Faculty of Electrical and Electronic Engineering Technology

IMAGE COMPRESSION BY USING SINGULAR VALUE DECOMPOSITION WITH MATLAB SIMULATION

MUHAMAD IZZAT SYAFIQ BIN NORMAN

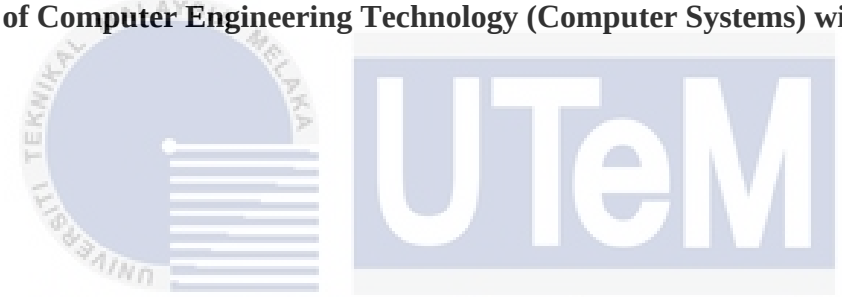
Bachelor of Computer Engineering Technology (Computer Systems) with Honours

2022

IMAGE COMPRESSION BY USING SINGULAR VALUE DECOMPOSITION WITH MATLAB SIMULATION

MUHAMAD IZZAT SYAFIQ BIN NORMAN

A project report submitted
in partial fulfillment of the requirements for the degree of
Bachelor of Computer Engineering Technology (Computer Systems) with Honours



Faculty of Electrical and Electronic Engineering Technology

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2022

DECLARATION

Aside from the information cited in the references, I certify that this project report entitled "IMAGE COMPRESSION BY USING SINGULAR VALUE DECOMPOSITION WITH MATLAB SIMULATION" was written entirely by me and represents my original research. The project report has not been accepted for any degree and is not being submitted for consideration for any other degree at the time of submission.

Signature : 

Student Name : MUHAMAD IZZAT SYAFIQ BIN NORMAN

Date : 4 JANUARI 2022

APPROVAL

I hereby declare that I have checked this project report and in my opinion, this project report is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Engineering Technology (Computer Systems) with Honours.

Signature : 

Supervisor Name : TS. DR. ROSTAM AFFENDI BIN HAMZAH

Date : 11 Januari 2022

Signature :

Co-Supervisor :

Name (if any)

Date :

DEDICATION

This research is lovingly dedicated to my parents,

Hapizah and Norman

To my beloved brother and sister,

Edzuan and Fathehah

And to my long-time partner,

Atikah

ABSTRACT

Network bandwidth and storage capacity are becoming increasingly constrained as the demand for multimedia products grows. That's why it's so critical to reduce redundant data and save as much storage and bandwidth as possible. Data compression, also known as source coding, is a technique used in computer science and information theory to encode information using fewer bits or other information-bearing units than an unencoded version of the same information. Because it reduces the consumption of expensive resources like memory and transmission bandwidth, compression is a cost-effective method of data storage. Singular Value Decomposition (SVD) is used as an image compression method in this research. In addition, a MATLAB programme based on Singular Value Decomposition will be created to mimic the image compression algorithm (SVD).

ABSTRAK

Lonjakan kadar penggunaan dan permintaan produk multimedia dalam masa beberapa tahun ini menyebabkan jalur lebar rangkaian dan kapasiti peranti memori tidak mengcukupi. Hasilnya, pemampatan data menjadi lebih penting untuk meminimumkan data lewah dan ini dapat mengurangkan ruang di dalam perkakasan simpanan dan juga mengurangkan kadar jalur levar yang diperlukan.Di dalam sains computer dan teori maklumat, pemampatan data adalah pengekodan maklumat yang menggunakan bit yang lebih sedikit atau unit yang mengandungi maklumat lain daripada versi yang tidak dikodkan. Pemampatan memberi kelebihan kerana dapat menjimatkan wang dengan mengurangkan penggunaan memory dan jalur lebar yang kadang kala mahal. Kajian ini membincangkan kelebihan menggunakan Penguraian Nilai Singular (SVD) sebagai kaedah pemampatan gambar dalam kajian ini. Di samping itu, program yang menerapkan algoritma ini juga akan dicipta di dalam aplikasi MATLAB untuk melihat fungsinya.

ACKNOWLEDGEMENTS

Thank you to my mentor, TS. DR. ROSTAM AFFENDI BIN HAMZAH, for all of your help and wisdom, as well as your patience.

A special thanks to my parents and family members for their support and prayers while I was in college.

Finally, I would like to thank all of my classmates, faculty members, and other unnamed individuals for their cooperation and assistance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

m	-	Matrix m
n	-	Matrix n
^{T}	-	Transpose
A	-	Image matrix
U	-	$m\ x\ m$ orthogonal matrix
S	-	$m\ x\ n$ matrix
V	-	$n\ x\ n$ orthogonal matrix
$\sum$	-	Summation

LIST OF ABBREVIATIONS

SVD	-	Singular Value Decompostion
JPEG/JPG	-	Joint Photograhic Experts Group
PNG	-	Portable Network Graphics
MSE	-	Mean-Square Error
PNSR	-	Peak Noise-to-Signal Ratio
CR	-	Compression Ratio

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

In today's digital society, technology is becoming an increasingly important part of our daily lives. We produce ever-increasing quantities of data every day. There are many different types of data out there, including text, audio, video, and photos. Humans frequently use images to convey information, which makes photos one of the most popular data-sharing methods. When smartphones were introduced, the amount of data being shared and the size of the data that was being shared both increased. The amount of memory and bandwidth needed to store and transmit all of these photos is staggering. Storing these data has proven to be prohibitively expensive due to the high cost of storage technology. Many researchers are now looking into ways to reduce the amount of data that can be stored and transmitted before it can be used. This is especially true for fields like artificial intelligence or pattern recognition. In order to reduce the amount of storage space required for an image file, this research will concentrate on picture compression using singular value decomposition.

Compression of digital images is a hot topic in the field of digital image processing. An image's storage and transmission bandwidth can be kept to a minimum by implementing various techniques for compression. When compressing images, the goal is to reduce the amount of information that is irrelevant or unnecessary, thus reducing storage space and increasing transmission speed. Image quality should not be reduced so much that the user is unable to decipher the content. It's important to use a suitable image compression method to

balance this trade-off the requirements of the application usually determine the compression/data quality trade-off.

For image compression, there are two main methods: lossy and lossless. To determine whether or not the original data can be recovered when a compressed file is decompressed, use the words "lossy" and "lossless." Lossless compression preserves all of the original data in a compressed file when it is decompressed. On the other hand, lossless compression shrinks a file without altering its content by removing redundant data permanently. When the file is uncompressed, only a small portion of the original data is recovered.

That which this paper discusses is the Singular Value Decomposition procedure (SVD). In the singular-value decomposition, a real or complex matrix is factored into its components using an extension of the polar decomposition to generalise the eigen decomposition, which only exists for square normal matrices. Examples include object detection, face recognition, field matching, and meteorological and oceanographic data processing.

1.2 Problem Statement

Currently available storage servers are becoming increasingly full due to the growing number of consumers who regularly share and exchange images. People working from home are increasing exponentially as a result of the pandemic, and the traffic of data exchange is becoming increasingly congested as the number of people working from home continues to rise.

The cost of equipment and data bandwidth prices went up as a result of this issue. SVD may be a potential solution to this problem, as this paper's research explores the possibility.

1.3 Project Objective

Methods for observing and simulating the application of the SVD algorithm as an image compression method using MATLAB are an important goal in this research. The following are the specific goals:

a) To investigate how Singular Value Decomposition can be used as an image compression approach.

b) To develop image compression script using SVD for both grayscale and colour images in MATLAB.

c) To reduce the size of the reconstructed image.

d) To analyse and compare the reconstructed image to the original image.

1.4 Scope of Project

The scope of this project are as follows:

a) Consumers who are just storing data on daily basis.

b) Consumers who prefer to lower their incoming and outgoing data to reduce the amount of bandwidth consumption.

c) Server operators that want to reduce the size data stored so it can reduce the cost of storage hardware.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In recent years, data compression has become more and more important in our daily lives. Powerful but unseen forces like data compression are at work in today's digital world. Without the numerous advancements in compression, our current computer age may not have taken off in the first place.

This chapter has a lot of. Additionally, this chapter will explain the fundamentals of SVD. Image compression, as well as the use of Singular Value Decomposition (SVD) in image compression, is examined.

2.2 Understanding Image Compression

An image compression technique is one in which the original image is encoded using a small number of bits, according to [10].It is the goal of picture compression to reduce the amount of redundant data in an image so that it can be stored or transmitted more efficiently. There are two types of image compression techniques, according to [5].

Lossless image compression:

Lossless compression preserves the integrity of the original image while reducing its file size. Here you can reassemble a compressed image. In this case, it is a reversible procedure. Some files will always be uncompressed by lossless data compression algorithms, and this is the case regardless of the type of algorithm used. [16]

According to [43] in their journal, attempts to compress previously compressed data as well as attempts to compress encrypted data often result in an expansion. This technique is

particularly well-suited for use in the medical field. In the case of medical images, technical drawings, clip art and comics, lossless compression is preferred for archival storage.

With image compression coding, the image is reduced to its smallest possible size, and the decoded image is shown on the monitor with the greatest possible accuracy. Researchers (Joshi et al., 2014) claim that current data compression methods may not have reached their theoretical limit.

Lossy image compression:

This is an irreversible compression approach, according to (Swathi et al., 2017), because the compound image cannot be reversed to its original image. Data and information that are no longer needed are deleted or rearranged in order to reduce and compress the file size. The most common application of this technique is to reduce the file size of extremely large bitmap images. For natural images, such as photographs, the lossy approach is ideally suited because it allows for a significant reduction in the bit rate while retaining some degree of fidelity. Lossy compression has a higher compression ratio than lossless compression. Fig. 1 shows the lossy compression algorithm.

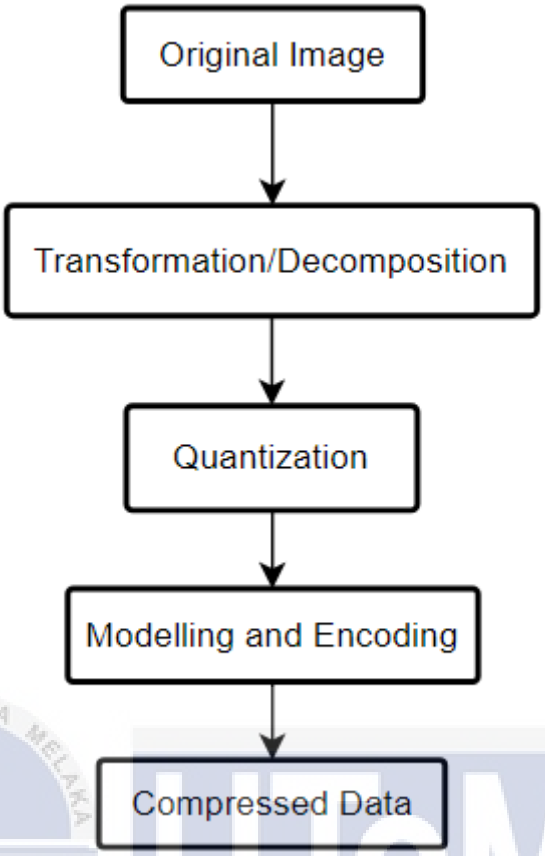


Figure 1 shows the lossy compression algorithm

The original image is processed in four stages: decomposition, quantization, modelling, encoding and compression. The image loses portion of its original size while maintaining its original quality, but in compressed form, as a result of this procedure. Lossy approach, as a schematic approach for compressing photos, could be used with any of the following methods:

i. Chroma subsampling: The human eye is more sensitive to variations in visual brightness than to color variances connected with it. As a result, this method makes use of the human eye by lowering or diminishing the image's chrominance data while enhancing the brightness data. This technique is used to decrease or compress an image to a lesser resolution while maintaining its quality.

ii. Transform coding: This method entails compressing natural data, such as photographic photographs, to a lossy or lossless format. The image is reversible in a lossless method, but the benefit is that it enables greater quantization of the image. Its method converts images into coefficient values, resulting in a low-resolution or low-quality output. There is no information loss, resulting in an equal number of coefficients and pixels altered. The coefficients are quantized, and the output is used to generate the final output using a symbol encoding approach.

iii. Fractal coding: Parts of textures and natural images are turned to mathematical data known as fractal codes, which are then employed to generate the encoded image. When this happens, the image's resolution is lost, making it resolution dependent. The input image's low self-similarity index is blamed for the image's degradation.

Lossy compression that produces negligible differences may be called visually lossless. Singular Value Composition is a type of lossy compression

2.2.1 Data Redundancy

It is the process of reducing the amount of data needed to convey a given amount of information. An important and commercially successful method in Digital Image Processing is picture compression, which attempts to reduce the number of bits needed for an image description by eliminating redundant information. This study by (seshaiah et al., 2016) found three types of redundancy in their datasets:

A. *Coding redundancy*: Coding redundancy consists of variable length code words selected as to match the statistic of the original source. In the case of Digital Image Processing, it is the image itself or processed version of its pixel values. Examples of image coding schemes that explore coding redundancy are the Huffman codes and the Arithmetic coding technique. [43][7]

B. *Spatial redundancy/Inter-pixel redundancy*: Here because the pixels of most 2D intensity arrays are correlated spatially that is each pixel is similar to or independent on neighbouring pixels, information is unnecessarily replicated in the representation of the correlated pixels. Similar to how [6] define which it said that original 2D pixel array is mapped into a different format. Other names of inter-pixel redundancy are spatial redundancy. Examples of this type of redundancy include Constant area coding and many Predictive coding algorithms. [7]

C. *Psycho-visual redundancy:* Human eyes is not fine-tuned to process every band of frequencies. Hence all the incoming data is not responded to with equal sensitivity. Some parts of the information will be more prominent than the others. This fact can be exploited when image redundancies are being removed. Psycho-visual redundancy makes use of this factor. [36]

2.3 Singular Value Decomposition

We'll be focusing on the singular value decomposition method for image compression in this paper. An SVD (Singular Value Decomposition) is an algorithm for reducing a real or complex matrix in linear algebra. [14]SVD outperforms other linear approximation techniques. As stated by [16], the least squares matrix decomposition, the SVD, is the best method for cramming the most signal energy into the smallest number of coefficients. SVM is a stable and

effective method for partitioning the system into a series of linearly independent components, each one of which contributes its own energy, according to the journal in question [16]. In order to produce a smaller image, the SVD implementation makes use of as much redundancy as possible. According to [30], this may work because the algorithm can delete only the parts of the image that are identical to each other. This does not compromise the quality of the image in any way.

Mathematicians use an algorithm known as singular value decomposition (SVD) in order to diagonalize matrices. They also stated [42]that there are numerous advantages to using SVD as an algebraic transform for image processing. These include the ability to compress images with maximum energy and to perform noise filtering and watermarking on images by using two distinct data and noise subspaces

2.3.1 The Techniques of SVD

In this subtopic, we will discuss about the techniques of SVD. In [42], [11] journals, both of them explain the techniques of SVD. The techniques of SVD are as follows.

Assume A is a m x n matrix. When you apply SVD to A, you get a product of orthogonal matrices, diagonal matrices, and another orthogonal matrix.

$$A = USV^T$$

Where,

A = image matrix

U = *m* x *m* orthogonal matrix

S = *m* x *n* matrix

V = *n* x *n* orthogonal matrix

A given matrix is split into a product of orthonormal matrices and a diagonal matrix using Singular Value Decomposition techniques. The theoretical approach for performing SVD is as follows

i) Determine the image matrix's Eigen values. Obtain unique values (square root of Eigen values).

ii) As a diagonal matrix, S matrix, arrange singular values in decreasing order.

iii) Using image matrix, say *A*, obtain AA^T and A^TA.

iv) Determine the Eigen vector of the matrices above. These vectors are transformed into U and V matrices columns.

v) Now, using S, U and V matrices represent A matrix

Figure below will show how matrices created after splitting the original image would look.

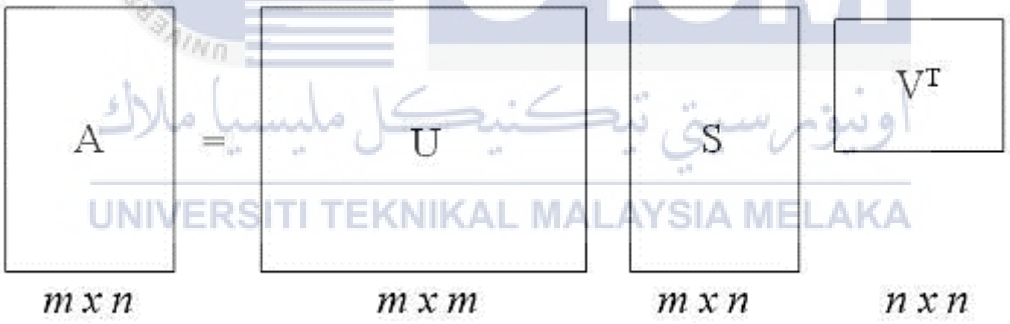


Figure 2 shows the matrices that were created after the original picture matrix was split

2.3.2 Image Compression Using SVD.

In [11] journal, they also explain about image compression using SVD. We must remove the unneeded singular values in the S matrix after obtaining the U, S, and V values of the original image as shown before. After deleting some singular values, create compressed image A with the new diagonal matrix. If we use SVD to represent the image matrix, then A may be written as

$$A = USV^T$$

In a particular way A can also be written as [33]:

$$A = u_1\,\sigma_1\,v_1^T + u_2\,\sigma_2\,v_2^T + \cdots + u_i\,\sigma_i\,v_i^T + u_n\,\sigma_n\,v_n^T$$

or

$$A = \sigma_1\,u_1\,v_1 + \sigma_2\,u_2\,v_2 + \cdots + \sigma_i\,u_i\,v_i + \sigma_n\,u_n\,v_n$$

The terms above are in order of dominance from greatest to the lease.

Singular values with small enough values are dropped when image compression does not perform the sum to the very last Singular Values. The values that fall below the required rank are rounded to the nearest integer[19].

2.3.3 SVD Image Compression Measures

The following performance measure can be used to determine how much an image has been compressed. [27] An image compression method known as SVD can be evaluated by its compression factor and image quality.[9] This ratio can be used to calculate the compression factor for an image. [2]

$$Compression\ Ratio = m * n\ /\ (k * (m + n + 1))$$

It measures the amount of compression an image has.[9]

If you want to compare the quality of compressed image A_k to the original image A, you can use the measurement of Mean Square Error (MSE). [2]

$$MSE = \frac{1}{mn}\sum_{y=1}^{m}\sum_{x=1}^{n}(f_A(x,y) - f_{A_k}(x,y))$$

CHAPTER 3

METHODOLOGY

3.1 Introduction

This project aims to show the process of image processing by using SVD in MATLAB. Methodology serves as a framework to pursue for completion of the project. To ensure that the project is completed successfully, the methods mention in this chapter should be followed to the letter.

3.2 Methodology

This project's flow was detailed in this chapter, from start to finish. To simplify and exemplify methods taken to perform this project, a flow chart is needed. It is created early in the project as a guide to ensure the efficiency of the project. In conjunction with that a Gantt chart is constructed to indicate how the project is planned and whether it is ahead of or behind the time.

3.3 The Project Flowchart

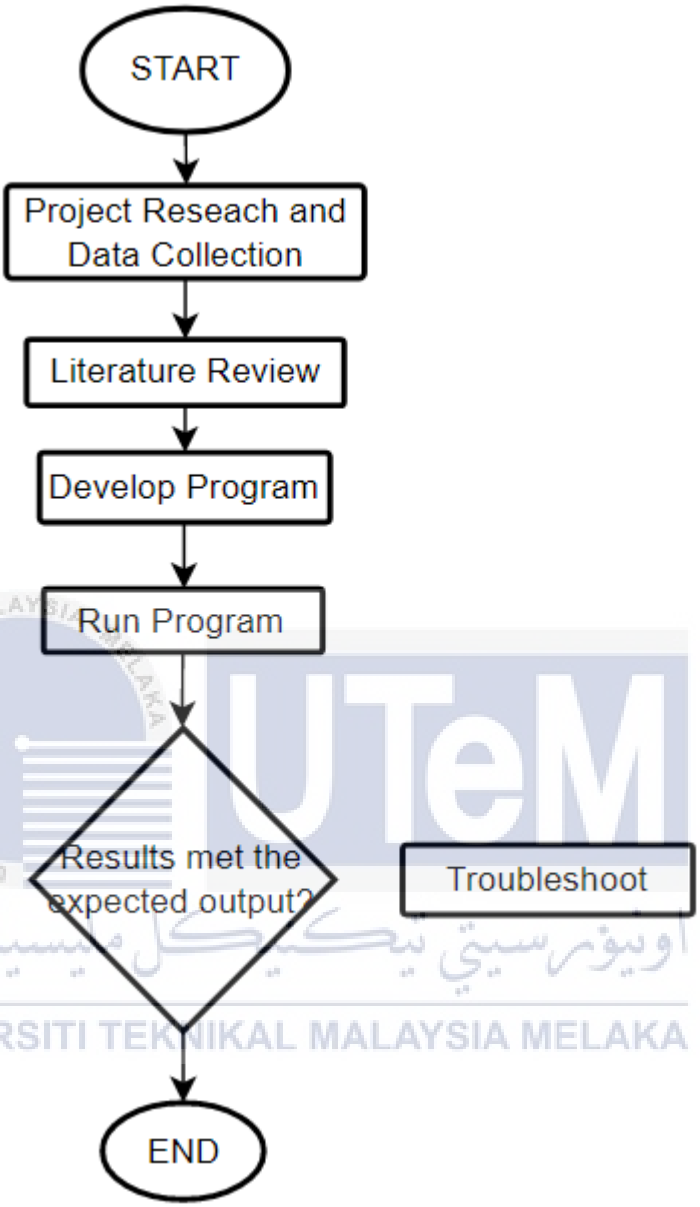


Figure 3 shows the flowchart of the development of this project

It's the project flowchart's job to make sure everything goes according to plan while the project is being completed. In the following sections, we'll go into greater detail about each of these steps.

3.3.1 Research and Data Collection

This project starts with a multitude of research regarding the scope of the project. During this process also some data were collected regarding the interest of this project. Because the issue of SVD being utilised in image compression is an old method compared to other compression methods, some of the data and research publication were a little out of date.

3.3.2 Literature Review

While working on this project, the literature review serves as a learning tool. During this stage, the majority of the knowledge obtained about the project's completion is gathered. Furthermore, a literature review aids in the analysis of the evolution of image compression between prior and current initiatives.

3.3.3 Develop Program

Development stage was the most crucial part of this project. This is where the heart of the project will be constructed and used for implementation. Flowchart below represents the process of the project's program.

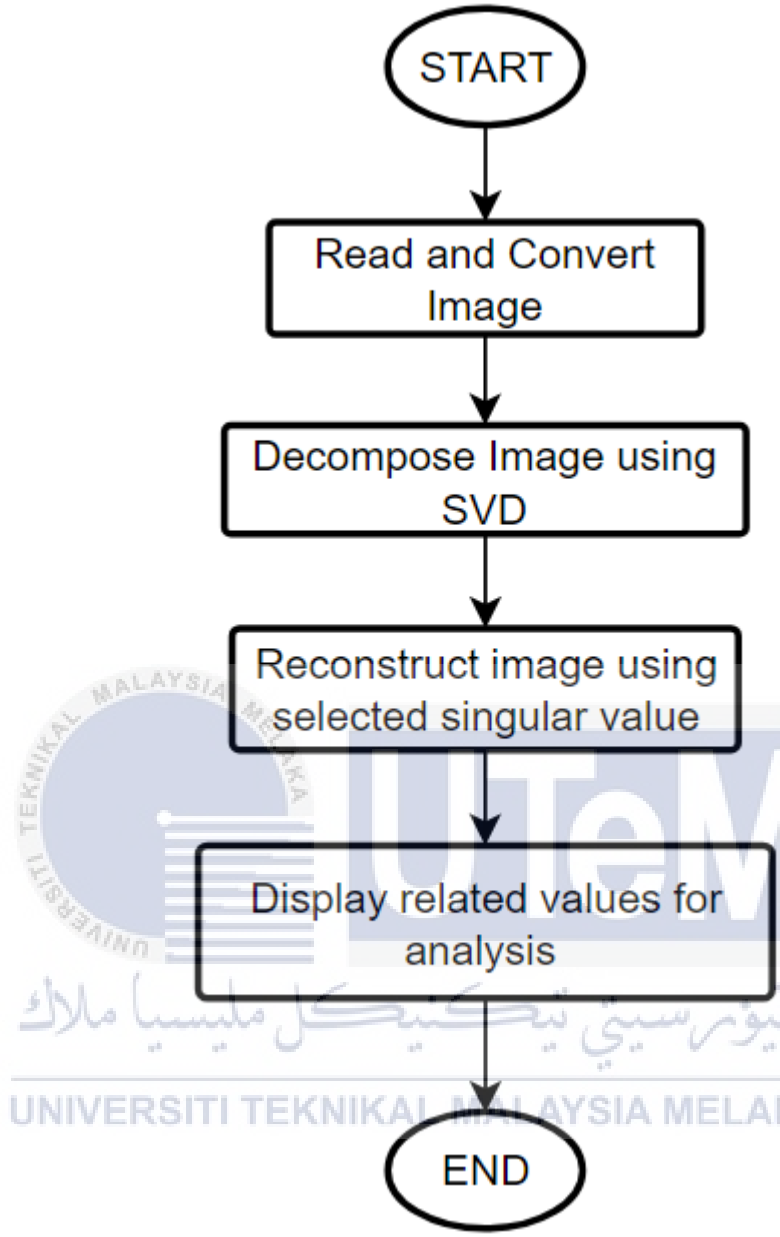


Figure 4 shows the program flowchart

The program's goal is to compress an image using SVD in MATLAB. To begin, the software uses the image previously entered into the code as its starting point. The image is then rescaled and reprojected twice as accurately. The image is then decomposed in MATLAB using the SVD() function. This function will be explored in greater depth in a subsequent section of the document. Decomposition of the image allows it to be reconstructed and the quality of the image can be determined by using any number of

singular values (diagonals of S). In addition to the reconstructed images, all values regarding the intend to analyses the outputs such as compression ratio and PSNR will also be outputted alongside the reconstructed image and all the values will be save it a separate file.

3.3.4 Equipment

As this project consists solely of writing and running a MATLAB program, only a few pieces of equipment are required to complete it. The following items are required:

- MATLAB software
- Personal computer / Desktop

Multi paradigm programming language and numeric computation environment MATLAB (short for "matrix laboratory") is developed by MathWorks. MATLAB allows for the manipulation of matrices, the visualisation of functions and data, the implementation of algorithms, the creation of user interfaces, and the integration of programmes written in other languages. Programming languages run smoothly in MATLAB's modern environment. Debugging, editing, and object-oriented programming can all be done in the same environment, making it ideal for highly complex data structures. What makes this programme such an effective tool for education and research is its unique set of features and capabilities. MATLAB has a number of advantages over traditional computer languages when it comes to solving technological problems. The software package is now considered a standard tool in most institutions and industries around the world. There are a wide variety of computations possible due to the powerful integrated routines An easy-to-understand layout that provides immediate results. This software includes many device boxes useful for optimizatin, signal and image processing, and other tasks, and all of the specific program?Smes are then assembled in toolbox packages.

Figure 5 shows the logo of MATLAB

Figure 6 shows the overview of MATLAB

3.4 Singular Value Decomposition

The content that people want to share can be quickly and efficiently shared using compression techniques. It would take far too long and consume far too much bandwidth if data were not compressed before transmission. It's possible to achieve this by encoding the blocks of pixels that are transmitted.

Singular Value Decomposition had already been used extensively in telecommunications and computer science before this method was introduced. With this method, current publications show a significant improvement in image compression. This method is in high demand for information processing and communication, along with other common multimedia applications and related products.

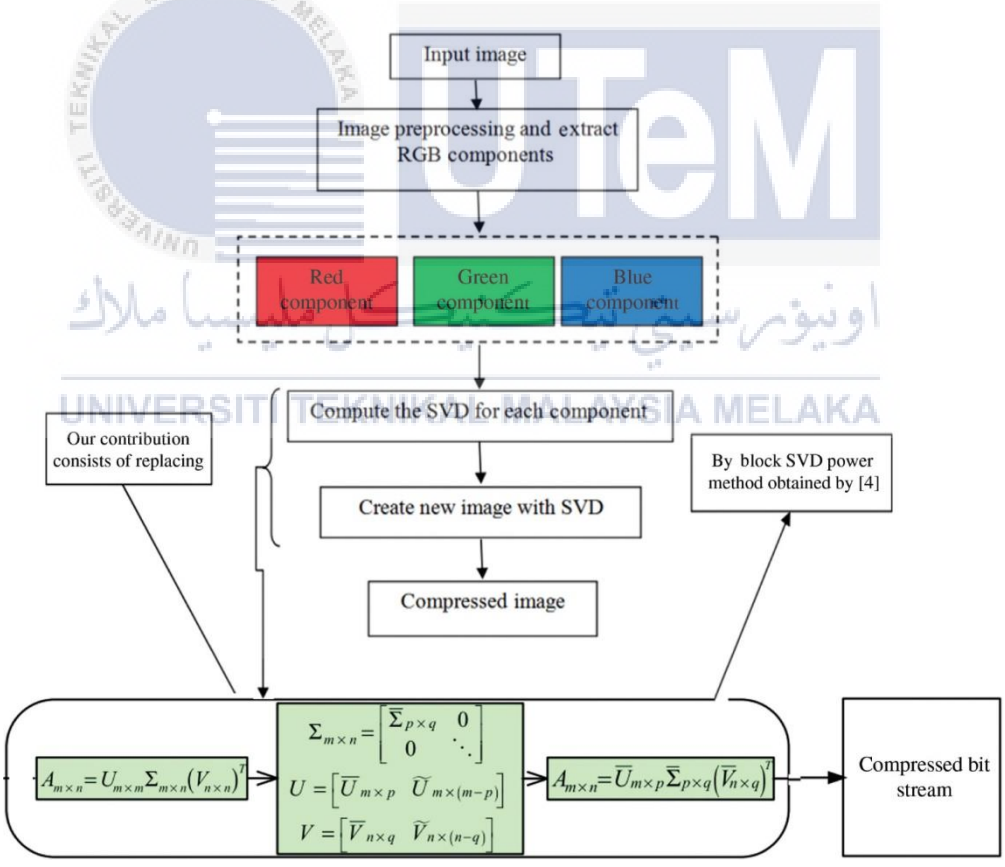


Figure 7 shows the image compression block diagram

A large, highly variable set of data is reduced to a smaller, lower-dimensional space that more clearly reveals the underlying structure of the data and arranges it from the greatest degree of variability down to the slightest degree.

Linear Algebra's singular value decomposition is a factorization of a real or complex matrix, whether square or non-square, according to the definition. There are few steps in mathematical to calculate SVD of matrix:

1. Given an input image matrix A
2. First, calculate AA^T and A^TA
3. Use AA^T to find the eigenvalues and eigenvectors to form the columns of U: $(AA^T - \lambda I)\ \ddot{x} = 0$
4. Use A^TA to find the eigenvalues and eigenvectors to form the columns of V: $(A^TA - \lambda I)\ \ddot{x} = 0$
5. Divide each eigen vectors by its magnitude to form the columns of U and V.
6. Take the square root of the eigenvalues to find the singular values, and arrange them in the diagonal matrix Sin descending order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$
7. In MATLAB: [U,W,V] =svd (A, 0)

3.5 Expected Outcome

It is predicted that the program will have been constructed and run satisfactorily by the end of the project. All of the codes and functions work as expected. The program would also pass all the basic check such as really compression an image and also can compress multiple formats of images.

3.6 Summary

In this chapter, a method for developing an image compression software in MATLAB using the concept of SVD is presented. Main objective of proposed method: create an application that can demonstrate the results of SVD compression in pictures. In addition, the techniques were developed to take advantage of data that is freely available. An image compression algorithm known as SVD has been demonstrated to be feasible, not to produce the best picture compression.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the results and analysis on the image compression by using singular value decomposition with MATLAB. As previously stated, this project uses only computer simulation that uses MATLAB without any additional hardware devices. As a result, this project's simulation will be shown and explained in detail. A simple program was created with MATLAB programming language simulate how would a image compression by using SVD inside MATLAB.

4.2 Analysis Method

With the script working and the program can finally deconstruct an image and reconstruct it with using singular value decomposition, we will need some way to validate the efficiency of said program. By properly analyze the reconstructed image, we can safety conclude the practicality of using singular value decomposition in image compression.

4.2.1 Compression Ratio

An algorithm's ability to reduce the size of its output is measured in terms of its data compression ratio, or compression power. Uncompressed size divided by compressed size is a common way to express it. Data compression ratio is defined as the ratio between the uncompressed size and compressed size:

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

Figure 8 shows the formula of compression ratio

A compression ratio of 10/2 = 5 is often notated as an explicit ratio, 5:1 (read "five" to "one"), or as an implicit ratio, 5/1, when compressing a file's storage size from 10 MB to 2 MB. This formula holds true for both compression and decompression, where the uncompressed size is that of the original and the uncompressed size is that of the reproduction.

4.2.2 Peak Signal-to-Noise Ratio

When comparing two images, a PSNR block calculates the peak signal-to-noise ratio (PSNR), which is expressed in decibels. This ratio is used to compare the original image's quality to that of a compressed one. The compressed or reconstructed image's quality improves with increasing PSNR.

Mean-square error (MSE) and PSNR are used to measure image compression quality and are used to compare the quality of the compression. PSNR measures the peak error, while MSE measures the cumulative squared error between compressed and uncompressed images. The smaller the MSE error, the better.

The PSNR is calculated by first calculating the mean-squared error using the following formula:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2.$$

Figure 9 shows the formula of mean-square error

In the previous equation, *M* and *N* are the number of rows and columns in the input images. Then the block computes the PSNR using the following equation:

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\
&= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE).
\end{aligned}
$$

Figure 10 shows the formula or PSNR

The input image data type's maximum range of variation is represented by the variable R in the previous equation. If the input image is a double-precision floating point data type, then R is 1. " R is 255 if the data type is 8-bit unsigned integer, etc.

Computing PSNR for Colour Images

It is possible to calculate the PSNR of a colour image using a variety of methods. It's possible to calculate the PSNR for colour images by first converting them to one of several colour spaces that separates the intensity (luma) channel from the others, such as YCbCr. The Y (luma) in YCbCr is a weighted average of the R, G, and B components of the chrominance. When it comes to determining a letter's importance, G is given more weight than any other letter. Only the luma channel should be used to calculate PSNR.

4.3 Result and Analysis

For this project, the aim of it is to use SVD as a viable image compression. Since we are comparing the results of original and the compressed image, the output of the program should have the original with the compressed image file. This is because, with both of the image, we can verify whether or not the image has been compressed or not.

To achieve the objective of this project bnwSVD.m and colourSVD.m is proposed as our script to run on and both scripts have its individual purpose. This way we can easily observe and analyses the images.

.

4.3.1 bnwSVD.m

Since JPEG or JPG are the most image format used currently, this program will try to compress a JPEG format image. The image is named image.jpg, which shows a photo of a famous theoretical physicist, Albert Einstein size with dimension of 1000 x 910.

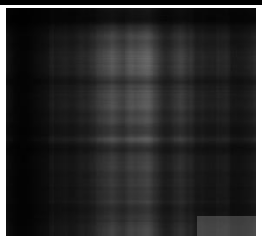
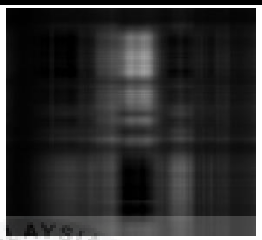
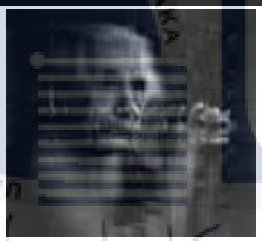
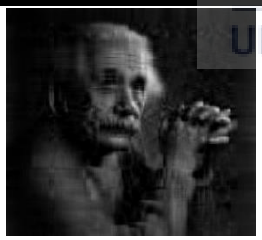
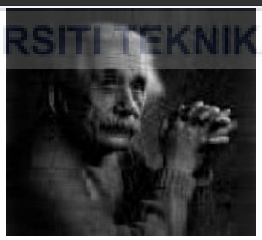
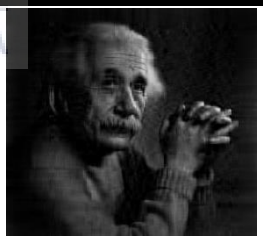
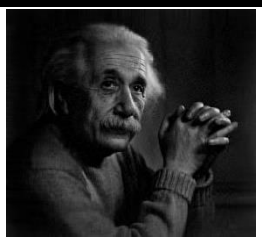
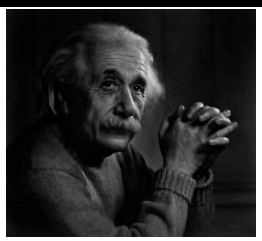
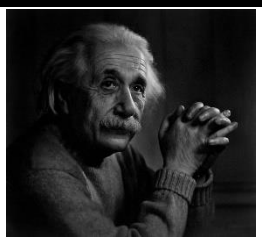
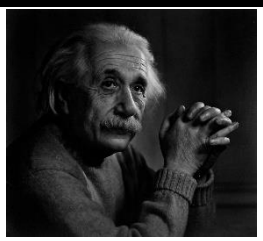
1 MODE	2 MODES	4 MODES	6 MODES
			
Size: 24.2 KB	Size: 27.2 KB	Size: 31.0 KB	Size: 33.9 KB
8 MODES	**10 MODES**	**12 MODES**	**14 MODES**
			
Size: 36.7 KB	Size: 38.8 KB	Size: 40.3 KB	Size: 41.5 KB
16 MODES	**18 MODES**	**20 MODES**	**25 MODES**
			
Size: 42.8 KB	Size: 43.9 KB	Size: 45.1 KB	Size: 47.4 KB
50 MODES	**75 MODES**	**100 MODES**	**ORIGINAL IMAGE**
			
Size: 55.8 KB	Size: 60.9 KB	Size: 64.4 KB	Size: 89.2 KB

Table 1 shows the compilation of all compressed image and the original with respective to its mode and image size for bnwSVD.m

In Table 1 we can see the quality of the compress image starts from the worst to the original image. This is to show the use of multiple modes, so we can compare which photo looks nearly identical to the original photo.

In addition to that, in the same table we can see the size of images decreases as the number of modes decreases. This verify that the program is indeed compressing the image given. We can see that these modes do actually save on memory quite a bit, more than halving the amount of memory used at mode 20.

Mode(s)	Compression Ratio
1	4.46
2	3.98
4	3.48
6	3.18
8	2.94
10	2.79
12	2.68
14	2.60
16	2.52
18	2.46
20	2.39
25	2.28
50	1.94
75	1.77
100	1.68

Table 2 shows the compression ratio of the results from bnwSVD.m

From table above, we can see that the compression ratio at higher number of singular values decreases exponentially which indicates that if we use higher number of singular values, we will get compression ratio not far farm the compression ratio when we use 100 number of singular values. Also, as we can see the compression ratio is only around 1.5:1 to

4.5:1 which is quite low but that seems to be because of the size of the original image is already small.

Mode(s)	PNSR
1	19.17
2	21.18
4	23.26
6	24.85
8	26.05
10	26.93
12	27.67
14	28.32
16	28.83
18	29.31
20	29.75
25	30.78
50	34.18
75	36.43
100	38.24

Table 3 shows the PNSR values of the results from bnwSVD.m

Table 3 shows the value of PSNR between compress and original image increases. Also at higher number of singular values use (modes) the value of PNSR seems to be less significant increase. We can safely say the as the number of singular values used increases the values of PSNR will increase insignificantly which tells us that at higher number of singular values the value of PSNR will not be far from the value of PSNR at mode 100.

As we can see the program can successfully compress an image. Even though this image is using JPG format, theoretically this program also can be use with any image format. However, the program only outputs a black and white image as its output. All the information gathers also only valid if the user wants to compress the image and convert it into black and white.

4.3.2 colourSVD.m

This script solves the problem faced in bnwSVD.m, this program will take any coloured images and reconstruct it with singular value decomposition with full colour. In this case, we are going to use ‘chicken.png’ as our test image. The dimension of ‘chicken.png’ is 3904x2604 with the image size of 12.7mb.

Image ‘chicken.png’ was chosen for few purposes, this to ensure that the program works with PNG photo and the size of image is enormous compared to previous image.

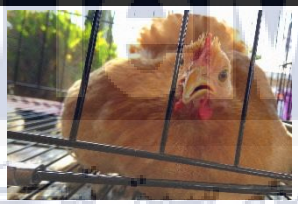
1 MODE	2 MODES	4 MODES	6 MODES
			
Size: 272 KB	Size: 304 KB	Size: 365 KB	Size: 412 KB
8 MODES	10 MODES	12 MODES	14 MODES
			
Size: 441 KB	Size: 462 KB	Size: 483 KB	Size: 503 KB
16 MODES	18 MODES	20 MODES	25 MODES
			
Size: 519 KB	Size: 536 KB	Size: 550 KB	Size: 580 KB
50 MODES	75 MODES	100 MODES	150 MODES
			
Size: 670 KB	Size: 731 KB	Size: 771 KB	Size: 832 KB
200 MODES	250 MODES	300 MODES	350 MODES
			
Size: 868 KB	Size: 901 KB	Size: 932 KB	Size: 960 KB
400 MODES	450 MODES	500 MODES	ORIGINAL IMAGE
			
Size: 986 KB	Size: 0.96 MB	Size: 1.00 MB	Size: 12.7 MB

Table 4 shows the compilation of all compressed image and the original with respective to its mode and image size for colourSVD.m

In Table 2 we can see the quality of the compress image starts from the smallest mode use ‘1’ until the original image. This way we can easily see that around mode 50 and above, the image is indistinguishable from the original image.

Furthermore, from Table 2 we can see the size of images decreases to a great extent at mode 1 until mode 500 compared to the size of the original image. Also, from mode 1 we can see that the size of reconstructed image increases as the mode until mode 500, which follows our theory of using singular value decomposition. This verifies that the program indeed reconstructs an image, and the reconstructed image is compressed by using the method of SVD.

Mode(s)	Compression Ratio
1	47.91
2	42.81
4	35.72
6	31.59
8	29.56
10	28.20
12	26.97
14	25.91
16	25.11
18	24.30
20	23.67
25	22.45
50	19.43
75	17.82
100	16.89
150	15.66
200	15.01
250	14.47
300	13.98
350	13.57
400	13.21
450	12.91
500	12.65

Table 5 shows the compression ratio of the results from colourSVD.m

From Table 5, we can see the compression ratio of the image reconstructed by the program colourSVD.m. We can see that at higher modes, there are less insignificant different on the compression size. Interestingly, the compression ratio of the reconstructed image is much more at 10:1, compared to the compression ratio we see during the run of program bnwSVD.m. However, this might be because of the very large size of original photo used here compared to the one use in bnwSVD.m.

Mode(s)	PNSR
1	13.57
2	15.64
4	17.35
6	18.23
8	19.04
10	19.85
12	20.52
14	21.07
16	21.54
18	21.99
20	22.41
25	23.35
50	26.35
75	28.02
100	29.22
150	30.95
200	32.27
250	33.32
300	34.20
350	34.97
400	35.66
450	36.32
500	36.94

Table 6 shows the PNSR values of the results from colourSVD.m

From the table above we can see the pattern of PSNR is almost identical with the pattern of PSNR in bnwSVD.m despite the compression ratio is much higher here. Also, we can see that at higher values of mode there is less significant increase in the different between

their PSNR values. Furthermore, it shows that any photo above mode 50 is good enough since it barely increases in PSNR and it is visually good enough.

4.4 Summary

Depending on the task, image compression can be a basic method of image processing. Singular value decomposition (SVD) is used in this project to reduce the image size. As a result, the compressed image quality can be maintained given the reduced image volume. The paper, journal, web, and other studies were the first steps in this project's development. This is the process of learning by reading and analyzing the approach proposed in such reviews of literature. It demonstrates the differences in approaches and contains all the pertinent information required to carry out the project as planned.

A flowchart is used to track the progress of this project throughout its development. As long as the project flowchart is used, hiccups can be improvised, and flaws can be controlled. MATLAB software is the primary tool for this project, as previously stated. This software alone will be sufficient to meet all of the project's goals. In order to study the use of singular value decomposition (SVD) in image compression, we have already written a script.

Earlier in this chapter the result of the project is shown where both of the script bnwSVD.m and colourSVD.m successfully deconstructed and reconstructed the image that was assigned to it. Also, both scripts have proven that every reconstructed image has been decrease in its size which prove that the image is compressed. In bnwSVD.m we can see that there is no high number of compression ratio compared to colourSVD.m, but this might be because the size of the original image.

While using both script is using different type of image format (JPG for bnwSVD.m and PNG for colourSVD.m) it is doubt that any image format would work when using the scripts. The only thing matter if the user wants the output in black and white or colour image.

In both scripts, we can see the curve of PSNR graph is pretty similar. Even though colourSVD.m is run with more modes (up to 500 mode) where is negligible difference between both of the program's PNSR graph. Each graph shows a minor change in PNSR value after mode 50.

Overall, it can be concluded that at around 50 modes or above, there it is safe to say the reconstructed image will be cut from the same cloth when comparing with the original image. The program also achieves the purpose of compressing an image with using the method of SVD. It is also shown that the method of SVD for compressing image is very viable.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter will conclude all the results, analysis, observation, objectives and evaluation needed based on project completion. The suggestion of future work will also\sbe addressed because of this issue.

5.2 Conclusion

As a result, this project proves that the method of singular value decomposition or SVD can be used for compressing images. MATLAB also shows that is it a great platform to use for any image processing needed. Both scripts were written in MATLAB programming language and run on the MATLAB software itself.

Also, with success of the program reconstructing the image that was deconstructed by using SVD, we can study the effectiveness of the method. The result was very pleasing as the image ‘chicken.png’ were able to obtain a whopping compression ratio of 10:1 while being able to be very indistinguishable from the original image. With use of some clever functions, we were also able to study and analyze the differences between both image by using compression ratio and PNSR values.

There is so much potential when using SVD in image compression. In conclusion, the project objectives were met, and the project is a success.

5.3 Future Works

For future improvements, the accuracy of the analysis and the script itself could be enhanced as follows:

- Wider range of image format tested with both scripts
- Some sort of combination of both script and to use it as a program for end user
- Better analysis methods

REFERENCES

[1] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, 2017.

[2] L. Cao, "Singular Value Decomposition Applied To Digital Image Processing," pp. 1–15.

[3] M. Rehman, M. Sharif, and M. Raza, "Image compression: A survey," *Res. J. Appl. Sci. Eng. Technol.*, vol. 7, no. 4, pp. 656–672, 2014, doi: 10.19026/rjaset.7.303.

[4] G. Vijayvargiya, S. Silakari, and R. Pandey, "A Survey: Various Techniques of Image Compression," vol. 11, no. 10, 2013, [Online]. Available: http://arxiv.org/abs/1311.6877.

[5] I. Charles and I. C. Udousoro, "The Application of Selective Image Compression Techniques," vol. 6, no. 4, pp. 116–120, 2019, doi: 10.11648/j.se.20180604.12.

[6] A. Hameed Khaleel, I. Q. Abduljaleel, and A. H. Khaleel, "Significant Medical Image Compression Techniques: A Review," *TELKOMNIKA Telecommun. Comput. Electron. Control*, vol. 19, no. 5, 2021, doi: 10.12928/TELKOMNIKA.v18i1.xxxxx.

[7] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 33, no. 2, pp. 119–140, 2021, doi: 10.1016/j.jksuci.2018.05.006.

[8] M. R. e Souza and H. Pedrini, "Adaptive Lossy Image Compression Based on Singular Value Decomposition," *J. Signal Inf. Process.*, 2019, doi: 10.4236/jsip.2019.103005.

[9] M. B. V. seshaiah, M. R. K N, and S. Michahial, "Image Compression using Singular Value Decomposition," *Ijarcce*, vol. 5, no. 12, pp. 208–211, 2016, doi: 10.17148/ijarcce.2016.51246.

[10] M. A. Joshi, M. S. Raval, Y. H. Dandawate, K. R. Joshi, and S. P. Metkar, "Introduction to Image Compression," *Image and Video Compression*, pp. 18–21, 2014, doi: 10.1201/b17738-4.

[11] H. R. Swathi, S. Sohini, Surbhi, and G. Gopichand, "Image compression using singular value decomposition," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 263, no. 4, 2017, doi: 10.1088/1757-899X/263/4/042082.

[12] R. Ashin, A. Morimoto, R. Vaillancourt, and M. Nagase, "Image compression with multiresolution singular value decomposition and other methods," *Math. Comput. Model.*, vol. 41, no. 6–7, pp. 773–790, 2005, doi: 10.1016/j.mcm.2003.12.014.

[13] Zaida Victoria Narcisa Betancourth Aragón, “No Title□□□□□□□□□ ”,ئئئئئ □□□□□, vol. 3, no. 2, pp. 148–150, 2010, [Online]. Available: https://repositorio.flacsoandes.edu.ec/bitstream/10469/2461/4/TFLACSO-2010ZVNBA.pdf.

[14] K. El Asnaoui, “Image Compression Based on Block SVD Power Method,” *J. Intell. Syst.*, vol. 29, no. 1, pp. 1345–1359, 2020, doi: 10.1515/jisys-2018-0034.

[15] F. Yeganegi, V. Hassanzade, and S. M. Ahadi, “Comparative Performance Evaluation of SVD-Based Image Compression,” *26th Iran. Conf. Electr. Eng. ICEE 2018*, pp. 464–469, 2018, doi: 10.1109/ICEE.2018.8472544.

[16] K. Mounika, D. S. N. Lakshmi, and K. Alekya, “SVD Based Image Compression,” *Int. J. Eng. Res. Gen. Sci.*, 2015.

[17] L. Keviczky, R. Bars, J. Hetthéssy, and C. Bányász, “Control engineering: MATLAB exercises,” *Adv. Textb. Control Signal Process.*, pp. 1–275, 2019, doi: 10.1007/978-981-10-8321-1.

[18] A. Thakur, R. Kumar, A. Bath, and J. Sharma, “Design of Image Compression Algorithm Using Matlab,” no. February, 2014.

[19] K. M. Aishwarya, R. Ramesh, P. M. Sobarad, and V. Singh, “Lossy image compression using SVD coding algorithm,” *Proc. 2016 IEEE Int. Conf. Wirel. Commun. Signal Process. Networking, WiSPNET 2016*, pp. 1384–1389, 2016, doi: 10.1109/WiSPNET.2016.7566363.

[20] U. Bhade, S. Kumar, P. Dwivedy, S. Soofi, and A. Ray, “Comparative study of DWT, DCT, BTC and SVD techniques for image compression,” *Int. Conf. Recent Innov. Signal Process. Embed. Syst. RISE 2017*, vol. 2018-January, pp. 279–283, 2018, doi: 10.1109/RISE.2017.8378167.

[21] J. Izadian and M. Jalili, “A Hybrid SVD Method Using Interpolation Algorithms for Image Compression A Hybrid SVD Method Using Interpolation Algorithms for Image Compression,” no. June, 2014.

[22] D. I. Processing and M. Page, “Introduction to image processing in Matlab,” *Image Process.*, no. March, pp. 1–18, 2011.

[23] “Lossless Image Compression Using Wavelets.” 2000.

[24] K. V. Indukuri, A. A. Ambekar, and A. Sureka, “Similarity analysis of patent claims using natural language processing techniques,” *Proc. - Int. Conf. Comput. Intell. Multimed. Appl. ICCIMA 2007*, vol. 4, pp. 169–175, 2008, doi: 10.1109/ICCIMA.2007.386.

[25] S. K. Singh and S. Kumar, “A framework to design novel SVD based color image compression,” *EMS 2009 - UKSim 3rd Eur. Model. Symp. Comput. Model. Simul.*, pp. 235–240, 2009, doi: 10.1109/EMS.2009.100.

[26] S. K. Singh and S. Kumar, “Singular value decomposition based sub-band decomposition and multi-resolution (SVD-SBD-MRR) representation of digital colour images,” *Pertanika J. Sci. Technol.*, vol. 19, no. 2, pp. 229–235, 2011.

[27] J. P. Pandey and L. Singh Umrao, “Digital Image Processing using Singular Value Decomposition,” *SSRN Electron. J.*, pp. 2–4, 2019, doi: 10.2139/ssrn.3350278.

[28] W. Renkjumnong, “ScholarWorks @ Georgia State University SVD and PCA in Image Processing,” 2007.

[29] B. Khan, M. Poonam, and M. M. Talib, “Matlab Based Image Compression Using Various Algorithm,” *Int. J. Trend Sci. Res. Dev.*, vol. Volume-2, no. Issue-4, pp. 1638–1642, 2018, doi: 10.31142/ijtsrd14394.

[30] H. R. Swathi, S. Sohini, Surbhi, and G. Gopichand, “Image compression using singular value decomposition,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 263, no. 4, 2017, doi: 10.1088/1757-899X/263/4/042082.

[31] A. J. Hussain, A. Al-Fayadh, and N. Radi, “Image compression techniques: A survey in lossless and lossy algorithms,” *Neurocomputing*, 2018, doi: 10.1016/j.neucom.2018.02.094.

[32] M. Singh, S. Kumar, S. Singh, and M. Shrivastava, “Various Image Compression Techniques: Lossy and Lossless,” *Int. J. Comput. Appl.*, vol. 142, no. 6, pp. 23–26, 2016, doi: 10.5120/ijca2016909829.

[33] S. Verma and J. P. Krishna, “Image Compression and Linear Algebra Mathematics Behind the SVD,” 2013.

[34] A. AbuBaker, M. Eshtay, and M. AkhoZahia, “Comparison Study of Different Lossy Compression Techniques Applied on Digital Mammogram Images,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 12, pp. 149–155, 2016, doi: 10.14569/ijacsa.2016.071220.

[35] V. Cheepurupalli, S. Tubbs, K. Boykin, and N. Naheed, “Comparison of SVD and FFT in image compression,” *Proc. - 2015 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2015*, pp. 526–530, 2016, doi: 10.1109/CSCI.2015.56.

[36] T. R. Hsing and K. H. Tzou, “Video Compression Techniques: a Review.,” vol. 5, no. 1, pp. 1521–1526, 1984, doi: 10.14445/22315381/ijett-v13p264.

[37] M. Singh, S. Kumar, S. Singh, and M. Shrivastava, “Various Image Compression Techniques: Lossy and Lossless,” *Int. J. Comput. Appl.*, vol. 142, no. 6, pp. 23–26, 2016, doi: 10.5120/ijca2016909829.

[38] F. Mentzer and M. Tschannen, “1. Introduction Fabian Mentzer ETH Zurich,” *IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020.

[39] H. R. Swathi, S. Sohini, Surbhi, and G. Gopichand, “Image compression using singular value decomposition,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 263, no. 4, 2017, doi: 10.1088/1757-899X/263/4/042082.

[40] S. Anwar Lashari, R. Ibrahim, N. S. A. Md Taujuddin, N. Senan, and S. Sari, “Thresholding And Quantization Algorithms for Image Compression Techniques: A Review,” *Asia-Pacific J. Inf. Technol. Multimed.*, vol. 07, no. 01, pp. 83–89, 2018, doi: 10.17576/apjitm-2018-0701-07.

[41] P. Aguilera, “Comparison of different image compression formats ECE 533 Project Report,” pp. 5–9, 2007.

[42] R. A, “SVD Based Image Processing Applications: State of The Art, Contributions and Research Challenges,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 7, pp. 26–34, 2012, doi: 10.14569/ijacsa.2012.030703.

[43] R. Kaur and P. Choudhary, “A Review of Image Compression Techniques,” *Int. J. Comput. Appl.*, 2016, doi: 10.5120/ijca2016909658.

APPENDICES

Activity	W1	W2	W3	W4	W6	W7	W8	W9	W10	W11	W12	W13	W14
Title Confirmation													
Introduction													
Journal Study													
Literature Review													
Methodology													
Initial Results													
Full Report and Turnitin													
Slide Preparation													
Presentation of PSM 1													

Appendix A Gantt Chart of Progression of PSM 1

Activity	W1	W2	W3	W4	W5	W7	W8	W9	W10	W11	W12	W13	W14
Planning Requirement Project													
Analysis of PSM 1													
Improving program													
Results and Analysis													
Conclusion													
Submit Draft Report													
Finalize Report													
Create Poster													
Slide Preparation and Video													
Presentation of PSM 2													

Appendix B Gantt Chart of Progression of PSM 2

No	Task Name	Start Date	End Date	Duration
1.	Briefing	17th March 2021	17th March 2021	1 day
2.	Title Selection	18th March 2021	18th March 2021	1 day
3.	Project title Discussion with SV	25th March 2021	25th March 2021	1 day
4.	Research and learning about Image Compression	18th March 2021	30th June 2021	105 days
5.	Research and learning about Singular Value Decomposition	18th March 2021	30th June 2021	105 days
6.	Chapter 1: Introduction	6th May 2021	12th May 2021	7 days
7.	Journal Study	18th March 2021	26th May 2021	70 days
8.	Chapter 2: Literature Writing	13th May 2021	26th May 2021	14 days
9.	Chapter 3: Methodology	27th May 2021	3rd June 2021	8 days
10.	Design Initial Program	4th June 2021	13th June 2021	10 days
11.	Chapter 4: Initial Result and Analysis	14th June 2021	16th June 2021	3 days
12.	Full report checks and submit	17th June 2021	17th June 2021	1 day
13.	Slide Preparation & Video Making	18th June 2021	18th June 2021	1 day
14.	Submit presentation video and documentation to panels and supervisor.	18th June 2021	18th June 2021	1 day
15.	Presentation of PSM 1	21st June 2021	21st June 2021	1 day

Appendix C Milestones of PSM 1

No	Task Name	Start Date	End Date	Duration
1.	PSM 2 Implementation Briefing	6th Oct 2021	6th Oct 2021	1 day
2.	PSM 1 Revision	13th Oct 2021	19th Oct 2021	6 days
3.	Writing the script for colourSVD.m	20th Oct 2021	1st Nov 2021	12 days
4.	Progress update with SV	3rd Nov 2021	3rd Nov 2021	1 day
5.	Completing colourVD.m and improving bnwSVD.m	10th Nov 2021	6th Jan 2022	57 days
6.	Chapter 4: Result and Discussion	15th Dis 2021	6th Jan 2022	22 days
7.	Chapter 5: Conclusion and Recommendations	1st Jan 2022	5th Jan 2022	4 days
8.	Submit first draft for SV evaluation	7th Jan 2022	7th Jan 2022	1 day
9.	Polishing report based on SV's comment	8th Jan 2022	10th Jan 2022	2 days
10.	Creating poster	6th Jan 2022	11th Jan 2022	5 days
11.	Creating video for presentation	6th Jan 2022	11th Jan 2022	5 days
12.	Second submission of report and poster for SV's evaluation	10th Jan 2022	10th Jan 2022	1 day
13.	Submit video and finalize report with documentation to panels and SV	11th Jan 2022	11th Jan 2022	1 day
14.	Presentation of PSM 2 (QnA session)	12th Jan 2022	12th Jan 2022	1 day
15.	Final submission of report on ePSM platform	31st Jan 2022	31st Jan 2022	1 day

Appendix D Milestones of PSM 2

```
close all
clear all
clc

displayError = [];
numofSVals = [];
valuePSNR = [];
imgSize = [];
compressionRatio = [];
bnwData = [];

a=imread('!!ae.jpg');
fileinfo = dir('!!ae.jpg');
sizeofImg = (fileinfo.bytes/1024);
original = sizeofImg;
bnwData(16).name =sprintf('Original.jpg');
bnwData(16).size = sizeofImg;
compressionratio = original/sizeofImg;
bnwData(16).compressionRatio = compressionratio;

a=rgb2gray(a);
[rows, columns] = size(a); %for psnr
aDouble=double(a);
%imwrite(uint8(aDouble), '!!aeoriginal.jpg');
[U,S,V]=svd(aDouble);

N = 1;
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
D=U*C*V';
figure;
buffer = sprintf('Image output using %d singular values', N);
imshow(uint8(D));
imwrite(uint8(D), sprintf('%dbw.jpg', N));

%getting image size
filename = sprintf('%dbw.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

%calculating and adding PSNR
squaredErrorImage = (double(a) - double(D)) .^ 2;
mse = sum(sum(squaredErrorImage)) / (rows * columns);
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

%calculating error rate & storing its values
error=sum(sum((aDouble-D).^2));
displayError = [displayError; error];
numofSVals = [numofSVals; N];

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];
```

```
%saving values in bnwData
valuePSNR = [valuePSNR; PSNR];
bnwData(N).name =sprintf('Image %dbw.jpg',N);
bnwData(N).size = sizeofImg;
bnwData(N).mse = mse;
bnwData(N).pnsr = PSNR;
bnwData(N).compressionRatio = compressionratio;
imgSize = [imgSize; sizeofImg];

figure;
for N=2:2:20
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
D=U*C*V';
subplot(2,5,N/2);
buffer = sprintf('Image output using %d singular values', N);
imshow(uint8(D));
imwrite(uint8(D), sprintf('%dbw.jpg', N));

%getting image size
filename = sprintf('%dbw.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

squaredErrorImage = (double(a) - double(D)) .^ 2;
mse = sum(sum(squaredErrorImage)) / (rows * columns);
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

error=sum(sum((aDouble-D).^2));
displayError = [displayError; error];
numofSVals = [numofSVals; N];

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
bnwData((N/2)+1).name =sprintf('Image %dbw.jpg',N);
bnwData((N/2)+1).size = sizeofImg;
bnwData((N/2)+1).mse = mse;
bnwData((N/2)+1).pnsr = PSNR;
bnwData((N/2)+1).compressionRatio = compressionratio;

end
figure;
for N=25:25:100
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
D=U*C*V';
subplot(2,2,N/25);
buffer = sprintf('Image output using %d singular values', N);
imshow(uint8(D));
```

```
imwrite(uint8(D), sprintf('%dbw.jpg', N));

%getting image size
filename = sprintf('%dbw.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

squaredErrorImage = (double(a) - double(D)) .^ 2;
mse = sum(sum(squaredErrorImage)) / (rows * columns);
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

error=sum(sum((aDouble-D).^2));
displayError = [displayError; error];
numofSVals = [numofSVals; N];

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
bnwData((N/25)+11).name =sprintf('Image %dbw.jpg',N);
bnwData((N/25)+11).size = sizeofImg;
bnwData((N/25)+11).mse = mse;
bnwData((N/25)+11).pnsr = PSNR;
bnwData((N/25)+11).compressionRatio = compressionratio;
end

figure;
title('Error rate in compression')
plot(numofSVals, displayError);
xlabel('Number of Singular Values used');
ylabel('Error rate between compress and original image');
grid on;

figure;
title('PSNR values in compression')
plot(numofSVals, valuePSNR);
ylabel('Value of PSNR between compress and original image');
xlabel('Number of Singular Values used');
grid on

figure;
title('Compression Ratio')
plot(numofSVals,compressionRatio);
ylabel('Compression Ratio');
xlabel('Number of Singular Values used');
grid on

%seperate files for values
save ('fileSize','bnwData');
```


Appendix E script for bnwSVD.m

```
close all
clear all
clc

displayError = [];
numofSVals = [];
colourData = [];
valuePSNR = [];
sizeDifference = [];
compressionRatio = [];
psnrData = [];
imgSize = [];

filename = 'chicken.png';
fileinfo = dir('chicken.png');
sizeofImg = (fileinfo.bytes/1024);
original = sizeofImg;
colourData(24).name =sprintf('Original.jpg');
colourData(24).size = sizeofImg;
compressionratio = original/sizeofImg;
colourData(24).compressionRatio = compressionratio;

[X, map] = imread(filename);
figure('Name','ORIGINAL component of the imported image');
imshow(X);
%imwrite(X, '!original.jpg');
[rows columns ~] = size(X);

R = X(:,:,1);
G = X(:,:,2);
B = X(:,:,3);

Rimg = cat(3, R, zeros(size(R)), zeros(size(R)));
Gimg = cat(3, zeros(size(G)), G, zeros(size(G)));
Bimg = cat(3, zeros(size(B)), zeros(size(B)), B);
%{
figure('Name','RED component of the imported image');
imshow(Rimg);
imwrite(Rimg, '!red.jpg');
figure('Name','GREEN component of the imported image');
imshow(Gimg);
imwrite(Gimg, '!green.jpg');
figure('Name','BLUE component of the imported image');
imshow(Bimg);
imwrite(Bimg, '!blue.jpg');
%}
Red =double(R);
Green = double(G);
Blue = double(B);

N = 1;
% Compute values for the red image
[U,S,V]=svd(Red);
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dr=U*C*V';
%{ Rebuild the data back into a displayable image and show it
%figure;
```

```
buffer = sprintf('Red image output using %d singular values', N);
Rimg = cat(3, Dr, zeros(size(Dr)), zeros(size(Dr)));
%imshow(uint8(Rimg));
imwrite(uint8(Rimg), sprintf('%dred.jpg', N));
title(buffer);
% Compute values for the green image
[U2, S2, V2]=svd(Green);
C = S2;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dg=U2*C*V2';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Green image output using %d singular values', N);
Gimg = cat(3, zeros(size(Dg)), Dg, zeros(size(Dg)));
%imshow(uint8(Gimg));
imwrite(uint8(Gimg), sprintf('%dgreen.jpg', N));
title(buffer);
% Compute values for the blue image
[U3, S3, V3]=svd(Blue);
C = S3;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Db=U3*C*V3';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Blue image output using %d singular values', N);
Bimg = cat(3, zeros(size(Db)), zeros(size(Db)), Db);
%imshow(uint8(Bimg));
imwrite(uint8(Bimg), sprintf('%dblue.jpg', N));
title(buffer);

% Thake the data from the Red, Green, and Blue image
% Rebuild a colored image with the corresponding data and show it
figure;
buffer = sprintf('Colored image output using %d singular values', N);
Cimg = cat(3, Dr, Dg, Db);
imshow(uint8(Cimg));
imwrite(uint8(Cimg), sprintf('%dcolor.jpg', N));
title(buffer);

%Calculating and adding PSNR

% Calculate mean square error of R, G, B.
mseRImage = (double(X(:,:,1)) - double(Cimg(:,:,1))) .^ 2;
mseGImage = (double(X(:,:,2)) - double(Cimg(:,:,2))) .^ 2;
mseBImage = (double(X(:,:,3)) - double(Cimg(:,:,3))) .^ 2;
mseR = sum(sum(mseRImage)) / (rows * columns);
mseG = sum(sum(mseGImage)) / (rows * columns);
mseB = sum(sum(mseBImage)) / (rows * columns);
% Average mean square error of R, G, B.
mse = (mseR + mseG + mseB)/3;
% Calculate PSNR (Peak Signal to noise ratio).
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

%getting image size
```

```
filename = sprintf('%dcolor.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

%saving values
numofSVals = [numofSVals; N];
psnrData = [psnrData; PSNR];
valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
colourData(N).name =sprintf('%dcolor.jpg',N);
colourData(N).size = sizeofImg;
colourData(N).compressionRatio = compressionratio;
colourData(N).mse = mse;
colourData(N).pnsr = PSNR;



for N=2:2:20
% Recompute modes for the red image - already solved by SVD above
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dr=U*C*V';
% Rebuild the data back into a displayable image and show it

%figure;
buffer = sprintf('Red image output using %d singular values', N);
Rimg = cat(3, Dr, zeros(size(Dr)), zeros(size(Dr)));
%imshow(uint8(Rimg));
imwrite(uint8(Rimg), sprintf('%dred.jpg', N));
title(buffer);
% Recompute modes for the green image - already solved by SVD above
C = S2;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dg=U2*C*V2';
% Rebuild the data back into a displayable image and show it

%figure;
buffer = sprintf('Green image output using %d singular values', N);
Gimg = cat(3, zeros(size(Dg)), Dg, zeros(size(Dg)));
%imshow(uint8(Gimg));
imwrite(uint8(Gimg), sprintf('%dgreen.jpg', N));
title(buffer);
% Recompute modes for the blue image - already solved by SVD above
C = S3;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Db=U3*C*V3';
% Rebuild the data back into a displayable image and show it

%figure;
buffer = sprintf('Blue image output using %d singular values', N);
Bimg = cat(3, zeros(size(Db)), zeros(size(Db)), Db);
```

```
%imshow(uint8(Bimg));
imwrite(uint8(Bimg), sprintf('%dblue.jpg', N));
title(buffer);
% Thake the data from the Red, Green, and Blue image

% Rebuild a colored image with the corresponding data and show it
figure;
buffer = sprintf('Colored image output using %d singular values', N);
Cimg = cat(3, Dr, Dg, Db);
imshow(uint8(Cimg));
imwrite(uint8(Cimg), sprintf('%dcolor.jpg', N));
title(buffer);

%Calculating and adding PSNR

% Calculate mean square error of R, G, B.
mseRImage = (double(X(:,:,1)) - double(Cimg(:,:,1))) .^ 2;
mseGImage = (double(X(:,:,2)) - double(Cimg(:,:,2))) .^ 2;
mseBImage = (double(X(:,:,3)) - double(Cimg(:,:,3))) .^ 2;
mseR = sum(sum(mseRImage)) / (rows * columns);
mseG = sum(sum(mseGImage)) / (rows * columns);
mseB = sum(sum(mseBImage)) / (rows * columns);
% Average mean square error of R, G, B.
mse = (mseR + mseG + mseB)/3;
% Calculate PSNR (Peak Signal to noise ratio).
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

%getting image size
filename = sprintf('%dcolor.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

%saving values
numofSVals = [numofSVals; N];
psnrData = [psnrData; PSNR];
valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
colourData((N/2)+1).name =sprintf('%dcolor.jpg',N);
colourData((N/2)+1).size = sizeofImg;
colourData((N/2)+1).compressionRatio = compressionratio;
colourData((N/2)+1).mse = mse;
colourData((N/2)+1).pnsr = PSNR;

end
for N=25:25:100
% Recompute modes for the red image - already solved by SVD above
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dr=U*C*V';
% Rebuild the data back into a displayable image and show it
```

```
%figure;
buffer = sprintf('Red image output using %d singular values', N);
Rimg = cat(3, Dr, zeros(size(Dr)), zeros(size(Dr)));
%imshow(uint8(Rimg));
imwrite(uint8(Rimg), sprintf('%dred.jpg', N));
title(buffer);
% Recompute modes for the green image - already solved by SVD above
C = S2;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dg=U2*C*V2';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Green image output using %d singular values', N);
Gimg = cat(3, zeros(size(Dg)), Dg, zeros(size(Dg)));
%imshow(uint8(Gimg));
imwrite(uint8(Gimg), sprintf('%dgreen.jpg', N));
title(buffer);
% Recompute modes for the blue image - already solved by SVD above
C = S3;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Db=U3*C*V3';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Blue image output using %d singular values', N);
Bimg = cat(3, zeros(size(Db)), zeros(size(Db)), Db);
%imshow(uint8(Bimg));
imwrite(uint8(Bimg), sprintf('%dblue.jpg', N));
title(buffer);
% Thake the data from the Red, Green, and Blue image
% Rebuild a colored image with the corresponding data and show it
figure;
buffer = sprintf('Colored image output using %d singular values', N);
Cimg = cat(3, Dr, Dg, Db);
imshow(uint8(Cimg));
imwrite(uint8(Cimg), sprintf('%dcolor.jpg', N));
title(buffer);

%Calculating and adding PSNR

% Calculate mean square error of R, G, B.
mseRImage = (double(X(:,:,1)) - double(Cimg(:,:,1))) .^ 2;
mseGImage = (double(X(:,:,2)) - double(Cimg(:,:,2))) .^ 2;
mseBImage = (double(X(:,:,3)) - double(Cimg(:,:,3))) .^ 2;
mseR = sum(sum(mseRImage)) / (rows * columns);
mseG = sum(sum(mseGImage)) / (rows * columns);
mseB = sum(sum(mseBImage)) / (rows * columns);
% Average mean square error of R, G, B.
mse = (mseR + mseG + mseB)/3;
% Calculate PSNR (Peak Signal to noise ratio).
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

%getting image size
filename = sprintf('%dcolor.jpg',N);
fileinfo = dir(filename);
```

```
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

%saving values
numofSVals = [numofSVals; N];
psnrData = [psnrData; PSNR];
valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
colourData((N/25)+11).name =sprintf('%dcolor.jpg',N);
colourData((N/25)+11).size = sizeofImg;
colourData((N/25)+11).compressionRatio = compressionratio;
colourData((N/25)+11).mse = mse;
colourData((N/25)+11).pnsr = PSNR;

end
for N=150:50:500
% Recompute modes for the red image - already solved by SVD above
C = S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dr=U*C*V';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Red image output using %d singular values', N);
Rimg = cat(3, Dr, zeros(size(Dr)), zeros(size(Dr)));
%imshow(uint8(Rimg));
imwrite(uint8(Rimg), sprintf('%dred.jpg', N));
title(buffer);
% Recompute modes for the green image - already solved by SVD above
C = S2;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Dg=U2*C*V2';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Green image output using %d singular values', N);
Gimg = cat(3, zeros(size(Dg)), Dg, zeros(size(Dg)));
%imshow(uint8(Gimg));
imwrite(uint8(Gimg), sprintf('%dgreen.jpg', N));
title(buffer);
% Recompute modes for the blue image - already solved by SVD above
C = S3;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
Db=U3*C*V3';
% Rebuild the data back into a displayable image and show it
%figure;
buffer = sprintf('Blue image output using %d singular values', N);
Bimg = cat(3, zeros(size(Db)), zeros(size(Db)), Db);
%imshow(uint8(Bimg));
imwrite(uint8(Bimg), sprintf('%dblue.jpg', N));
title(buffer);
% Thake the data from the Red, Green, and Blue image
% Rebuild a colored image with the corresponding data and show it
figure;
```

```
buffer = sprintf('Colored image output using %d singular values', N);
Cimg = cat(3, Dr, Dg, Db);
imshow(uint8(Cimg));
imwrite(uint8(Cimg), sprintf('%dcolor.jpg', N));
title(buffer);

%Calculating and adding PSNR

% Calculate mean square error of R, G, B.
mseRImage = (double(X(:,:,1)) - double(Cimg(:,:,1))) .^ 2;
mseGImage = (double(X(:,:,2)) - double(Cimg(:,:,2))) .^ 2;
mseBImage = (double(X(:,:,3)) - double(Cimg(:,:,3))) .^ 2;
mseR = sum(sum(mseRImage)) / (rows * columns);
mseG = sum(sum(mseGImage)) / (rows * columns);
mseB = sum(sum(mseBImage)) / (rows * columns);
% Average mean square error of R, G, B.
mse = (mseR + mseG + mseB)/3;
% Calculate PSNR (Peak Signal to noise ratio).
PSNR = 10 * log10( 255^2 / mse);
message = sprintf('The mean square error is %.2f.\nThe PSNR = %.2f', mse,
PSNR);
xlabel(message);

%getting image size
filename = sprintf('%dcolor.jpg',N);
fileinfo = dir(filename);
sizeofImg = (fileinfo.bytes/1024);
title(buffer);

%calculating compresssion ratio
compressionratio = original/sizeofImg;
compressionRatio = [compressionRatio ;compressionratio];

%saving values
numofSVals = [numofSVals; N];
psnrData = [psnrData; PSNR];
valuePSNR = [valuePSNR; PSNR];
imgSize = [imgSize; sizeofImg];
colourData((N/50)+13).name =sprintf('%dcolor.jpg',N);
colourData((N/50)+13).size = sizeofImg;
colourData((N/50)+13).compressionRatio = compressionratio;
colourData((N/50)+13).mse = mse;
colourData((N/50)+13).pnsr = PSNR;
end

figure;
title('PSNR values in compression')
plot(numofSVals, valuePSNR);
ylabel('Value of PSNR between compress and original image');
xlabel('Number of Singular Values used');
grid on
figure;
title('Compression Ratio')
plot(numofSVals, compressionRatio);
ylabel('Compression Ratio');
xlabel('Number of Singular Values used');
grid on
```


Appendix F script for colourSVD.m