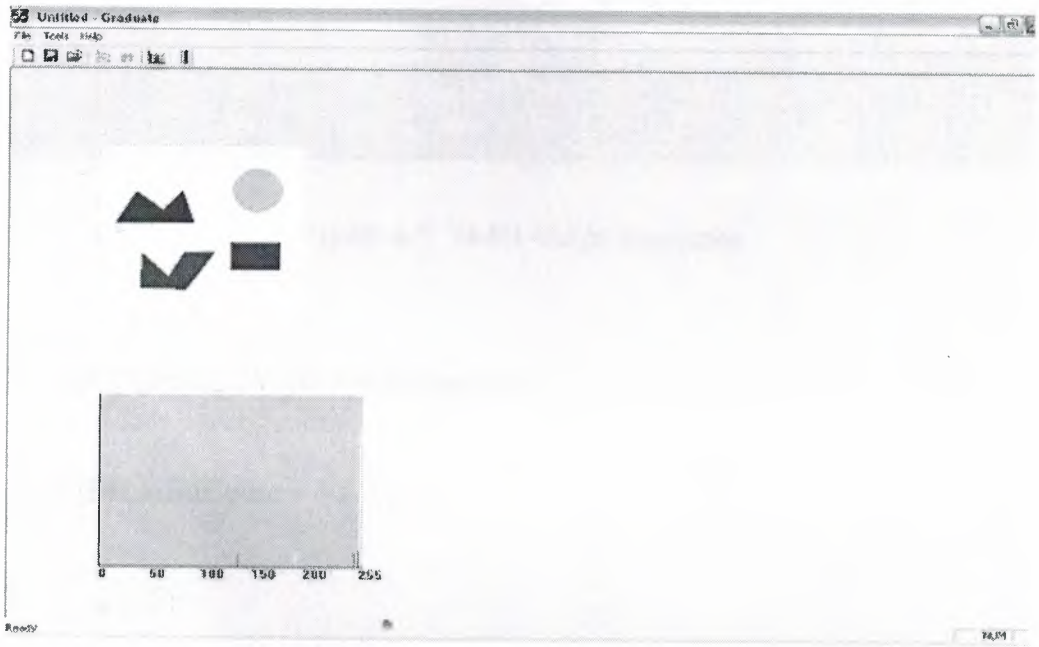


Figure 4.4. 8-bit Image histogram

Histogram 24_bits: the following function is established to relatively calculate the repetition of Red, Green, and Blue in each pixel in the image. Bearing in mind, that we have her 256 different combinations of colors for each of the three colors. To do so, we created a special canvas form the Rgb dialog to perform the drawing on, Note that each pixel her is three byte long.

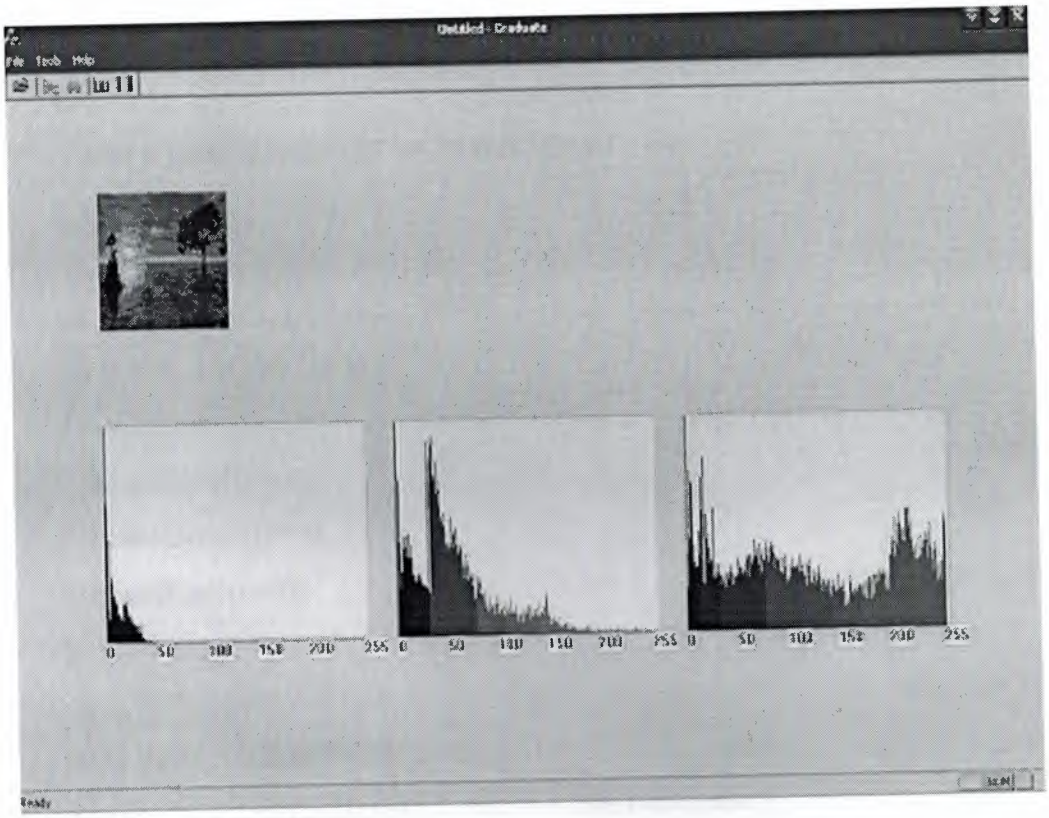

Figure 4.5. 24-bit image histogram

```
void CGraduateView::OnHistogram()
{
if (BH.biBitCount = = 8 || gray )
{
    x = 90;
    y = 180 + BH.biHeight;
    for (i = 0 ; i < 256 ; i++)
 CountColor[i] = 0;
    for (i =0 ; i < BH.biWidth; i++)
    for (k = 0; k < BH.biHeight; k++)
 {
   j = cimage[k][i];
  CountColor[j]++;
 }
for (i = 0 ; i < 256 ; i++)
```

```
fprintf(q, "the Count of The color %d = %ld\n", i, CountColor[i]) ;
Display1(CountColor);
}
   else if ( BH.biBitCount == 24 && !gray )
{

int R,G,B;
for (i = 0 ; i < 256 ; i++)
{
   countBlue[i] = 0;
   countGreen[i] = 0;
   countRed[i] = 0;
}

for (i =0 ; i < BH.biWidth; i++)
for (k = 0; k < BH.biHeight*3 && i1 < BH.biHeight ; )
{
   R = mimage[i][k+2];
   countRed[R]++;
   BW[R][0] = 0;
   BW[R][1] = 0;
   BW[R][2] = R;
   k = k + 3;
}
x =  150 + 259*2;
y = 180 + BH.biHeight;
Display1(countRed);
for (i =0 ; i < BH.biWidth; i++)
for (k = 0; k < BH.biHeight*3 && i1 < BH.biHeight; )
{
B = mimage[i][k];
countBlue[B]++ ;
BW[B][0] = B;
BW[B][1] = 0;
```

```
BW[B][2] = 0;
k = k + 3;
}
  x = 90;
  y = 180 + BH.biHeight;
Display1(countBlue);

for (i =0 ; i < BH.biWidth; i++)
for (k = 0; k < BH.biHeight*3 && i1 < BH.biHeight; )
{
  G = mimage[i][k+1];
  countGreen[G]++;
  BW[G][0] = 0;
  BW[G][1] = G;
  BW[G][2] = 0;
  k = k + 3;
}
  x =  379 ;
  y = 180 + BH.biHeight;
Display1(countGreen);
        }
}
```

## 2- Gray Level

The Gray level option is done on the 24 Bitmap image before filtering, or an error message would show.

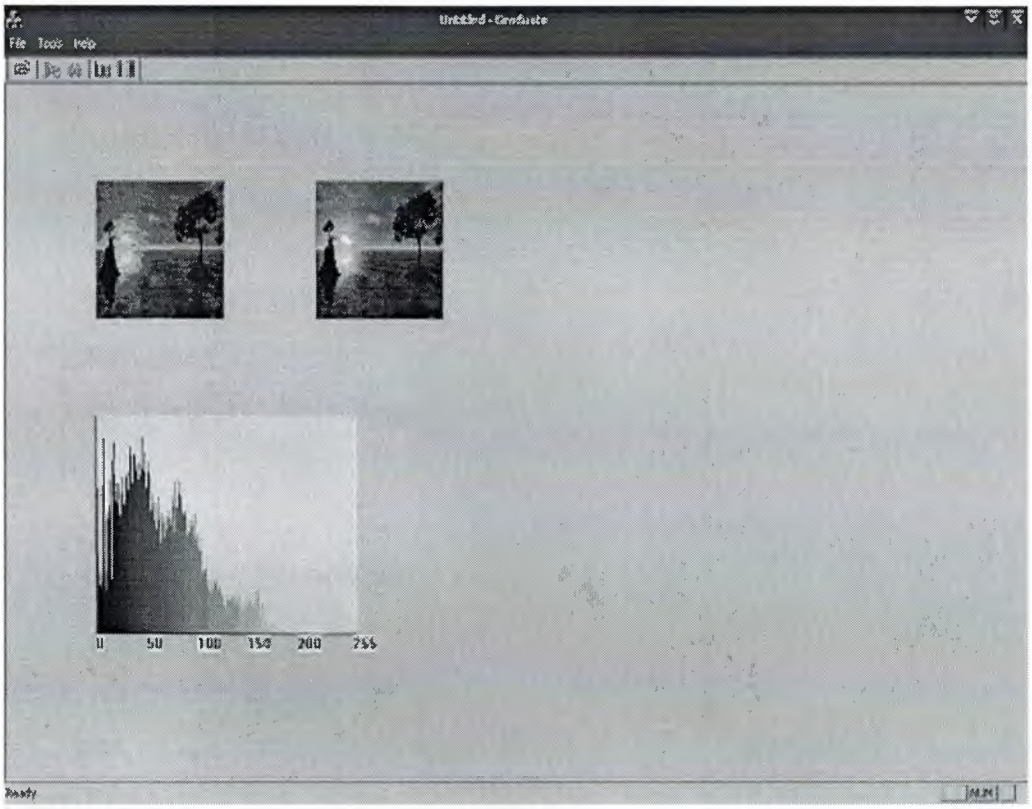


Figure 4.6. Gray level image histogram

```
void CGraduateView::OnGraylevel()
{
   gray = true;
   if(BH.biBitCount = = 8)
{
   y = 90;
   x = 180 + BH.biWidth;
   int value;
   for (i = 0 ; i < BH.biHeight; i++)
   for (j = 0; j < BH.biWidth ; j++)
{

k = image[i][j];
```

```
value = (int)(0.59 * Color[k][1] + 0.3 * Color[k][2] + 0.11 * Color[k][0]);
BW[value][0] = BW[value][1] = BW[value][2] = value;
cimage[i][j] = value;

}
        }
else if(BH.biBitCount == 24)
{
   y = 90;
   x = 180 + BH.biWidth;
   int value;
   for (i = 0 ; i < BH.biHeight; i++)
{
  k = 0;
  for (j = 0; j < BH.biWidth*3 ; )

{
value = (int)(0.59 * mimage[i][j+1] + 0.3 * mimage[i][j+2] + 0.11 *
mimage[i][j]);
BW[value][0] = BW[value][1] = BW[value][2] = value;
cimage[i][k] = value;
j = j + 3;
k++;
                    }
                }
}

Display(cimage,BW);
}
```

**3- Filtering**

Filtering was performed on both the 8_bit and the 24_bitmap images as well the concept of filtering is to make the displayed images noisy and blur, filtering comes in two different degrees:

3x3 Filtering, and the 5x5 Filtering the Filtering (3x3) is called for the two types of images.

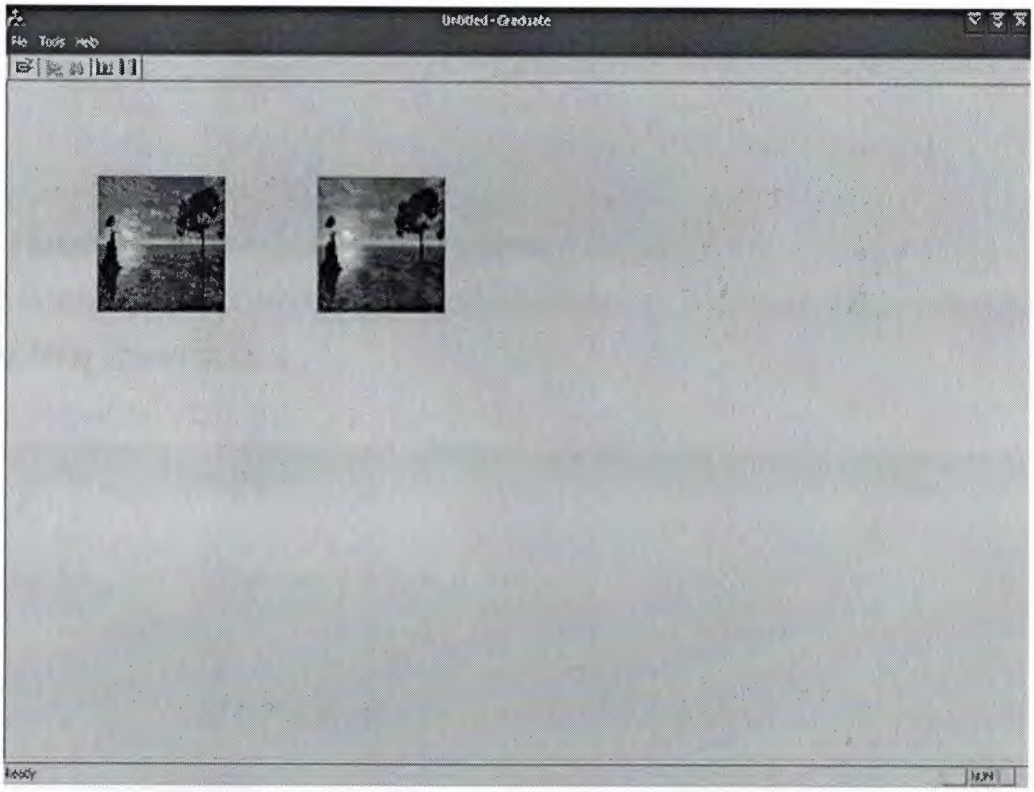


Figure 4.7. (3*3) gray level filtration

```
void CGraduateView::OnFilter3()
{
   y = 90;
   x = 180 + BH.biWidth;
 for (i = 0 ; i < BH.biHeight; i++)
for (j = 0; j < BH.biWidth ; j++)
imagebw[i][j] = cimage[i][j];
for (i = 1 ; i < BH.biHeight-1; i++)
for (j = 1; j < BH.biWidth - 1; j++)
```

```
{
imagebw[i][j] = (cimage[i][j] + cimage[i-1][j-1] + cimage[i][j-1] +
cimage[i+1][j-1] + cimage[i-1][j] + cimage[i+1][j] + cimage[i-1][j+1] +
cimage[i][j+1] + cimage[i+1][j+1])/9;
}
for (i = 0 ; i < BH.biHeight; i++)
for (j = 0; j < BH.biWidth ; j++)
cimage[i][j] = imagebw[i][j] ;
Display(cimage,BW);
}
```

The F**iltering (5x5)** is called for the two types of images.

The following is the **Filtering (5x5)** performed also on 24_bitmaps after performing the **Gray level** operation on it.

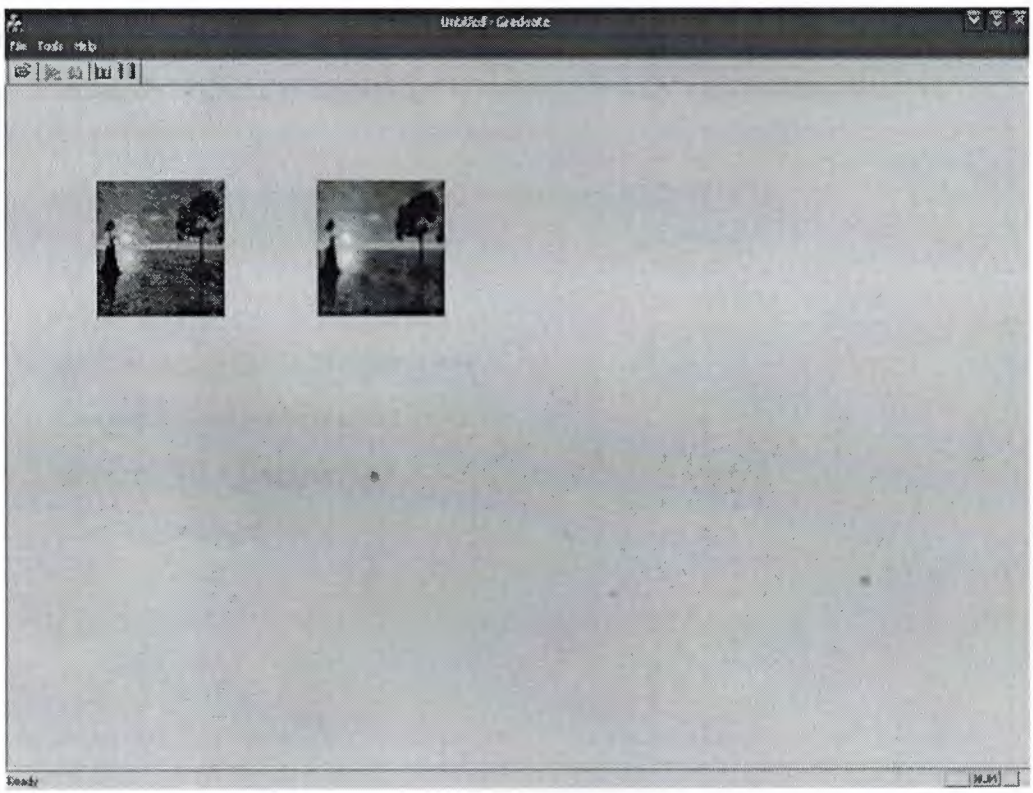

Figure 4.7. (5*5) gray level filtration

```
void CGraduateView::OnFilter5()
        {
           y = 90;
           x = 180 + BH.biWidth;
         for (i = 0 ; i < BH.biHeight; i++)
        for (j = 0; j < BH.biWidth ; j++)
        imagebw[i][j] = cimage[i][j];
        for (i = 2 ; i < BH.biHeight-2; i++)
        for (j = 2; j < BH.biWidth - 2; j++)
        {
        imagebw[i][j] = (cimage[i][j] + cimage[i-1][j-1] + cimage[i][j-1] +
        cimage[i+1][j-1] + cimage[i-1][j] + cimage[i+1][j] + cimage[i-1][j+1] +
        cimage[i][j+1]+cimage[i+1][j+1]+cimage[i-2][j-2]+cimage[i-1][j-2]+
cimage[i][j-2]+
        cimage[i+1][j-2]+cimage[i+2][j-2]+cimage[i-2][j-1]+cimage[i+2][j-1]+
cimage[i-2][j]+
        cimage[i+2][j]+cimage[i-2][j+1]+cimage[i+2][j+1]+cimage[i-2][j+1]+
cimage[i+1][j+2]+
       cimage[i][j+2]+ cimage[i+1][j+2]+ cimage[i+2][j+2])/25;

        }
        for (i = 0 ; i < BH.biHeight; i++)
        for (j = 0; j < BH.biWidth ; j++)
        cimage[i][j] = imagebw[i][j] ;
        Display(cimage,BW);

        }
```

**4- Edges**

Edging was performed on both the 8_bit and the 24_Bitmap images as well.

This function show how edging is called form the menu bar.

```
void CGraduateView::OnEdges()
{
  int l;
  y = 90;
  x = 180 + BH.biWidth;
  for (i = 0 ; i < BH.biHeight; i++)
  for (j = 0; j < BH.biWidth ; j++)
  imagebw[i][j] = 255;
  for (i = 0 ; i < BH.biHeight -1; i++)
  for (j = 0; j < BH.biWidth -1; j++)
  {
  l = abs(cimage[i][j] - cimage[i+1][j+1]) + abs(cimage[i+1][j] - cimage[i][j+1]);
       if (l > 0 )
  imagebw[i][j] = 0;

 else
     imagebw[i][j] = 255;
}

if(BH.biBitCount = = 8)
{
   Display(imagebw,Color);
}
  else if(BH.biBitCount = = 24)
{
  CClientDC cd(this);
  for (i = 0 ; i < BH.biWidth; i++)
  for (k = 0 ; k < BH.biHeight; k++)
{
   if (imagebw[i][k] = = 0)
```

```
  cd.SetPixel(k+x, i+y,RGB(0,0,0));
else if (imagebw[i][k] = = 255)
cd.SetPixel(k+x, i+y,RGB(255,255,255));
                                        }
              }

        }
```

Edging can performed on the 24_bitmap images after performing the **Gray Level** operation.

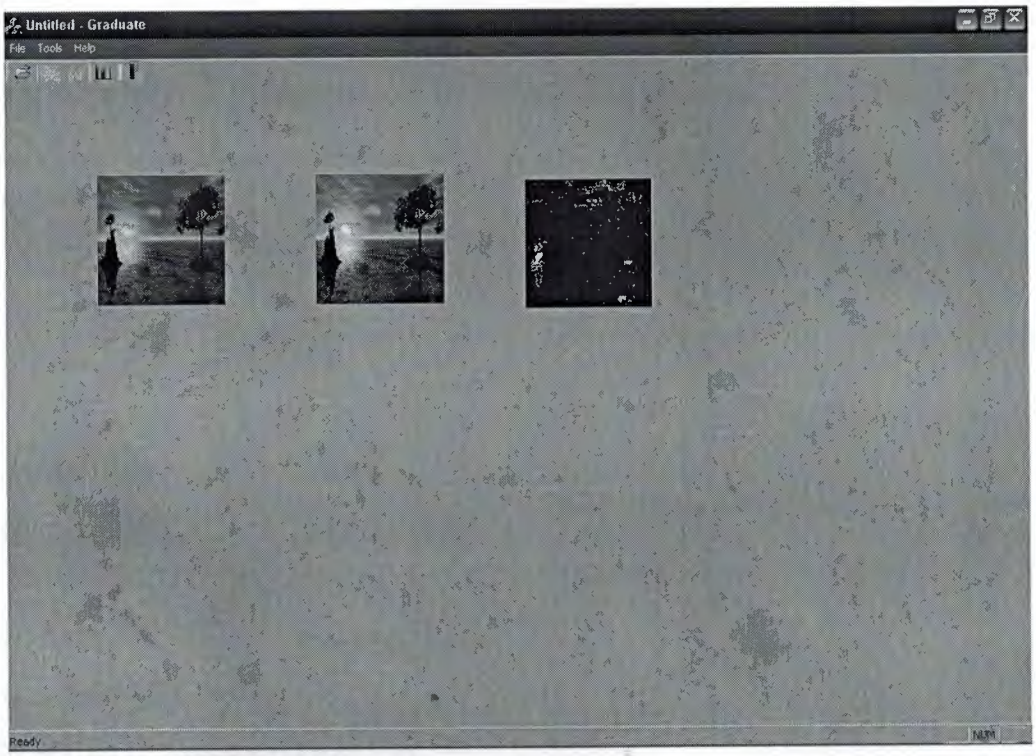


Figure 4.8. 24-bit image Edge detection

```
void CGraduateView::Edge()
{
   int l;

   for (i = 0 ; i < BH.biHeight; i++)
   for (j = 0; j < BH.biWidth ; j++)
   imagebw[i][j] = 255;
   for (i = 0 ; i < BH.biHeight -1; i++)
   for (j = 0; j < BH.biWidth -1; j++)
{
   l = abs(cimage[i][j] - cimage[i+1][j+1]) + abs(cimage[i+1][j] - cimage[i][j+1]);
if (l > 0 )
   imagebw[i][j] = 0;
else
   imagebw[i][j] = 255;

}
   for(k = 0 ; k < y ; k++)
{
   distency[k][2] = 0;
   for(i = point[k][3] ; i <= point[k][4] ; i++)
{
   for(j = point[k][1] ; j <= point[k][2] ; j++)
   if (imagebw[j][i] == 0)
  distency[k][2]= distency[k][2] + 1;
}
 distency[k][2] = distency[k][2]/1000 ;
                  }
 }
```

**5- Threshold**

Threshold was performed on both 8_bit and 24_Bitmaps as well, when we select the threshold option, the two type of threshold will appear Normal threshold and binary threshold.

Threshold can performed on the 24_bitmap images after performing the **Gray Level** operation.

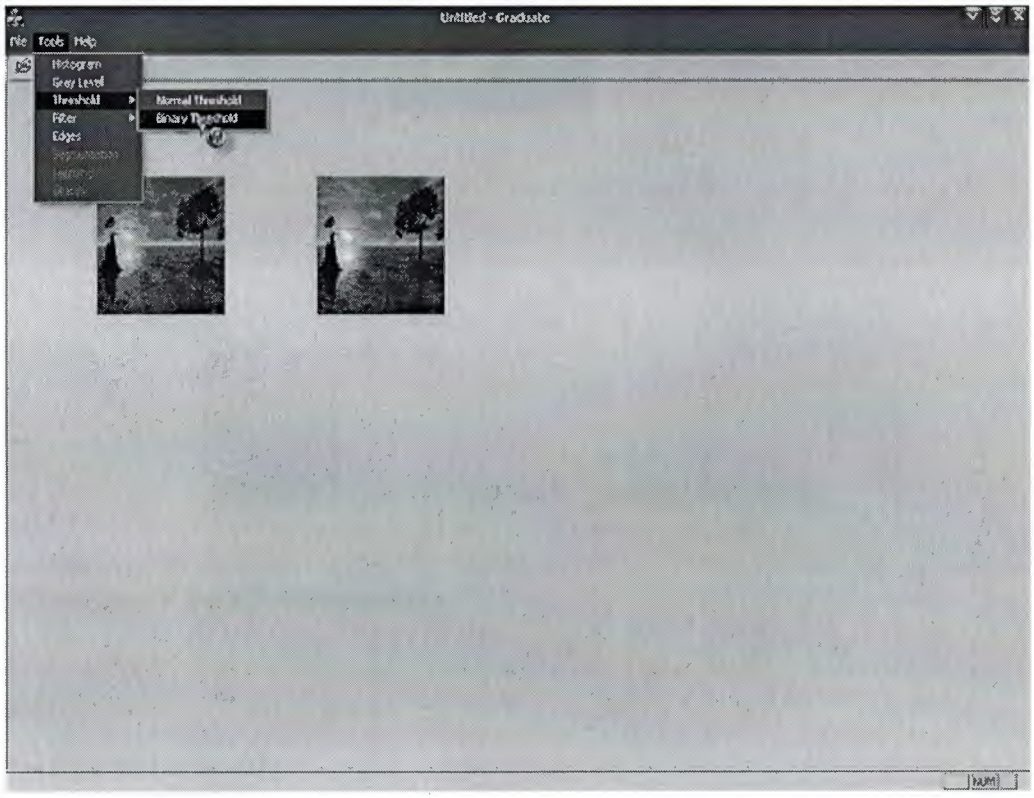


Figure 4.9. 24-bit image normal threshold

By selecting the normal threshold option a normal dialog will appear, and you have to input one value between 0 and 255.

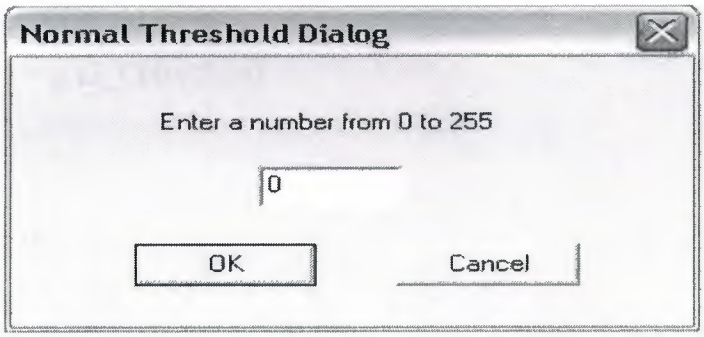


Figure 4.10. Normal Threshold Dialog

After entering the value the image will change taking the value, making in to point of black and white.

Figure 4.11. Apply Normal Threshold to image

```
void CGraduateView::OnNthreshold()
{
   y = 90;
   x = 180 + BH.biWidth;
   d.m_ColorNum = 0;
   if ( d.DoModal() == IDOK)
{
  threesho = true;
  for (i = 0 ; i < BH.biWidth; i++)
  for (j = 0; j < BH.biHeight; j++)
  if (cimage[i][j] <= d.m_ColorNum)
  imagebw[i][j] = 255;
else
  imagebw[i][j] = 0;
  if(BH.biBitCount = = 8)
```

```
{
  Display(imagebw,Color);
}
 else if(BH.biBitCount == 24)
{
  CClientDC cd(this);
  for (i = 0 ; i < BH.biWidth; i++)
  for (k = 0 ; k < BH.biHeight; k++)
{
  if (imagebw[i][k] == 0)
 cd.SetPixel(k+x, i+y,RGB(0,0,0));
 else if (imagebw[i][k] == 255)
 cd.SetPixel(k+x, i+y,RGB(255,255,255));
                                        }
                }

        }

}
```

By selecting the Binary threshold option the Binary dialog will appear, and you have to input two values between 0 and 255 and the first value must be less than the second.

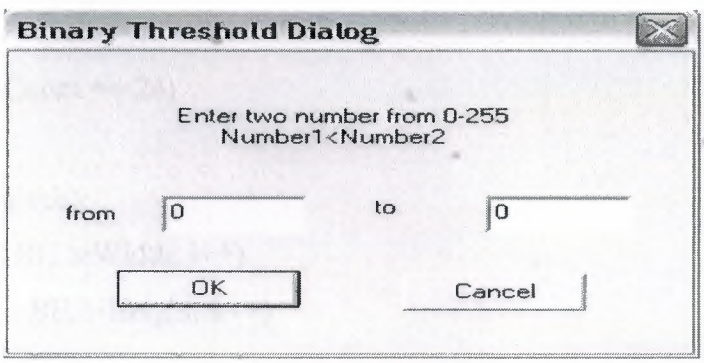


Figure 4.12. Binary Threshold Dialog

```
void CGraduateView::OnBtreshold()
{
  y = 90;
  x = 180 + BH.biWidth;
  d2.m_ColorNum1 = 0;
  d2.m_ColorNum2 = 0;
if ( d2.DoModal() = = IDOK)
{
  threesho = true;
if (d2.m_ColorNum1 > d2.m_ColorNum2)
   MessageBox("Color number 1 > Color number 2","error");
else
{
  for (i = 0 ; i < BH.biWidth; i++)
  for (j = 0; j < BH.biHeight; j++)
if (cimage[i][j] >= d2.m_ColorNum1 && cimage[i][j] <= d2.m_ColorNum2)
  imagebw[i][j] = 255;
else
 imagebw[i][j] = 0;
}
if(BH.biBitCount = = 8)
{
  Display(imagebw,Color);
}
else if(BH.biBitCount == 24)
{
  CClientDC cd(this);
  for (i = 0 ; i < BH.biWidth; i++)
  for (k = 0 ; k < BH.biHeight; k++)
{
if (imagebw[i][k] = = 0)
  cd.SetPixel(k+x, i+y,RGB(0,0,0));
else if (imagebw[i][k] == 255)
  cd.SetPixel(k+x, i+y,RGB(255,255,255)); }}}}
```

## 4.4 Image Segmentation

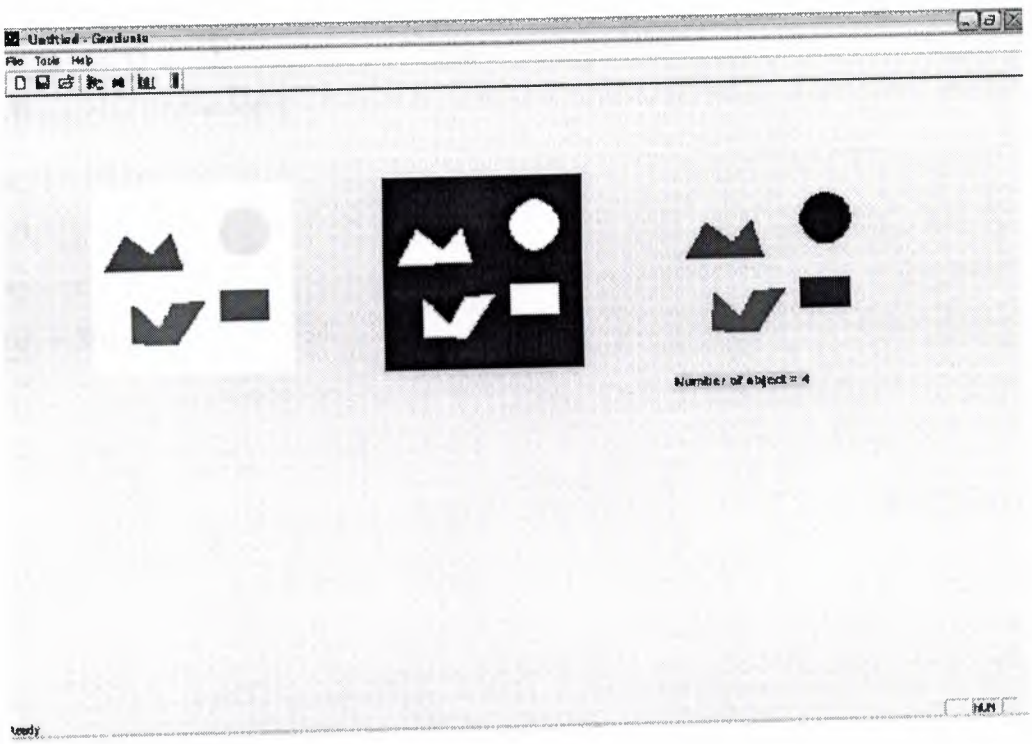


Figure 4.13. Image Segmentation

```
void CGraduateView::OnSegmentation()
{
   segment = true;
    x = 0;
     int   c = 0;
    y = 0;
for (i = 0 ; i < 20000 ; i++)
    seg[i] = 202;
for (i = 0 ; i < BH.biHeight; i++)
{
for (j = 0; j < BH.biWidth ; j++)
{ if (c = = 0)
{ if (imagebw[i][j] = = 255 )
{ seg[y] = j;
  c = 1;
```

```
  y++;
}
 }
else if (c = = 1)
{ if (imagebw[i][j] = = 0 )
{ seg[y] = j-1;
 c = 0;
 y++;
seg[y] = 201;
y++;
}
  }
       }
y++;
}

lon = y;
for (i =0; i < lon ; i++)
{ fprintf(q, "the seg[%d]\t = %d",i,seg[i] );
  fprintf(q,"\n");

}
counting();
PreCount();
Displayseg();

}
```

```
void CGraduateView::counting()
{
  x = 0;
  count = 0;
for (i = 0 ; seg[i] = = 202 && i < lon ;i++);
for (j = i ; seg[j] != 202 && j < lon ; j++)
    if (seg[j] = = 201)
 {
  count++;
   seg[j] = count;
  }
    x = i;
  for ( k = j+1 ; k <= lon - 3   ; )
  {
      if (seg[k] != 202 )
  {
       if (seg[x] >= seg[k]  &&  seg[x+1] <= seg[k+1])
          overflag = true;
       else if (seg[x] <= seg[k]  &&  seg[x+1] >= seg[k+1])
          overflag = true;
       else if (seg[x] >= seg[k] &&  seg[x] <= seg[k+1])
          overflag = true;
       else if (seg[x+1] >= seg[k] &&  seg[x+1] <= seg[k+1])
          overflag = true;
       else i f ( seg[x+1] = =  seg[x] && seg[k] = = seg[k+1] &&(seg[x]+1 = = seg[k] ||
   seg[k]  +1 = = seg[x]))
          overflag = true;
       else if (seg[x]+1 = = seg[k] || seg[k] +1 = = seg[x])
          overflag = true;
       else if (seg[x+1]+1 = = seg[k] || seg[k] +1 = = seg[x+1])
                overflag = true;
       else if (seg[x]+1 = = seg[k+1] || seg[k+1] +1 = = seg[x])
                overflag = true;
```

```
    else if (seg[x+1]+1 = = seg[k+1] || seg[k+1] +1 = = seg[x+1])
        overflag = true;
else
        overflag = false;
if (overflag)
{
  seg[k+2] = seg[x+2];
if (seg[k+3] = = 202 && seg[x+3] = = 202)
{ k = k + 4; x = x+4; }
  else if (seg[k+3] != 202 && seg[x+3] != 202)
{ k = k + 3; x = x+3; }
  else if (seg[k+3] != 202 && seg[x+3] = = 202)
{ k = k + 3;}
  else if (seg[k+3] = = 202 && seg[x+3] !=202)
{k = k + 4;
for (i = x ; seg[i] != 202; i++);
x = i+1;
        }
          }
   else
{
    if ( seg [x+3] == 202 )
{
     count++;
     seg[k+2] = count;
     if (seg[k+3] = = 202)
{ k = k + 4; x = x+4;}
      else
     k= k + 3;
}
      else
    x = x+3;
}
```

```
}
  else
{
  for (i = k ; seg[i] = = 202 && i < lon ;i++);

  for (j = i ; seg[j] != 202 && j < lon; j++)
       if (seg[j] = = 201)
{
   count++;
   seg[j] = count;
}
  x = i;
  k = j +1;
}

}
  for (i =0; i < lon ; i++)
   {
     fprintf(q, "the seg[%d]\t = %d",i,seg[i] );
     fprintf(q,"\n");

}
```

**Object Count**

```
void CGraduateView::ObjectCount()
{
for(k = 0 ; k < y ; k++)
   distency[k][3] = 1;
for(k = 0 ; k < y ; k++)
{  i = point[k][0];
for (j = 0; j < lonpre; j++)
{  if (i == precount[j][0])
    distency[k][3] = distency[k][3] +1;} }
```

**Object Recount**

```
void CGraduateView::PreCount()
{
    y = 0;
    x = 0;
for (i = 0 ; seg[i] = = 202 && i < lon ;i++);
for (j = i ; seg[j] != 202 && j < lon ; j++);
    x = i ;
for ( k = j+1 ; k <= lon - 3 && y < 20000 && x < lon   ; )
{
    if (seg[k] != 202 )
{
    if (seg[x] >= seg[k]  &&  seg[x+1] <= seg[k+1])
     overflag = true;
else if (seg[x] <= seg[k]  &&  seg[x+1] >= seg[k+1])
    overflag = true;
else if (seg[x] >= seg[k] &&  seg[x] <= seg[k+1])
    overflag = true;
else if (seg[x+1] >= seg[k] &&  seg[x+1] <= seg[k+1])
   overflag = true;
else if (seg[x+1] = = seg[x] && seg[k] = = seg[k+1] &&(seg[x]+1 = = seg[k] || seg[k]
+1 = = seg[x]))
   overflag = true;
else if (seg[x]+1 = = seg[k] || seg[k] +1 = = seg[x])
   overflag = true;
else if (seg[x+1]+1 = = seg[k] || seg[k] +1 = = seg[x+1])
   overflag = true;
else if (seg[x]+1 = = seg[k+1] || seg[k+1] +1 = = seg[x])
   overflag = true;
else if (seg[x+1]+1 == seg[k+1] || seg[k+1] +1 == seg[x+1])
   overflag = true;
else
   overflag = false;
```

```
    if (overflag)
{
    if (seg[k+2] = = seg[x+2])
{
    if ( seg [x+3] = = 202 )
{
    if (seg[k+3] == 202)
{
    k = k +4;
    x = x + 4;
     i = x;
 }
    else
{
    k = k +3;
    x = i;
     }
}
else  x = x+3;
}
else
{
   precount[y][1] = seg[k+2];
   precount[y][0] = seg[x+2];
   y++;
   count = count -1;
    if ( seg [x+3] == 202 )
{
    if (seg[k+3] == 202)
{
    k = k +4;
    x = x + 4;
    i = x;
```

```
}
  else
{
   k = k +3;
   x = i;
     }
}
   else  x = x+3;
     }
}
   else
{
   if ( seg [x+3] == 202 )
{
   if (seg[k+3] == 202)
{
   k = k +4;
   x = x + 4;
   i = x;
}
    else
{
   k = k +3;
   x = i;
   }
}
    else x = x+3;
    }
}
    else {
for (i = k ; seg[i] = = 202 && i < lon ;i++);
for (j = i ; seg[j] != 202 && j < lon; j++);
x = i;
 k = j +1;
```

```
    }
}
    for (i = 0; i < y ; i++)
    for (j = 0; j < y; j++)
if (precount[i][1] = = precount[j][0])
   precount[j][0] = precount[i][0];
    for (i =0; i < y ; i++)
 {
     fprintf(q,"%d %d", precount[i][0] ,precount[i][1] );
     fprintf(q,"\n");
    }
      lonpre = y;
      for(x = 0; x < y; x++)
     {
   for (i = 0 ; seg[i] == 202 && i < lon ;i++);
     j = i;
   for(k = j ; k < lon ; )
      {
      if (seg[k] != 202 )
     {
       if (seg[k+2] == precount[x][1])
       seg[k+2] = precount[x][0];
       if ( seg [k+3] == 202 )
       k = k +4;
      else
        k = k +3;
      }
      else
       {
      for (i = k ; seg[i] == 202 && i < lon ;i++);
         j = i ;
         k = j ;
              }
       }
```

```
}
for (i =0; i < lon ; i++)
 {
    fprintf(q, "the seg[%d]\t = %d",i,seg[i] );
    fprintf(q,"\n");
                }
}
```

## 4.5 Learning

By clicking on learn option, a dialog will appear to input the name of the object and the number of the objects.

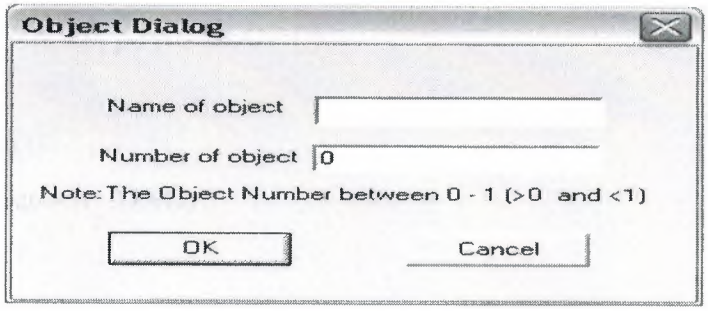


Figure 4.14. Learning object dialog

```
void CGraduateView::OnLearning()
{
    Distency();
    ObjectCount();
     Edge();
for(i = 0 ; i< count ; i++)
 {
    d3.m_output = 0;
    d3.m_name = "";
    wait1[0][0] = 0.9;
    wait1[0][1] = 0.8;
    wait1[1][0] = 0.7;
    wait1[1][1] = 0.6;
    wait1[2][0] = 0.5;
    wait1[2][1] = 0.4;
    wait1[3][0] = 0.3;
    wait1[3][1] = 0.1;
    wait2[0] = 0.2;
    wait2[1] = 0.9;
  if (d3.DoModal() = = IDOK)
{
   t = d3.m_output;
```

```
    name = d3.m_name ;

m_pSet->MoveFirst() ;
    found = false;
    while (!found && !m_pSet->IsEOF ())
{
    m_pSet->Edit();
    if (t = = m_pSet->m_IDNO)
    found = true;
else
    m_pSet->MoveNext() ;
                        }
        ////////////////////////////////////////////////////////////
    if(!found)
 {
    m_pSet->MoveLast();
    e =  0.09;
    my = 0;
    while (e >= 0.00000001 || e <= -0.00000001  && my < 3000000)
 {
    my++;
 for(k = 0; k < 2; k++)
    hidden[k] = 0;
 for(j = 0; j < 2; j++)
 {
 for(k = 0 ; k < 4; k++)
      hidden[j] = hidden[j] + distency[i][k] * wait1[k][j];
      hidden[j] =  1/(1+exp(hidden[j]*-1));
   }
      net = 0;
 for (k =0 ; k<2 ; k++)
      net = net + wait2[k] * hidden[k];

    o = 1 / (1 + exp (net * -1));
```

```
    e = o - t;
    if (e >= 0.00000001 || e <= -0.00000001)
{
    s = (t- o)* (1-o) * o ;
for(k = 0;k<2;k++)
    w[k] =  0.05* s * hidden[k];
for(k=0; k<2;k++)
{
     s1[k]=hidden[k]*(1 - hidden[k]) * wait2[k] * s;
for(j= 0 ;j < 4 ; j++)
     w1[j][k]= 0.05* s1[k] * distency[i][k];
 }
for(k=0;k<2;k++)
         wait2[k]=wait2[k] + w[k];
for(k=0;k<2;k++)
for (j=0;j<4;j++)
     wait1[j][k]=wait1[j][k] + w1[j][k];
         }

}
for (k =0; k < 4 ; k++)
{
for(j = 0; j<2; j++)
    fprintf(q,"\t%f", wait1[k][j] );
    fprintf(q,"\n");
}
for(j = 0; j<2; j++)
{
    fprintf(q,"%f", wait2[j] );
    fprintf(q,"\n");
}
    fprintf(q,"%f \n", o );

    CRecordset* pSet = OnGetRecordset();
```

```
if(pSet->CanUpdate() && !pSet->IsDeleted ())

pSet->Edit ();
if(!UpdateData())
return;
pSet->Update ();

m_pSet->AddNew();
m_pSet->m_IDNO = t;
m_pSet->m_name = name;
m_pSet->m_w11 = wait1[0][0];
m_pSet->m_w12 = wait1[0][1];
m_pSet->m_w21 = wait1[1][0];
m_pSet->m_w22 = wait1[1][1];
m_pSet->m_w31 = wait1[2][0];
m_pSet->m_w32 = wait1[2][1];
m_pSet->m_w41 = wait1[3][0];
m_pSet->m_w42 = wait1[3][1];
m_pSet->m_wh1 = wait2[0];
m_pSet->m_wh2 = wait2[1];

m_pSet->Update ();
m_pSet->Requery();
m_pSet->MoveLast ();
}
else
MessageBox("The Record has been existed","error");
            }
      }

}


void CGraduateView::Distency()
```

```
{
   BOOL flag = true;
    y = 0;
    x = 0;
for (i = 0 ; seg[i] == 202 && i < lon ;i++);
    x = i ;
for (j = i ; j < lon && y < 20000; )
{
   if (seg[j] != 202)
{
for (k = 0 ; k < y && k < 20000; k++)
{
   if ( point[k][0] == seg[j+2])
{
    flag = false;
    break;
}
   else flag = true;
  }
   if (flag)
{
     point[y][0] = seg[j+2];
     point[y][1] = x;
     point[y][2] = x;
     point[y][3] = seg[j];
     point[y][4] = seg[j+1];
    sum[y] = seg[j+1] - seg[j] +1;
      y++;
   }
   else
{
   point[k][2] = x;
   if (seg[j] < point[k][3])
  point[k][3] = seg[j];
```

```
if (seg[j+1] > point[k][4])
point[k][4] = seg[j+1];
sum[k] = sum[k] +(seg[j+1] - seg[j] +1);

 if (seg[j+3] == 202)

 j = j+4;
 x++;

else
   j = j + 3;
 }
else
{
for (i = j ; seg[i] == 202 && i < lon ;i++)
   x++;
   j = i;
 }
for (i = 0 ; i < y; i++)
{
    int value ;
    value = (point[i][2] - point[i][1] +1) * (point[i][4] - point[i][3]+1);
    fprintf(q,"%d \n", value);
    distency[i][0] = (float) sum[i]/ value;
    if ((point[i][4] - point[i][3]+1) <= (point[i][2] - point[i][1] +1))
    distency[i][1] = (float) (point[i][4] - point[i][3]+1) /(point[i][2] - point[i][1] +1) ;
else
     distency[i][1] = (float) (point[i][2] - point[i][1]+1) /(point[i][4] - point[i][3] +1) ;
              }

}
```

## 4.6 Search

By clicking on Search option, the program will try to find the nearest image close to the object and how percent it is close.

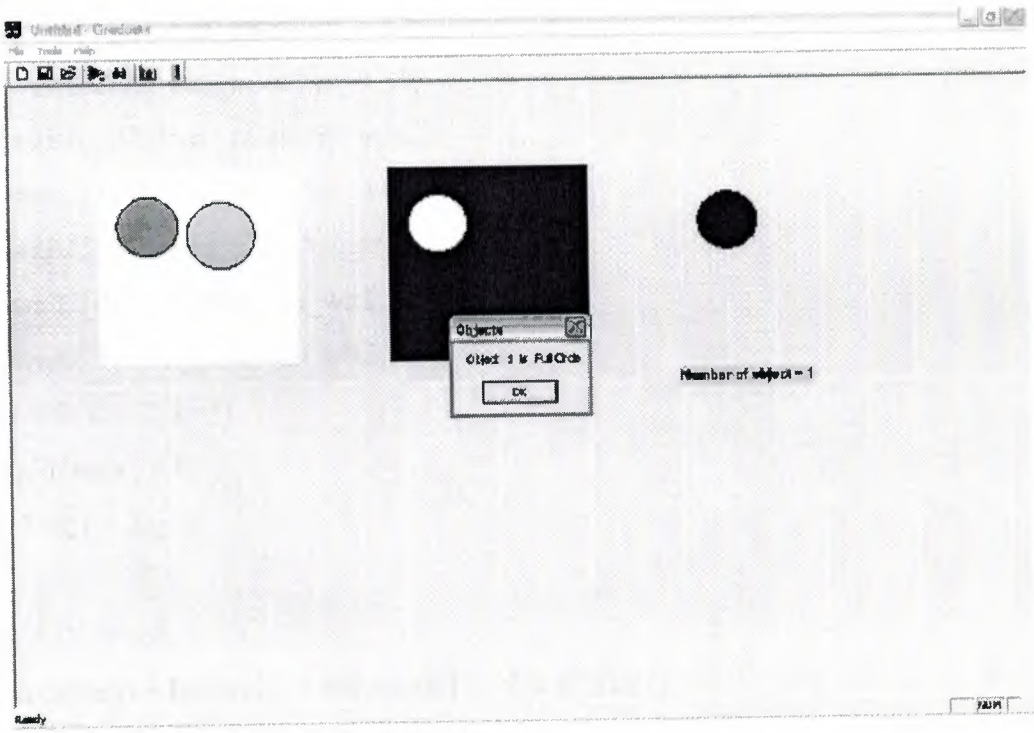


Figure 4.14. presentation graph for searching process

```
void CGraduateView::OnSearch()
{
    Distency();
    ObjectCount();
     Edge();
      msg = "";
for(i = 0; i < count; i++)
{
      m_pSet->MoveFirst() ;
      found = false;
      while (!found && !m_pSet->IsEOF ())
{
      m_pSet->Edit();
      t = m_pSet->m_IDNO;
```

```
    name = m_pSet->m_name;
    wait1[0][0] = m_pSet->m_w11 ;
    wait1[0][1] = m_pSet->m_w12;
    wait1[1][0] = m_pSet->m_w21;
    wait1[1][1] = m_pSet->m_w22;
    wait1[2][0] = m_pSet->m_w31;
    wait1[2][1] = m_pSet->m_w32;
    wait1[3][0] = m_pSet->m_w41;
    wait1[3][1] = m_pSet->m_w42;
    wait2[0] = m_pSet->m_wh1;
    wait2[1] = m_pSet->m_wh2;
for(k = 0; k < 2; k++)
    hidden[k] = 0;
for(j = 0; j < 2; j++)
{
for(k = 0 ; k < 4; k++)
    hidden[j] = hidden[j] + distency[i][k] * wait1[k][j];
    hidden[j] =  1/(1+exp(hidden[j]*-1))
}
    net = 0;
for (k =0 ; k<2 ; k++)
    net = net + wait2[k] * hidden[k];
    o =  1 / (1 + exp(net * -1));
    e = 0.5 * pow(o-t,2);
    if (e <= 0.0000001)
  {
    pointer.Format("%d", (i+1));
    name1[i] = "Object   "+ pointer +"  is   "+ name;
    found = true;
        }
  else
  {
    pointer.Format("%d", (i+1));
    name1[i] = "Object   "+ pointer +"  is not found";
```

```
Set->MoveNext() ;
        }
}

 0; i < count; i++)
sg = msg + name1[i] +"\n";
essageBox(msg,"Objects");
_pSet->MoveLast ();
```

# CONCLUSION

Image processing techniques are developed with the goal of manipulate some es within the image for some application such as weather prediction photographs mphasizing some features within an image which are not clear to the human or nition system.

Neural networks are developed with the goal of modeling information processing earning in the brain applied to a number of practical applications in various fields, ding computational molecular biology.

Artificial neural networks offer an ability to perform tasks outside the scope of tional processors. They can recognize patterns within vast data sets and then ralize those patterns into recommended courses of action. Neural networks learn, are not programmed.

Yet, even though they are not traditionally programmed, the designing of neural vorks does require a skill. It requires an "art." This art involves the understanding of various network topologies, current hardware, current software tools, the application e solved, and a strategy to acquire the necessary data to train the network. This art her involves the selection of learning rules, transfer functions, summation functions, how to connect the neurons within the network.

The art of neural networking requires a lot of hard work as data is fed into the tem, performances are monitored, processes tweaked, connections added, rules dified, and on and on until the network achieves the desired results.

The project fulfills the request of combining image processing with neural network recognize the objects within the image. In this system user can teach the computer to entify simple objects within the image, manipulate the image such as changing it to ay level, make a filtration, object segmentation, edge detection.

# REFERENCES


[1] Kenneth R. Castleman, *Digital Image Processing*, Prentice-Hall, 1996.

[2] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Second Ed, Prentice-Hall, 2001.

[3] Wayne Niblack, *An Introduction to Digital Image Processing,Prentice*-Hall International, 1985.

[4] Anil K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, 1989.

[5] Wegman, E.J., DePriest,D.J. (eds) (1986) *STATISTICAL IMAGE PROCESSING graphics,* NY,Marcel Dekker Inc.

[6] Kashman Adnan,*Image Processing lectures note* ,Near East University ,2003.

[7] Anderson, James A. and Rosenfeld, E. (eds.) (1988) *Neurocomputing.*

[8] Bishop, Christopher M. (1995*) Neural Networks for Pattern Recognition.*

[9] Chauvin, Yves (1989) *'A Back-Propagation Algorithm with Optimal Use of Hidden Units.*

[10] Chauvin, Yves (1990) 'Dynamic Behavior of Constrained Back-Propagation Networks.

[11] Churchland, P. and Sejnowski, T, "*The Computational Brain*", MIT Press Cambridge, 1992.

[12] Bishop, C. M., "*Neural Networks for Pattern Recognition*". Oxford University Press, 1995.

[13] Hopfield, J. J., "Neural networks and physical systems with emergent computational abilities", *Proceedings of the National Academy of Sciences*, 79:2554, 1982

[14] Kashman Adnan,neural networks (N.N) lectures note ,near east university ,2003.

[15] Herbert Schildt. Greg Guntle, Borland C++ Builder, McGraw-Hill, 2003.

[16] Microsoft Access tutorials help.