DWIT COLLEGE

DEERWALK INSTITUTE OF TECHNOLOGY

Tribhuvan University

Institute of Science and Technology

IMAGE COMPRESSION USING DEEP AUTOENCODER

A PROJECT REPORT

Submitted to

Department of Computer Science and Information Technology

DWIT College

In partial fulfillment of the requirements for the Bachelor's Degree in Computer Science and Information Technology

Submitted by

Ashim Regmi

August, 2016

DWIT College

DEERWALK INSTITUTE OF TECHNOLOGY

Tribhuvan University

SUPERVISOR’S RECOMENDATION

I hereby recommend that this project prepared under my supervision by ASHIM REGMI entitled **“IMAGE COMPRESSION USING DEEP AUTOENCDODER”** in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for the evaluation.

…………………………………………

Sarbin Sayami

Assistant Professor

IOST, Tribhuvan University

DWIT College

DEERWALK INSTITUTE OF TECHNOLOGY

Tribhuvan University

LETTER OF APPROVAL

This is to certify that this project prepared by ASHIM REGMI entitled **“IMAGE COMPRESSION USING DEEP AUTOENCODER”** in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

…………………………………… Rituraj Lamsal [Internal Examiner] Lecturer IOST, TU.	………………………………………… Hitesh Karki Chief Academic Officer DWIT College
……………………………………….. Jagdish Bhatta [External Examiner] IOST, Tribhuvan University	……………………………………….. Sarbin Sayami [Supervisor] Assistant Professor IOST, Tribhuvan University

ACKNOWLEDGEMENT

First of all, I would like to thank DWIT College for providing me with the opportunity and resources need for this project. Also, I am really thankful to my respected and esteemed guide Mr. Sarbin Sayami who helped me to complete this project.

Secondly, I would like to give special thanks to Ms.Nikita Gautam who helped me during the project.

At the end, I would like to express my sincere thanks to all my friends and others who helped me directly or indirectly during this project work.

Tribhuvan University

Institute of Science and Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

... ...

Ashim Regmi

Date: August, 2016

ABSTRACT

Big enterprises and organizations store a vast amount of images. The current technologies of image compression use similar characteristics within an image to compress the image. Deep Autoencoder neural network trains on a large set of images to figure out similarities between the images in the set. The network then uses those similarities to compress them and represent them using fewer codes than usually possible from current compression techniques. This document builds on to demonstrate how Deep Autoencoder neural network can be used to train on and compress large sets of images. MNIST handwritten digits dataset is used as training and testing set. 28 x 28 image is converted to a vector of size 784 x 1 which is then fed to the network. The 784 x 1 vector is then gradually compressed in each iteration of the training phase of the network. Once the vector is compressed, the middle hidden layer of the network will hold a compressed feature vector which can be stored or transmitted. This compressed feature vector later can be converted back to obtain a near approximation of the original image.

Keywords: Image compression, Deep Autoencoder

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

CT	Computed Tomography
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machine
SGD	Stochastic Gradient Descent
MNIST	Mixed National Institute of Standards and Technology
GUI	Graphical User Interface

CHAPTER 1: INTRODUCTION

1.1 Background

In many cases, group of images stored in a file system are almost identical for example, X-ray images in a medical database, large set of fingerprint images in a police database, CT scan images of brain, etc. These images have similar histograms, edge distributions and similar pixel intensities in same areas. These similarities in a set of images can be helpful to reduce size when storing, transmitting and analyzing them by applying inter-image encoding techniques. Deep Autoencoders are one of the techniques that encode inter-image pixels. (Karadimitriou, 1996)

In digital image compression, there are four types of redundancies that can be used to compress images:

- Coding redundancy
- Inter pixel redundancy
- Psycho visual redundancy
- Set redundancy

Coding, inter pixel and psycho visual redundancy are exploited by the current methods of image compressions. These three types of redundancies appear in monochrome images. However, in a set of similar images, significant amount of inter-image redundancy is present. The so called similar images have following features:

- similar pixel intensities in the same areas,
- comparable histograms,
- similar edge distributions and
- analogous distribution of features.

The correlation between images is measured by product-moment correlation coefficient, or Pearson’s r. When images having above features are measured using the coefficient, the correlation obtained is high. This statistical correlation is due to the presence of inter-image redundancy. This redundancy is so called set redundancy. This set redundancy can be used to improve image compression ratio. (Karadimitriou, 1996)

1.2 Problem Statement

The existing technology of image compression JPEG, MPEG-4 VTC, JPEG-LS, PNG and JPEG 2000 standards compress images by transforming, quantizing and encoding the quantized pixels. These technologies encode general traits in images and fail to encode common traits in a set of images. Successful applications of neural networks to encode common traits in large set of images have been well established. (D. Santa-Cruz, 2000)

1.3 Objective

The main goal of this project is:

- To use Deep Autoencoder neural network to compress gray level images to obtain a 4:1 compression ratio on MNIST handwritten digits dataset.

1.4 Scope

This project demonstrates the use of Deep Autoencoder neural network to compress 28 x 28 pixel gray scale image to a size of 14 x 14 image.

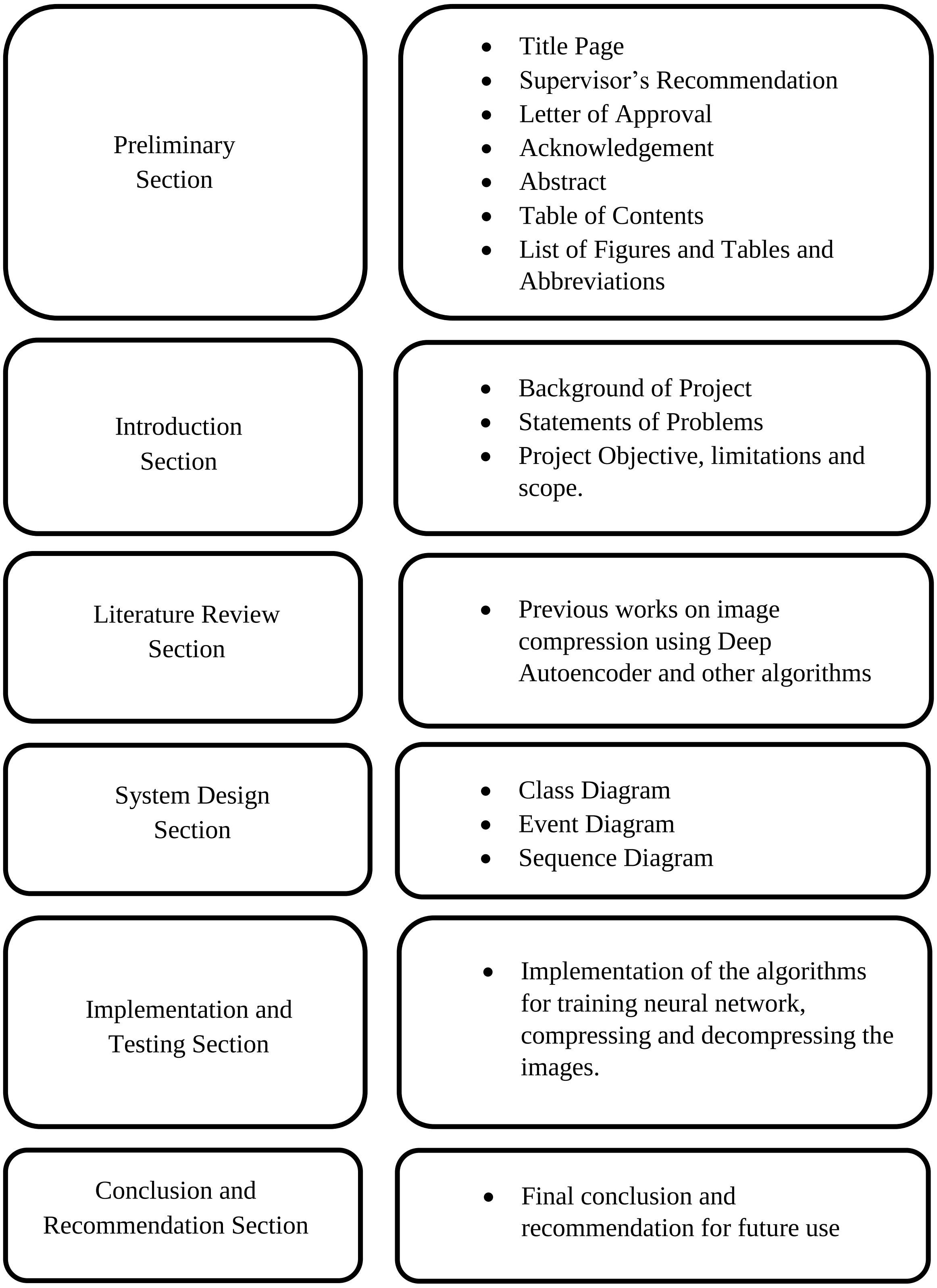
It is able to reconstruct only the approximation of the original image.

1.5 Limitation

- This application cannot compress colored images.
- This application can compress images of size 28 x 28 only.

1.6 Outline of Document

Hereafter, the document is outlined as follows:



Preliminary Section	• Title Page • Supervisor's Recommendation • Letter of Approval • Acknowledgement • Abstract • Table of Contents • List of Figures and Tables and Abbreviations
Introduction Section	• Background of Project • Statements of Problems • Project Objective, limitations and scope.
Literature Review Section	• Previous works on image compression using Deep Autoencoder and other algorithms
System Design Section	• Class Diagram • Event Diagram • Sequence Diagram
Implementation and Testing Section	• Implementation of the algorithms for training neural network, compressing and decompressing the images.
Conclusion and Recommendation Section	• Final conclusion and recommendation for future use

Figure 1 - Outline of document

CHAPTER 2: REQUIREMENT AND FEASIBILITY ANALYSIS

2.1 Literature Review

Deep Autoencoders perform nonlinear dimensionality reduction on input images to reduce them to a compressed representation. It has been used a number of times in the past to compress images. The results obtained from these implementations and the insights gained from their work are mentioned below.

2.1.1 Image compression using Huffman coding

Huffman Coding is a lossless compression algorithm which assigns codes to the pixels based on the frequency of occurrence. Pixels occurring more frequently will have shorter codes whereas pixels that occur less frequently will have the same length codes. The result is variable length codes that contain integral number of bits. The codes end up having unique prefix property. The codes are stored on a binary tree. The binary tree is built using the codes, starting from the leaves and moving up to the root of the tree.

An experiment was performed to compress a gray scale image using Huffman coding algorithm. Huffman coding was used to reduce the number of bits required to represent each pixel. Experiment results showed that up to a 0.8456 compression ratio was obtained on the image. (D. Santa-Cruz, 2000)

2.1.2 Comparison with PCA and JPEG

Anand Atreya and Daniel O’Shea have presented their use of Deep Autoencoders. Their first step was to train the autoencoder. First the images were fed into the nonlinear encoding layer. This encoding layer gradually compressed the input image pixels into a compressed feature vector. After the encoding was complete, the compressed feature vector was again fed into the decoding layer which then reversed the process of the encoding layer to obtain the reconstruction of input images. The difference between the reconstruction and input images was regarded as error to further minimize the difference. Also, a sparsity parameter was used to ensure sparsity of the activation vectors in each layer except the output layer.

The next step was to find an ideal set of quantization values using k-means clustering to encode the activation values.

Finally, they compared this compression algorithm with other common algorithms like JPEG and PCA. They found that Deep Autoencoder image compression algorithm outperformed JPEG in high compression and reduced quality scheme. Also, Deep Autoencoder showed better non-linear representation of the input image than that of PCA and hence Deep Autoencoder had better reconstruction quality. They observed that Deep Autoencoder was able to find out statistical regularities in a specific domain of images which was not possible by JPEG. (O'Shea, 2009)

2.1.3 Representing and generalizing nonlinear structure in data

G. E. Hinton and R. R. Salakhutdinov have also presented the use of autoencoder to reduce the dimensionality of data. They trained a stack of Restricted Boltzmann machines (RBMs) to pre-train the neural network with each RBM consisting a layer of feature detectors. After the pretraining, the RBMs were unrolled to get full layers of the neural network which at last were fine-tuned using Backpropagation algorithm to reduce the data reconstruction error.

RBM is a two layer network in which the input binary pixels are connected to the hidden layer using symmetrical weights. The binary pixels are the nodes in the visible layer and the hidden layer activations are updated using sigmoid function on the sum of all the products of inputs and their respective weights. Again a sigmoid function is used on the sum of all the products of hidden values and their respective weights. The hidden layer here acts as a feature detector. The number of nodes in the hidden layer is always kept smaller than the number of pixels in the input image so that the hidden layer does not end up learning an identity function. With gradual decrease in number of pixels in the hidden layers, more RBMs are added onto the network with first layer of feature detectors as visible layer until a desired encoding state is reached. It is mentioned in the paper that each layer of the detectors captures strong, high-order correlations and this is an efficient way to progressively reveal low-dimensional, nonlinear structure. After pretraining the network, the encoder part of the neural network is then unrolled to get the decoder part where same weights of autoencoder are used. The weights are later adjusted in the fine tuning stage to get optimal reconstruction. They in general concludes that backpropagation together with

Deep Autoencoder is a brute force towards efficient representing and generalizing nonlinear structure in data. (R. R. Salakhutdinov, 2006)

2.1.4 Deep Autoencoders as feature detectors

Andrew Y. Ng and others have demonstrated the use of large-scale unsupervised learning algorithms for building high-level features which includes the use of stack of RBMs to construct autoencoders.

They have presented the training using unlabeled data to create high-level, class-specific feature detectors. They have trained a 9-layer locally connected sparse autoencoder with pooling and local contrast normalization on a large dataset of images (the model has 1 billion connections, the dataset has 10 million 200 x 200 pixel images). They trained the network using model parallelism and asynchronous stochastic gradient descent (SGD) on a cluster of 1,000 machines (16000 cores) for 3 days. They proved that it is possible to train a face detector without having to label images as containing a face or not.

They have used sparse Deep Autoencoder with three important constituents: local receptive fields, pooling and local contrast normalization. The local receptive field method is used to scale the autoencoder to large images. The idea behind this method is that each feature in the autoencoder can connect only to a small region of the lower layer. Also, to achieve invariance to local deformations, they employed local L2 pooling and local contrast normalization. L2 pooling allowed to learn invariant features. The deep autoencoder was constructed by replicating three times the same stage composed of local filtering, local pooling and local contrast normalization. The output of one stage is input to the next one and the overall model is lastly interpreted as a nine-layered network. The first and second sublayers are known as filtering and pooling respectively. The third sublayer performs local subtractive and divisive normalization and it is inspired by biological and computational models. The weights are not shared which allow the learning of more invariances other than translational invariances. In a nutshell, they show that Deep Autoencoders perform well as feature detectors in case of images. (Andrew Y. Ng, 2012)

2.1.5 The enhanced compression model

The enhanced compression model compresses 8 bits per pixels to 3 bits per pixels. Also, the compression model used is lossless and perfectly recovers the image. Present compression models follow one of the basic theoretical compression models depicted in Figure 3 and Figure 4. Figure 3 presents the lossless compression model and Figure 4 presents the lossy compression model. The lossless and lossy compression differ only in the "Quantization" step as show below. Compression is generally obtained by eliminating different kind of redundancy. Pixel mapping reduces the interpixel redundancy, Quantization reduces psychovisual redundancy and symbol encoding in final step eliminates the coding redundancy.

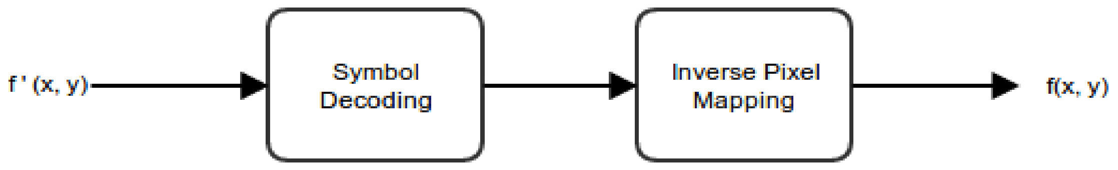


Figure 2 - Decompressing image

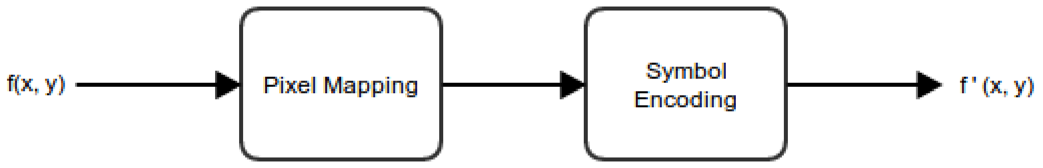


Figure 3 - Lossless compressing model

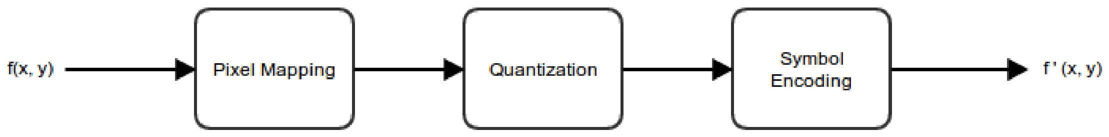


Figure 4 - Lossy compression model

f(x, y) : origin image,

f `(x, y) : compressed image.

Above compression models are sufficient for compressing individual images. However, for a set of similar images containing set redundancy the general compression models are not sufficient. The enhanced compressed model proposes to handle set redundancy in image compression by introducing ‘Set Mapping’ in compression step and ‘Inverse Set Mapping’ in decompression step.

Deep Autoencoder learns the set mapping and inverse set mapping between multiple images automatically. In a way, the Deep Autoencoder is similar to the enhanced compression model proposed. (Karadimitriou, 1996)

2.2 Requirement Analysis

2.2.1 Functional requirements

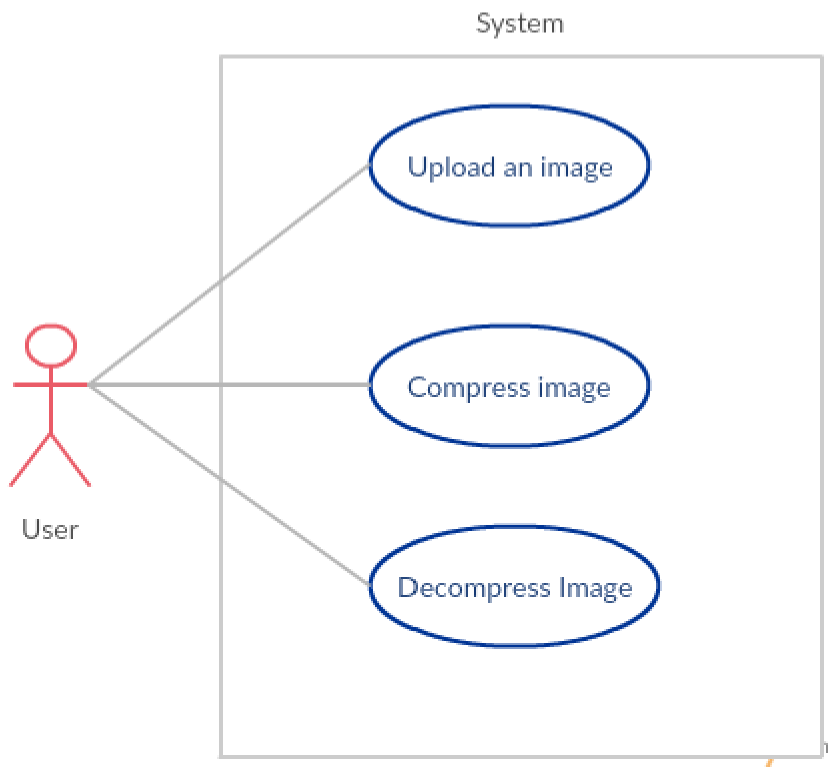


Figure 5 - Functional requirements

Figure 5 depicts the functions performed by a user on the system. The user can upload an image, compress the image or decompress the uploaded image.

2.2.2 Non-functional requirements

The non-functional requirements for this project are:

- Image must be of size 28 x 28
- Image must be grayscale or monochrome image
- Image can be compressed only after the application has been trained on a set of images from same domain.

2.3 Feasibility Analysis

2.3.1 Schedule feasibility

This project comprises following components:

- Deep Autoencoder Neural Network Implementation in Octave
- Compressor function in Java
- Decompressor function in Java
- User Interface in Java Swing

The critical path schedule is depicted in Figure 6 and the schedule for the implementation of the above components is represented in Table 1 as:

Table 1 - Activities of project with their duration

Legend	**Activity**	**Duration**	**Precedence**
A	Deep Autoencoder Neural Network in Octave	4 weeks	-
B	Compressor function in Java	1 week	A
C	Decompressor function in Java	1 week	A, B
D	User Interface in Java Swing	1 week	-

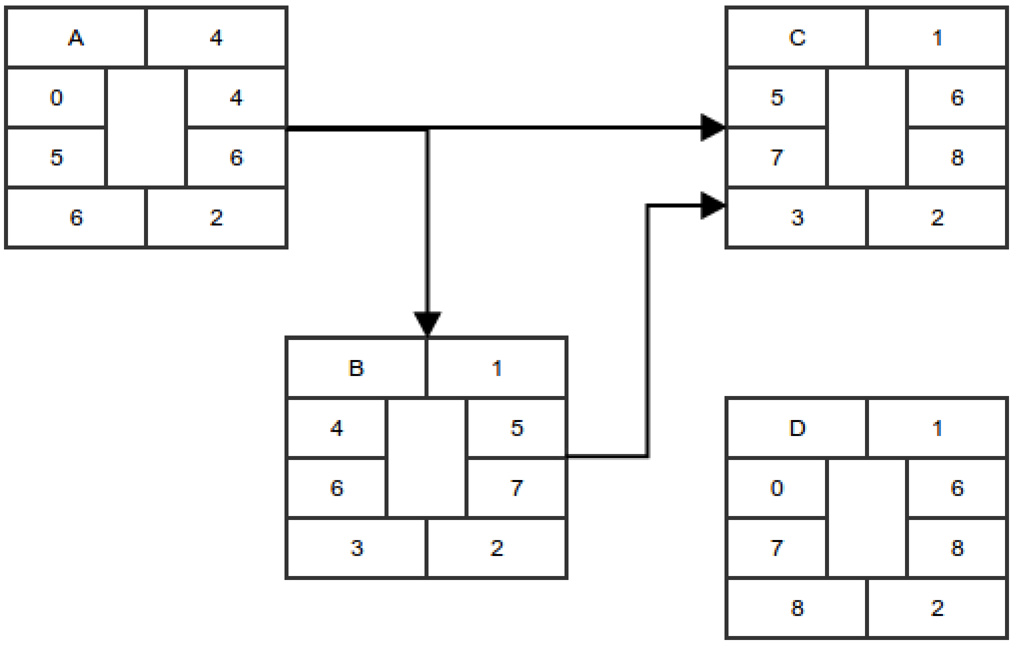


Figure 6 - Critical path schedule

Task A can be completed in 4 weeks. So, task B can only be started after 4 weeks and ends before 5th week. Task C has precedence of A and B. So, task C can only be started when task A and task B complete successfully. Task D is independent of other tasks. So, it can be started anytime between 1st and 8th week. The project must be completed in 8 weeks. The late finish time of other tasks according to that deadline is shown in Figure 5.

2.3.2 Operational feasibility

This application takes an image, compresses it and saves the image on the disk. It can also take an already compressed image and generate its approximate reconstruction i.e. the original image. Since, the working mechanism of this application doesn't result in any complications, it is operationally feasible.

2.3.3 Technical feasibility

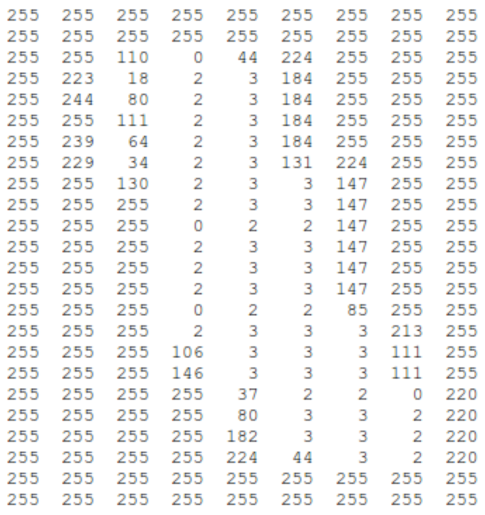
The neural network model is implemented using open source GNU Octave. Also, the compressor, decompressor functions and user interface are implemented in Java. This application doesn't require additional technologies other than those mentioned above and is technically feasible.

CHAPTER 3: SYSTEM DESIGN

3.1 Methodology

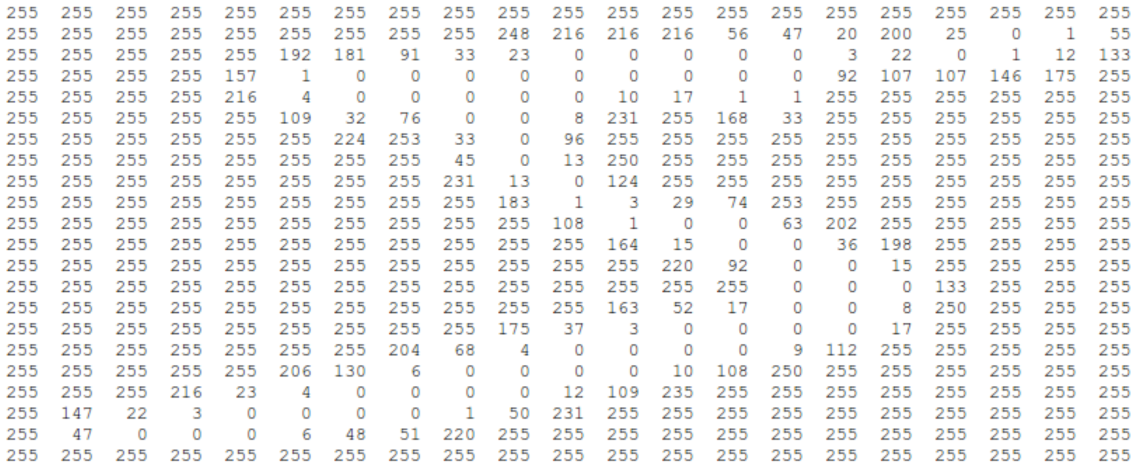
3.1.1 Data collection

The handwritten digits images from MNIST database is used as the training and testing dataset. The images are of size 28 x 28 (784 total pixels in each image) and hence the size of input layer is 784. There are altogether 60000 images in the MNIST database. First 20000 images are used as training data and 1000 images are used as testing data. The data samples are shown in Figure 7 and Figure 8.



```
  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255
  255  255  110    0   44  224  255  255  255
  255  223   18    2    3  184  255  255  255
  255  244   80    2    3  184  255  255  255
  255  255  111    2    3  184  255  255  255
  255  239   64    2    3  184  255  255  255
  255  229   34    2    3  131  224  255  255
  255  255  130    2    3    3  147  255  255
  255  255  255    2    3    3  147  255  255
  255  255  255    0    2    2  147  255  255
  255  255  255    2    3    3  147  255  255
  255  255  255    2    3    3  147  255  255
  255  255  255    2    3    3  147  255  255
  255  255  255    0    2    2   85  255  255
  255  255  255    2    3    3    3  213  255
  255  255  255  106    3    3    3  111  255
  255  255  255  146    3    3    3  111  255
  255  255  255  255   37    2    2    0  220
  255  255  255  255   80    3    3    2  220
  255  255  255  255  182    3    3    2  220
  255  255  255  255  224   44    3    2  220
  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255
```

Figure 8 - Data sample showing digit 1



```
  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255  248  216  216  216   56   47   20  200   25    0    1   55
  255  255  255  255  255  192  181   91   33   23    0    0    0    0    0    3   22    0    1   12  133
  255  255  255  255  157    1    0    0    0    0    0    0    0    0    0   92  107  107  146  175  255
  255  255  255  255  216    4    0    0    0    0    0   10   17    1    1  255  255  255  255  255  255
  255  255  255  255  255  109   32   76    0    0    8  231  255  168   33  255  255  255  255  255  255
  255  255  255  255  255  255  224  253   33    0   96  255  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255   45    0   13  250  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  231   13    0  124  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255  183    1    3   29   74  253  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255  255  108    1    0    0   63  202  255  255  255  255  255
  255  255  255  255  255  255  255  255  255  255  255  164   15    0    0   36  198  255  255  255  255
  255  255  255  255  255  255  255  255  255  255  255  255  220   92    0    0   15  255  255  255  255
  255  255  255  255  255  255  255  255  255  255  255  255  255  255    0    0    0  133  255  255  255
  255  255  255  255  255  255  255  255  255  255  255  163   52   17    0    0    8  250  255  255  255
  255  255  255  255  255  255  255  255  255  175   37    3    0    0    0    0   17  255  255  255  255
  255  255  255  255  255  255  255  204   68    4    0    0    0    0    9  112  255  255  255  255  255
  255  255  255  255  255  206  130    6    0    0    0    0   10  108  250  255  255  255  255  255  255
  255  255  255  216   23    4    0    0    0    0   12  109  235  255  255  255  255  255  255  255  255
  255  147   22    3    0    0    0    0    1   50  231  255  255  255  255  255  255  255  255  255  255
  255   47    0    0    0    6   48   51  220  255  255  255  255  255  255  255  255  255  255  255  255
  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255  255
```

Figure 7 - Data sample showing matrix of digit 5

3.1.2 Network architecture setup

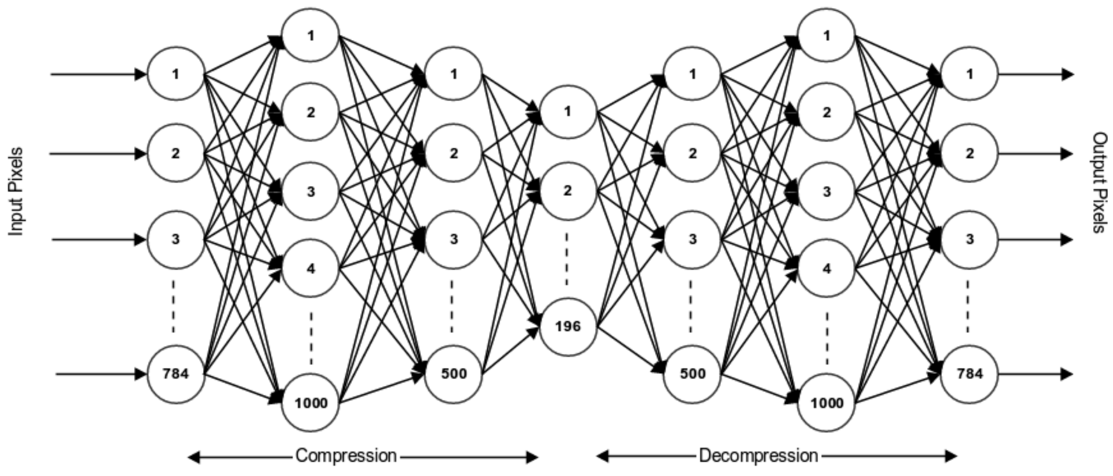


Figure 9 - Deep Autoencoder architecture

The neural network used here is a Deep Autoencoder which consists of two symmetrical deep belief networks that typically have two to four layers of encoding hidden layers and another two to four layers of decoding hidden layers. Each pair of layers in the deep belief network is part of a stack of Restricted Boltzmann Machines (RBMs). Each pair consists of two layers: a visible layer and a hidden layer (feature detecting layer).

The encoding layer consists of three hidden layers with each of size 1000, 500 and 196 respectively. The first hidden layer is sized 1000 nodes because expanding the size in this way compensates the incomplete representation provided by the sigmoid belief units. Hence, the encoding layer compresses the 784 pixels into 196 values.

3.1.3 Training

The encoding layer is first pre-trained by forming 3 RBMs: input layer and first hidden layer, first hidden layer and second hidden layer and second hidden layer and third hidden layer. Each RBMs are pre-trained at first to produce a 196 size compressed feature vector. Then, the RBMs are unrolled to form the decoding layer of the deep belief network. Hence, the decoding layer consists of two hidden layers and one output layer of size 500, 1000 and 784 respectively. The decoding layer then transforms the compressed feature vector by feed forwarding it through the layers to obtain the approximate reconstruction of the input image.

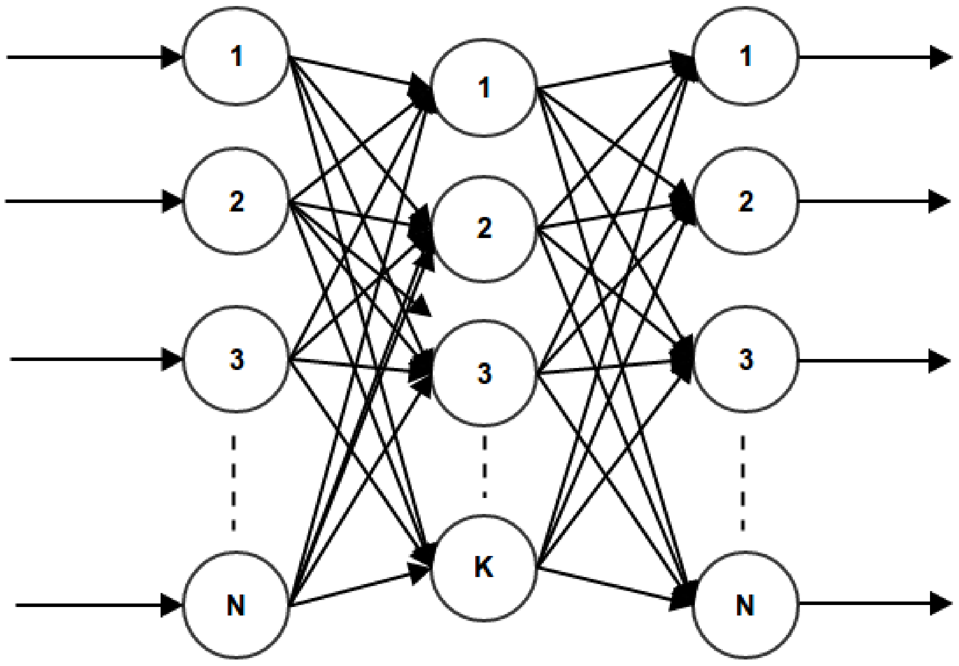


Figure 10 - Restricted Boltzmann Machine

Each RBM is of the form as shown in Figure 10. K is the size of the middle hidden layer. This hidden layer learns the features of the input layer which is of size N. These machines when placed progressively decreasing the K's value form an encoding layer which gradually generalize over the original input.

The gradient for the sigmoid function of the nodes in the network was computed as:

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1 - g(z))$$

where sigmoid is given by:

$$\text{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}.$$

The cost function used for the network was:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right]$$

J - cost function

m - no. of images

y - target images(in this project, input images)

h - network output

3.2 System Design

3.2.1 Class diagram

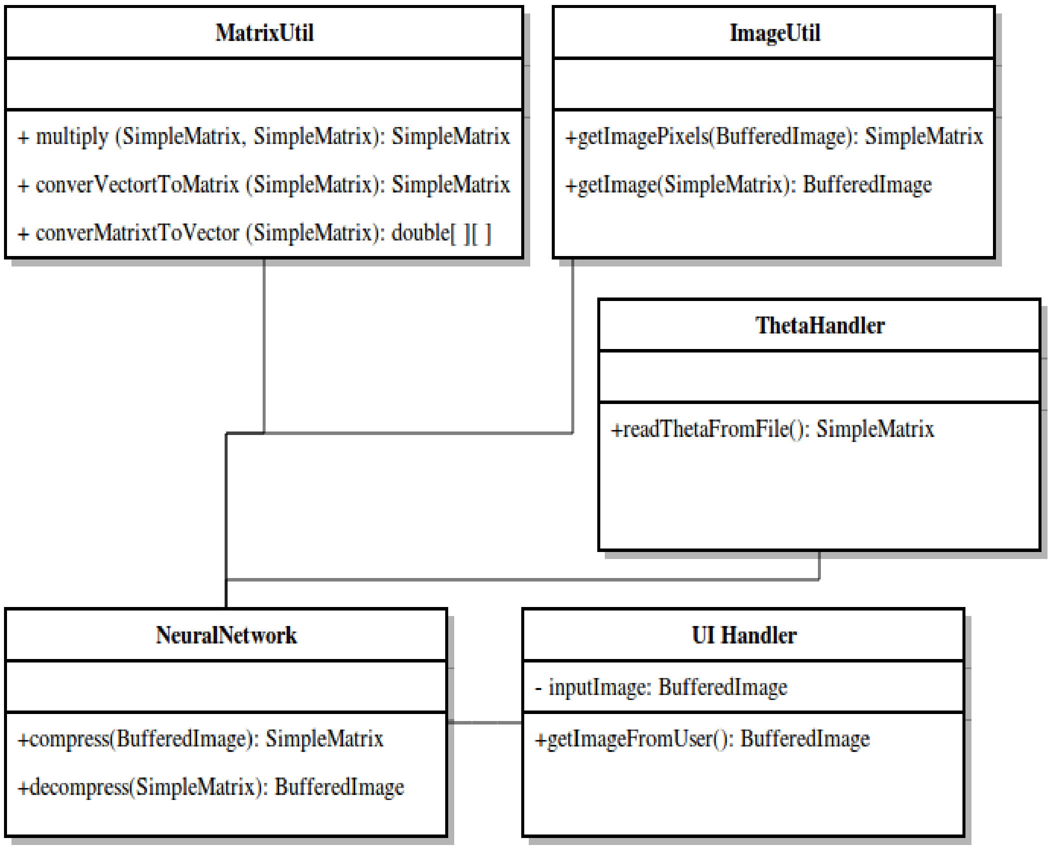

MatrixUtil
+ multiply (SimpleMatrix, SimpleMatrix): SimpleMatrix
+ converVectortToMatrix (SimpleMatrix): SimpleMatrix
+ converMatrixtToVector (SimpleMatrix): double[ ][ ]
ImageUtil
+getImagePixels(BufferedImage): SimpleMatrix
+getImage(SimpleMatrix): BufferedImage
ThetaHandler
+readThetaFromFile(): SimpleMatrix
NeuralNetwork
+compress(BufferedImage): SimpleMatrix
+decompress(SimpleMatrix): BufferedImage
UI Handler
- inputImage: BufferedImage
+getImageFromUser(): BufferedImage

Figure 11 - Class diagram

There are mainly five classes in the application namely: MatrixUtil, ImageUtil, NeuralNetwork, UI Handler and ThetaHandler as shown in Figure 11.

UI Handler does the work of receiving input image from the user and display the output image. The input image is obtained and stored in the buffer in java.awt.image.BufferedImage object. When user presses the ‘Compress’ button, compress method of NeuralNetwork class is called which compresses the image using three other classes: ThetaHandler, MatrixUtil and ImageUtil. ThetaHandler class reads theta (weights) from file, MatrixUtil class provides methods for multiplying two matrices, converting matrix to vector and vector back to matrix and ImageUtil class provides methods to get pixels from image and get image back from pixel values.

3.2.2 Event diagram

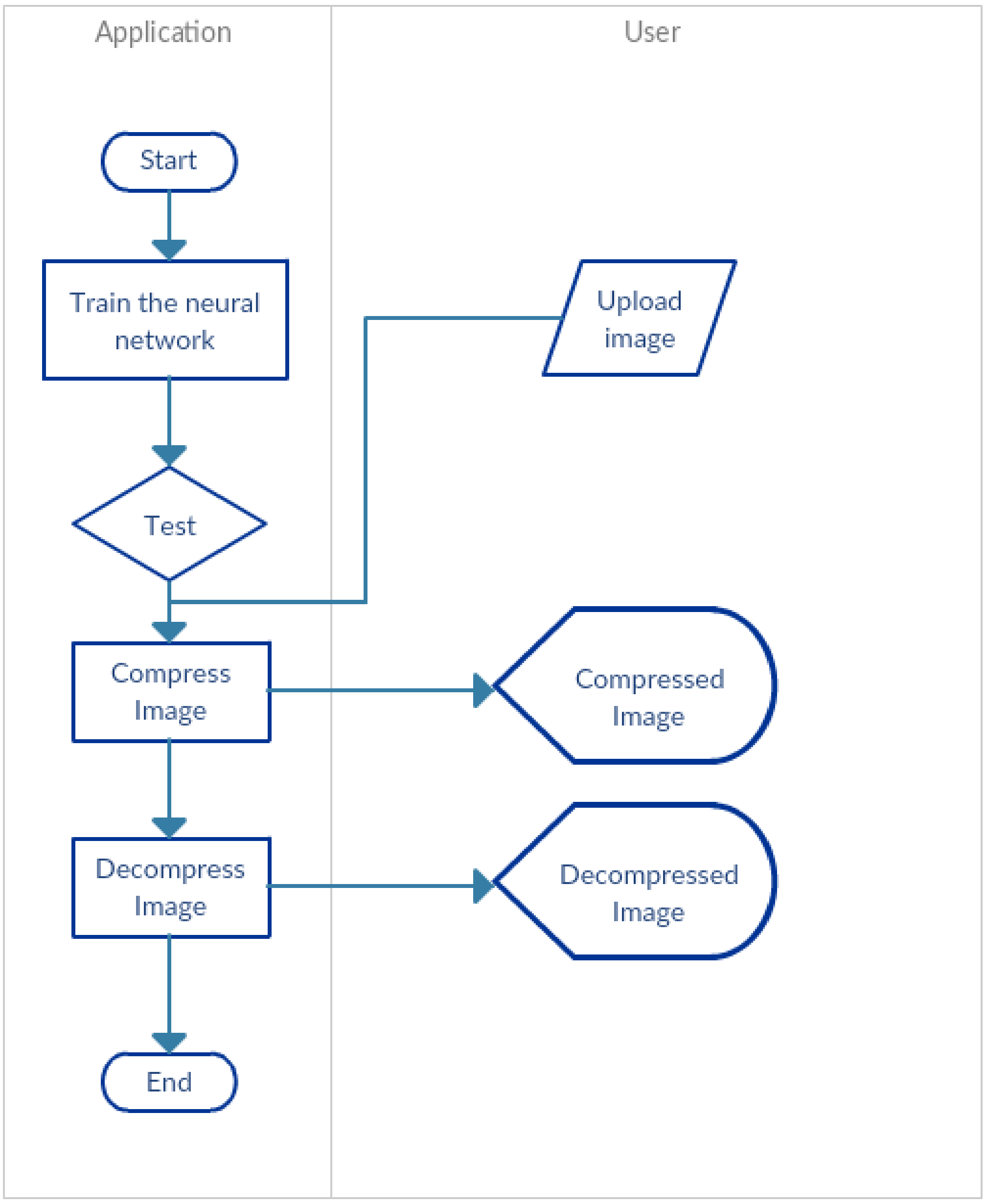


Figure 12 - Event diagram

The events in the application is shown in Figure 12. First of all, the network is trained for 10000 iteration. In this process, the network learns to compress and reconstruct the original image as perfectly as possible. Then the network's output is tested.

Secondly, the user uploads an image to the application. Then, the application compresses the image and returns the compressed image.

In the final step, user uploads an already compressed image which application decompresses and returns the reconstruction of original image.

3.2.3 Sequence diagram

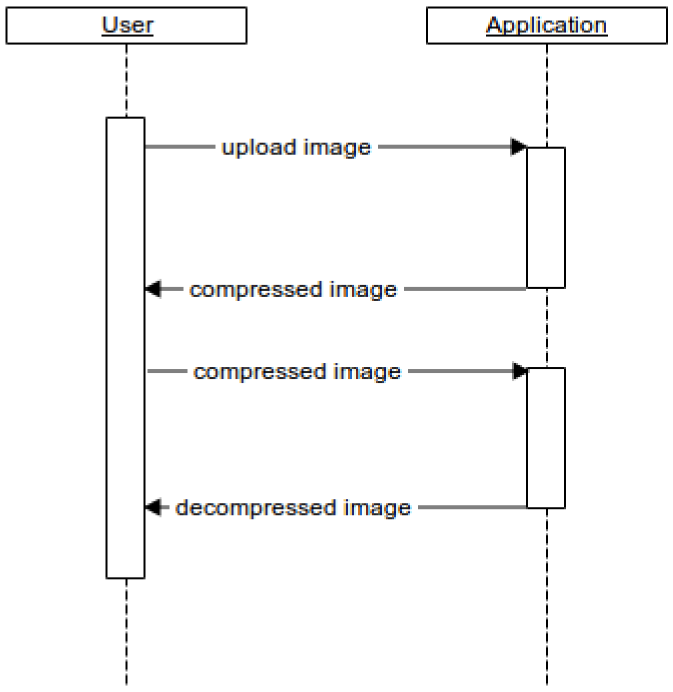


Figure 13 - Sequence diagram

The sequence of events in the application is shown in Figure 13. This diagram shows how the user interacts with the application.

CHAPTER 4: IMPLEMENTATION AND TESTING

4.1 Implementation

4.1.1 Tools used

Efficient Java Matrix Library (EJML)
It is a matrix library for manipulating matrices. The SimpleMatrix class of the library is used in this project to hold the pixels of the image, multiply the matrices and hold the values of hidden layers.

GNU Octave
It is an open source version of the MATLAB. It is mostly used for numerical computations. The deep autoencoder neural network was implemented in Octave. The cost function of the network was minimized using fmincg function available in Octave.

Java Swing
It is a GUI toolkit for Java. It provides GUI components that can be used to build a UI design. The user interface for the application was implemented and designed using Java Swing.

4.1.2 Major classes and methods

1. NeuralNetwork.java

It is the major class which contains two methods: compress( ) and decompress( ).

- compress(BufferedImage) : SimpleMatrix

The compress method takes a BufferedImage.java object and compresses the image in the object and returns a compressed matrix representation of the image.

- decompress(SimpleMatrix) : BufferedImage

Meanwhile, the decompress method takes the compressed matrix returned by compress method and generates best possible approximation of the original image.

The compress and decompress methods use the weights that are obtained after training the neural network in Octave.

2. MatrixUtil.java

This class contains three methods: multiply ( ), convertMatrixToVector ( ) and convertVectorToMatrix ( ).

- multiply(SimpleMatrix, SimpleMatrix) : SimpleMatrix

This method takes two matrices and returns the multiplication result.

- convertMatrixToVector (SimpleMatrix) : SimpleMatrix

This method takes a matrix of arbitrary size and converts it to a vector which is a one-dimensional vector.

- convertVectorToMatrix (SimpleMatrix) : SimpleMatrix

This method takes a one dimensional matrix or a vector and converts it to a two dimensional matrix.

4.2 Testing

The quality of the image reconstructed by the Deep Autoencoder can be measured using a number of techniques. In this project, Mutual Information (MI) is used to compare the reconstructed images with original images.

Given two images I1 and I2, the MI between them can be calculated as:

$$MI(I1, I2) = Entropy(I1) + Entropy(I2) - JointEntropy(I1, I2)$$

where

$$Entropy(I) = -\sum_{x} p(x) log p(x)$$

Also, joint entropy can be calculated as:

$$JointEntropy(I1, I2) = -\sum_{x,y} p(x,y) log p(x,y)$$

where p(x, y) is the joint probability distribution of pixels associated with images I1 and I2. The joint entropy decreases when there is a one-to-one mapping between the pixels in

A and B and increases when the statistical relationship between the pixels diminish. Mutual information finds the most complex overlapping between any two images. A complex one-to-one mapping is indicated by a maximum mutual information. (Daniel B. Russakoff. Carlo Tomasi, 2004)

The mutual information is calculated for 1000 test images in 5, 100, 1000 and 10000 iterations which are shown in the Table 4.1 below:

Table 2 - Mutual Information for different iterations

Iterations	**Mutual Information**
5	1.5044
100	1.8495
1000	2.3086
10000	2.3177

CHAPTER 5: MAINTENANCE AND SUPPORT PLAN

The neural network implementation of the application is written in GNU Octave. Later on, this implementation can be shifted to Java and the user can be allowed to train on custom images of different sizes.

Also, the performance evaluation of the application will be constantly measured. The occurrence of any errors will be fixed.

CHAPTER 6: CONCLUSION AND RECOMMENDATION

6.1 Conclusion

This application is able to compress the MNIST handwritten digits images up to 4:1 compression ratio. The output obtained transforms the 0-255 values in original image to only two gray scale values 0 or 255 in the reconstructed image. This is due to the thresholding operation performed in the training phase.

6.2 Recommendations

The neural network implemented in this application doesn’t pre-train the neural network. The network can be pre-trained using stacked Restricted Boltzmann Machines (RBMs) to obtain approximation of initial weights. This significantly improves the reconstruction quality and training time of the neural network.

APPENDIX

Figure: Before compression

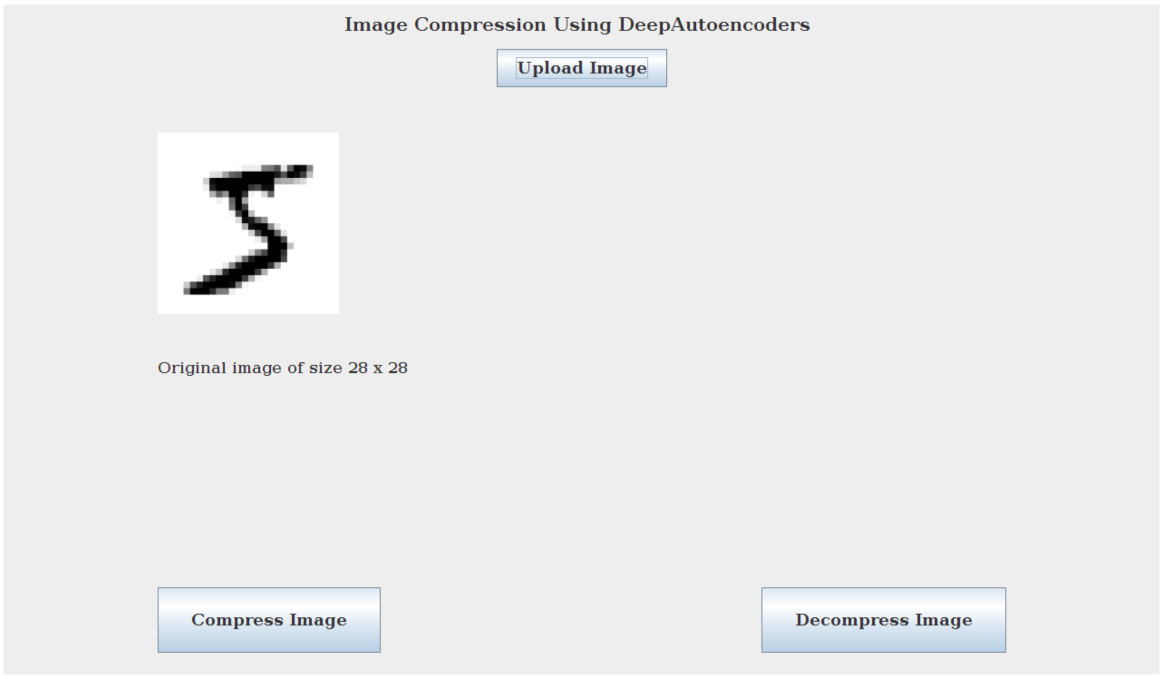


Image Compression Using DeepAutoencoders
Upload Image
Original image of size 28 x 28
Compress Image
Decompress Image

Figure: After compression

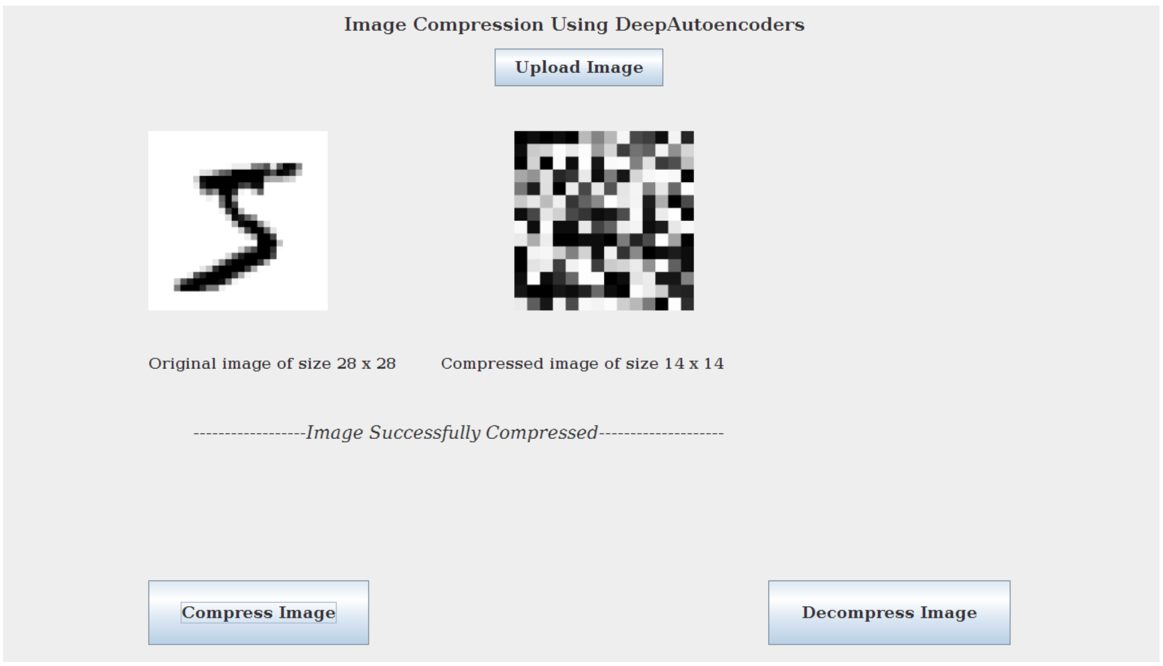


Image Compression Using DeepAutoencoders
Upload Image
Original image of size 28 x 28
Compressed image of size 14 x 14
-----------------Image Successfully Compressed-------------------
Compress Image
Decompress Image

Figure: After image is reconstructed from compressed form

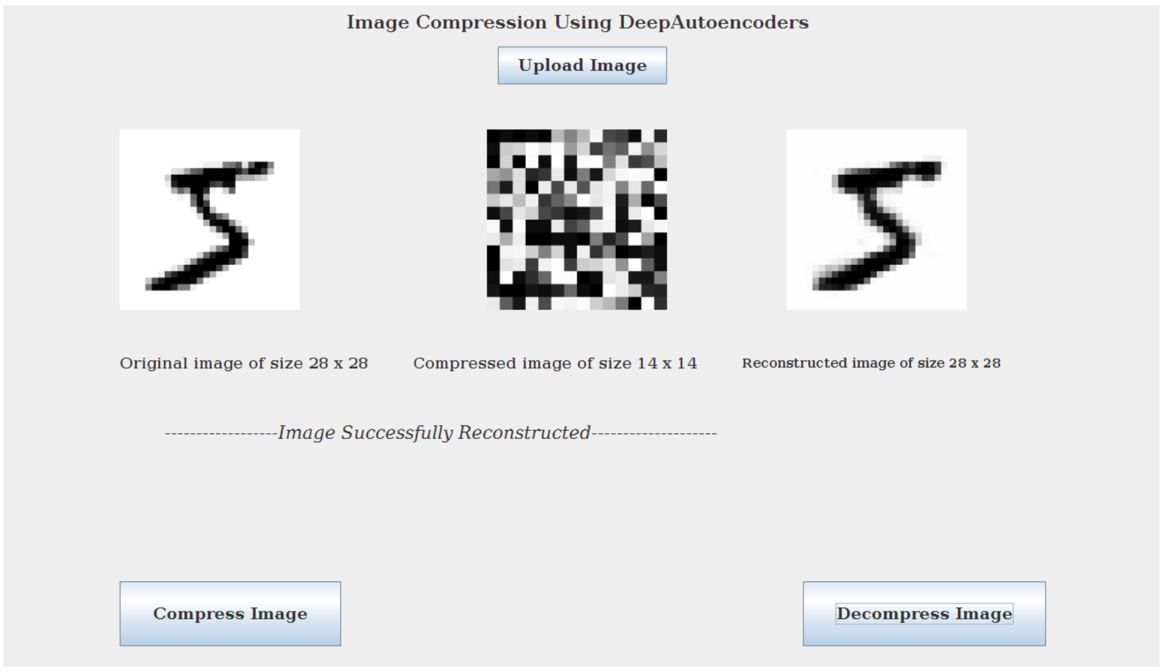


Image Compression Using DeepAutoencoders
Upload Image
Original image of size 28 x 28
Compressed image of size 14 x 14
Reconstructed image of size 28 x 28
Compress Image
Decompress Image

------------------*Image Successfully Reconstructed*--------------------

REFERENCES

Andrew Y. Ng, Q. V. (2012). Building High-level Features Using Large Scale Unsupervised Learning. *29th International Conference on Machine Learning*, (pp. 2-5). Edinburgh, Scotland, UK.

D. Santa-Cruz, T. E. (2000). JPEG 2000 still image coding versus other standards.

Daniel B. Russakoff. Carlo Tomasi, T. R. (2004). Image Similarity Using Mutual Information of Regions. In J. M. Tomas Pajdla, *Computer Vision - ECCV 2004* (pp. 596-607). Berlin: Springer Berlin Heidelberg.

Karadimitriou, K. (1996). *SET REDUNDANCY, THE ENHANCED COMPRESSION MODEL, AND METHODS FOR COMPRESSING SETS OF SIMILAR IMAGES.* Louisiana: Department of Computer Science, Louisiana State University.

O'Shea, A. A. (2009). *Novel Lossy Compression Algorithms with Stacked Autoencoders.* Stanford University.

R. R. Salakhutdinov, G. E. (2006, July 28). Reducing the Dimensionality of Data with Neural Networks. *SCIENCE VOL 313*, pp. 505-507.