Chapter 1
INTRODUCTION

1.1 Background of the Study

It is said that picture is worth a thousand words. This clarifies the difference between the human's capability to perceive linguistic and visual information. Likewise, visual representation tends to be more efficient than the spoken or written word when it comes to perceiving information. Language, in an evolutionary perspective, develops instantaneously and is expressed in only a single species. Unlike in visual representation, in one form or another, has been existing since hundreds of millions of years ago and is shared by a numerous organisms. But why should there be such a difference between the two representations in such an advances being as ourselves?

Nowadays, photos are being handled by the latest gadgets that has been developed over the years such as phones, laptops, desktops and even in company computers. These hi-tech gadgets, especially laptops, gain the capacity to run any programs like softwares that process digital images. And these softwares rapidly advances in its smartest and simplest way. In relation to these development, today's digital imaging software have reach the level of photo editing and even its 3-D representation. Its user interfaces intended to be friendly to advance users as well as first-time users.

The development of internet raise the growth of digital imaging. Online editing, sharing and viewing of photos are now allowed on the internet. Thus, online websites like Instagram, Facebook, Pinterest are emerging one after the other to give humans the capability to share their photos and to viewed other's phots also whether they are professionals or not. This have changed the way people's understanding in the field of photography.

Photography has gone from being a luxury medium of communication and sharing to more of a fleeting moment in time. Subjects have also changed. Pictures used to be primarily taken of people and family. Now, we take them of anything. We can document our day and share it with everyone with the touch of our fingers. *(Wikipedia, Digital Imaging)*

As smart devices continue to transpire, digitals images, when it comes to the transmission and sharing, have increased excessively. The more these devices includes cameras and provides the users to share the captured images directly to the Internet, the more these storage devices are grasping the necessity of effectual storing of huge amount of image data. These contributes to inadequate bandwidth of network and storage of memory device. Therefore, the theory of compressing the data becomes more and more significant for reducing the data redundancy save more hardware space and transmission bandwidth.

Methods such as image compressions has been an active area of research since the beginning of digital image processing. The idea of this is that it converts the original image with few bits. Since images can be represented in a two-dimensional signal, a lot of digital compression techniques are being studied in order to meet the needs of image compression. Without significantly reducing its quality, it is useful because of the fact that it reduces the bandwidth consumed during the transferring of images from one medium to another and the cost it takes in a hard disk's space. It is beneficial in many applications especially in progressive transmission, image browsing and multimedia application. Its general objective is to obtain the image quality and yet occupy less space

This study focuses on compressing images using neural network. This chapter discuss about the main objectives of the study and the limitations that it can handle. And also the problem that is being experienced today that serve as the basis for the researchers to conduct this study. Chapter 2 briefly summarizes some of the works that have been done by other researchers which is correlated in this study. Chapter 3 provides the ideas and concepts for image compression which helps the researchers to gain understanding on their chosen topic. Chapter 4 is all about the approaches made to implement the concepts gathered in the study. It introduces some diagrams which can help to have a deeper understanding about the basis for the procedures used. Chapter 5 summarizes the results and discuss the core ideas developed and analyzed by the researchers. Chapter 6 stated some of the constraints experienced by them during the process of developing the system and some standards to be considered in conducting the study

1.2 Statement of the Problem

Images has a larger file size than document files or any other simple text files , therefore its transmission over a network demands an extra bandwidth . Studies and researches have been conducted in order to find ways on how to easily transfer images to another medium in a short amount of time. Compressing the image file size is one of the possible solution for this problem.

In this paper a neural network based image compression method is presented. Neural networks has the capabilities of providing a solution for data compression by its ability to represent the internal data of the images, which is considered important in creating the program

1.3 Objective of the Study

1.3.1 General Objective

To implement a neural network based image compression using MATLAB

1.3.2 Specific Objective

1.3.2.1 To be knowledgeable on how to use MATLAB's GUIDE in creating a graphical user interface window for easy access of compressing and decompressing images.

1.3.2.2 To provide a high compression ratio with minimal distortion.

1.3.2.3 To show the difference between the original images, compressed image, and decompressed image.

1.3.2.4 To compute for the peak signal to noise ratio, mean square error and the compression ratio after compressing the image file.

1.3.2.5 To utilize a neural network based on back propagation algorithm.

1.3.2.6 To discuss the steps in compressing and decompressing the image

1.3.2.7 To know the importance of data compression and to be able use its benefits

1.4 Significance of the Study

The development of this study will benefit those people who attach photos when sending their emails, who likes a collection of their captured images on their favorite storage devices because compressing the images they want to send or to store will save the hard disk's space and the bandwidth it takes to download or to upload the images.

This study will also provide an advantage to web designers who want to create faster loading web pages which can make their website more accessible to others. This image compression will also save them a lot of unnecessary bandwidth by providing high-quality image with fraction of file size. Other important application that will benefit in this study are televideo conferencing, remote sensing (satellite imagery), document and medical imaging, facsimile transmission and control of remotely piloted vehicles in military space and hazardous waste management.

1.5 Scope and Limitation of the Study

1.5.1 Scope of the Study

To be able to arrive at the desired output for compressing the images, this study chose to use MATLAB for the development of the program. Neural network based on back propagation method will be the technique used in creating this process of encoding and decoding the images. This study used standard grayscale images like Lena, Cameraman and pepper having a dimensions of 512x512, 256x256 and 128x128 of any file type for the actual test of the program. It is a must that the bit depth has a value of 8

1.5.2 Limitation of the study

The researcher's implementation of the study only worked with grayscale images. This study cannot compressed colored standard test images and also images with dimensions aside from 256 x 256, 128 x 128 and 512x512. And bit depth having a value aside from 8.

Chapter 2
REVIEW OF RELATED LITERATURE

2.1 Artificial neural network

Artificial neural networks are simplified models of the biological neuron system. A neural network is a highly interconnected network with a large number of processing elements called neurons in an architecture inspired by the brain. Artificial neural networks are massively parallel adaptive networks which are intended to abstract and model some of the functionality of the human nervous system in an attempt to partially capture some of its computational strengths. A neural network can be viewed as comprising eight components which are neurons, activation state vector, signal function, pattern of connectivity, activity aggregation rule, activation rule, learning rule and environment. They are considered as the possible solutions to problems and for the applications where high computation rates are required.*(Anjana and Sreeja 2012)*

Neural networks have, since the very beginning of practical application, proven to be a powerful tool for signal analysis, features extraction, data classification, pattern recognition, etc. Owing to the capabilities of learning and generalization from observation data of the past researchers, the networks have been widely accepted by engineers and researchers as a tool for processing of experimental data. This is mainly because neural networks reduce enormously the computational efforts needed for problem solving and, owing to their massive parallelity, considerably accelerate the computational process. This was reason enough for intelligent network technology to leave soon the research laboratories and to migrate to industry, business, financial engineering, etc. For instance, the neural-network based approaches developed and the methodologies used have efficiently solved the fundamental problems of time series analysis, forecasting, and prediction using collected observation data and the problems of on-line modelling and control of dynamic systems using sensor data.

Artificial Neural Networks have been applied to many problems, and have demonstrated their superiority over classical methods when dealing with noisy or incomplete data. One such application is for data compression. Neural networks seem to be well suited to this particular function, as they have an ability to preprocess input patterns to produce simpler patterns with fewer components. This compressed information (stored in a hidden layer) preserves the full information obtained from the external environment. The compressed features may then exit the network into the external environment in their original uncompressed form*. (Mehare, Shibu 2012)*

Recently there has been a tremendous growth of interest in the field of neural networks. Yet the wide range of applications and architectures which fall under this category make a specific definition of the term “neural network" difficult. A neural network can be defined as a “massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use". Generally, neural networks refer to a computational paradigm in which a large number of simple computational units, interconnected to form a network, perform complex computational tasks. There is an analogy between such a computational model and the functioning of complex neurobiological systems. Higher-order neurobiological systems, of which the human brain is the ultimate example, perform extremely complex tasks using a highly connected network of neurons, in which each neuron performs a relatively simple information processing task. *(Dony, 1995)*

Neural networks can be trained to represent certain sets of data. After decomposing an image using the discrete wavelet transform (DWT), a neural network may be able to represent the DWT coefficients in less space than the coefficients themselves. After splitting the image and the decomposition using several methods, neural networks were trained to represent the image blocks. By saving the weights and bias of each neuron, an image segment can be approximately recreated. Compression can be achieved using neural networks. Current results have been promising except for the amount of time needed to train a neural network. *(Gillespie)*

There are many methods with which to train a neural network. Training implies modifying the weights and bias of each neuron until an acceptable output goal is reached. During training, if the output is far from its desired target, the weights and biases are changed to help achieve a lower error. Normally, neural networks are used to generalize a function. Since there is noise in almost any data, the neural network should not be exact to better generalize the function. However, since a neural network will be used to approximate only one section of the image in the proposed approach, it is acceptable for the network to be exact. *(Gillespie)*

2.2 Image Processing

Image Processing is a very interesting and a hot area where day-to-day improvement is quite inexplicable and has become an integral part of own lives. Image processing is the analysis, manipulation, storage, and display of graphical images. An image is digitized to convert it to a form which can be stored in a computer's memory or on some form of storage media such as a hard disk. This digitization procedure can be done by a scanner, or by a video camera connected to a frame grabber board in a computer. Once the image has been digitized, it can be operated upon by various image processing operations. Image processing is a module that is primarily used to enhance the quality and appearance of black and white images. It also enhances the quality of the scanned or faxed document, by performing operations that remove imperfections. Image processing operations can be roughly divided into three major categories, Image Enhancement, Image Restoration and Image Compression. Image compression is familiar to most people. It involves reducing the amount of memory needed to store a digital image. *(Reddy, Vikram, Rao 2012)*

Digital image presentation requires a large amount of data and its transmission over communication channels is time consuming. To rectify that problem, large number of techniques to compress the amount of data for representing a digital image have been developed to make its storage and transmission economical. One of the major difficulties encountered in image processing is the huge amount of data used to store an image. Thus, there is a pressing need to limit the resulting data volume. Image compression techniques aim to remove the redundancy present in data in a way, which makes image reconstruction possible. Image compression continues to be an important subject in many areas such as communication, data storage, computation etc. *(Reddy, Vikram, Rao 2012)*

Image Processing is an area of investigation that uses several techniques and algorithms in order to interpret and understand the information contained in a digital image. Such algorithms may be classifiedoptimization. For any of these tasks, it is necessary to interpret information with a certain amount of uncertainty associated to it, which is typically done using Computational Intelligence techniques such as Fuzzy Logic (FL), Genetic Algorithms (GA) and Artificial Neural Networks (ANN). The interest on ANNs has been on the rise due to them being inspired on the nervous system, their usefulness for solving pattern recognition problems and their parallel architectures. *(Quintana, Murguia and Hinojos 2012)*

2.3 Image Compression

In the literature, methods such as neural networks and genetic algorithms have been developed to discover the future of image compression. Rahman successfully compressed and decompressed image data using counter propagation neural network. J. Jiang, on his survey, discussed various up-to-date image compression neural network which he classified into three different categories according to the nature of their application and design. These include direct development of neural learning algorithms for image compression, neural network implementation of traditional image compression algorithms and indirect applications of neural networks to assist with those existing image compression techniques. Panda 2012 implemented back propagation neural network on his image compression system. He observed that the convergence time for the training of back propagation neural network is fast. Ilic and Berkovic concluded that the best results of the compression of the digital grayscale images can be achieved by taining the network with the SCG and RPROP back propagation algorithms. At the end of their research, the result is the image with the satisfied quality and satisfied training time even for the compression rate of 16.

The need for effective data compression is evident in almost all applications where storage and transmission of digital images are involved. For examples an 1024 X1024 color image with 8 bit/pixel generates 25Mbits data, which without compression requires about 7 minutes of transmission time over 64 kbps line. A Compact Disk with storage capacity of 5 gigabytes can only hold about 200 uncompressed images. Reducing the image size by applying compression techniques is usually possible because images contain a high degree of spatial redundancy due to correlation between neighbouring pixels. *(Mahmoud, 2007)*

Data compression is often referred to as coding, where coding is very general term encompassing any special representation of data which satisfies a given need. As with any communication, compressed data communication only works when both the sender and receiver of the information understand the encoding scheme. For example, this text makes sense only if the receiver understands that it is intended to be interpreted as characters representing the English language. Similarly, compressed data can only be understood if the decoding method is known by the receiver. It is useful because it helps to reduce consumption of expensive resources such as hard disk space or transmission bandwidth i.e. a data file that is supposed to takes up 50 kilobytes (KB) could be downsized to 25 kilobytes (KB), by using data compression software. A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Data compression has important application in the areas of data transmission and data storage. Compressed data required smaller storage size and reduce the amount of data that need to be transmitted. Hence, it increases the capacity of the communication channel.*(Mehare, Shibu 2012).*

Images play an important role in our lives. They are used in many applications. In our study we have applied different type of compression technique on different type of images for a PSNR value and compression ratio. In this paper, the problem of compressing an image in MATLAB environment has been taken. A MATLAB compression approach has been implemented for compressing images. After analysis we have found that, scene Images Wavelet provides the better result but compression ratio is high in case of BTC and visual quality is better of BTC. For face Images BTC perform the most compression. FIC provide almost same result as compare to BTC, but fractal image compression take a long time for compression, so we can use BTC as compare to FIC when time in main concern. We analyzed the PSNR obtained of compressed after each compression technique and decides which technique can provide maximum PSNR for a particular image *(Thakur, Kumar, Bath, Sharma 2014)*

Compression of a specific type of data entails transforming and organizing the data in a way which is easily represented. Images are in wide use today, and decreasing the bandwidth and space required by them is a benefit. With images, lossy compression is generally allowed as long as the losses are subjectively unnoticeable to the human eye. The human visual system is not as sensitive to changes in high frequencies. This piece of information can be utilized by image compression methods. After converting an image into the frequency domain, we can effectively control the magnitudes of higher frequencies in an image. *(Gillespie)*

Image compression is playing a key role in the development of various multimedia computer services and telecommunication applications. As image needs a huge amount of data to store it, there is pressing need to limit image data volume for transport along communication links. An ideal image compression system must yield good quality compressed images with good compression ratio, while maintaining minimal time cost. The goal of image compression techniques is to remove redundancy present in data in a way that enables image compression technique. There are numerous lossy and lossless image compression techniques. For the still digital image or video, a lossy compression is preferred. Wavelet-based image compression provides substantial improvements in picture quality at higher compression ratios. Contrary to traditional techniques for image compression, neural networks can also be used for data or image compression*. (Subbarao, Khasim, Thati, Sastry 2013)*

Chapter 3
Theoretical and Conceptual Framework

This chapter will discuss the main idea of neural networks application for the problem of image compression. The main concept of this study is to create such a network wherein the input and output layer having the same number of neurons is connected to the middle (hidden) layer with smaller number of neuron – which is the requirement for compression apprehension.

The goal of such compression is the reconstruction of the original image, which will be the input for the network; hence the output of the network that solve this class of problems must be equal to its input. Network training begins with the initialization of its weight and requires a set of examples i.e. appropriate input-output pairs. Weights in the network are iteratively adjusted throughout the training time to minimize the error function of the network, which is so-called MSE (Mean Square Error between the calculated and expected network output). The procedure is repeated until satisfied level of deviation between calculated and expected output is reached.

3.1 Theoretical Framework

3.1.1 Advantage of Image Compression

The benefits that we can take advantage on after this study is that it provides a believable cost savings involved with sending less data over the switched-telephone network where the cost of the call is really based upon its duration. It's not only reduce storage requirements but also its overall execution of time during the transferring, downloading or uploading images. It reduces the probability of error-transmission since fewer bits are transferred. And lastly, it provides a level of security against unlawful monitoring

3.1.2 Data Redundancies

Redundancy do contain in an image data and it can be remove. It has three types:

a. Coding redundancy (statistical redundancy):

A code is a system of symbols representing an information. Each bits of information is represented by a set of code symbols. Pixels in an image are represented by a sequence of code words. Thus, coding redundancy is present when less than ideal code words are used. It can occur when codes are pick without considering the relative probabilities of intensities. This type of coding can be reversed and usually implemented using look up tables (LUTs). Some of image coding schemes that make use of coding redundancy are the Huffman codes and the arithmetic coding technique. We use entropy to measure coding redundancy.

b. Inter pixel redundancy (spatial redundancy)

This is sometimes called inter frame redundancy or geometric redundancy. It can be used in several ways, one of which is by predicting a pixel value based on the values of its neighboring pixels. In order to do so, the original 2-D array of pixels is commonly mapped into a different format, e.g., an array of differences between adjacent pixels. If the original image pixels can be reconstructed from the transformed data set, the mapping is said to be reversible. Intensity values in an image typically vary slowly in space or time over large portions of an image. Examples of compression techniques that utilize the inter pixel redundancy are Constant Area Coding (CAC), (1-D or 2-D) Run Length Encoding (RLE) techniques, and many predictive coding algorithms such as Differential Pulse Code Modulation (DPCM). We use correlation to measure this redundancy.

c. Psycho visual redundancy (perceptual redundancy)

Based on the experiments on the psychophysical aspects of human vision, it has been proven that the human eye, when it comes to receiving visual information, doesn’t respond with equal sensitivity. This have led to the aim of the image compression wherein the data that is psycho visually redundant will be removed. Then the result of these technique will be smaller in size and quality. Removing this type of redundancy is a lossy process and the lost information cannot be recovered. The method used to remove this type of redundancy is called quantization which means the mapping of a broad range of input values to a limited number of output values

3.1.3 The Rate-Distortion Theory

Segregating the image data into different classes is the first step in compressing an image. Each class is allocated a portion of the total bit budget, depending on the importance of the data it contains, such that the compressed image has the minimum possible distortion. This procedure is called Bit Allocation. The Rate-Distortion theory is used for solving the problem of allocating bits to a set of classes, or for bit rate control in general. The theory aims at reducing the distortion for a given target bit rate, by optimally allocating bits to the various classes of data. One approach to solve the problem of Optimal Bit Allocation using the Rate-Distortion theory is given below.

1. Initially, all classes are allocated a predefined maximum number of bits.
2. For each class, one bit is reduced from its quota of allocated bits, and the distortion due to the reduction of that 1 bit is calculated.
3. Of all the classes, the class with minimum distortion for a reduction of 1 bit is noted, and 1 bit is reduced from its quota of bits.
4. The total distortion for all classes D is calculated.
5. The total rate for all the classes is calculated as $R = p(i) * B(i)$, where p is the probability and B is the bit allocation for each class.
6. Compare the target rate and distortion specifications with the values

3.2 Conceptual Framework

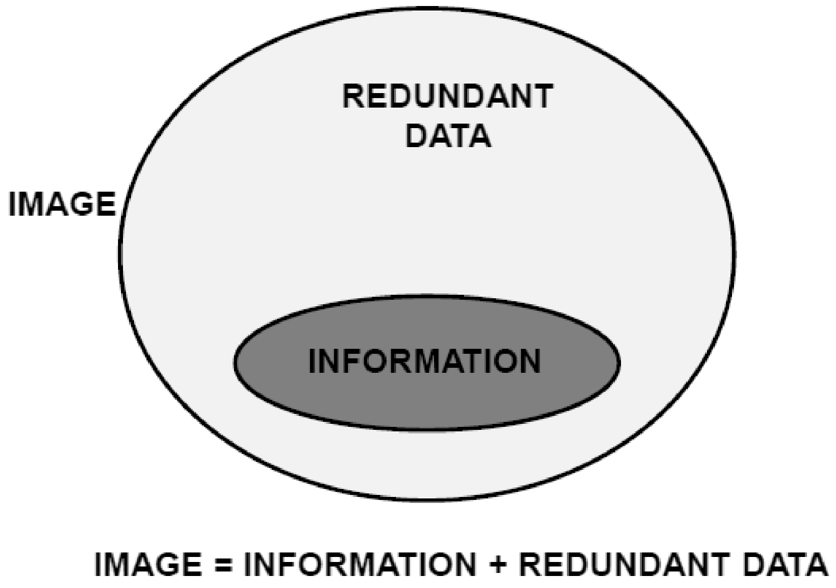


Figure1. Information vs. Data

3.2.1 Back Propagation Neural Network (BPNN)

The neural network used in this study is based on Back propagation neural network. Back propagation algorithm was first presented in the 1970s, but its significance wasn't fully appreciated until a famous 1986 paper was released by David Rumelhart, Geoffrey Hinton, and Ronald Williams. It has the simplest architecture of ANN that has been utilized for image compression but the disadvantage of it is very slow convergence. It architecture has three layers: input, hidden, and output layers. An image has an internal data representation and during the training phase, these data is being fed into the input layer and disseminated to the hidden layer and then to the output layer. This manner is called the forward pass of the back propagation algorithm in which each node in hidden layer gets input from all the nodes from input layer, which are multiplied with their appropriate weights and then summed it all. The compressed features of the image are stored in the hidden layer, therefore the smaller the number of hidden neurons, the higher the compression ratio. The output of the hidden node is the nonlinear transformation of this resulting sum. Similarly, the input of the output layer is from all the nodes from the hidden layer, which is also multiplied with appropriate weights and then summed. The output of this node is the non-linear transformation of the resulting sum. After these process, the output data of the output layer are compared with the target output values. The target output values are those that we attempt to teach our network. The error between these values is calculated and propagated back toward hidden layer. This is called the backward pass of the back propagation algorithm. The error is used to update the connection strengths between nodes, i.e. weight matrices between input-hidden layers and hidden-output layers are updated. During the testing phase, no learning takes place i.e., weight matrices are not changed. Each test vector is fed into the input layer. The feed forward of the testing data is similar to the feed forward of the training data.

3.2.2 Image Compression Technique

Image compression techniques are broadly divided in following two major categories.

- Lossless image compression
- Lossy image compression

3.2.2.1 Lossless Compression

If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. It is generally used for applications that cannot allow any difference between the original and reconstructed data. Lossless compression is sometimes preferred for artificial images such as technical drawings, icons or comics. This is because lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossless compression methods may also be preferred for high value content, such as medical imagery or image scans made for archival purposes

a) Run length coding

Removing repeated data values and replacing a value-length pairs is the aim of run length coding where value is the intensity of pixel and length refers to the number of consecutive pixels with the intensity value. It ignore row boundaries in image and save additional spaces, which is what image compression is all about. How does run length coding is applied? First, it scan the left, right, top and bottom pixels of the image and create pairs of value and length. Values and lengths are stored using a given bits per pixels between the possible ranges.

$$\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\
4 & 4 & 5 & 5 & 5 & 6 & 6 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
7 & 0 & 7 & 0 & 7 & 0 & 7 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\quad => \quad
\begin{array}{c}
(0,2)(1,6) \\
(1,3)(2,5) \\
(2,4)(3,4) \\
(4,2)(5,3)(6,3) \\
(6,8) \\
(6,1)(7,7) \\
(0,1)(7,1)\ldots(0,1) \\
(0,8)
\end{array}$$

Figure 2. Example of Run Length Coding

b) Entropy Coding

After the process of quantization, the data is sent to entropy coder to give an additional compression. Its aim is to achieve less average length of the image by creating and assigning a minimum number of bits required for each unique representation of data that the input has according to the probability of each symbol or data. .

c) Huffman Coding

The pixels in the image are treated as symbols. These symbols are assigned in a number of bits based on their frequent occurrence in the image. Symbols which occur more frequently are assigned in a smaller number of bits while symbols which occur less are assigned in a larger number of bits. A variable length code, which has been derived based on an estimated possibility of occurrence of a symbol, is used for encoding a symbol in the image. This type of encoding algorithm takes advantage of the coding redundancy in images.

d) Lempel-Ziv-Welch Coding

It is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. It became the first widely used universal image compression method on computers. A large English text file can typically be compressed via LZW to about half its original size. LZW is a dictionary based coding and this can be static or dynamic. In static dictionary coding, dictionary is fixed during the process of encoding and decoding unlike in dynamic where dictionary is being updated in the process. Its algorithm is simple to implement, and has the potential for very high quantity in hardware implementations. It was the algorithm of the widely used UNIX file compression utility compress, and is used in the GIF image format.

3.2.2.2 Lossy Compression Methods:

Lossy compression, as the name implies, allow some loss of information and data cannot be recovered or reconstructed exactly from the compressed data. It is being used in some applications like in facsimile transmission, broadcast television, video-conferencing, and, etc. wherein the exact representation of the image is not necessary as long as the difference between the original and the compressed data is still acceptable. it provides a higher compression ratio than the lossless compression. Major performance considerations of a lossy compression scheme include compression ratio, signal to noise ratio and speed of encoding & decoding. Lossy image compression techniques include following schemes:

a) Scalar Quantization

The most common type of quantization is known as scalar quantization. Scalar quantization, typically denoted as Y=Q (x), is the process of using a quantization function Q to map a scalar (one-dimensional) input value x to a scalar output value Y. Scalar quantization can be as simple and intuitive as rounding high-precision numbers to the nearest integer, or to the nearest multiple of some other unit of precision.

b) Vector Quantization

Vector quantization (VQ) is a classical quantization technique from signal processing which allows the modeling of probability density functions by the distribution of prototype vectors. It was originally used for image compression. It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. The density matching property of vector quantization is powerful, especially for identifying the density of large and high-dimensioned data. Since data points are represented by the index of their closest centroid, commonly occurring data have low error, and rare data high error. This is why VQ is suitable for lossy data compression. It can also be used for lossy data correction and density estimation

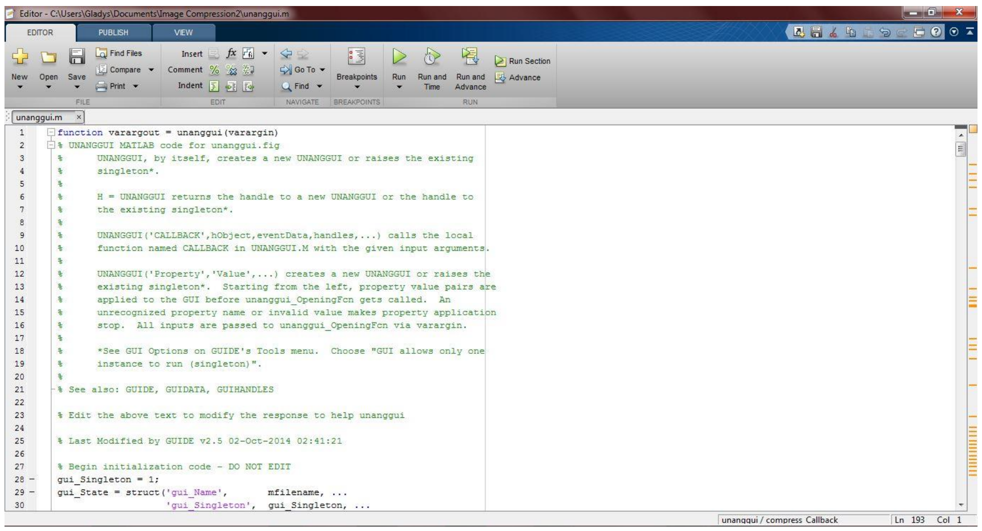
Chapter 4
Methodology

The chapter will discuss about the procedures and techniques used by the researchers.

4.1 Interfacing the Image Compression System

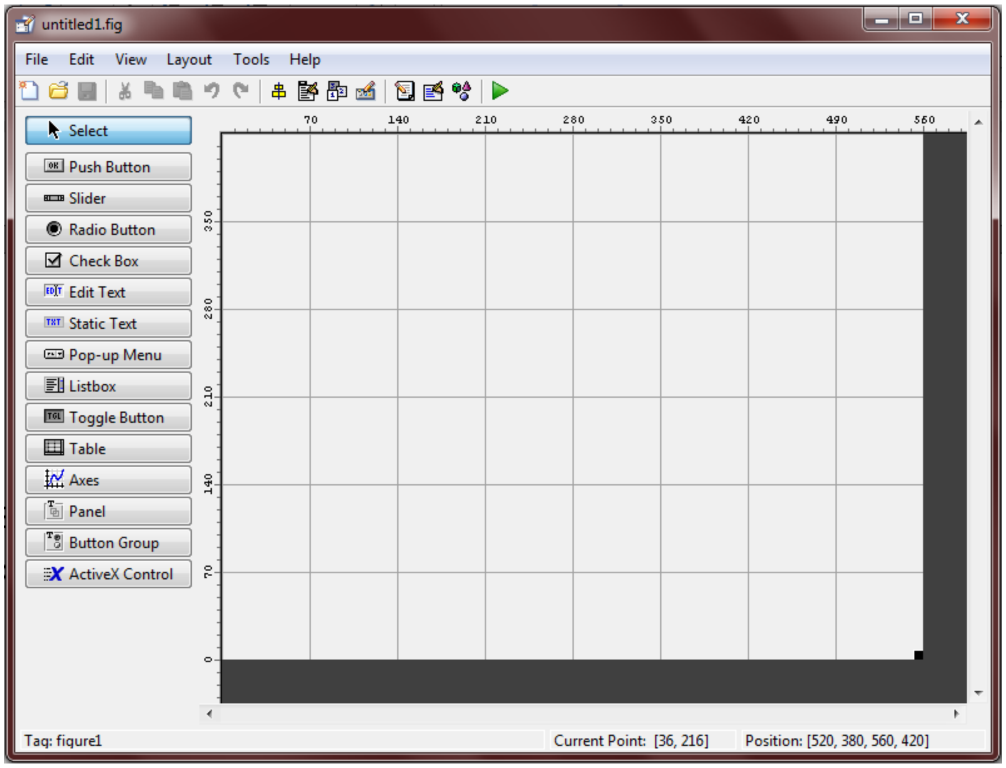
4.1.1 Creating a Graphical User Interface Window (with GUIDE)

cThe researchers aim to create their system interface using MATLAB Graphical User Interface Development Environment. To create a GUI, click on New on the Home tab, then select the Graphical User Interface. The GUIDE Quick Start window will pop up on the screen. Go to the ‘Create New GUI’ tab then select ‘Blank GUI (Default)’. Save the new blank GUI file, with an extension ‘*.fig’ by checking the ‘Save new figure as’ below the tab and browse or create the folder wherein the GUI will be saved. After this, an editor and a ‘*.fig’ window will be shown on the screen. This is where the desired GUI window can be edited: the ‘*.fig’ window for the layout of the GUI and the editor for the function and process of the GUI.



```
function varargout = unanggui(varargin)
% UNANGGUI MATLAB code for unanggui.fig
%      UNANGGUI, by itself, creates a new UNANGGUI or raises the existing
%      singleton*.
%
%      H = UNANGGUI returns the handle to a new UNANGGUI or the handle to
%      the existing singleton*.
%
%      UNANGGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in UNANGGUI.M with the given input arguments.
%
%      UNANGGUI('Property','Value',...) creates a new UNANGGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before unanggui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to unanggui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help unanggui

% Last Modified by GUIDE v2.5 02-Oct-2014 02:41:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

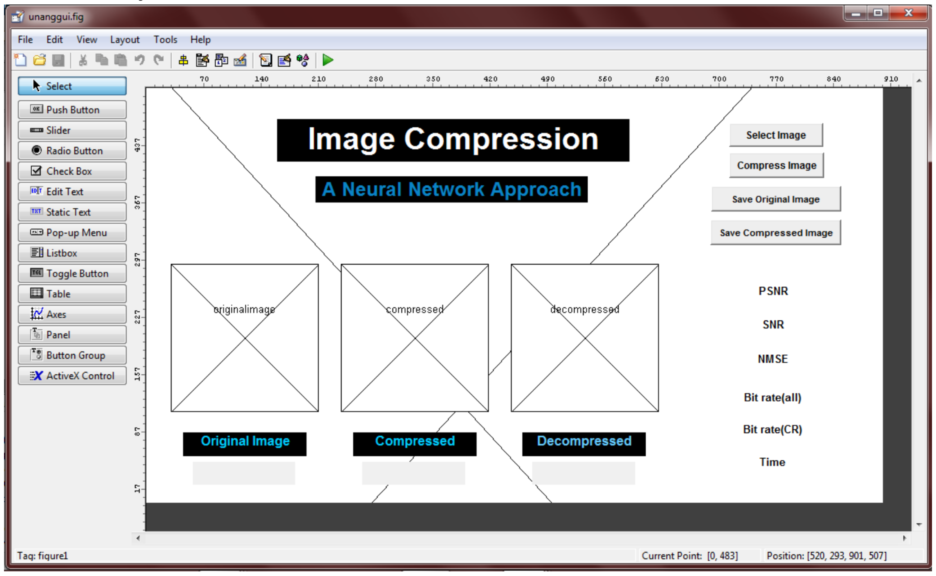
Figure 3. The Editor Window



Tag: figure1 | Current Point: [36, 216] | Position: [520, 380, 560, 420]

Figure 4. Graphical User Interface Development Environment (GUIDE)

Select buttons, axes (for viewing image or for plotting function), or static texts, and modify the string and tag of each selected object on the property inspector to call its functions on the editor. The property inspector allows to modify the necessary adjustment to be made on the object including the size of the object, its color, button function and etc.



Tag: figure1 | Current Point: [0, 483] | Position: [520, 293, 901, 507]

Figure 5. Image Compression GUI layout in MATLAB GUIDE

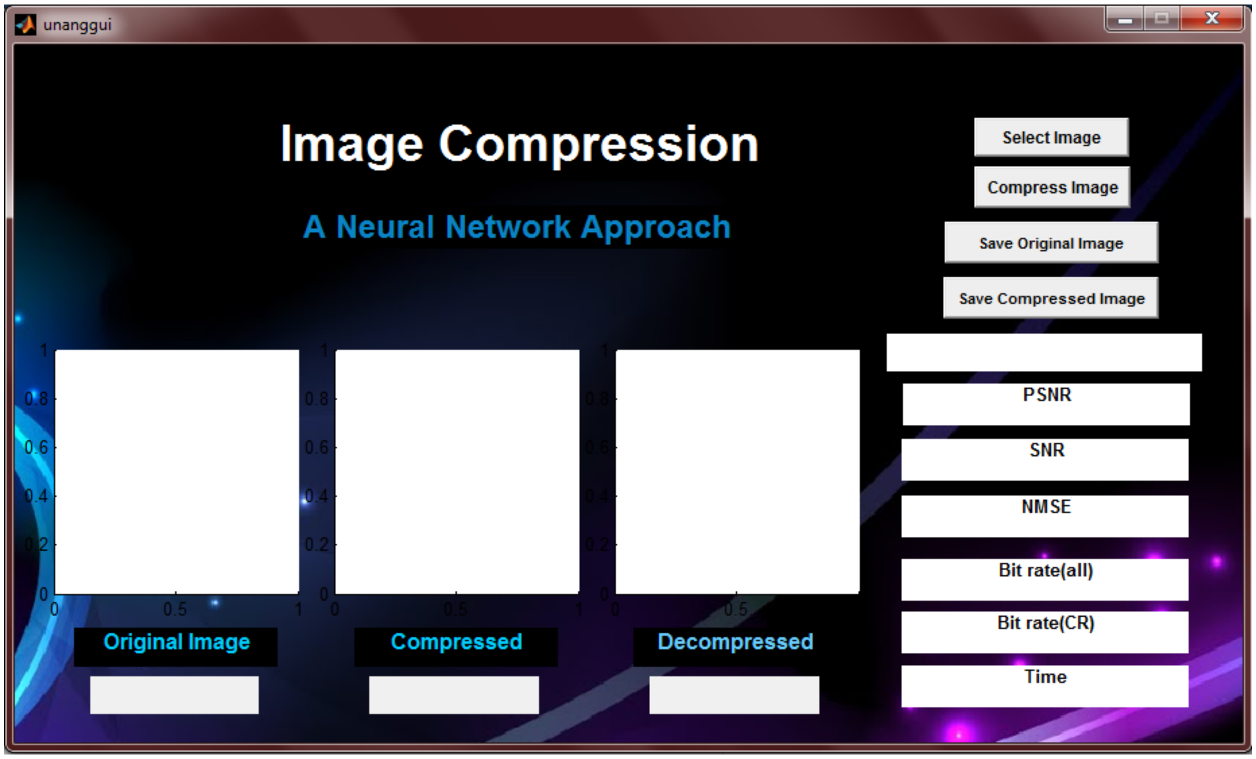


Figure 6. The researcher's Image compression system GUI

4.2 The Image Compression Process

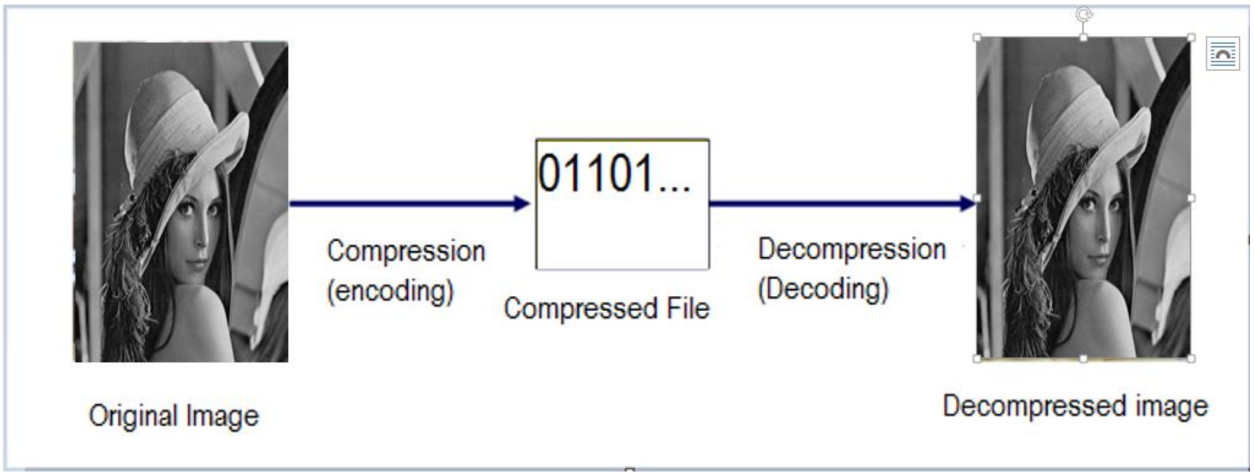


Figure 7. Block diagram of Image Compression

4.2.1 Constructing a Network Architecture

Network architecture has been defined by the researchers to establish the system.

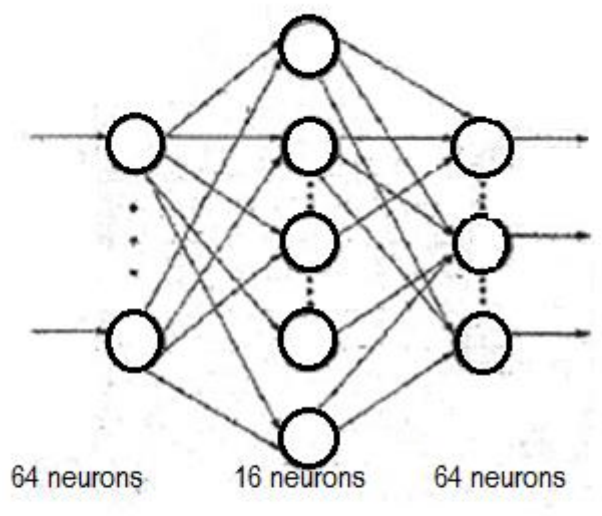


Figure 8. Typical Neural Network Architecture (Ilic and Berkovic)

The chosen number of input and output neurons should be the same and decision about its quantity must be assessed properly because it is solely related to the whole system that is being created. In image compression, for example, selected number of input and output neurons depends on the number of two dimensional array of pixels of an image which each elements of pixels are values from 0 to 255. Then the constructed network will compress the grayscale image. Blocks of size 8x8 pixels are extracted from the image, which means that values of 64 pixels used in network training are presented to the network input layer. Therefore, input layer, same with the output layer, has 64 neurons and choice about the number of neurons in the hidden layer is free but it has to be less than the number of neurons in input layer.

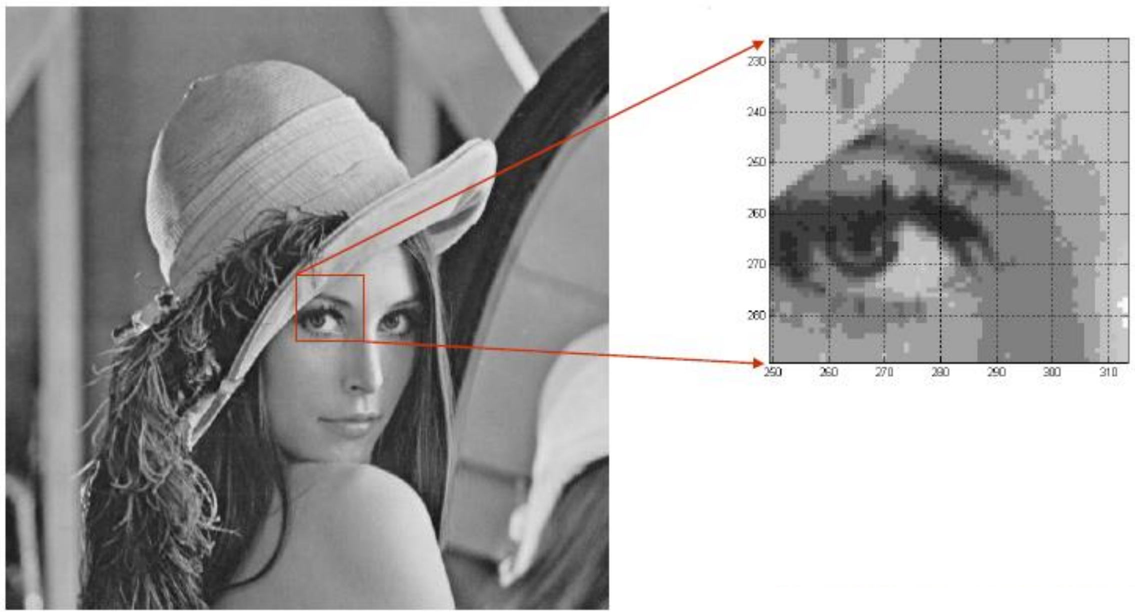


Figure 9. Lena's extracted Block of Image

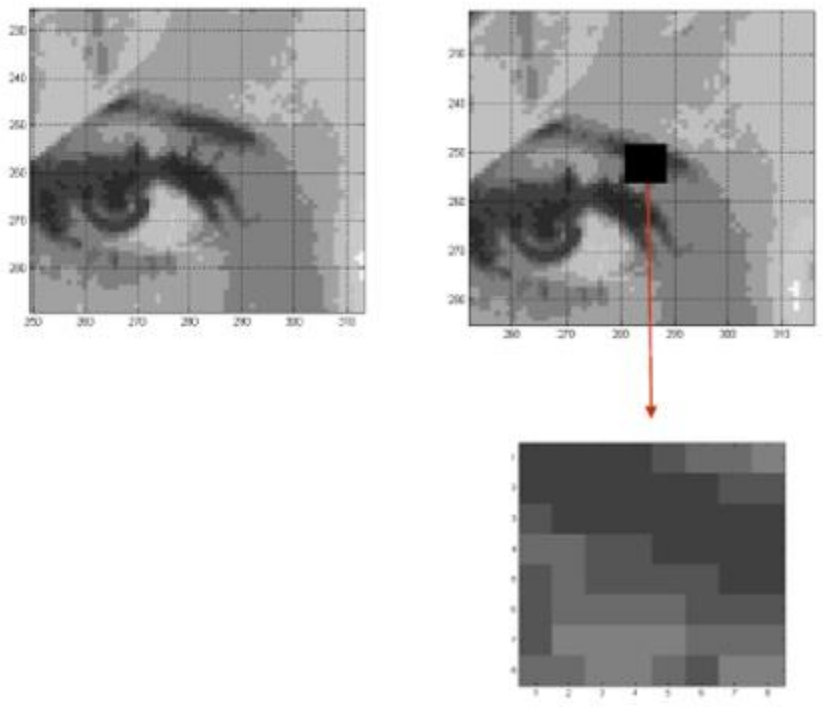
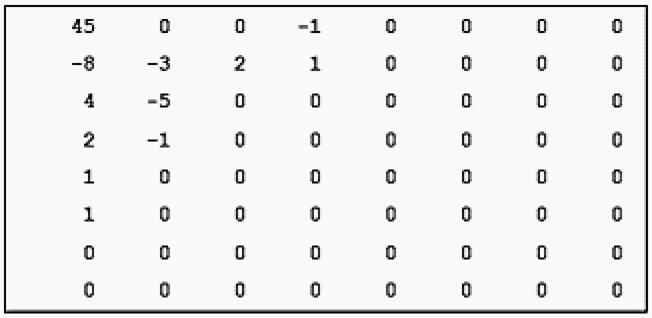


Figure 10. 8x8 Blocked to be processed

4.2.2 Creating Input Values

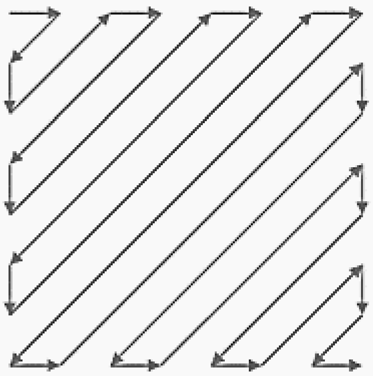
Image is divided into 8x8 pixels blocks and this will be the input for the network. The activation function, which is the sigmoid function requires each neuron to be in the range of 0 to 1, because normally, the image have input values ranging from 0- 255. The process of linearly transformation of image values from 0-255 into another range appropriate for the network is called Normalization. It is attained through the division of pixel values with its biggest possible value (255). Network is more operative when the input and output values are limited only to the interval [0,1].

Initially, image is a 2-dimensional blocks. Therefore linearization is required to transform the 2-dimensional block image into 1-dimensional vector for the representation of the input into the neural network. Linearization involves the methods of scanning rows, columns or labyrinth. To simplify the learning process, segmentation is used. It is the process of dividing it into non overlapping blocks with equal size.



45	0	0	-1	0	0	0	0
-8	-3	2	1	0	0	0	0
4	-5	0	0	0	0	0	0
2	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Normalized input values



Zigzag Scan Procedure

```
Result = 45,0,-8,4,-3,0,-1,2,-5,2,1,-1,0,1,0,0,0,0,0,0,1,EOB
```

Figure11 .Process of creating input values

4.2.3 Training Pairs Preparation

For the training of the network to be possible, two weight matrices must be represented. The first matrix, which the researchers assigned in a variable v, represents the weight connections between the input and hidden layer. And the second matrix w, is for the connections between the hidden and the output layer. This has been initialized into small random values because these training pairs affect the whole process of training the network. Its value, as other researchers have utilized also, must be ranging between numbers having a values of 0.1 to 0.9, but it still depends on the user of the network. Large values can lead to the instability of the network while smaller values can cause excessively slow learning

4.2.3 Network Training

The researchers chose the appropriate learning algorithm for the network which is the back propagation algorithm. Algorithm is responsible for the training of the network by modifying its weight. The desired output has been provided so that the weights circulating in the network will be updated by the algorithm according to the quadratic error between output and the desired one. The error must be in its highest minimum value in order for the network to achieve its goal. The weights matrices are being remembered during the training process. These stored parameters provides the possibility of presenting to the network a completely new image that is also still need to be processed and in order to apply algorithm of compression and decompression.

The algorithm used gradient of preformation function in order to identify the way how to modify weights and to minimize the function. For the weights w in the network according to the gradient rule, new value is calculated following the expression:

$$W_t\ (new) = W_i\ (old) + \alpha(Y - O)\ X_i$$

Where:

a - constant that maintain the learning rate

X_i - input of the neuron i

Y - calculated output from the neuron

O - expected output from the neuron

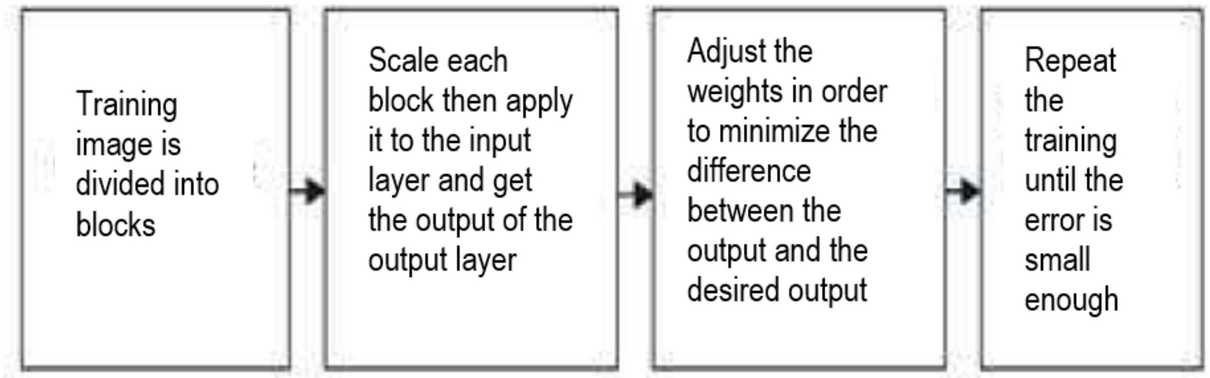

Training image is divided into blocks
Scale each block then apply it to the input layer and get the output of the output layer
Adjust the weights in order to minimize the difference between the output and the desired output
Repeat the training until the error is small enough

Figure 12. Block Diagram of Network Training

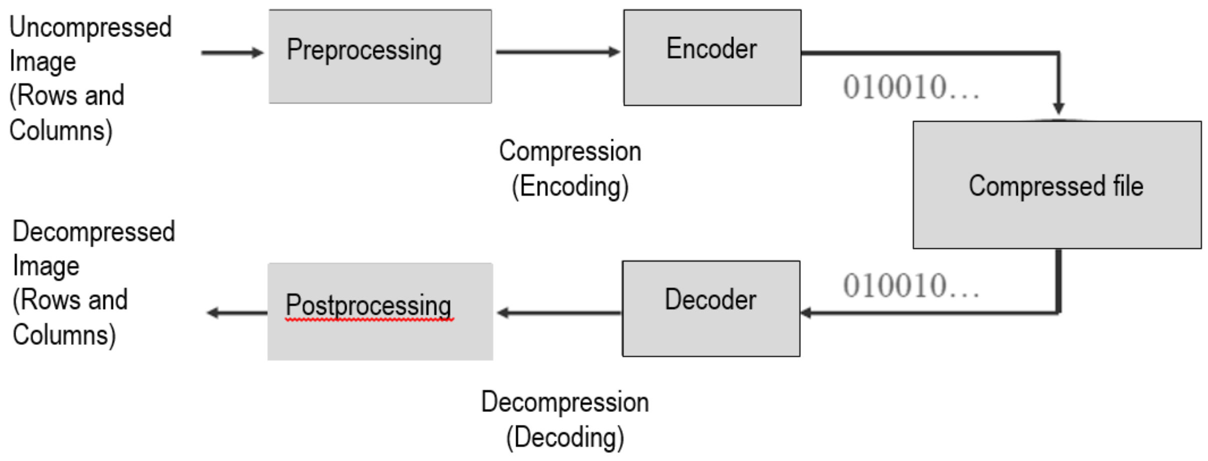


Figure 13. Image Compression Model

4.2.4 Compression

To remove the redundancies and to create the main process of compressing the image, the encoder is designed. After the network training, new inputs and expected outputs (image that is going to be compressed) are transferred into the network input layer. The first stage of encoding is called mapping. A mapper is used to reduce some data redundancies like spatial and temporal redundancies. This operation is generally reversible and may or may not reduce directly the amount of data required to represent the image. Some methods used for mapping are run length coding, predictive coding and transform coding. With the used of the stored weights, hidden layer output is calculated and then attained values are quantified with 8 bits and remembered.

Next stage for compressing the image is called the quantizer. Quantizer reduce psychovisual redundancies. It is irreversible unlike the mapper and the symbol encoder. Keeping the irrelevant information is its goal out of the compressed representation. It must be omitted when the desired error-free compression has been attained. Uniform quantization and nonuniform quantization are the examples for quantizer.

After the data is being reduce by the mapper and the quantizer, it will now reduce the coding redundancies stored in the image using a symbol encoder like Huffman coding, arithmetic coding and LZW. This is the third and the final stage wherein it generates a fixed length code to represent the quantizer output and maps the output in accordance with the code. And same with quantizer, this operation is irreversible. The figure 10 below shows the block diagram of an encoder

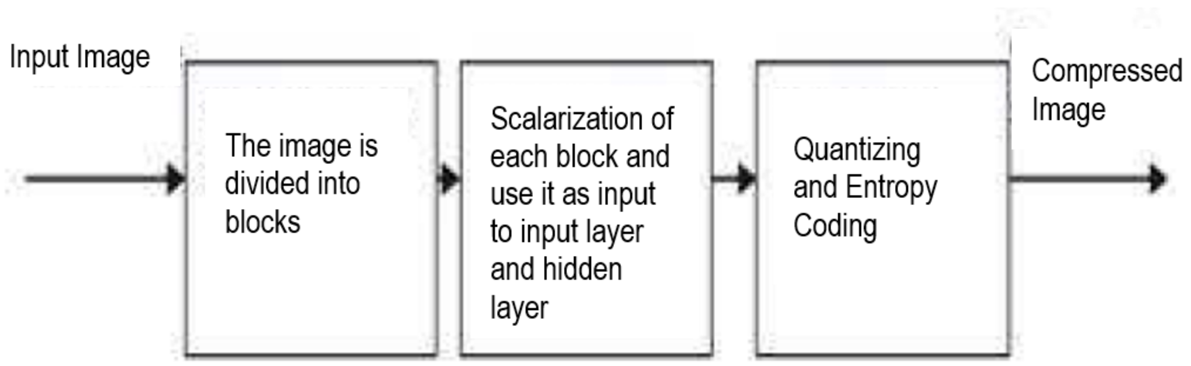

Input Image
The image is divided into blocks
Scalarization of each block and use it as input to input layer and hidden layer
Quantizing and Entropy Coding
Compressed Image

Figure 14. Encoding Steps (Mehare, Shibu 2012)

4.2.5 Decompression

Decompression simply reverse the path of compression. The reconstruction of the compressed image involves the reading the stored data (compressed image) to set as hidden layer outputs, and then, again using the previously stored weight matrices, calculate the network output with the input to the output layer and weights between hidden and output layer. This process of reconstruction the image is called post processing. The decoder (decompression) has two stages: a symbol decoder and an inverse mapper. They only perform the operation of encoder's symbol encoder and mapper in reversible way. The reason why decoder do not have quantizer is because the quantizer is irreversible, so this block is no longer included in the general decoder model.

To decompress the image; first decode the entropy coding then apply it to the output of the hidden layer and get the output of the OL scale it and reconstruct the image. Fig. 11 show the decoder block diagram.

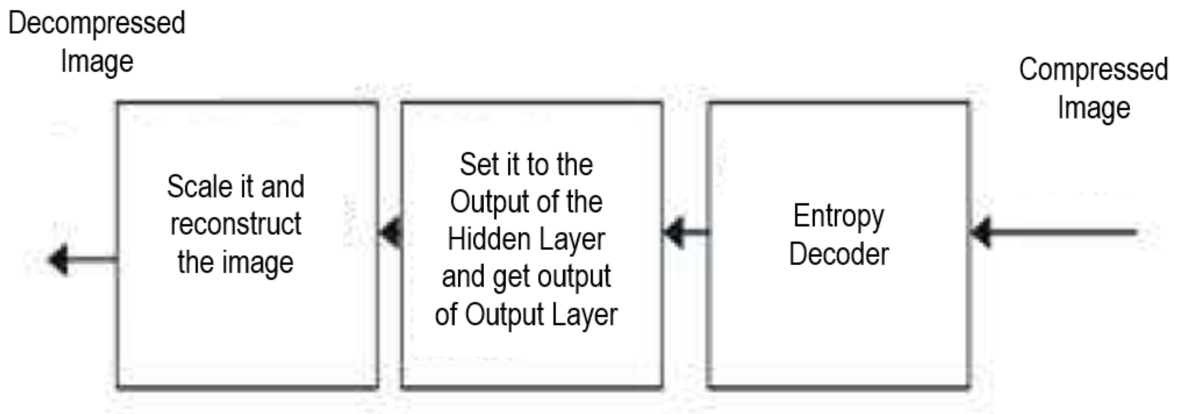

Decompressed Image
Scale it and reconstruct the image
Set it to the Output of the Hidden Layer and get output of Output Layer
Entropy Decoder
Compressed Image

Figure 15. Decoding steps (Mehare, Shibu 2012)

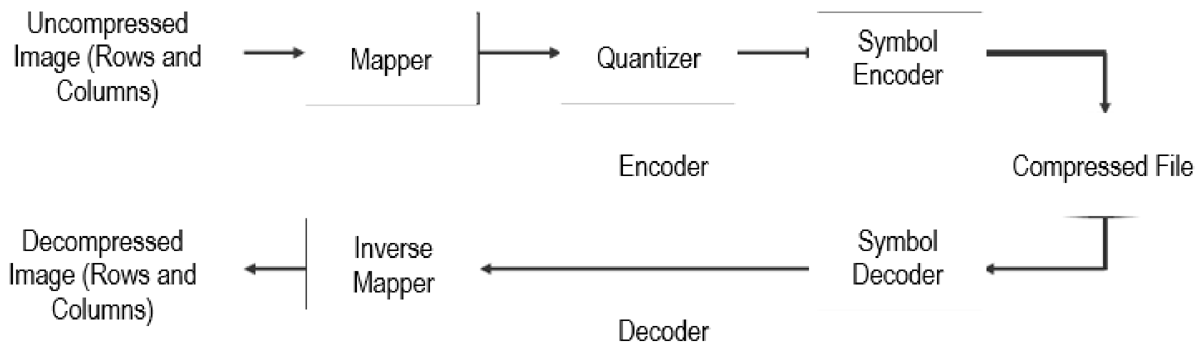


Figure 16. Process of encoding and decoding

Chapter 5
Results, Discussions and Conclusions

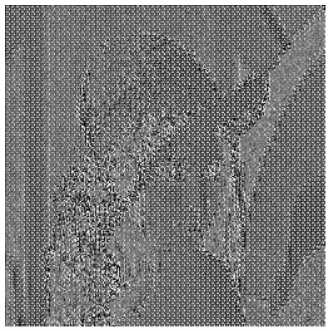
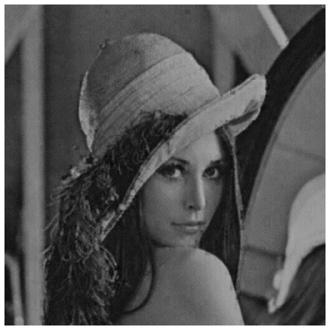
After the system has been modified and tested, this part of the study will be discussing about the topics connected and the result of the process made by the researchers based on some given criteria and also the final insights of the researchers after all these processes

5.1 Results and Discussion

Researchers made use of the Lena, cameraman and pepper as test images. The results obtained using the network with 64 neurons in input and output layer and 16 neurons in its hidden layer will be shown in the succeeding tables by means of fidelity criteria. Fidelity criteria is used to measure information loss in image compression and it is basically divided into two types: subjective fidelity criteria and objective fidelity criteria.

5.1.1 Subjective Fidelity Criteria

It is measured by human observation. The table below shows the subjective differences of the original image, the result of the encoding process which is the compressed image and the decompressed image.

Name	Original Image	Compressed Image	Decompressed Image
Lena			

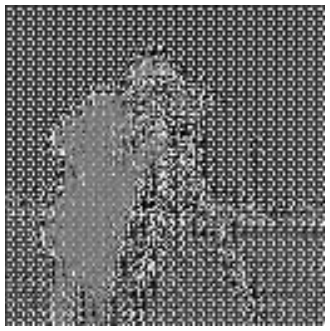
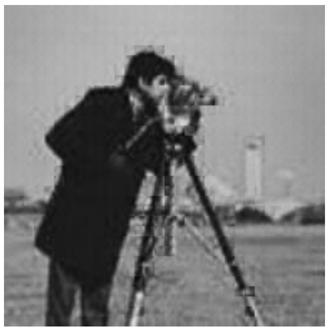
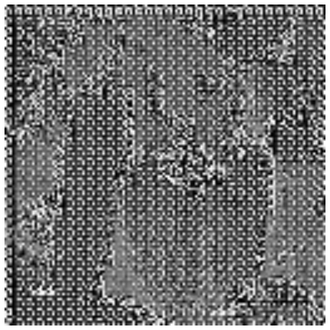
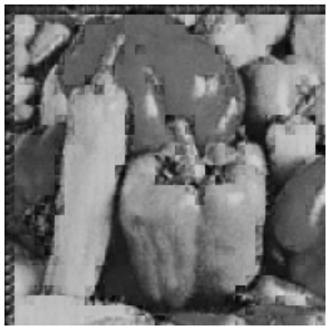
Cameraman			
Peppers			

Table 1. Compression and Decompression results of sample images

This result indicates that decompression of images is possible but the quality, based on this type of criteria, will depend on the dogmatic observation of human. Researchers have provided a subjective evaluations as the basis for rating the image quality.

Impairment	Quality	Comparison
5 - Imperceptible	A - Excellent	+2 much better
4 - Perceptible, not annoying	B - Good	+1 better
3 - Somewhat annoying	C - Fair	0 the same
2 - Severely annoying	D - Poor	-1 worse
1 - unusable	E - Bad	-2 much worse

Table 2. Sample Subjective Evaluation

Impairment test is used to evaluate how bad are the decompressed images while quality test is for how good the decompressed images are. Comparison test weigh the result between the original image and the decompressed image

5.1.2 Objective Fidelity Criteria.

It measures mathematically the amount of error between the original image and the compressed/decompressed image. The researchers made use of some formulas in measuring the quality of the decompressed images. Such formulas used were signal-to-noise ratio (SNR), peak signal-to-noise ratio (PSNR) and normalized-mean-square-error (NMSE). Table 2 summarizes the comparison of the properties of the images after the process of compressing and decompressing the image, which have utilized by the researchers to calculate some quality measures to be discussed further in this section.

before compression					after compression		
filename	dimension	bit depth	size	type	size	save as	Time
Lena	512x512	8	256kb	bmp	24.3kb	jpeg	14 seconds
pepper	256x256	8	39.2kb	png	8.69kb	jpeg	29 seconds
cameraman	256x256	8	65kb	bmp	6.39kb	jpeg	10 seconds

Table 3. Detailed properties of the images before and after compression

It is shown that the file size of the original images is reduced in a certain ratio. Image of Lena, from the original file size of 256kb, became 24.3 kb after the compression process. Same with the other test images like cameraman and pepper in which have reduced from 39.2kb to 8.69kb and 65kb to 6.39kb.

a. *Mean Square Error*

The mean squared error is computed based on the target image and the decompressed image. It should be as small as possible so that the quality of the reconstructed image is near to the target image. For ideal decompression, the value of MSE should be zero. The formula used to compute the MSE is

$$\text{MSE} = \frac{1}{NM}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]$$

The table below shows the result of the MSE of each images.

Image	MSE
Lena	0.010125

cameraman	0.008943
pepper	0.047913

Table 4. Calculated MSE of the test images

Based on the given results, having a values between 0.01 to 0.04, it is shown that the reconstructed images are near to the target images since the value of the MSE of each test images is close to zero .

b. Peak signal-to-noise ratio

The PSNR value is the difference between the original image and its decoded image. The larger the PSNR value, the better will be the decoded image quality. The PSNR computes by the following equation:

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{MSE}$$

The table below shows the result of the PSNR of each images

Image	PSNR
Lena	29.3509db
cameraman	26.2659 dB
pepper	18.8124 dB

Table 5. PSNR of each sample test images

If the results obtained a lower MSE and a high PSNR, it literally means that the decoded image is a better one.

c. Compression ratio

Compression ratio is calculated as the ratio between the original uncompressed image file and the decompressed image. This is basically the data fed to the input layer neurons and the out from the hidden layer neurons. The researcher's network has 64 input neuron and 16 hidden layer neuron, thus using the formula

$$\text{CR} = \left(1 - \frac{H}{I}\right) * 100\%$$

Where

H – Number of hidden layer neurons (16)

I – Number of input layer neurons (64)

The compression ratio of the network is 75%.

Another way of solving the compression ratio is based on the file sizes of the original and compressed image. The table below shows the calculated ratio based on the formula

$$CR = \frac{\text{uncompressed file size}}{\text{compressed file size}}$$

Image	original file size	compressed file size	compression ratio
lena	256kb	24.3kb	10.535
cameraman	39.2kb	8.69	4.5109
pepper	65kb	6.39kb	10.1721

Table 6. Compression Ratio between the original file size and the compressed file size

d. Bit Rate

This can be calculated as the average number of bits per pixel for the encoded image. The table provided is the result of the bit rate in the network. Both the bit rate and the distortion measure must be showed for any meaningful comparison of the network's performance.

Image	Bit Rate
lena	2.0313 bpp
cameraman	2.125 bpp
pepper	2.125 bpp

Table 7. Bit rate of the test images

5.2 Conclusion

Neural network based image compression technique has been used for the successful compression and decompression of the image. Based from this study, it can be concluded that image compression system can be applied using artificial neural network. Neural networks are well suited to the task of processing the image data because of its organizational structure and it provides an internal representation of data, which is needed in the process of image compression. MATLAB is used for the system, wherein it is capable of handling matrices of images. Neural Network approaches to image compression have been shown to perform as well as or better than standard approaches thus it should be considered as an alternative method to other traditional techniques of image compression.

Bibliography

Anjana B. and Mrs Shreeja R.: ‘Image Compression: An Artificial Neural Network Approach’

K. M. Ashikur Rahman and Chowdhury Mofizur Rahman: ‘A New Approach for Compressing Color Images using Neural Network’ Department of Computer Science and Engineering. Bangladesh University of Engineering and Technology, Dhaka-1000, Bangladesh

J. Jiang: ‘Image compression with neural networks: A survey’

S .S. Panda, M.S.R.S Prasad, MNM Prasad, Ch. SKVR Naidu: ‘Image Compression using Back Propagation Neural Network’

Dubravka Ilic, Ivana Berkovic: ‘Grayscale Image Compression Using Back Propagation Algorithm’

K.P.Paradeshi : IMAGE COMPRESSION BY EZW COMBINING HUFFMAN AND ARITHMETIC ENCODER

Jamshid Nazari and Okan K. Ersoy: Implementation of back-propagation neural networks with MatLab

Vaishali Mehare, Sini Shibu: A Neural Network Approach to Improve the Lossless Image Compression ratio

Vijayvargiya, Silakari and Pandey : A Survey: Various Techniques of Image Compression

Mahmud Hasan, Kamruddin Md. Nur: A Lossless Image Compression Technique using Location Based Approach

Wilford Gillespie : Still Image Compression Using Neural Network

G.M.Padmaja, 2 P.Nirupama : Analysis of Various Image Compression Techniques

Juan A. Ramírez-Quintana, Mario I. Chacon-Murguia and Jose F. Chacon-Hinojos: Artificial Neural Image Processing Applications: A Survey

M. Venkata Subbarao, N.Sayedu Khasim, Jagadeesh Thati and M. H. H.Sastry : Hybrid Image Compression using DWT and Neural Networks

Robert D. Dony : Neural Network Approaches to Image Compression

http://en.wikipedia.org/wiki/Digital_imaging

Appendices

Appendix A. Weight connections between the input and hidden layer.

-10.165	13.286	5.343	0.676	-0.382	-3.493	-2.883	1.845	-4.747	0.523	-6.590
12.768	-6.099	-5.501	-0.971	-1.001	0.471	2.378	-1.834	1.693	-0.663	1.921
3.171	1.818	6.550	-1.024	-1.367	0.780	4.447	0.605	1.544	-0.281	2.619
-10.366	5.979	5.068	0.929	0.239	-1.944	-1.709	1.939	-2.772	0.331	-2.466
7.602	-4.404	-5.127	0.220	0.181	1.622	3.281	0.463	1.560	-0.266	2.231
-0.101	2.359	-0.446	0.400	-0.456	-2.221	-3.041	0.993	-3.216	0.125	-2.337
-4.588	-0.245	-0.084	-1.104	-0.587	-2.201	-2.324	-2.332	-0.673	-1.248	0.183
-2.573	0.998	0.775	1.164	1.505	0.136	-0.013	-1.024	-0.843	0.755	-1.115
-0.176	1.612	-1.349	1.450	1.010	1.560	-0.499	0.206	-0.188	1.135	-3.132
8.897	-9.577	-15.396	-1.941	-0.998	1.651	-2.218	-0.755	3.327	-3.183	0.887
-6.198	1.586	1.716	-1.391	-1.072	-2.280	-1.801	0.850	-1.036	-1.312	0.700
-4.154	2.530	0.122	0.865	0.641	-2.469	-2.188	-0.306	-2.729	0.816	-2.021
3.384	1.746	1.523	-0.928	-1.153	0.561	0.518	-2.934	0.442	-0.092	0.191
8.036	-6.595	-4.200	4.076	3.193	4.805	6.535	5.775	2.193	2.083	1.320
-3.519	1.956	-0.405	-1.129	-0.536	-3.407	-3.550	-1.400	-2.209	-0.557	-2.864
5.550	-4.551	2.041	1.096	1.274	2.233	5.321	-0.679	1.981	1.328	2.006
1.741	-4.842	4.204	1.548	1.556	4.154	2.602	2.126	0.990	0.444	0.639
2.028	-4.210	-3.195	-0.331	0.507	0.141	0.550	-2.042	2.128	0.341	0.102
-2.591	-0.722	-0.194	-0.421	0.423	2.979	2.914	-0.980	3.348	-0.087	5.084
-5.222	3.348	4.689	-0.392	-0.257	4.053	0.712	1.192	0.336	-0.720	0.488
0.627	0.565	0.210	-2.329	-0.818	3.916	0.669	0.857	0.970	-1.785	2.569
0.327	-5.113	-3.005	-0.620	-0.053	2.350	-3.700	1.587	-0.575	-1.853	0.166
-4.764	-0.982	3.076	-0.315	0.547	-0.455	-1.115	0.125	-0.202	0.416	0.353
4.960	1.893	-2.760	-2.023	-1.709	1.948	-1.573	-2.902	0.791	-2.897	-0.873
-6.970	1.299	11.218	0.821	0.908	-3.853	0.218	-1.251	-1.234	1.384	-0.821
1.309	-2.426	-2.141	2.009	0.921	-0.520	0.370	1.641	-0.930	0.614	-1.151
-5.390	-0.287	2.664	-0.805	-0.143	-8.247	-2.706	-3.916	-0.636	0.807	-1.094
-4.325	0.518	-3.100	0.006	0.361	-2.697	-4.256	1.416	-2.462	-0.011	-3.214
-2.062	6.609	2.262	-1.470	-0.866	-0.635	-1.020	-0.677	-1.932	0.462	-1.628
-8.331	4.396	3.351	0.255	0.968	-6.136	-4.373	-0.615	-2.810	0.925	-4.355
-3.786	4.001	8.471	-0.905	-0.166	-1.316	0.423	-1.121	-0.652	0.052	-0.104
2.345	3.692	2.850	-2.127	-2.686	0.326	0.311	-0.938	-0.300	-2.137	2.766
-4.136	4.471	9.403	-1.577	-1.414	0.689	1.084	-2.635	-0.224	-0.392	1.283
3.196	-5.778	-10.167	0.427	0.515	1.716	-5.349	1.677	-0.990	-0.569	-2.505
8.873	-2.231	0.687	-2.738	-2.208	3.178	6.291	-2.747	3.766	-1.224	5.989
-5.075	6.935	4.414	2.232	2.231	-3.299	-0.446	-0.360	-3.743	3.716	-4.908
6.844	-4.680	-3.432	1.187	0.772	2.592	5.308	1.208	3.123	0.525	3.025

7.665	-8.034	-6.585	-0.404	0.351	4.185	2.670	0.280	4.303	-0.369	4.183
7.873	-7.673	-2.141	-0.702	0.037	3.262	3.067	-0.416	3.885	-1.130	2.405
-5.139	1.809	-0.634	1.380	0.704	-2.421	0.298	0.721	0.341	1.140	-0.232
1.515	-1.410	5.088	-0.013	0.095	0.957	1.260	0.004	0.371	-0.099	2.081
0.510	4.265	2.501	1.061	0.178	-3.905	1.887	-1.101	-1.501	1.765	-0.881
-0.510	-0.043	-1.447	-0.135	0.052	2.305	-0.133	-1.208	2.127	-1.277	2.337
-0.767	1.061	-1.627	-1.720	-1.146	2.977	2.637	-0.703	1.795	-1.079	4.090
3.757	-0.022	-2.575	1.849	1.349	2.301	2.773	3.020	1.187	1.552	0.632
4.376	-2.913	-2.614	-0.518	-0.587	-3.913	-3.339	0.389	-2.303	-0.268	-4.239
-4.142	-0.021	3.701	-0.434	-0.854	-2.197	-0.599	-0.783	0.472	-0.503	-0.620
-0.081	-1.172	-5.301	2.541	1.522	1.390	1.124	2.333	1.627	0.399	0.884
-0.100	-0.511	-2.847	-0.334	-0.266	0.374	-2.575	1.749	-1.893	-0.803	-2.116
5.766	-0.646	-4.325	-1.475	-1.535	0.846	1.206	-0.010	1.474	-1.601	2.909
3.282	0.710	-1.969	-0.363	0.054	1.931	-0.432	-0.235	-0.695	0.011	-0.414
-7.456	3.344	-2.593	0.484	0.275	-5.065	-6.652	1.180	-2.252	0.517	-4.204
-0.249	0.191	-0.084	0.936	0.046	-0.952	-0.840	-0.380	-2.274	0.693	-3.357
-5.817	6.812	4.351	-1.474	-1.141	-3.780	-3.801	-1.725	-4.003	0.682	-4.313
1.564	0.252	-5.357	0.846	-0.478	1.467	1.522	2.092	1.586	-0.540	3.085
-3.991	5.278	-4.214	2.945	1.761	-1.475	-2.880	3.323	-3.216	2.287	-4.399
-1.456	-4.906	-4.162	0.168	0.208	0.028	1.592	1.284	2.416	-0.713	2.728
-0.759	-1.069	1.777	1.842	0.707	-0.390	-0.550	0.899	-2.056	1.186	-3.425
0.255	-2.392	-4.115	0.931	0.462	1.137	3.687	1.679	2.197	0.390	3.597
-1.734	5.336	-0.413	-0.102	-0.455	-1.925	-5.641	-2.713	-5.825	0.036	-7.407
4.214	0.518	4.101	-0.375	-0.253	4.035	5.018	0.477	3.636	0.817	4.900
2.919	-8.340	4.672	-0.873	-0.179	3.222	2.477	-0.635	2.776	-1.179	3.531
-5.771	-1.875	5.604	-0.977	-0.750	-3.333	-0.842	-2.243	-1.117	0.912	-0.040
4.554	0.375	-0.285	-2.177	-2.248	5.011	0.464	-2.694	1.350	-2.564	2.314

Appendix B. Weight connections between the hidden and output layer.

Column 1-11

-0.9385	-0.9199	-1.0166	-0.9694	-0.6889	-0.4161	-0.2376	-0.0393	-1.2504	-1.3357	-1.4287
-0.0734	-0.1532	-0.1777	-0.3280	-0.5703	-0.6524	-0.5800	-0.2669	-0.5874	-0.5987	-0.4767
-0.3608	-0.4809	-0.3636	-0.2786	-0.2114	-0.1840	-0.2137	-0.2978	-0.3266	-0.5652	-0.5480
-0.3602	-0.9968	-1.3364	-1.4291	-1.3212	-0.8383	-0.2807	0.5740	0.0100	-0.9370	-1.6255
-0.9874	-1.7524	-2.0886	-1.5704	-0.5851	0.6368	1.4817	2.0793	-0.9465	-1.3913	-1.4164
-0.2426	-0.5340	-0.6572	-0.1943	0.5258	1.0792	1.0269	0.7578	0.3280	0.0716	-0.0529
3.5135	3.8287	3.7189	2.9813	1.9707	1.0484	0.6900	0.6619	3.1211	3.3864	3.2657
-0.1610	0.0241	0.3948	0.5355	0.4074	-0.3113	-1.3198	-2.4973	-0.5860	-0.4235	-0.0687

0.7981	-0.1264	-1.3211	-2.5885	-3.3917	-3.5337	-2.8794	-1.9196	0.8086	0.1816	-0.7536
-1.5546	-1.6010	-1.7276	-1.9581	-2.2573	-2.7645	-3.3655	-4.1239	-1.0419	-0.9448	-0.9349
-3.5623	-2.6542	-1.5299	-0.6754	-0.4162	-0.4064	-0.6599	-0.8710	-3.1295	-2.4382	-1.4527
0.0901	-0.0191	-0.1166	-0.1723	-0.1486	-0.0867	0.0042	0.1495	-0.0296	-0.0978	-0.1413
-0.0631	1.1885	1.6540	1.9588	2.0211	1.4979	0.6624	-0.2370	-0.0017	1.1088	1.3209
-1.3489	-0.9683	-0.3864	0.1880	0.5605	1.0170	1.3922	1.7119	-1.4949	-1.1243	-0.5659
0.0560	0.0965	-0.0229	-0.3559	-0.7583	-0.9429	-0.7903	-0.6504	-0.0634	0.0313	-0.0726
-0.0465	-0.0723	-0.0989	-0.1161	-0.1378	-0.1077	-0.0637	-0.0340	-0.1088	-0.1369	-0.1422

Column 12-22

-1.2276	-0.8143	-0.3849	-0.0699	0.2022	-1.3349	-1.4787	-1.5520	-1.3060	-0.7946	-0.2893
-0.4940	-0.6516	-0.7002	-0.5570	-0.2176	-0.9088	-0.8547	-0.6003	-0.5259	-0.6001	-0.5904
-0.4554	-0.3005	-0.1740	-0.1025	-0.1411	-0.1868	-0.5378	-0.6605	-0.6147	-0.4043	-0.1870
-2.1090	-2.2269	-1.7077	-0.9070	0.3511	0.4447	-0.6909	-1.8063	-2.7387	-3.0931	-2.4636
-0.7299	0.2359	1.2116	1.7075	1.7495	-0.8283	-0.9475	-0.6445	0.1741	1.0348	1.6017
0.4613	1.2447	1.7892	1.7158	1.3314	0.5838	0.3019	0.2862	0.8491	1.7349	2.2997
2.4666	1.3256	0.3990	0.1253	0.2241	3.0046	3.1927	2.9739	2.0260	0.8993	-0.0635
0.2153	0.3004	-0.2503	-1.1460	-2.2720	-1.0371	-0.9862	-0.6535	-0.1589	0.1539	-0.1812
-1.7395	-2.3142	-2.3012	-1.6896	-0.9335	0.3482	-0.0483	-0.5335	-1.0017	-1.2276	-1.0200
-1.1975	-1.5809	-2.3536	-3.1734	-4.1485	-0.5819	-0.3645	-0.2039	-0.4061	-0.8288	-1.8263
-0.7380	-0.5615	-0.5931	-0.7639	-0.7666	-2.2818	-1.7564	-1.0691	-0.7053	-0.6784	-0.7397
-0.1607	-0.1495	-0.1104	-0.0764	0.0137	-0.1230	-0.1275	-0.1222	-0.1014	-0.0949	-0.1091
1.3057	1.0218	0.4268	-0.4209	-1.1075	-0.2664	0.6343	0.6159	0.3068	-0.1788	-0.8821
-0.0071	0.3808	0.8455	1.2507	1.5595	-1.4890	-1.1440	-0.6826	-0.1310	0.2647	0.7313
-0.3791	-0.7538	-0.9844	-0.8908	-0.8279	-0.3085	-0.2326	-0.2807	-0.5689	-0.8875	-1.0865
-0.1541	-0.1493	-0.1408	-0.0954	-0.0651	-0.1453	-0.1823	-0.1671	-0.1674	-0.1489	-0.1418

Column 23-33

0.0439	0.2918	-1.1982	-1.2879	-1.3694	-1.0985	-0.5689	-0.1653	0.0478	0.1444	-0.8940
-0.4779	-0.1520	-1.0759	-0.9215	-0.6300	-0.5010	-0.5851	-0.6006	-0.4936	-0.1941	-1.0090
-0.0232	-0.0364	-0.1165	-0.5372	-0.7543	-0.7338	-0.4710	-0.2053	-0.0309	-0.0538	-0.1459
-1.2823	0.2941	0.6390	-0.4449	-1.8431	-2.9718	-3.4640	-2.6870	-1.2424	0.6435	0.6201
1.6132	1.2372	-0.5337	-0.4033	0.0246	0.6426	1.0690	1.1428	0.7762	0.1638	-0.1348
2.1644	1.7992	0.6613	0.5972	0.5798	1.1533	1.7893	2.2872	2.2661	2.0839	0.7119
-0.2465	-0.1336	2.8606	2.9288	2.7040	1.7399	0.6226	-0.2209	-0.4193	-0.3667	2.7974
-0.9498	-1.9622	-1.3491	-1.4508	-1.1639	-0.6301	-0.1670	-0.2406	-0.7898	-1.6663	-1.3883

-0.5296	0.0668	-0.1865	-0.4643	-0.5326	-0.5407	-0.3938	0.0224	0.4675	0.8298	-0.6125
-2.9013	-4.0510	-0.2885	0.1140	0.5461	0.5076	0.0865	-1.0251	-2.2670	-3.6330	-0.2652
-0.7497	-0.6836	-1.1868	-0.8118	-0.6111	-0.6284	-0.8041	-0.9467	-0.8455	-0.6049	-0.2642
-0.1343	-0.0937	-0.1345	-0.1454	-0.0901	-0.0524	-0.0463	-0.0930	-0.1666	-0.1664	-0.0869
-1.5471	-1.9888	-1.0937	-0.3757	-0.4055	-0.6961	-1.2005	-1.7402	-2.1742	-2.2369	-2.1563
1.1628	1.4239	-1.4672	-1.2052	-0.7123	-0.1253	0.3531	0.8827	1.2238	1.3725	-1.4987
-1.0430	-1.0032	-0.5799	-0.5861	-0.5644	-0.8124	-1.1188	-1.3070	-1.3179	-1.3585	-0.7207
-0.1170	-0.0916	-0.1493	-0.1733	-0.1607	-0.1563	-0.1304	-0.1333	-0.1219	-0.1041	-0.1201

Column 34-44

-0.8908	-0.9103	-0.7107	-0.3098	-0.0674	-0.0968	-0.2281	-0.6442	-0.4996	-0.5037	-0.3574
-0.8126	-0.5500	-0.4771	-0.6531	-0.7054	-0.6247	-0.3083	-1.0524	-0.7573	-0.5130	-0.5172
-0.5602	-0.7749	-0.7458	-0.4726	-0.2207	-0.1108	-0.2052	-0.3184	-0.6430	-0.8008	-0.6818
-0.2859	-1.5737	-2.6035	-3.0238	-2.3181	-0.7872	1.1695	0.6706	0.0374	-0.9072	-1.8056
-0.0508	0.1549	0.3785	0.3280	0.0613	-0.3985	-0.9482	0.0812	-0.0532	-0.1614	-0.5407
0.8473	0.7771	1.1823	1.6169	2.0479	2.1345	2.1243	0.6273	0.8609	1.0764	1.3527
2.7010	2.4261	1.5969	0.6524	-0.0898	-0.3138	-0.3705	2.6819	2.4711	2.0456	1.2942
-1.6132	-1.4930	-1.0491	-0.4724	-0.3532	-0.7055	-1.4313	-1.0379	-1.4527	-1.5743	-1.2436
-0.8588	-0.7566	-0.5082	-0.1536	0.4276	0.9130	1.1943	-0.9439	-1.2211	-1.1188	-0.7636
0.2994	0.8617	1.0474	0.7340	-0.2766	-1.5870	-3.0472	-0.4016	0.3242	1.0801	1.4995
-0.0987	-0.2454	-0.6141	-0.9719	-1.1529	-1.0354	-0.7186	0.3475	0.3346	-0.0301	-0.5768
-0.1255	-0.0757	-0.0502	-0.0453	-0.0938	-0.1691	-0.1635	0.0029	-0.0858	-0.1035	-0.0967
-1.4697	-1.3162	-1.2909	-1.4550	-1.7374	-1.9379	-1.8276	-3.2429	-2.6351	-2.2194	-1.7771
-1.2750	-0.7458	-0.0347	0.5946	1.1252	1.4060	1.4100	-1.3474	-1.1180	-0.6742	0.0999
-0.7909	-0.7564	-0.9578	-1.2604	-1.4684	-1.5237	-1.6278	-0.5948	-0.6772	-0.6029	-0.7781
-0.1438	-0.1642	-0.1629	-0.1463	-0.1276	-0.1222	-0.1103	-0.0949	-0.1123	-0.1482	-0.1558

Column 45-55

-0.1339	-0.1036	-0.3417	-0.6331	-0.5318	-0.3106	-0.2696	-0.2051	-0.1019	-0.2399	-0.5595
-0.7455	-0.8283	-0.7415	-0.4099	-1.1046	-0.8048	-0.5731	-0.6148	-0.8622	-0.9332	-0.7973
-0.3769	-0.1720	-0.1621	-0.3500	-0.5261	-0.7565	-0.7808	-0.5647	-0.2305	-0.0690	-0.1739
-2.1335	-1.5214	-0.1724	1.5856	0.7283	0.4150	-0.1287	-0.7670	-1.1873	-0.7940	0.2690
-1.0683	-1.6517	-2.0892	-2.3963	0.0820	-0.3717	-0.8347	-1.6459	-2.4380	-3.1676	-3.4906
1.6001	1.8789	2.0980	2.3790	0.6146	0.9766	1.3396	1.6206	1.7260	1.9929	2.3303
0.5420	-0.0963	-0.3990	-0.5500	2.5687	2.1936	1.7015	0.9444	0.3540	-0.2592	-0.5744
-0.7312	-0.5464	-0.7185	-1.3061	-0.5488	-1.0843	-1.3699	-1.2419	-0.8262	-0.6660	-0.7874
-0.2885	0.3381	0.8481	1.1654	-0.9685	-1.3291	-1.4169	-1.1919	-0.7326	-0.0924	0.3777
1.3841	0.5214	-0.7069	-2.1412	-0.5324	0.2036	1.0317	1.6576	1.7333	1.1019	0.0254
-1.0746	-1.2834	-1.1223	-0.7814	0.3519	0.3095	-0.0744	-0.6277	-1.1288	-1.3043	-1.0863
-0.0735	-0.0795	-0.1022	-0.0788	0.0884	-0.0737	-0.1697	-0.1875	-0.1191	-0.0563	-0.0235
-1.4785	-1.3234	-1.2966	-1.0318	-3.9278	-3.3541	-2.7548	-1.9623	-1.2579	-0.7279	-0.4873
0.8091	1.3296	1.4657	1.2958	-1.1209	-0.9456	-0.5127	0.2441	0.9681	1.4002	1.3761
-1.1024	-1.3911	-1.5499	-1.7074	-0.2037	-0.2365	-0.1682	-0.3676	-0.7078	-1.0701	-1.3240
-0.1429	-0.1164	-0.1069	-0.0928	-0.0744	-0.0876	-0.1359	-0.1493	-0.1454	-0.1130	-0.0749

Column 56-64

-0.8906	-0.5783	-0.3174	-0.2362	-0.1865	-0.1605	-0.3614	-0.6768	-0.9258
-0.4594	-1.2996	-1.0138	-0.7602	-0.7745	-0.9832	-1.0167	-0.8372	-0.4521
-0.4022	-0.6850	-0.8023	-0.7299	-0.4220	-0.0778	0.0462	-0.0761	-0.3575
1.6667	0.9962	1.0087	0.7291	0.2209	-0.4164	-0.3257	0.3926	1.3973
-3.5328	-0.3653	-1.0883	-1.8228	-2.9415	-3.8655	-4.5578	-4.7686	-4.6116
2.7771	0.5020	1.0357	1.5047	1.8703	1.9886	2.3406	2.7352	3.2586
-0.8687	2.3846	1.9181	1.3532	0.5721	0.0443	-0.5629	-0.9652	-1.3023
-1.2682	0.0130	-0.5556	-1.0154	-1.0932	-0.8138	-0.7260	-0.8733	-1.3075
0.7156	-0.8550	-1.2975	-1.5572	-1.5051	-1.1120	-0.5843	-0.1891	0.1789
-1.3402	-0.5818	0.1131	0.9734	1.6737	1.9702	1.5623	0.7074	-0.5512
-0.7236	0.1215	0.0819	-0.2128	-0.6592	-1.0574	-1.1649	-0.8906	-0.5271
0.0328	0.1347	-0.0822	-0.2335	-0.2661	-0.1468	-0.0260	0.0518	0.1511
-0.1908	-4.3756	-3.7496	-3.0650	-1.9843	-0.9936	-0.1713	0.1549	0.4612
1.0490	-0.7191	-0.5897	-0.2339	0.3814	1.0011	1.2716	1.1415	0.6809
-1.5929	0.3269	0.2996	0.3710	0.1667	-0.1810	-0.5879	-0.9297	-1.2675
-0.0618	-0.0638	-0.0891	-0.1193	-0.1411	-0.1258	-0.0855	-0.0281	-0.0086

Appendix C. Internal Data Representation of neurons in the network for Lena (from MATLAB workspace)

input	input of hidden layer	output of hidden layer (Compressed Data)	input of output layer	output of output layer (Decompressed Data)
0.2148	-0.9111	0.2868	-1.3164	0.2114
0.1992	-0.6798	0.3363	-1.3281	0.2095
0.2305	-1.9084	0.1292	-1.2814	0.2173
0.2734	-0.8045	0.3091	-1.1444	0.2415
0.2734	-0.8144	0.3070	-0.9825	0.2724
0.2773	1.2181	0.7717	-0.8036	0.3093
0.3008	0.2555	0.5635	-0.6849	0.3352
0.3359	-0.7776	0.3148	-0.6113	0.3518
0.2227	-0.7655	0.3175	-1.2625	0.2205
0.2383	-0.6390	0.3455	-1.2071	0.2302
0.2734	-0.6450	0.3441	-1.1138	0.2472
0.2969	1.5426	0.8238	-0.9416	0.2806
0.3203	-1.0909	0.2514	-0.7672	0.3171
0.3203	0.6253	0.6514	-0.6288	0.3478
0.3516	-0.2934	0.4272	-0.5803	0.3589
0.3633	1.4937	0.8166	-0.5725	0.3607
0.2188			-1.2697	0.2193
0.2539			-1.1504	0.2404
0.2891			-0.9799	0.2729
0.3008			-0.7535	0.3201
0.3750			-0.5826	0.3583
0.3516			-0.4726	0.3840
0.3438			-0.4904	0.3798
0.3711			-0.5254	0.3716
0.2617			-1.2123	0.2293
0.3047			-1.0634	0.2567
0.3242			-0.8497	0.2995
0.3438			-0.6067	0.3528
0.3516			-0.4515	0.3890
0.3633			-0.3888	0.4040
0.3672			-0.4333	0.3933
0.3477			-0.4986	0.3779
0.2891			-1.1188	0.2462
0.3125			-0.9656	0.2758
0.3438			-0.7645	0.3177

0.3633			-0.5612	0.3633
0.3906			-0.4410	0.3915
0.3984			-0.3955	0.4024
0.3789			-0.4349	0.3930
0.3750			-0.4773	0.3829
0.3086			-0.9394	0.2810
0.3086			-0.8101	0.3079
0.3594			-0.6300	0.3475
0.3750			-0.4773	0.3829
0.3867			-0.4117	0.3985
0.4063			-0.3807	0.4060
0.3906			-0.3847	0.4050
0.3828			-0.3972	0.4020
0.3672			-0.7460	0.3217
0.3516			-0.6370	0.3459
0.3789			-0.5160	0.3738
0.4063			-0.4107	0.3988
0.3906			-0.3807	0.4060
0.4023			-0.3416	0.4154
0.4102			-0.3177	0.4212
0.4219			-0.3030	0.4248
0.3672			-0.5436	0.3673
0.3516			-0.4705	0.3845
0.3789			-0.3877	0.4043
0.4063			-0.3297	0.4183
0.3906			-0.3067	0.4239
0.4023			-0.2642	0.4343
0.4102			-0.2195	0.4453
0.4219			-0.1775	0.4557

Appendix D. Internal Data Representation of neurons in the network for cameraman (from MATLAB workspace)

input	input of hidden layer	output of hidden layer (Compressed Data)	input of output layer	output of output layer (Decompressed Data)
0.4609	-1.6095	0.1667	-0.1021	0.4745
0.4414	0.0156	0.5039	-0.1860	0.4536
0.4531	-0.3527	0.4127	-0.1608	0.4599
0.4844	-0.8046	0.3091	-0.0641	0.4840

0.5000	-0.8691	0.2954	0.0411	0.5103
0.4922	1.6234	0.8353	0.0945	0.5236
0.4805	0.8544	0.7015	0.0447	0.5112
0.4766	-0.5946	0.3556	-0.0436	0.4891
0.4609	-1.5328	0.1776	-0.2209	0.4450
0.4414	-0.9285	0.2832	-0.2748	0.4317
0.4531	-1.7751	0.1449	-0.2327	0.4421
0.4805	1.0928	0.7489	-0.1120	0.4720
0.4922	-1.0623	0.2569	0.0129	0.5032
0.4844	0.4021	0.5992	0.0554	0.5138
0.4727	-0.8844	0.2923	-0.0074	0.4981
0.4688	8.6905	0.9998	-0.1176	0.4706
0.4375			-0.3531	0.4126
0.4180			-0.3881	0.4042
0.4297			-0.3164	0.4216
0.4609			-0.1660	0.4586
0.4727			-0.0305	0.4924
0.4648			0.0112	0.5028
0.4570			-0.0719	0.4820
0.4531			-0.1752	0.4563
0.3984			-0.4818	0.3818
0.3867			-0.4892	0.3801
0.3984			-0.3919	0.4033
0.4297			-0.2296	0.4428
0.4453			-0.1188	0.4703
0.4375			-0.0833	0.4792
0.4258			-0.1491	0.4628
0.4258			-0.2283	0.4432
0.3750			-0.5426	0.3676
0.3672			-0.5227	0.3722
0.3828			-0.4351	0.3929
0.4141			-0.2928	0.4273
0.4375			-0.2005	0.4501
0.4336			-0.1649	0.4589
0.4219			-0.2019	0.4497
0.4180			-0.2295	0.4429
0.3750			-0.5588	0.3638
0.3711			-0.5170	0.3735
0.3945			-0.4139	0.3980

0.4258			-0.2971	0.4263
0.4492			-0.2224	0.4446
0.4492			-0.1814	0.4548
0.4414			-0.1783	0.4555
0.4453			-0.1676	0.4582
0.3867			-0.5034	0.3767
0.3867			-0.4601	0.3870
0.4063			-0.3563	0.4119
0.4453			-0.2461	0.4388
0.4688			-0.1855	0.4538
0.4688			-0.1336	0.4666
0.4648			-0.1024	0.4744
0.4648			-0.0687	0.4828
0.3867			-0.4411	0.3915
0.3867			-0.3914	0.4034
0.4102			-0.2910	0.4278
0.4453			-0.1911	0.4524
0.4727			-0.1322	0.4670
0.4727			-0.0693	0.4827
0.4688			-0.0226	0.4943
0.4688			0.0232	0.5058

Appendix E. Internal Data Representation of neurons in the network for peppers (from MATLAB workspace)

input	input of hidden layer	output of hidden layer (Compressed Data)	input of output layer	output of output layer (Decompressed Data)
0.6328	1.7496	0.8519	0.3462	0.5857
0.6953	0.0415	0.5104	0.1081	0.5270
0.5742	-1.3833	0.2005	-0.1906	0.4525
0.7422	0.0422	0.5105	-0.2259	0.4438
0.8203	-0.9409	0.2807	0.0139	0.5035
0.6992	3.9976	0.9820	0.3577	0.5885
0.5742	2.7172	0.9380	0.6002	0.6457
0.7734	-2.0719	0.1119	0.9433	0.7198
0.7852	-1.7329	0.1502	0.2397	0.5596
0.5898	-0.7302	0.3251	-0.0926	0.4769
0.2383	-2.9310	0.0506	-0.4342	0.3931
0.6484	-0.9695	0.2750	-0.4750	0.3834

0.7344	-1.1892	0.2334	-0.2562	0.4363
0.6992	0.2678	0.5666	0.1423	0.5355
0.6914	-2.7305	0.0612	0.5119	0.6252
0.8008	9.3035	0.9999	0.9804	0.7272
0.8164			0.2824	0.5701
0.6367			-0.0900	0.4775
0.5469			-0.4559	0.3880
0.5117			-0.5922	0.3561
0.7852			-0.3934	0.4029
0.7344			0.0161	0.5040
0.7578			0.4352	0.6071
0.9141			0.9524	0.7216
0.8125			0.3302	0.5818
0.7070			0.1331	0.5332
0.6875			-0.2367	0.4411
0.6953			-0.3755	0.4072
0.6484			-0.2884	0.4284
0.7422			0.0306	0.5076
0.7422			0.4175	0.6029
0.9570			0.9156	0.7142
0.7773			0.4385	0.6079
0.6328			0.4437	0.6091
0.7344			0.1802	0.5449
0.7422			0.0444	0.5111
0.6875			0.0300	0.5075
0.7266			0.2017	0.5502
0.7695			0.4473	0.6100
0.9766			0.8361	0.6976
0.5898			0.4062	0.6002
0.6367			0.6563	0.6584
0.7930			0.5938	0.6442
0.8008			0.4534	0.6114
0.7539			0.3531	0.5874
0.7656			0.3630	0.5898
0.7578			0.4438	0.6092
0.9570			0.7280	0.6744
0.6992			0.3188	0.5790
0.6836			0.7128	0.6710
0.7930			0.8299	0.6963

0.8086			0.7359	0.6761
0.7656			0.5401	0.6318
0.7891			0.4076	0.6005
0.7422			0.4262	0.6050
0.9492			0.6175	0.6496
0.6641			0.1147	0.5286
0.7383			0.6247	0.6513
0.7695			0.8672	0.7042
0.7930			0.8248	0.6953
0.7734			0.5563	0.6356
0.7930			0.3573	0.5884
0.7461			0.3275	0.5812
0.9063			0.5097	0.6247

Appendix F. Image Compression GUI Source code

```
function varargout = unanggui(varargin)
% UNANGGUI MATLAB code for unanggui.fig
%      UNANGGUI, by itself, creates a new UNANGGUI or raises the existing
%      singleton*.
%
%      H = UNANGGUI returns the handle to a new UNANGGUI or the handle to
%      the existing singleton*.
%
%      UNANGGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in UNANGGUI.M with the given input arguments.
%
%      UNANGGUI('Property','Value',...) creates a new UNANGGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before unanggui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to unanggui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help unanggui

% Last Modified by GUIDE v2.5 04-Oct-2014 15:16:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @unanggui_OpeningFcn, ...
                   'gui_OutputFcn',  @unanggui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


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

% Choose default command line output for unanggui


% Update handles structure
guidata(hObject, handles);

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


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

% Get default command line output from handles structure
%varargout{1} = handles.output;
```

```
% --- Executes on button press in browse.
function browse_Callback(hObject, eventdata, handles)
% hObject    handle to browse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%[Filename] = uigetfile({'*.bmp';'*.jpg';'*.png';'*.*'},'Select Picture');
%axes(handles.originalimage);
%imshow(Filename);

[Filename] = uigetfile({'*.bmp';'*.gif';'*.*';},'Select Picture');
axes(handles.originalimage);
imshow(Filename);

y = getimage(axes.originalimage);
info = imfinfo(y, bmp)
set(handles.fsorig, 'String', [num2str(round(info.FileSize/1024)) ' kB'])
% --- Executes on button press in compress.

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



  I=64;                    % Number of neurons in Input and output layer
  H=16;                    % Number of neurons in hidden layer
  Sqrt_H=fix(sqrt(H));        % Input and Output Block size:  Sqrt_H* Sqrt_H
  Sqrt_I=fix(sqrt(I));       % Compressed Block size:  Sqrt_I* Sqrt_I

  Time = clock;       % Start execution clock


  InputImage = getimage(handles.originalimage);
  [M,N]=size(InputImage);
  No=M*N/I;                      % Number of input blocks

  [v,w,v_b,w_b]=WeightsRead(I,H);     % Return weights of network 64_16_64

  Mrg=zeros(M/2,N/2);                % Compressed image
  Dmrg=zeros(M,N);                   % Decompressed image

  set(handles.compresstext,'String','Start Compressing/Decompressing . . . ');
  g = 'Complete';
  set(handles.compresstext,'String',g);
```

```
for u=1:No
   [x]=PatternNext(InputImage,I,u);     % Read next block of image(x vector) that the block size=64
   x=double(x)/256;               % Normalize to [0,1]

   h_in=(x'*v)'+v_b;              % Input of hidden layer
   h=f1(h_in);                    % Output of hidden layer
   % h is the compressed data

   k=1;
   for i=1:  Sqrt_H
      for j=1: Sqrt_H
         tmp(i,j)=h(k);
         k=k+1;
      end
   end

   Mrg=PatternAdd(Mrg,tmp,M/2,N/2,u);

   y_in=w_b+(h'*w)';                 % Input of output layer
   y=f1(y_in);                    % Output of output layer
   % y is decompressed data
   k=1;
   for i=1: Sqrt_I
      for j=1: Sqrt_I
         tmp(i,j)=y(k);
         k=k+1;
      end
   end
   Dmrg=PatternAdd(Dmrg,tmp,M,N,u);
   clear tmp;
end

Dmrg=Dmrg*256;
Mrg=Mrg*256;

% **************************************************
% ****             Outputs                     ***
% **************************************************
% Compute the PSNR
Psnr=PSNR(InputImage,Dmrg);
snr=SNR(InputImage,Dmrg);
nmse=NMSE(InputImage,Dmrg);


% Computing Bit rate
```

```
    % BitRate(BlockSize,NoOfBlocks,NoOfHiddenNeroun,NoOfBitsOut,NoOfBitsWeight)
    NoOfBitsOut=8;
    NoOfBitsWeight=8;
    Bitrate=BitRate(I,No,H,NoOfBitsOut,NoOfBitsWeight);


    a = ['PSNR= ',num2str(Psnr),'dB' ];
    set(handles.psnr,'String',a);
    b =['SNR= ',num2str(snr),' dB'];
    set(handles.snr,'String',b);
    c =['NMSE= ',num2str(nmse)];
    set(handles.nmse,'String',c);
    d=['Bit rate(all)= ',num2str(Bitrate),' bpp'];
    set(handles.bitrateall,'String',d);
    e =['Bit rate(CR)= ',num2str(H/I),' bpp'];
    set(handles.bitratecr,'String',e);


    axes(handles.originalimage)
    imshow(InputImage)
    axes(handles.compressed)
    imshow(uint8(fix(Mrg)))
    axes(handles.decompressed)
    imshow(uint8(fix(Dmrg)))


%     figure, imshow(uint8(fix(Dmrg)))
%    subplot(1,2,2),imshow(uint8(fix(Dmrg)))



    Ttime= etime(clock,Time);           % All time in Sec.
    Thour=fix(Ttime/3600);
    Tmp=round(rem(Ttime,3600));
    Tmin=fix(Tmp/60);
    Tsec=round(rem(Tmp,60));
      % elapsed time
    f=['Time: ',int2str(Thour),':',int2str(Tmin),''':',int2str(Tsec),''''''];
    set(handles.time,'String',f);




% [In,n,h,m,nop,err,file_I2H,file_I2H_B,file_H2O,file_H2O_B]=ParamRead
% Read input parametere from a file
% return In=image matrix and
%       n=number of input nerons,  h=number of hidden nerons
%       m=number of output nerons, nop=number of input training patterns
%       err= error tolerance for terminatig network training
%       file...= files that contain weights of network


function [v,w,v_b,w_b]=WeightsRead(I,H)

    file_I2H='I2H.wgt';        % Input2Hidden weight
    file_H2O='H2O.wgt';        % Hidden2Output weight
    file_I2H_B='I2H_B.wgt';    % Input2Hidden bias weight vector
    file_H2O_B='H2O_B.wgt';


    v=ReadFromFile(file_I2H,I,H);
    w=ReadFromFile(file_H2O,H,I);

    fid = fopen(file_I2H_B,'r+');
    v_b=fscanf(fid,'%f');
    fclose(fid);

    fid = fopen(file_H2O_B,'r+');
    w_b=fscanf(fid,'%f');
    fclose(fid);

% Res=PatternAdd(Mrg,Add,M,N,I):
% Merg the blocks of image and return the result
% adding the Ith block Add to Merg, M*N is the size of final image

function Res=PatternAdd(Mrg,Add,M,N,I)

     t=zeros(M,N);
    [M2,N2]=size(Add);

    in=fix((I-1)*N2/N)*M2;
    jn=(mod((I-1)*N2,N));

    for i=in+1:in+N2
        for j=jn+1:jn+N2
            Mrg(i,j)=Add(i-in,j-jn);
        end
    end
    Res=Mrg;

function val=BitRate(BlockSize,NoOfBlocks,NoOfHiddenNeroun,NoOfBitsOut,NoOfBitsWeight)
```

```
% val=BitRate(BlockSize,NoOfBlocks,NoOfHiddenNeroun,NoOfBitsOut,NoOfBitsWeight)
% Return the value val of bit rate that is necessary for coding outputs of hidden
% layer and weights from hidden to output layer,
% Parameters:
% BlockSize= size of each block in pixel, i.e. 8*8=(64)
% NoOfBlocks= number of blocks of image=(all pixels of image)/BlockSize,
% NoOfHiddenNeroun= number of neurons in hidden layer i.e. (16),
% NoOfBitsOut= number of bits that require for coding hidden layer outputs i.e.integer(16)
% NoOfBitsWeight= number of bits that require for coding hidden to output layer weights i.e.float(32)

val=
((NoOfBlocks*NoOfHiddenNeroun*NoOfBitsOut)+(BlockSize*NoOfHiddenNeroun*NoOfBitsWeight))/(Block
Size*NoOfBlocks);

% o=f1(x):
% Active function for Hidden layer nerons (sigmoid)

function o=f1(x)
   tmp1=1+exp(x.*-1);
   o=(tmp1.^(-1));


% o=f1_p(x):
% Derivative of Active function for Hidden layer nerons


function o=f1_p(x)
   o=f1(x).*(1-f1(x));


% [x]=PatternNext(Input,N,No):
% [x]=PatternNext(Input,N,No)
% Read next ith block of image Input by each block size=N
% return x=input vector = t=target vector
%
% Assumed that Image(Input) is power of 2 in both dimentions.

function [x]=PatternNext(Input,N,No)
   tmp=sqrt(N);
   [i,j]=size(Input);
   i=i/tmp;
   j=j/tmp;
   in=fix((No-1)/i)*tmp;
   jn=(mod(No-1,i))*tmp;
   k=1;
   for i=in+1:in+tmp
```

```
        for j=jn+1:jn+tmp
            %x(i-in,j-jn)=Input(i,j);
            x(k,1)=Input(i,j);  %(i-1)*tmp+j
            k=k+1;
        end
    end

% Input=NextTrainPattern(P_Counter);
% Return the next training pattern

function Input=NextTrainPattern(P_Counter)

    tmp = mod(P_Counter,22);
    file=['TrainingSet\train' ,int2str(tmp),'lena.bmp'];
    Input=imread('lena.bmp');

% val=NMSE(Im,Im_hat):
% Return the value val of Normalized-Mean-Square-Error (NMSE)
% for image Im and it's compressed vertion Im_hat.

function val=NMSE(Im,Im_hat)

[M,N]=size(Im);
tmp1=0;
tmp2=0;
for i=1:M
    for j=1:N
        tmp1=tmp1+(double(Im(i,j))-double(Im_hat(i,j)))^2;
        tmp2=tmp2+double(Im(i,j))^2;
    end
end
val=tmp1/tmp2;

% val=PSNR(Im,Im_hat):
% Return the value val of Peak-Signal-to-Noise-Ratio(PSNR) in dB for
% the image Im and it's compressed version Im_hat,

function val=PSNR(Im,Im_hat)

[M,N]=size(Im);
tmp=0;
for i=1:M
    for j=1:N
        tmp=tmp+(double(Im(i,j))-double(Im_hat(i,j)))^2;
    end
end
```

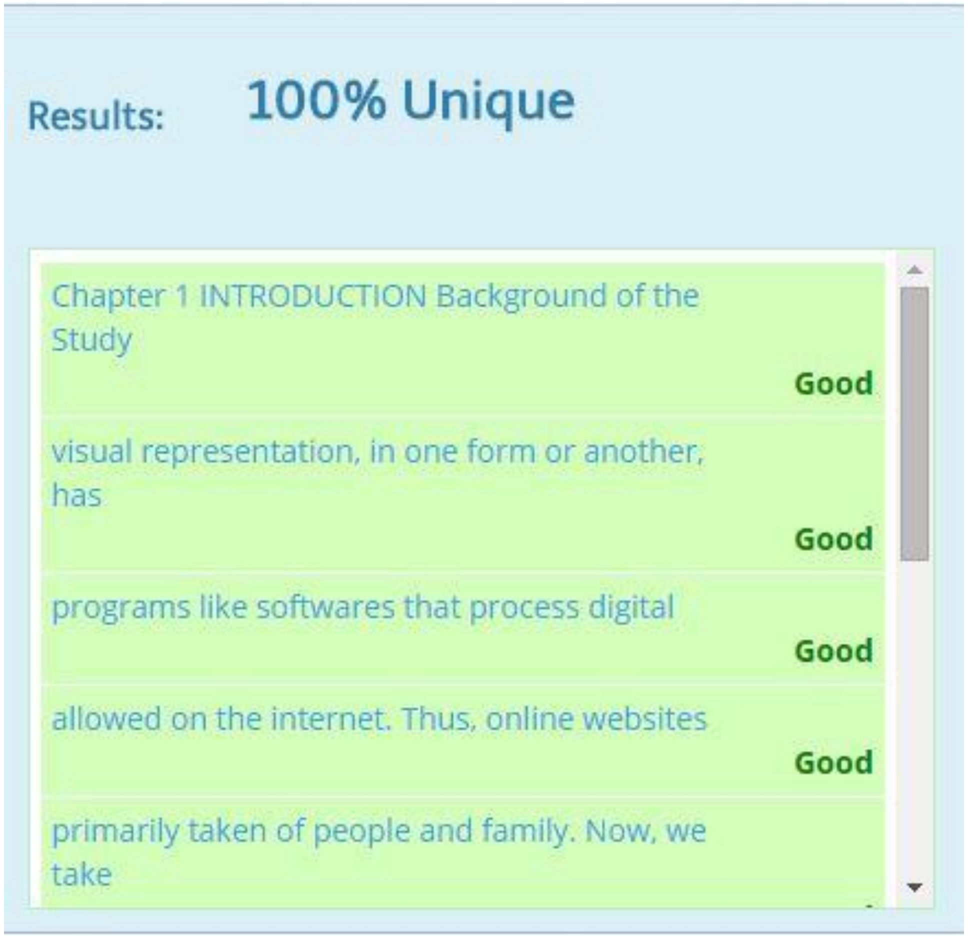
```
tmp=tmp/(M*N);
val=10*log10(256^2/tmp);

% [Out]=ReadFromFile(FileName,Rows,Cols):
% Read Rows*Cols data from FileName and return in Out,


function [Out]=ReadFromFile(FileName,Rows,Cols)
    fid = fopen(FileName,'r+');
    Out=fscanf(fid,'%f');
    fclose(fid);

    tmp=zeros(Rows,Cols);
    k=1;
    for i=1:Rows
        for j=1:Cols
            tmp(i,j)=Out(k);
            k=k+1;
        end
    end
    Out=tmp;

% Save_res(Vec,Fname):
% Save the trained weights:
% Vec=  Data
% Fname=Output file name

function Save_res(Vec,Fname)

    No=length(Vec);
    if No~=length(Fname)
        disp('Error! Number of data and output file name should be same.');
        return;
    end

    for i=1:No
        Data=Vec(i);
        file=Fname(i);
        file=[file,'.wgt'];
        fid = fopen(file,'w');
        if fid==-1
            disp('Error! cannot create the output file');
            return;
        end
        [M,N]=size(Data);
        %fprintf(fid,'The input to hidden layer:\n\n    ');
        for i=1:M
            for j=1:N
                fprintf(fid,'%6.2f    ',Data(i,j));
            end
            fprintf(fid,'\n');
        end

        fclose(fid);

    end %for


% Save_w(v,w,Name1,Name2)
% Save the trained weights:
% v=encoder weights(input to hidden layer)
% w=decoder weights(hidden to output layer)

function Save_w(v,w,Name1,Name2)
%    file=input('Output file name for encoding weights:');
%    save file 'v' -ASCII;
    file=Name1;
    file=[file,'.wgt'];
    fid = fopen(file,'w');
    if fid==-1
        disp('Error! cannot create the output file');
        return;
    end
    [M,N]=size(v);
    %str=['The input to hidden layer [',int2str(M),'*',int2str(N),'] :\n\n'];
    %fprintf(fid,str);
    for i=1:M
        for j=1:N
            fprintf(fid,'%10.6f    ',v(i,j));
        end
        fprintf(fid,'\n');
    end
    fclose(fid);

    %*******************************************************************
    file=Name2;
    file=[file,'.wgt'];
    fid = fopen(file,'w');

    if fid==-1
        disp('Error! cannot create the output file');
        return;
    end

    [M,N]=size(w);
    %str=['The hidden to output layer [',int2str(M),'*',int2str(N),'] :\n\n'];
    %fprintf(fid,str);
    for i=1:M
        for j=1:N
            fprintf(fid,'%10.6f    ',w(i,j));
        end
        fprintf(fid,'\n');
    end
    fclose(fid);



% Save_w(v,w,Name1,Name2)
% Save the trained weights:
% v=encoder weights(input to hidden layer)
% w=decoder weights(hidden to output layer)
%
function Save_w_Ver2(v,w,Name1,Name2)
%    file=input('Output file name for encoding weights:');
%    save file 'v' -ASCII;
    file=Name1;
    file=[file,'.wgt'];
    fid = fopen(file,'w');
    if fid==-1
        disp('Error! cannot create the output file');
        return;
    end
    [M,N,P]=size(v);
    %str=['The input to hidden layer [',int2str(M),'*',int2str(N),'] :\n\n'];
    %fprintf(fid,str);
    for i=1:M
        for j=1:N
            for k=1:P
                fprintf(fid,'%10.6f    ',v(i,j,k));
            end
            fprintf(fid,'\n');
        end
        fprintf(fid,'\n\n\n');
    end
    fclose(fid);

    %*****************************************************************
    file=Name2;
    file=[file,'.wgt'];
    fid = fopen(file,'w');

    if fid==-1
        disp('Error! cannot create the output file');
        return;
    end

    [M,N,P]=size(w);
    %str=['The hidden to output layer [',int2str(M),'*',int2str(N),'] :\n\n'];
    %fprintf(fid,str);
    for i=1:M
        for j=1:N
            for k=1:P
                fprintf(fid,'%10.6f    ',w(i,j));
            end
            fprintf(fid,'\n');
        end
        fprintf(fid,'\n\n\n');
    end
    fclose(fid);


% val=SNR(Im,Im_hat)
% Return the value val of Signal-to-Noise-Ratio(SNR)in dB
% for image Im and it's compressed vertion Im_hat.

function val=SNR(Im,Im_hat)

tmp=NMSE(Im,Im_hat);
val=-10*log10(tmp);


function wait(times)

% wait procedure for specified time.

% WAIT(TIMES)
% TIMES - number of seconds (can be fractional).
% Stops procedure for N seconds.


if nargin ~= 1
  error('Usage: WAIT(TIMES)');
end
```

```
drawnow
t1 = clock;
while etime(clock,t1) < times,end;


% --- Executes during object creation, after setting all properties.
function decompressed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to decompressed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate decompressed

% --- Executes during object creation, after setting all properties.
function originalimage_CreateFcn(hObject, eventdata, handles)
% hObject    handle to originalimage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate originalimage

% --- Executes during object creation, after setting all properties.
function compressed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to compressed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate compressed

% --- Executes during object creation, after setting all properties.

% --- Executes during object creation, after setting all properties.
function compresstext_CreateFcn(hObject, eventdata, handles)
% hObject    handle to compresstext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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


% --- Executes during object creation, after setting all properties.
```

```
function psnr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to psnr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function snr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to snr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function nmse_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nmse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function bitrateall_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bitrateall (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function bitratecr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bitratecr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function axes15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes15


% --- Executes during object creation, after setting all properties.
function axes19_CreateFcn(hObject, eventdata, handles)

BackGr = imread('cool-wallpapers-hd-8092-8423-hd-wallpapers.jpg');
imshow(BackGr);
axes(handles.axes19)

% hObject    handle to axes19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes19


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
imsave(handles.originalimage);
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
imsave(handles.decompressed);
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.


% --- Executes during object creation, after setting all properties.
function fsorig_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fsorig (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
```

```
function fscomp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fscomp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function fsdecomp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fsdecomp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

Appendix G. Plagiarism check



Results: 100% Unique

Chapter 1 INTRODUCTION Background of the Study — Good

visual representation, in one form or another, has — Good

programs like softwares that process digital — Good

allowed on the internet. Thus, online websites — Good

primarily taken of people and family. Now, we take