

# **IMAGE COMPRESSION USING SVD AND SPARSE MATRICES**

A PROJECT REPORT

*Submitted by*

*AKULA DHANUSH – BL.EN.U4AIE19002*

*NVS PRADYUMNA – BL.EN.U4AIE19043*

*SATWIK REDDY SRIPATHI – BL.EN.U4AIE19059*

*for the course*

***19AIE203- Data Structures and Algorithms – 2***

*Guided and Evaluated by*

***D. RADHA***

***Asst. Prof(SG),***

***Dept. of CSE,***



**AMRITA SCHOOL OF ENGINEERING, BANGALORE**

**AMRITA VISHWA VIDHYAPEETHAM**

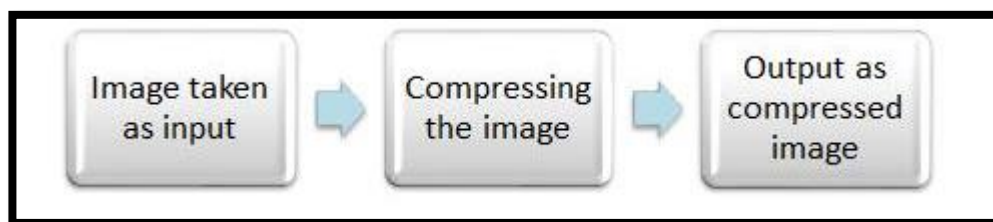
**BANGALORE-560 035**

## ABSTRACT

In today's world there are many images which would be sent as digital information every second and with the development in the technology there is also an rise in that. These transmitted images require a lot of storage and to reduce the storage we are compressing the image. In our project we are compressing the image for both the grey scale image and also an colour image. In the case of the grey scaled image we directly compress it using the singular value decomposition method and then we are using the sparse matrices where we convert the matrix into a sparse matrix and the we use sparse matrices to remove the unnecessary values in the data present in that image. While in the case of the colour image we are splitting the colour image into three different part which are on the red scale, the blue scale and the green scale after the splitting of the image into three different image we then compress those image using the singular value decomposition and then we are using the sparse matrices where we convert the matrix into an sparse matrix and the sparse matrices are used to remove the unnecessary data from those images and finally we combine those three images to get the compressed colour image. And we implemented our project in 2 programming languages which are Python and MATLAB.

## INTRODUCTION

In the present world there is a huge development in technology and with this rise in development of the technology each and every person is able to share the multimedia content. This sharing of the multimedia content is also rising and here as we say the multimedia content we mean the sharing of images. As there is a rise in the number of images that are required to be shared, the space required to store those images would also be high. To decrease the storage for the images in a device we use the method of image compression. Image compression is the process of encoding or converting an image file in such a way that it consumes less space than the original file. This is a type of compression technique that reduces the size of the image without degrading or affecting the quality of the image to a limited extent. Image compression does not physically reduce the size of the image but rather work on compressing the data present in the image into a smaller size. Image compression can benefit the users by having their pictures load faster and also for the web pages to use less space on the web host. Here for doing image compression we use the singular value decomposition method where we use sparse matrices while compressing the image. In our project we are using both the Mat Lab and the python programming languages to do the image compression. And the images which we consider in our project are either grey scale images or even the colour images. For the grey scale image first we completely blur the image that is compress the image to a certain level and after that compression we try to retrieve the image and by this process we are able to reduce the size of the image for example if the image is of first thousand kb then we compress it to hundred kb after that we are retrieving the image such that we get the image of approximate size around seven hundred to eight hundred kilo bytes and we even do this image compression for the colour images where first we divide the colour image into three parts red, blue and green and that splitting we are compressing each and every colour image with respectively and then at last after all the compression we combine those images to get an compressed colour image.



## SINGULAR VALUE DECOMPOSITION (SVD)

### Overview:

SVD is one of the most useful general-purpose tools in numerical linear algebra for data processing. It plays a vital role in dimensionality reduction when high dimensional data like in the case of megapixel images and high-quality videos which has great resolution which in turn result in large data. SVD reduces this high dimensionality dimensions into data with less dimensions with key features which are necessary for analyzing, understanding and describing this data. This feature of SVD makes it first step in most dimensionality reduction and machine learning techniques. It is also said, SVD is a data driven generalization of the Fourier transform. It also allows to tailor the coordinate system or transformation based on the data. It also helps in solving system of linear equations for non-square matrices which further help in linear regression models. SVD is basis for principal component analysis (PCA) which is one of the most widely used technique in all statistics for taking high dimensional data and trying to understand them in terms of dominant patterns and SVD distills high dimension data into key features or key correlations that are used to interpret the data. Thus, it is an effective tool in minimizing data storage and data transfer.

### Mathematical Overview:

Singular value decomposition is basically decomposing the given data matrix. It aims to approximate the original data with high dimensions with only few dimensions. In this process it factorizes the original data matrix into a product of an orthonormal matrix, diagonal matrix and another orthonormal matrix. Let  $A$  be an  $m \times n$  matrix and if we perform SVD to it the factorized matrices are  $U$ ,  $V$  and  $\Sigma$ , where  $U$  ( $m \times m$ ) and  $V$  ( $n \times n$ ) are orthonormal matrices and  $\Sigma$  ( $m \times n$ ) is a diagonal matrix.  $U$  contains left singular vectors of  $A$ ,  $V$  contains right singular vectors and  $\Sigma$  contains singular values of  $A$ , which are square roots of eigen values in such a way that  $\sigma_1 \geq \sigma_2 \geq \sigma_3 \dots \geq \sigma_N \geq 0$ , where  $\sigma = \sqrt{\lambda}$ .

Thus, matrix

$$A = U\Sigma V^T.$$

$$U = [u_1, u_2, u_3 \dots \dots \dots u_n], \Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}, V^T = \begin{bmatrix} V^T 1 \\ V^T 2 \\ \vdots \\ V^T n \end{bmatrix}$$

To perform SVD for a given data matrix  $A$  is as follows

1. Eigen values of the matrix  $A$  are to be found to get singular values.
2. Singular values found should be placed in decreasing order as a diagonal matrix which gives us  $\Sigma$ .
3. Compute  $AA^T$  and  $A^T A$  matrices.
4. Now eigen vectors of the above matrices should be found.

The above eigen vectors of  $AA^T$  will be columns of U and the eigen vectors of  $A^T A$  will be columns of V matrix.

1. Now with the equation  $U\Sigma V^T$  matrix A is represented.

Let us now compute SVD for a 2x2 matrix.

Let  $A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$  and

Follows  $A^T = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$

Now first we have to compute  $AA^T$  and  $A^T A$  matrices

Therefore,  $AA^T = \begin{bmatrix} 8 & 0 \\ 2 & 0 \end{bmatrix}$  and

$$A^T A = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$$

These following equations will help us in finding eigen values  $\lambda$ , and eigen vectors. And from these eigen values we can make singular values to make  $\Sigma$ .

$$|AA^T - \lambda I| = 0 \quad (1)$$

$$|A^T A - \lambda I| = 0 \quad (2)$$

From (1),

$$\begin{aligned} |AA^T - \lambda I| &= 0 \\ \left| \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| &= 0 \\ \left| \begin{bmatrix} 8-\lambda & 0 \\ 0 & 2-\lambda \end{bmatrix} \right| &= 0 \\ (8-\lambda)(2-\lambda) &= 0 \\ \lambda_1 &= 8 \\ \lambda_2 &= 2 \end{aligned}$$

Now we have eigen values  $\lambda_1, \lambda_2$  and from these we can compute singular values  $\sigma_1, \sigma_2$ , which on representing in decreasing order in a diagonal matrix.

$$\sigma_1 = \sqrt{\lambda_1} = \sqrt{8}$$

$$\sigma_2 = \sqrt{\lambda_2} = \sqrt{2}$$

$$\Sigma = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$$

Now columns of U will be unit eigen vectors of  $AA^T$ . So, we need to solve the following equation to with both the eigen values which we got to get U.

$$(AA^T - \lambda I)x = 0$$

$$\begin{aligned} (AA^T - \lambda I)x &= 0 \\ \left( \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) x_1 &= 0 \\ \begin{bmatrix} 8 - \lambda_1 & 0 \\ 0 & 2 - \lambda_1 \end{bmatrix} x_1 &= 0 \\ \begin{bmatrix} 8 - 8 & 0 \\ 0 & 2 - 8 \end{bmatrix} x_1 &= 0 \\ \begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix} x_1 &= 0 \\ x_1 &= \begin{bmatrix} -1 \\ 0 \end{bmatrix} \end{aligned}$$

Now we need to calculate the same with second eigen value to get second column of U.

$$\begin{aligned} (AA^T - \lambda I)x &= 0 \\ \left( \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) x_2 &= 0 \\ \begin{bmatrix} 8 - \lambda_2 & 0 \\ 0 & 2 - \lambda_2 \end{bmatrix} x_2 &= 0 \\ \begin{bmatrix} 8 - 8 & 0 \\ 0 & 2 - 8 \end{bmatrix} x_2 &= 0 \\ \begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix} x_2 &= 0 \\ x_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned}$$

Now collectively, columns of U will be unit vectors of  $x_1$  and  $x_2$ , which will give us  $u_1, u_2$ .

$$\text{Therefore, } U = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Similarly, with the equation  $(A^T A - \lambda I)x = 0$ , we will get  $v_1, v_2$ .

$$\begin{aligned} v_1 &= \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\ V &= \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \end{aligned}$$

By securing U,  $\Sigma$ , and V the factorization of A is realized and the SVD is complete.

$$A = U\Sigma V^T.$$

$$\begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

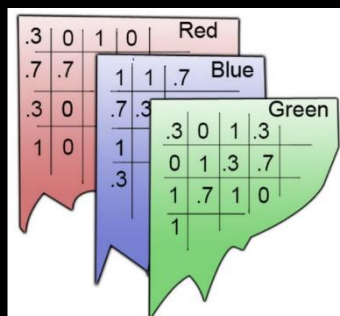
## IMAGE COMPRESSION

Image compression is a method used to reduce the high dimensionality data like images with high resolutions. This method goal is to reduce the storage space of the data and the cost of transmission. As the goal is to reduce the size of digital images and represent them in with lowest no. of bits. Redundancies in image can be taken as advantage to reduce the no. of bits. This redundancy can be defined as approximate repeating patterns present in image which are required to maintain the resolution and quality of the image. However, the quality of the image should not be compromised too much so that it becomes incomprehensible for the user. Algorithm or technique used for image compression should be taken in such a way that it balances both quality and storage. Image compression is classified into lossless and lossy compression. Lossless compression is reversible technique where it defines the entropy which limits the reduction of the image and thus, tends to produce copy of the original data. Its reversibility is because it doesn't degrade the quality of image much. Lossy compression is such technique where minute pattern repetitions and variations which human-eye cannot recognize. So, such features can be eliminated to reduce the storage space of the image. This technique is irreversible as quality of the image is altered to reduce the storage space.

Hence, from the above such technique should be picked where the main goal of image compression to reduce the storage space of the image without completely degrading the image. Therefore, algorithm or technique which comes under lossy compression suits best, where image once compressed cannot be reconstructed and acceptable loss of fidelity or quality takes place. Singular Value Decomposition (SVD) is such Image compression technique.

## SVD IN IMAGE COMPRESSION

Under this, first we have to know about the representation of the image. Image is generally represented as  $m \times n$  matrix, where  $m$  is no. of rows which signifies the pixel height of the image and  $n$  is no. of columns which signifies the pixel width of the image. Each subsequent value in the matrix talks about the intensity of the image. In matrix each element is each pixel's relative contribution of darkness, brightness or resolution of the image. As a result, characteristics of each pixel and their contribution to the image is represented by a value and these values are wholly represented as a matrix. In the case of grayscale images, the values 1 and 0 represent nature of each pixel, where 0 is given to pixels which are responsible for black and 1 is given to pixels which are responsible for white, which translates the relative grayness of each pixel. In the case of Colored images, they are composed of three basic colors: Red, blue, and Green. These colored images are stored as three layers which correspond to three primitive colors and each pixel has



Red			
.3	0	1	0
.7	.7	1	1
.3	0	.7	.3
1	0	1	.3

Blue			
1	1	.7	

Green			
.3	0	1	.3
0	1	.3	.7
1	.7	1	0
1			

three values, that is their contribution to each layer. Thus, each pixel store three values which are ranged from 0 for not colored and 1 for complete saturation and these values are assigned to each layer. This is the reason why colored images take more space than gray scaled images. For example, if a 5-megapixel image is stored and is represented by 3000x3000 matrix and if we consider 1 byte for each pixel for storage space, gray scale image needs 5MB storage space where as, if the image is colored each layer or component needs to be represent as a matrix and totally needs 15MB storage space.

Now this matrix representation of Image helps SVD to compress the image. As we discussed SVD is dimensions reduction tool and in the case of image dimensions are nothing but the whole data with some part of junk in it in the form of redundancy which also occupy storage space along with key featured data. So, implementation of SVD takes these redundancies as an advantage to reduce the dimension of the data. The details of redundancy areas in image helps SVD to delete or remove such areas in compressing the image. This property of SVD makes it work well. So, now when image is represented as a matrix, this matrix is used to perform SVD.

SVD is implemented on the image matrix and as a result its orthonormal and singular values matrices are produced as a result. Let A be the image matrix and after SVD it is represented as:

$$A = U\Sigma V^T.$$

Now this  $\Sigma$  matrix contain singular values in descending order in the form of diagonal matrix. The singular value which is able to interpret the key or major data of the image will become first non-zero element in the diagonal matrix. Similarly, depending on the amount of their contribution to represent the image data they are placed in descending order. So, the redundancy in image is caused because of the singular values which are in the last in diagonal matrix. The main or effective step is to take the effective rank or r value which represents how many non-zero singular values need to be taken. So, in the  $\Sigma$  matrix the values till 'r' are taken and are considered as the most contributed singular values and other values or values after 'r' are approximated to zero. So, when this  $\Sigma$  matrix is multiplied with both orthonormal matrices the values which are multiplied to after 'r' values will nearly become zero. So, this reduction process after SVD is based on the rank value of a singular matrix which is equal to number of non-zero elements and therefore the rank of image matrix A will be reduced as the number of singular values are approximated to 'r'. This means we can approximate the image matrix A by just considering few terms of the series before the 'r' value. Therefore, removing or erasing this data which is approximated to zero will reduce the dimension of the data and thus, a compressed image is formed.

Therefore, from the above analysis we infer that the image matrix is approximated to reduce the dimensions of it by adding the first few terms till 'r'. This signifies that the increase in 'r' value chosen will result in increase in quality and storage space. Hence, for a better image compression using SVD, an optimum 'r' value should be taken which balances both the quality and the memory required to store.



## IMPLEMENTATION

As part of our project on image compression using a very powerful technique of Singular Value Decomposition and making the matrices sparse in saving space is done in two programming platforms. One is in python programming language and the other one is MATLAB. This section is entirely on the implementation in both programming language. In both the cases, the idea is to take a picture as user input and compress it according to the singular values range provided by the user and storing these images in the system. Idea is to get an image which is of optimum clarity and consumes less space. As part of this project, we did two types of image compression. One way is to make image into a gray image and then compress it. Making an image gray will straight away reduce a lot of space and compressing it will further reduce the space it consumes. Other way is to compress a colored image. The following is details of how each of it was done in both programming languages.

### 1. PYTHON PROGRAMMING LANGUAGE

#### a. GRAY SCALED IMAGE COMPRESSION

Python language is a very powerful and useful language in image compression. It provides many built-in packages which helps a programmer work efficiently. These are the details of compressing a gray image.

This is an image of the pseudo code on gray image compression. These are the steps followed in order to compress an image converting it into a gray image.

```
Enter file name : spd2.jpg
Image size is : 500 x 500
Enter dividing factor : 1
Enter minimum singular values : 10
Enter maximum singular values : 300
Enter step size : 20
Done !!
```

```
## PSEUDO CODE OF GRAY IMAGE COMPRESSION
-> Original image input
-> Convert to gray scale
-> Perform SVD
-> Keep necessary value
-> Unnecessary values are made zero(SPARSE)
-> SVD on given range
-> Display and store images
-> Show error
```

This is how the file name is taken as input from user. The minimum and maximum values are range of singular values that a user wishes to have.

After the file is read, its dimensions are measured and according to the scale given by user, the image dimensions are modified. These modifications don't affect the image, it is only to display it to the user.

```
inImage=cv2.imread(filename)
[imRow, imCol, imH] = np.shape(inImage)
print("Image size is : ",imRow,'x',imCol)
dividingFactor=int(input("Enter dividing factor : "))
imShowingRowScale=int(imRow/dividingFactor)
imShowingColScale=int(imCol/dividingFactor)
inImageResize = cv2.resize(inImage, (imShowingRowScale, imShowingColScale))
cv2.imshow('Original Image', inImageResize)
cv2.waitKey(0)
cv2.imwrite('Original Image.jpg', inImage)
```

The NumPy package helps in calculating the dimensions of the image and the OpenCV package helps in reading image, showing it and writing into a location on our system.

```
inImage = cv2.cvtColor(inImage, cv2.COLOR_BGR2GRAY)
inImageResize = cv2.resize(inImage, (imShowingRowScale, imShowingColScale))
cv2.imshow('Gray_Scaled Image', inImageResize)
cv2.waitKey(0)
cv2.imwrite('Original Gray Image.jpg', inImage)
```

This is where the image is converted into a gray image and same as above, it is resized only to show the user. The OpenCV

command of cvtColor helps in conversion.

```
inImageD = np.double(inImage)
[U, S, VT] = np.linalg.svd(inImageD, full_matrices=True)
```

The image matrix is converted into double and SVD is performed on it.

After SVD is done, a new matrix is generated for the image with specific number of singular values. This is a function written which will modify the image matrix and get us an image with modified number of singular values. This modified image is shown and the same is saved in the system. Instead of passing parameter, the variables are made global so as to perform the variables which are common to all.

```
def newMatrixAfterSVDandErrorAddition(num):
    global displayError, singularVals, inImageD
    N=num
    gray_U = U[:, :N]
    gray_S = np.diag(S[:N])
    gray_VT = VT[:N, :]
    DN = gray_U @ gray_S @ gray_VT
    D = np.uint8(DN)
    DImageResize = cv2.resize(D, (imShowingRowScale, imShowingColScale))
    cv2.imshow('Image with ' + str(N) + ' singular values', DImageResize)
    cv2.waitKey(0)
    cv2.imwrite('Gray Image - '+str(N)+'.jpg', D)
    error = sum(sum((inImageD - DN) ** 2))
    displayError.append(error)
    singularVals.append(N)
```

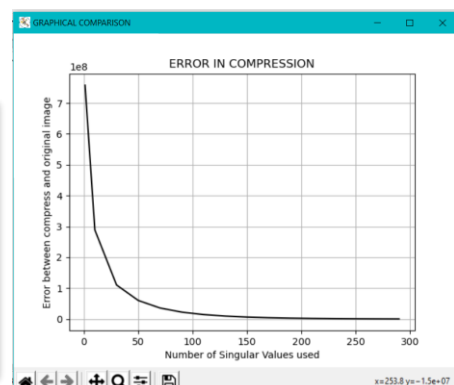
After this is done for a single value of singular value, the loop with specified range by the user starts and images at every instance are shown and downloaded at the same time. We see that the image quality keeps improving as the number of singular values keep increasing.

```
for N in range(startRange, endRange+1, stepSize):
    newMatrixAfterSVDandErrorAddition(N)
```

This is how the loop calls the above function for given range.

Once this is done, the amount of error of the image with respect to the original image is found and the same is plotted for better understanding.

```
def visualizeError():
    plt.figure('GRAPHICAL COMPARISON')
    plt.title('ERROR IN COMPRESSION')
    plt.plot(singularVals, displayError, 'k')
    plt.grid('on')
    plt.xlabel('Number of Singular Values used')
    plt.ylabel('Error between compress and original image')
    plt.show()
```



This is how the plot is for the example shown above. Let's see how the images at every instance were so that we understand it in a better way.

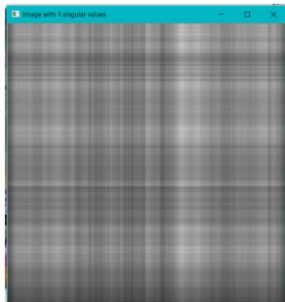


120 KB

This was the original image which was a colored image and this is how it was converted to gray image which looks like this. The following images will show how the quality of the images kept improving.



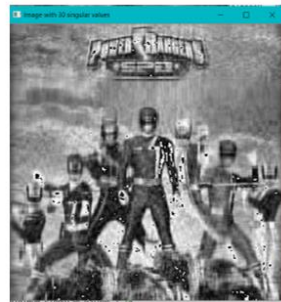
98 KB



1 - 42.6 KB



10 - 74 KB



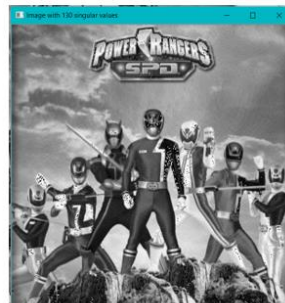
30 - 74 KB



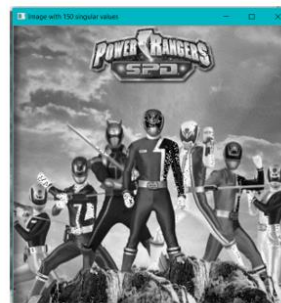
50 - 80 KB



70 - 88 KB



130- 98 KB



150- 100KB



250- 101KB



270 -102 KB



290 -102 KB

This is how the image quality kept improving and their corresponding size is show below them. In this way, a colored image can be made into a gray image and then compressed it into this size. So, we can infer that, more the quality of the image is, more is its size. So an image of

almost same quality but with less size is achieved and this can be used in data transfer. This technique come into picture when size of an image is more important than its quality. An image which is less in size and with optimum quality is what is required and that is achieved.

## b. COLORED IMAGE COMPRESSION

Every colored image is a different combination of Red, Green and Blue colors. Every other color can be achieved in certain combination of the red, green and blue colors. So, for a colored image to be compressed, it is not as directly as we did by converting it into a gray image. We have three color components Red, Green and Blue and we need to work on each of them. Each of the color components are to be separated first and then decomposed separately and then combine them to get a colored image which is decomposed.

```
## PSEUDO CODE OF COLORED IMAGE COMPRESSION
-> Original Image Input
-> Split into RGB components
-> Perform SVD
    -> SVD on RED
        -> New RED matrix
    -> SVD on GREEN
        -> New GREEN matrix
    -> SVD on BLUE
        -> New BLUE matrix
-> New COLOR Image
    -> Combine New RED, GREEN & BLUE
-> Keep necessary value
-> Unnecessary values are made zero(SPARSE)
-> SVD on given range
-> Display and store images
-> Show error
```

This is the pseudo code of a colored image compression. The file is read and then it is split into its components which are Red, Green and Blue. Singular Value Decomposition is done on these component matrices separately and then these new matrices are merged together to generate a new colored image. This image has modified number of singular values and thus has a different quality once modified.

Let's see the same example of what we did for gray scale image.

This is how user input of the file name is taken. The range of values for which the user wants images to be is taken here.

```
Enter file name : spd2.jpg
Image size is : 500 x 500
Enter dividing factor : 1
Enter minimum singular values : 1
Enter maximum singular values : 400
Enter step size : 50

Done !!
```

```
inImage=cv2.imread(filename)
[imRow,imCol,imH]=np.shape(inImage)
print("Image size is : ",imRow,'x',imCol)
dividingFactor=int(input("Enter dividing factor : "))
imShowingRowScale=int(imRow/dividingFactor)
imShowingColScale=int(imCol/dividingFactor)
inImageResize=cv2.resize(inImage, (imShowingRowScale,imShowingColScale))
cv2.imshow('Original Image',inImageResize)
cv2.imwrite('Original COLORED Image.jpg',inImage)
cv2.waitKey(0)
```

This is where the image is read and its dimensions are measured and a resized image is shown to the user as we did for gray scale image.

```
B,G,R=cv2.split(inImage)
```

This command splits the colored image into three color components. The OpenCV package helps in splitting the image.



```
zeros=np.zeros(inImage.shape[:2],dtype="uint8")
```

This command creates a matrix of zeros which of the size equal to that of the image. This will return a list [rows, columns] and the NumPy library will create a zeros matrix of the dimensions. This will be used to extract only one component form the color image and make other components zero.

```
Rimg=cv2.merge([zeros,zeros,R])
inImageRedResize=cv2.resize(Rimg, (imShowingRowScale,imShowingColScale))
cv2.imshow('RED',inImageRedResize)
cv2.imwrite('Original RED Image.jpg',Rimg)
cv2.waitKey(0)

Gimg=cv2.merge([zeros,G,zeros])
inImageGreenResize=cv2.resize(Gimg, (imShowingRowScale,imShowingColScale))
cv2.imshow('GREEN',inImageGreenResize)
cv2.imwrite('Original GREEN Image.jpg',Gimg)
cv2.waitKey(0)

Bimg=cv2.merge([B,zeros,zeros])
inImageBlueResize=cv2.resize(Bimg, (imShowingRowScale,imShowingColScale))
cv2.imshow('BLUE',inImageBlueResize)
cv2.imwrite('Original BLUE Image.jpg',Bimg)
cv2.waitKey(0)
```

Every colored image has RGB representation. In python it is different and the order is BGR. So, for given colored image, red, green and blue images are made. This is done by keeping the color component as it is and making other components zero.

This is the function which performs SVD of each of the components. The variables are made global as they are common to everything. This is how SVD is performed on RED color component. Similarly, it is done on GREEN & BLUE. This function can be called by specifying the singular values required to generate a new image.

```
def newMatrixSVDandError(num):
    global dividingFactor
    global Red,Green,Blue,Color
    global displayErrorRED,singularValsRED
    global displayErrorGREEN,singularValsGREEN
    global displayErrorBLUE,singularValsBLUE
    global displayErrorCOLOR,singularValsCOLOR
    N=num
    # on red component
    U_red, S_red, VT_red = np.linalg.svd(Red)
    U_RED = U_red[:, :N]
    S_RED = np.diag(S_red[:N])
    VT_RED = VT_red[:N, :]
    D_RED = U_RED @ S_RED @ VT_RED
    zeros_red = np.zeros([np.shape(D_RED)[0], np.shape(D_RED)[1]])
    RimgD = cv2.merge((zeros_red, zeros_red, D_RED))
    Rimg = np.uint8(RimgD)
    Rimg_Row_scale = int(np.shape(Rimg)[0] / dividingFactor)
    Rimg_COL_scale = int(np.shape(Rimg)[1] / dividingFactor)
    RimgResize = cv2.resize(Rimg, (Rimg_Row_scale, Rimg_COL_scale))
    cv2.imshow(('Red with ' + str(N) + ' singular values'), RimgResize)
    cv2.waitKey(0)
    cv2.imwrite(('Red Component - ' + str(N) + '.jpg'), Rimg)
    errorRED = sum(sum((Red - D_RED) ** 2))
    displayErrorRED.append(errorRED)
    singularValsRED.append(N)
```

```
# getting back colored image
CimgD = cv2.merge((D_BLUE, D_GREEN, D_RED))
Cimg = np.uint8(CimgD)
Cimg_ROW_scale = int(np.shape(Cimg)[0] / dividingFactor)
Cimg_COL_scale = int(np.shape(Cimg)[1] / dividingFactor)
CimgResize = cv2.resize(Cimg, (Cimg_ROW_scale, Cimg_COL_scale))
cv2.imshow(('Colored with ' + str(N) + ' singular values'), CimgResize)
cv2.waitKey(0)
cv2.imwrite(('RGB component - ' + str(N) + '.jpg'), Cimg)
```

This is the continuation of the above function. Once new matrices for the color components after performing SVD is generated, those matrices are merged

together to get back the colored image with the new generated color component matrices. All the images are show to the user and also saved to the system.

This is how the loop runs for the range specified by the user. As

```
for N in range(startRange, endRange+2, stepSize):
    newMatrixSVDandError(N)
cv2.destroyAllWindows()
```

we saw in the function, at every stage, images are show and also downloaded.

This is a function to plot graphs which show error. Error is the difference in values of the original image to that of the new generated values after SVD. This is shown for RED component, the same is done for the other color components.

```
def plotGraphs():
    global displayErrorRED, singularValsRED
    global displayErrorGREEN, singularValsGREEN
    global displayErrorBLUE, singularValsBLUE
    global displayErrorCOLOR, singularValsCOLOR
    # Plot RED error
    plt.figure('RED')
    plt.title('Error in compression')
    plt.plot(singularValsRED, displayErrorRED, 'r')
    plt.grid('on')
    plt.xlabel('Number of Singular Values used')
    plt.ylabel('Error between compress and original image')
```

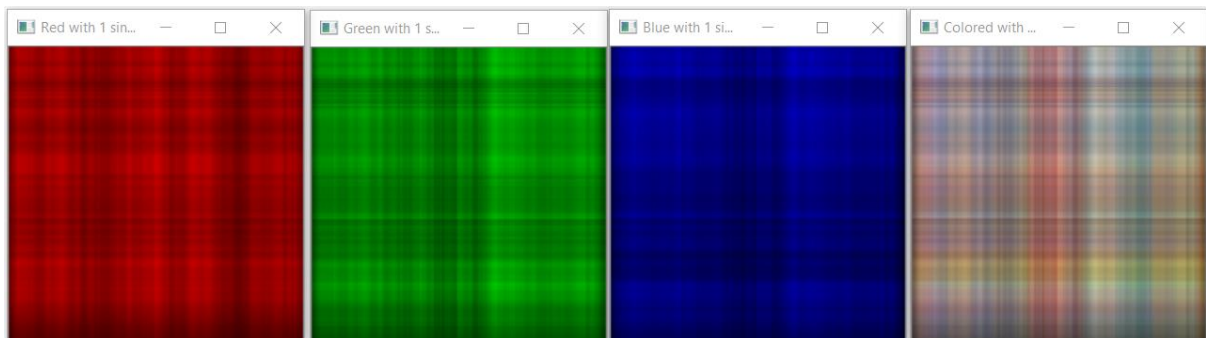
```
# subplot representation
plt.figure('ALL IN ONE')
plt.title('Error in compression')
plt.subplot(1,3,1)
plt.plot(singularValsRED, displayErrorRED, 'r')
plt.ylabel('Error between compress and original image')
plt.title('Error in RED')
plt.grid('on')
plt.subplot(1,3,2)
plt.plot(singularValsGREEN, displayErrorGREEN, 'g')
plt.xlabel('Number of Singular Values used')
plt.title('Error in GREEN')
plt.grid('on')
plt.subplot(1,3,3)
plt.plot(singularValsBLUE, displayErrorBLUE, 'b')
plt.title('Error in BLUE')
plt.grid('on')
plt.suptitle('Errors of each color components')
plt.show()
```

This will plot a graph which has subplots to show all the error in same graph.

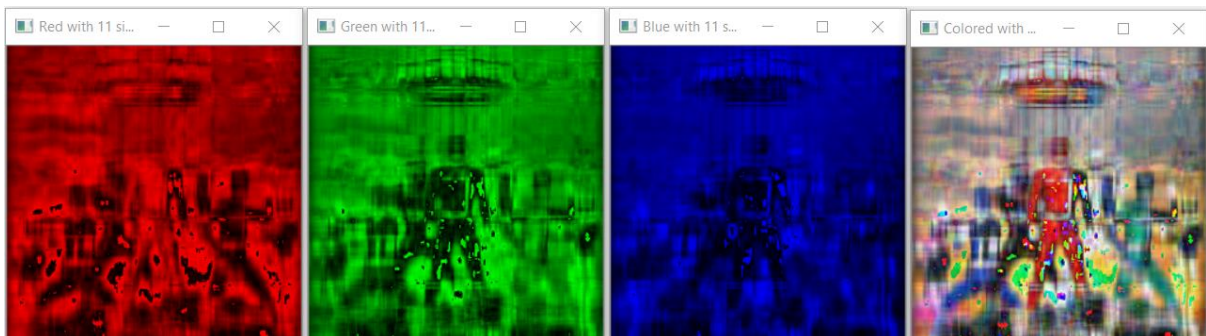


This is the original-colored image. Once this image is read, the original image is split into its color components. The Red, Green and Blue color components are extracted and shown below. The original color image is of size 120KB.

The following are the original color components in the colored image.

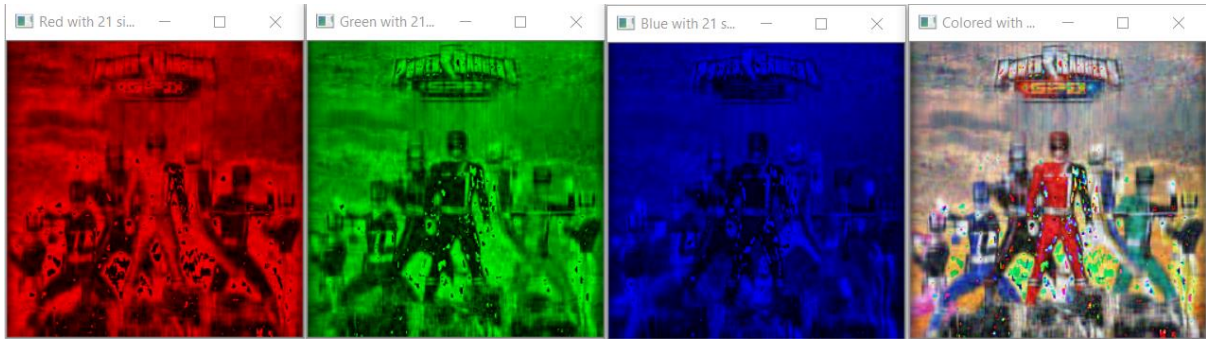


These are image components with 1 singular value, COLORED IMAGE=51.7KB

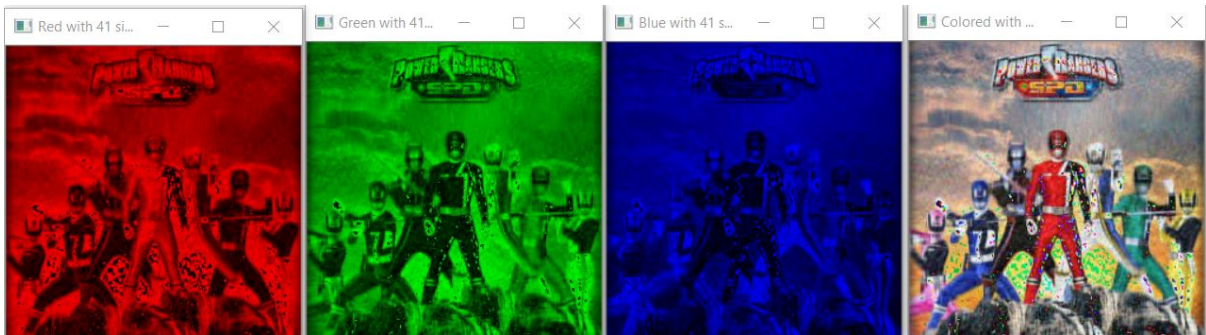


These are image components with 11 singular value, COLORED IMAGE=110KB

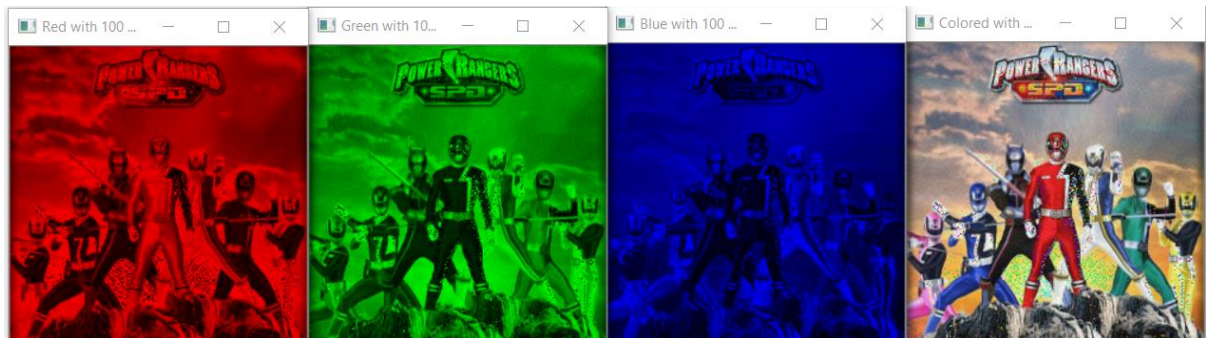




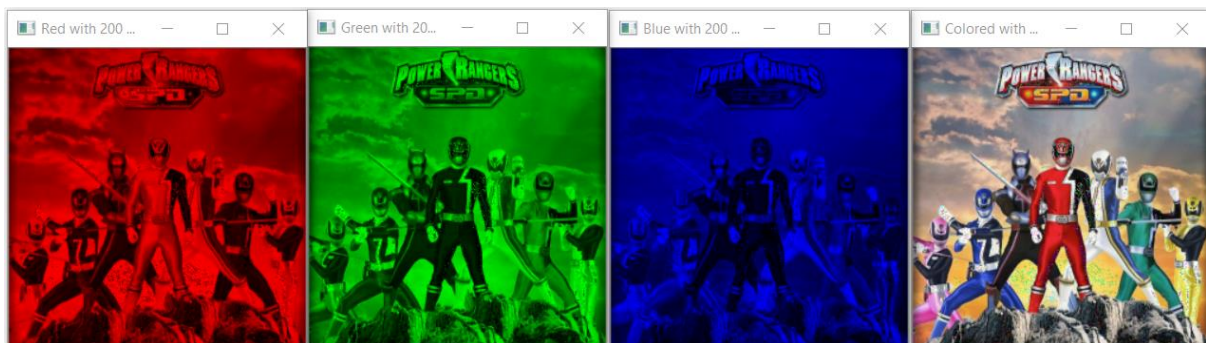
These are image components with 21 singular value, COLORED IMAGE=131KB



These are image components with 41 singular value, COLORED IMAGE=146KB

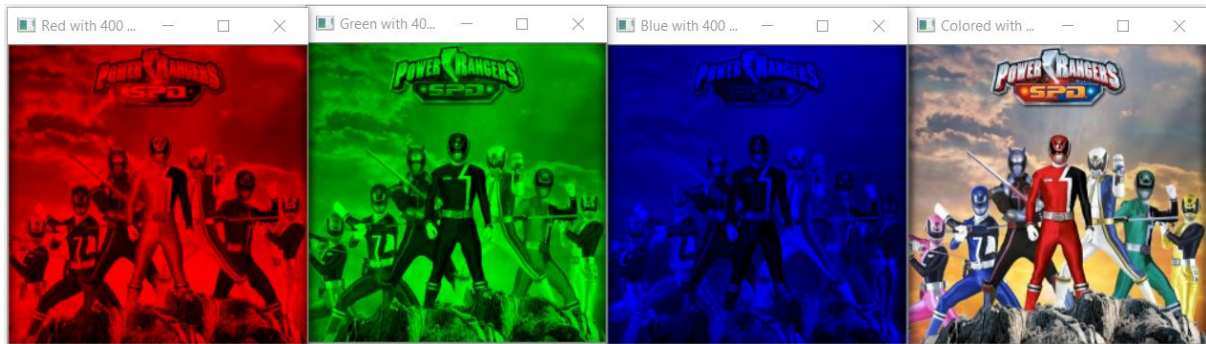


These are image components with 100 singular value, COLORED IMAGE=153KB

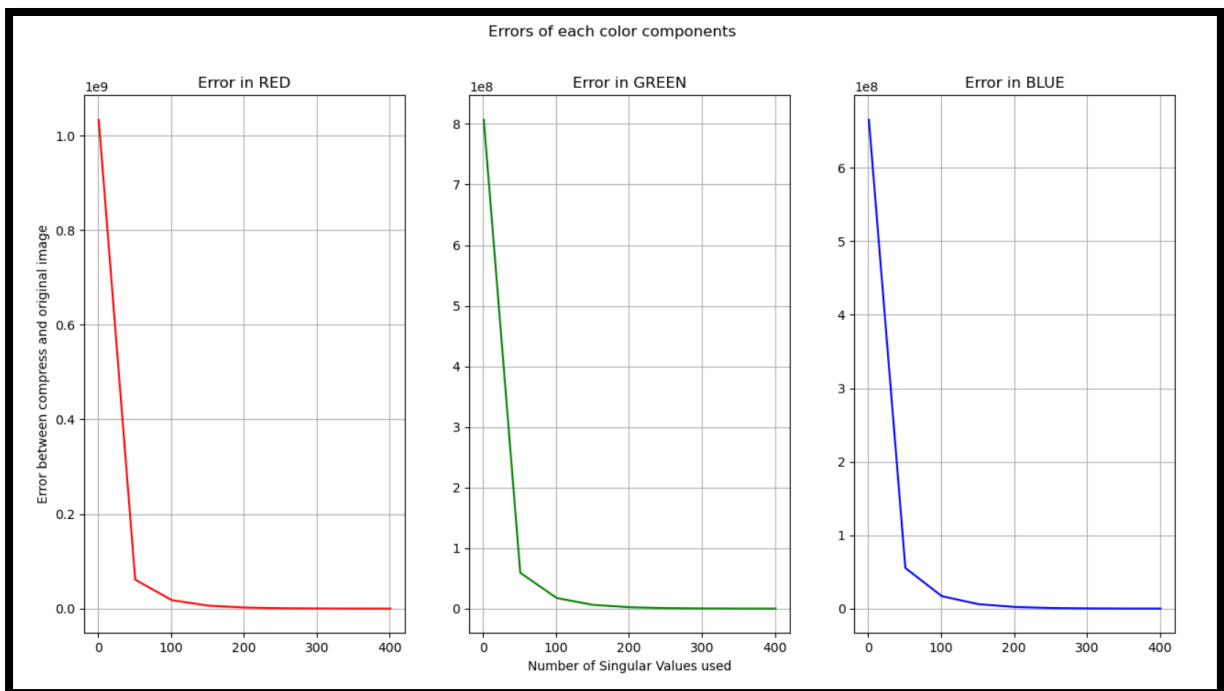


These are image components with 200 singular value, COLORED IMAGE=143KB





These are image components with 400 singular value, COLORED IMAGE=117KB



This is how the graph for error in each of the component looks like. As the singular values kept increasing, the error kept reducing. Finally, an image of optimum quality with less size is achieved.

## 2. MATLAB PROGRAMMING LANGUAGE

### a. GRAY SCALED IMAGE COMPRESSION

MATLAB programming language is very powerful when it comes to mathematical operations. Every programming language such as python uses MATPLOTLIB library which is of MATLAB only. Image compression is possible in this language also. The steps are even more simple compared to that of python.

This is the image of the pseudo code. It remains same for both the programming languages.

```
Enter file name (with"")      : 'spd2.jpg'
Enter minimum singular values : 1
Enter maximum singular values : 101
Enter step size : 10
Done !!!
>> |
```

```
## PSEUDO CODE OF GRAY IMAGE COMPRESSION
-> Original image input
-> Convert to gray scale
-> Perform SVD
-> Keep necessary value
-> Unnecessary values are made zero(SPARSE)
-> SVD on given range
-> Display and store images
-> Show error
```

This is how the input is taken from the user.

This is how the image is read from the system using imread command provided by MATLAB. After reading, the same image is shown and also saved to the system.

```
filename=input('Enter file name (with"")      : ');
filename=strcat('../',filename);
inImage=imread(filename);
figure('name','Original Colored Image')
imshow(filename)
title('Original Image')
imwrite(inImage,'Original Colored Image.jpg')
```

```
inImage=rgb2gray(inImage);
figure('name','Original image in GreyScale')
imshow(inImage)
title('Gray Scaled Image')
name=strcat('Original Gray Scaled','.jpg');
imwrite(inImage,name)
```

This is how the colored image is converted into gray image using rgb2gray command. This image is shown and also save to the system.

```
inImageD=double(inImage);
[U,S,V]=svd(inImageD);
```

The image matrix is then converted into double datatype as the SVD command in MATLAB supports only double values. The svd() command is used to perform SVD.

A temporary matrix stores the value of Sigma matrix S and this is manipulated according to the number of singular values being used. After modification, a new matrix is generated which represents a new image based on the above values is calculated and error is measured. The image at this instance is shown and also downloaded to the system.

```
N=1;
C=S;
C(N+1:end,:)=0;
C(:,N+1:end)=0;
D=U*C*V' ;
error=sum(sum((inImageD-D).^2));
displayError=[displayError;error];
singularVals=[singularVals;N];
```

```

for N=min:step:max
    C=S;
    C(N+1:end,:)=0;
    C(:,N+1:end)=0;
    D=U*C*V';
    error=sum(sum((inImageD-D).^2));
    displayError=[displayError;error];
    singularVals=[singularVals;N];
    buffer = sprintf('GrayScale - %d', N);
    figure('name',buffer)
    new_img=uint8(D);
    imshow(new_img);
    imwrite(new_img, strcat(buffer, '.jpg'));
    title(buffer);
end

```

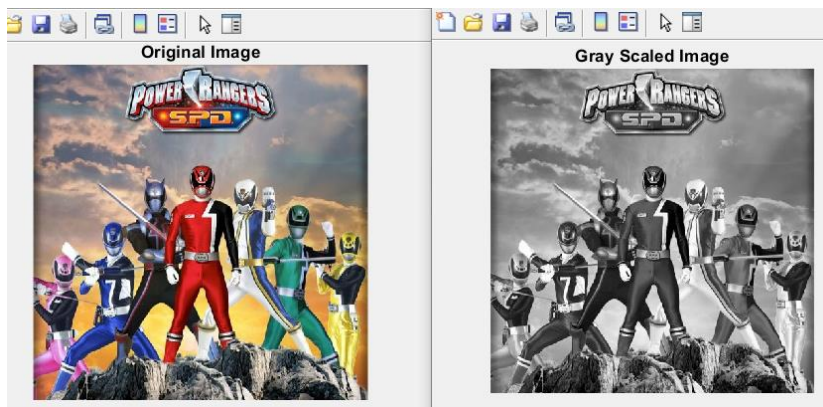
The same code loops for the range given by the user. At every instance, images are shown and downloaded to the system.

```

figure;
title('Error In Compression');
plot(singularVals, displayError);
grid on
xlabel('Number of Singular Values used');
ylabel('Error between compressed and original image');
fprintf("Done !!!\n")

```

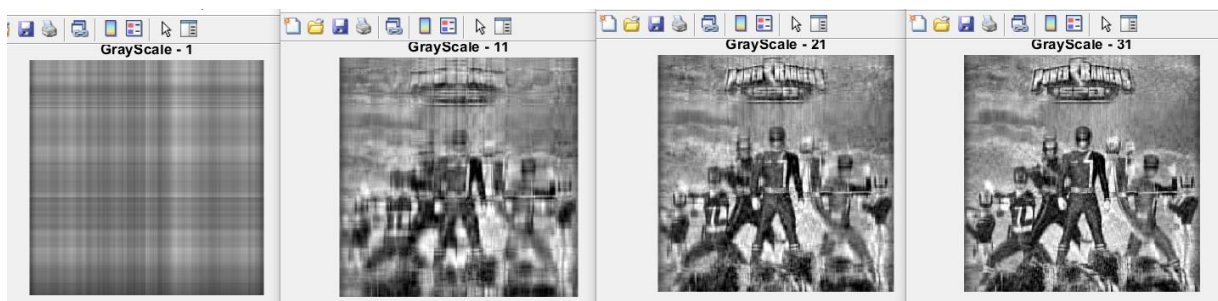
A graph which displays error is plot after all iterations.



Original -102 KB

Gray -45.6 KB

These are the original images. The colored image in the left is gray-scaled and shown as the right image.

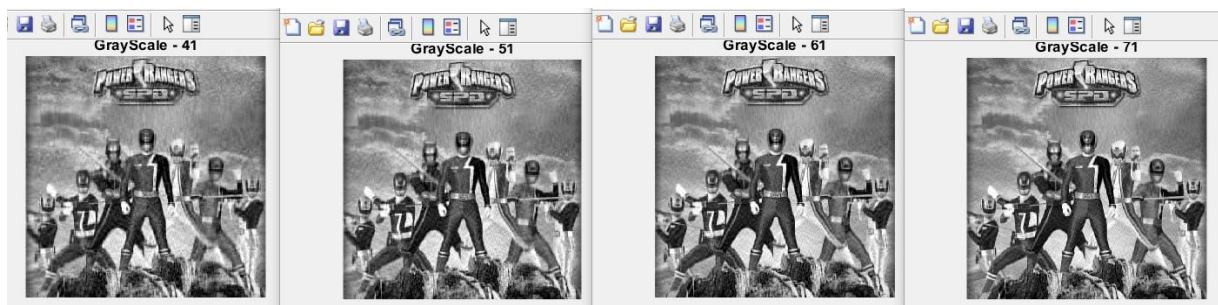


1 -18.9 KB

11 -32.4 KB

21 -38.9 KB

31 -41.2 KB

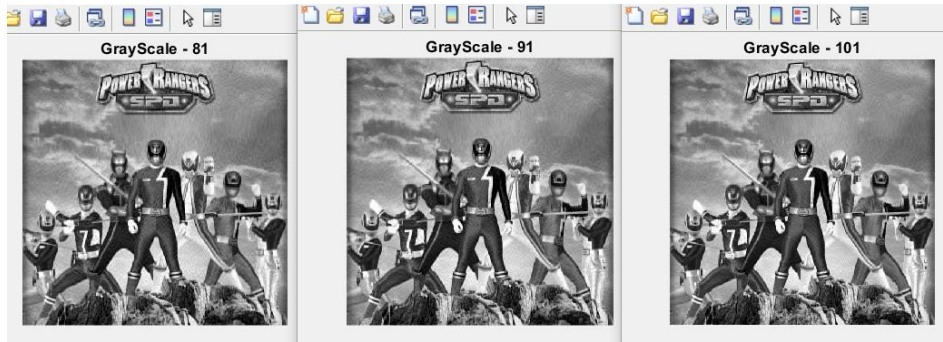


41 -43.4 KB

51 -45 KB

61 -46.5 KB

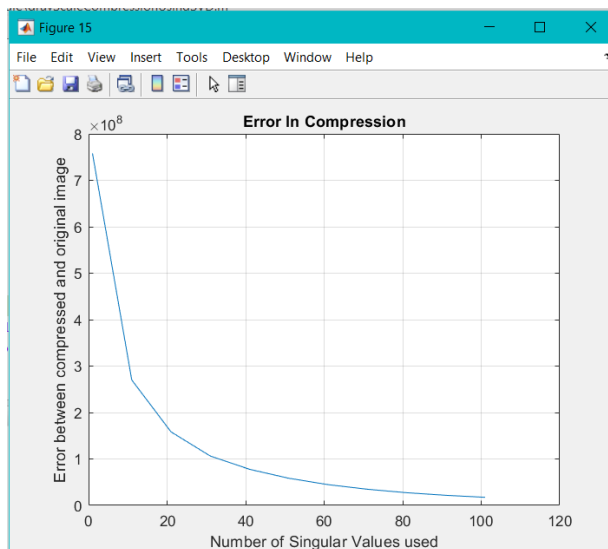
71 -47.5 KB



81 – 48.2 KB

91 -48.6 KB

101 -48.9 KB



This is the graph which is Error Vs Number of singular values. As we saw when we did gray image compression using python that as the singular values keep increasing, the quality increase, even her we see that the error reduces as the number of singular values keep increasing.

## b. COLORED IMAGE COMPRESSION

Again, every colored image is a mixture of three colors components-Red, Green and Blue. Every other color can be achieved from these colors. Let's see how is it implemented in this project using MATLAB.

```
filename=input('Enter file name (with"") : ');
filename=strcat('../',filename);
[X,map]=imread(filename);
figure('name','Original Colored Image')
imshow(X);
title('Original Image')
imwrite(X,'Original Colored Image.jpg');
```

This is how the file is read and the same is shown and downloaded to the system.



```
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);
```

This is how the components are separated from the original image and then individual images are made by keeping their respective component and making other components zero.

Color components of the original image are plot and also downloaded.

```
##### RED IMAGE
[U_red,S_red,V_red]=svd(Red);
rank(S_red);
N=1;
C_red=S_red;
C_red(N+1:end,:)=0;
C_red(:,N+1:end)=0;
D_red=U_red*C_red*V_red';
figure('name',sprintf('Images with %d singular values',N));
buffer = sprintf('Red image output using %d singular values', N);
Rimg=cat(3,D_red,zeros(size(D_red)),zeros(size(D_red)));
new_Rimg=uint8(Rimg);
subplot(1,4,1)
imshow(new_Rimg)
imwrite(new_Rimg, strcat(buffer, '.jpg'));
title(sprintf('R Component- %d ',N))
error_red=sum(sum((Red-D_red).^2));
displayErrorRED=[displayErrorRED;error_red];
singularValsRED=[singularValsRED;N];
```

```
figure('name','RGB color components separated')
% Red component
subplot(1,3,1)
imshow(Rimg)
title('R component')
imwrite(Rimg,'Original Red.jpg')
% Green component
subplot(1,3,2)
imshow(Gimg)
title('G component')
imwrite(Gimg,'Original Green.jpg')
% Blue component
subplot(1,3,3)
imshow(Bimg)
title('B component')
imwrite(Bimg,'Original Blue.jpg')
suptitle('Separated R G B Components')
```

This is how SVD is performed on the RED color component. RED image with modified matrix is plot and also downloaded. The same is done with other color components. The matrices are made double as SVD need double datatype inputs.

Once SVD is performed on RGB components, they have to be merged together to get a colored image with these modified

```
#### COMBINING THESE THREE BACK
buffer = sprintf('Colored image output using %d singular values', N);
Cimg=cat(3,D_red,D_green,D_blue);
new_Cimg=uint8(Cimg);
subplot(2,2,4)
imshow(new_Cimg)
imwrite(new_Cimg, strcat(buffer, '.jpg'));
title(sprintf('RGB Component- %d ',N))
suptitle(sprintf('R G B Images with %d singular values',N))
```

matrices. This is done by the command cat which means to concatenate the matrices together. The color image obtained by these new values is shown and downloaded.

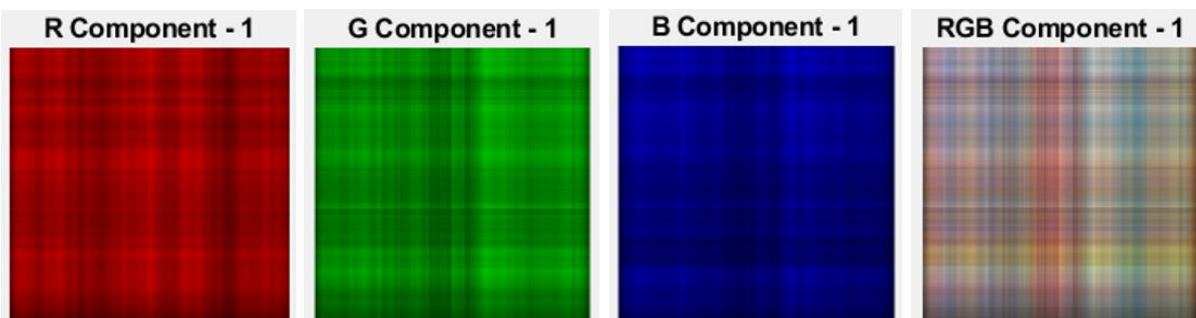
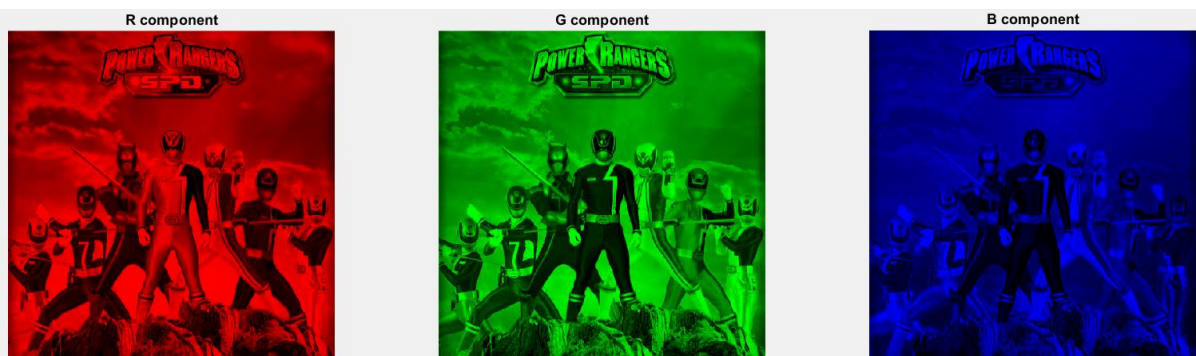
```
figure('name','Error Vs Singular Values');
hold on;
plot(singularValsRED, displayErrorRED,'r');
plot(singularValsGREEN, displayErrorGREEN,'g');
plot(singularValsBLUE, displayErrorBLUE,'b');
grid on
xlabel('Number of Singular Values used');
ylabel('Error between compress and original image');
title('RGB Error in compression');
legend('RED','GREEN','BLUE')
```

After the loop runs for the range specified by user, images are shown and downloaded and graph is plot to show error measured at every instance.

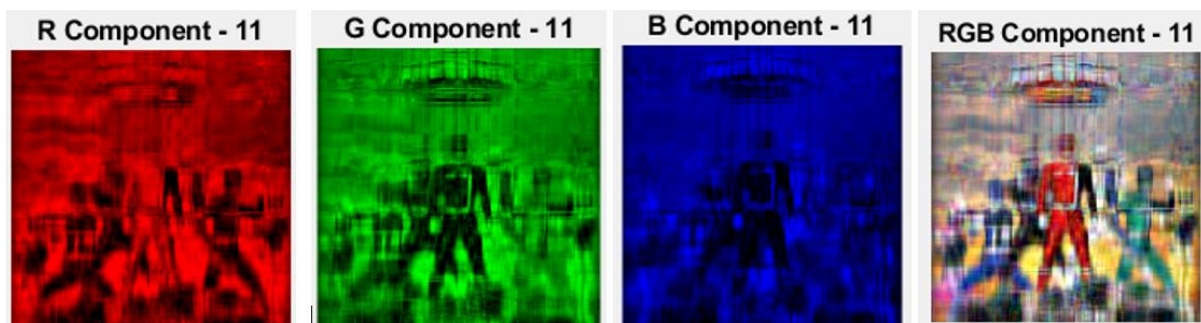


This is the original-colored image which is given as input. This is read and split into Red, Green and Blue components

The size of the colored image is 120KB.

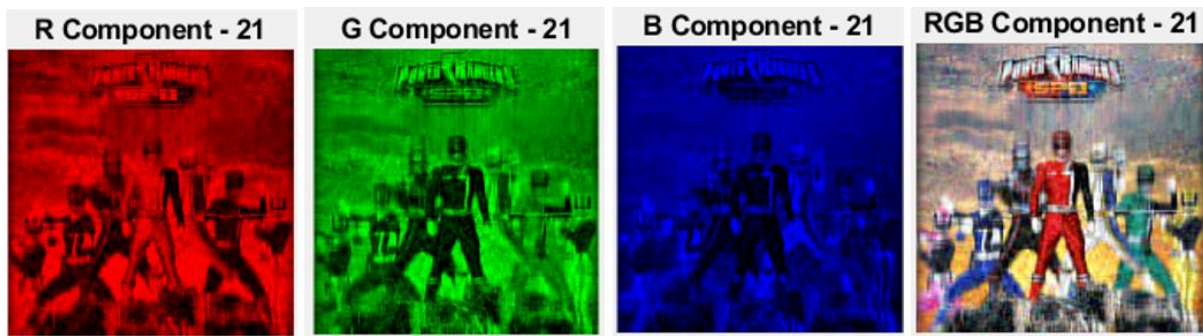


These are image components with 1 singular value, COLORED IMAGE=22.4KB

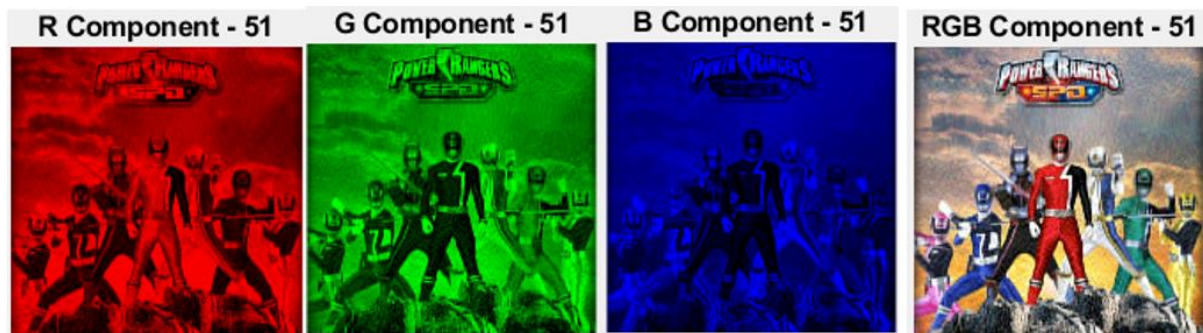


These are image components with 11 singular value, COLORED IMAGE=39.9KB

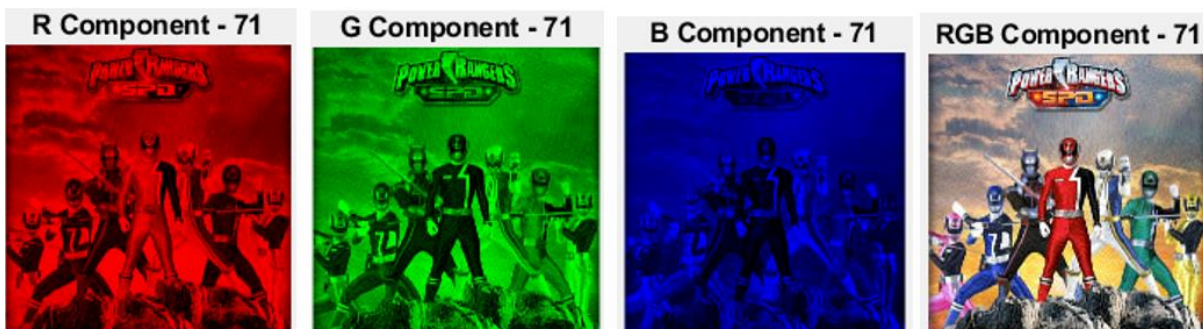




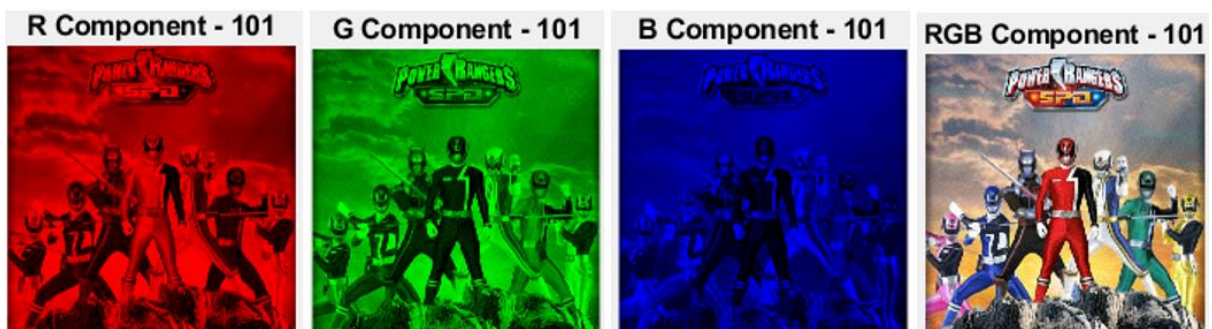
These are image components with 21 singular value, COLORED IMAGE=45.8KB



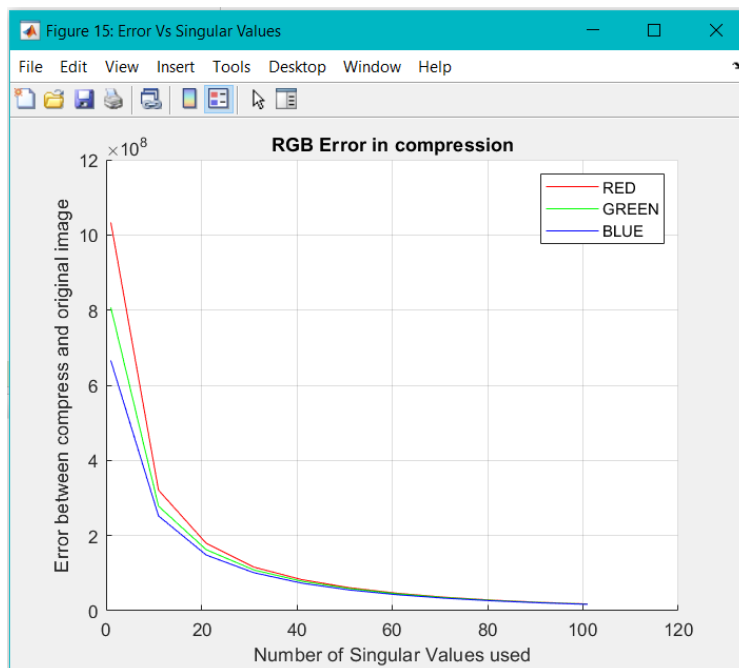
These are image components with 51 singular value, COLORED IMAGE=52.5KB



These are image components with 71 singular value, COLORED IMAGE=54.6KB



These are image components with 101 singular value, COLORED IMAGE=55.9KB



This is the graph which show error of each of the components RED, GREEN and BLUE from the original image. It is calculated by finding the difference from the original matrix to the values at every instance.

Implementation of image compression is done in both programming languages, all intermediate images and graphs are plot.

## CONCLUSION

The technique of Singular Value Decomposition has helped in manipulating a matrix and making calculations easy. When it comes to images, the image matrices are manipulated using this technique. An image size and quality depend on the number of singular values it is determined with. More the number of singular values considered; more is the quality of the image and memory space. With the above results achieved, an optimal number of singular values considered will help get an image with good quality and also not consuming lot of space. Making a matrix sparse helps in better representation of the data in it instead of keeping all the zeros, we can only represent the non-zero elements. This again constitutes in reducing space. Both the programming languages are equally efficient in compressing an image but compared to both the languages, MATLAB achieves less image size with the same quality provided as in python. With less space, digital data transfer has improved. The cost of transmission has reduced and the time consumption has reduced. Thus image processing plays a vital role in digital media communication and development.



## REFERENCES

- [1]. K. Mishra, S. Kumar Singh and P. Nagabhushan, "An Improved SVD based Image Compression," 2018 Conference on Information and Communication Technology (CICT), Jabalpur, India, 2018, pp. 1-5, doi: 10.1109/INFOCOMTECH.2018.8722414.
- [2]. Brady Mathews – “*Image compression using singular value decomposition*”- 12 December 2014, The University of Utah
- [3]. H. S. Prasantha, H. L. Shashidhara and K. N. Balasubramanya Murthy, "*Image Compression Using SVD*," International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), Sivakasi, Tamil Nadu, 2007, pp. 143-145, doi: 10.1109/ICCIMA.2007.386.
- [4]. Samruddhi Kahu and Reena Rahate, "Image Compression using singular Value Decomposition", *International Journal of Advancements in Research and Technology*, vol. 2, no. 8, August 2013.
- [5]. Gilbert Strang – “*Linear Algebra and its applications*”, “*Introduction to linear algebra*”. [https://www.academia.edu/32459792/ Strang G Linear algebra and its applications 4 5881001 PDF](https://www.academia.edu/32459792/Strang_G_Linear_algebra_and_its_applications_4_5881001_PDF)