PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

IMAGE COMPRESSION USING SVD

Submitted by

1.	Sai Shruthi S	PES2201800361
2.	Divya S	PES2201800272

Branch & Section : CSE , 'D'

PROJECT EVALUATION

(For Official Use Only)

Sl. No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor :

Signature of the Course Instructor :

INTRODUCTION

In today's day and age, sharing information has become widely digitized. Every year, people spend significant amounts of money on secondary storage devices, hard drives, digital cards etc. for personal and professional requirements of storing soft data. It is no surprise that over time, we find ourselves trying to optimize our memory usage, by erasing less important media on our electronic or storage devices, disabling auto-download preferences etc. This issue, however, has a greater magnitude of impact for large-scale users. Thus, this problem statement has been of great interest to engineers. Developments in the field of Memory Optimization include numerous sustainable techniques for saving memory space on data storage.

Speaking specific to Image - format media, one of the many approaches that can be taken to sustain memory space is compression.

Image compression is the process of minimizing the size of a graphic file in bytes, without degrading the quality of the file to an unacceptable level.

Consider the image you choose as your desktop background, that usually has dimensions 1280 x 1024. You would have to store 1,310,720 different pixel values - if it were a grayscale image. If it was colored, you would have to store triple the amount of values - having to keep track of 3,932,160 different numbers. If you think about one of those numbers equating to a byte on your computer, this sums up to 1.25MB for a grayscale image and 3.75MB for a colored image.

What we can do to save valuable memory on the image is to compress it. Thus, image compression is a very useful procedure that can help save storage space and increase transmission capacity considerably.

There are two kinds of image compression methods:

- **Lossless Compression** :

  This is a method of data compression used to reduce the size of a file while maintaining the same quality as before it was compressed. The data of the original file is rewritten in a more efficient way. Some image file formats like **PNG, RAW** and **GIF** use lossless compression. Because no quality is lost, the resulting files are typically much larger than files compressed with lossy compression. Algorithms used in **Lossy** compression are: **Transform coding, Discrete Cosine Transform, Discrete Wavelet Transform**, fractal compression etc.[1]

- **Lossy Compression**:

  This is a method of data compression in which the size of the file is reduced by eliminating data in the file. In doing so, image quality is sacrificed to decrease file size. Any data that the compression algorithm deems expendable is removed from the image, thereby reducing its size. Although the file doesn't have the same data as it did before compression was used, often this won't be noticeable, even though the resolution of the image has suffered. **JPEG** image format uses lossy compression. It is important to note that data that is compressed using lossy techniques generally cannot be recovered or reconstructed exactly. Certain lossy data compression algorithms include **Discrete Cosine Transform, Singular Value Decomposition.**

  For example in a **DSLR** camera, one has the option to save the photos in either RAW or **JPEG** format.The RAW format uses lossless compression and will tend to take up more space as unnoticeable data is not eliminated. Thus this image format is great for professional photo editors. JPEG on the other hand uses lossy compression and thus will not fill up the hard drive as fast, but some of the data is lost in the **conversion**.

1

An image is a collection of pixels. A coloured image is basically an m*n*3 array of coloured pixels, where each pixel is associated with three values that correspond to the red, green and blue colour components of the image at a specified spatial location. Therefore, the colour of any pixel is determined by the combination of the red, green, and blue intensities stored in each colour plane at the pixel's location. These red, green, and blue pixels range in saturation on a scale of 0 to 255 where **0 implies minimum brightness and 255 implies maximum brightness**. On the other hand, a grayscale image is one in which the red, green and blue components all have **equal intensities in RGB space**, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full colour image.

A large part of Linear Algebra involves operating matrices. A matrix is a table that stores data in the cells formed by intersection of rows and columns. Linear Algebra techniques are used to manipulate these matrices to convert them to convenient forms for analysis of the large amounts of data stored.

When we talk about an image, each pixel can be represented as a number and the columns and rows of the matrix hold the position of that value relative to its position on the image.

One of the techniques we can use for our matrix compression is **Singular Value Decomposition**.

SVD splits a matrix into three important submatrices to represent the data. Given a matrix A, size of A being $m \times n$ where m represents the number of rows in the matrix and n represents the number of columns, A can be broken down into three submatrices $\boldsymbol{A = U\Sigma V^{T}}$ where U is an orthogonal matrix of size $m \times m$, Σ is a diagonal matrix of size $m \times n$ and V^{T} is an orthogonal matrix of size $n \times n$.

In this report we explore **Lossy Image Compression** using **Singular Value Decomposition.**

REVIEW OF LITERATURE

Brief History :

Transform coding dates back to the late 1960s, with the introduction of fast Fourier transform (FFT) coding in 1968 and the Hadamard transform in 1969. An important development in **image** data **compression** was the discrete cosine transform (DCT), a lossy **compression** technique first proposed by Nasir Ahmed in 1972.

Reviews of Research Papers studied :

Paper - 1 [2] - Mathematics Behind Image Compression

The team found a decreasing exponential trend in compression ratio (original file size divided by the compressed file size) as the number of singular values increased. This trend makes sense because when all the singular values are used, the resulting matrix is identical to the original matrix, and the resulting image is the same as the original one. They would therefore have the same file size. In other words, as we take more singular values, we are incorporating more information about the image and thus the file size will increase. It is not very clear why this decreasing trend is exponential and not linear. A partial explanation might be that the computation complexity of the singular value decomposition is not in polynomial time and thus we see an exponential decay rather than a linear decay.

There was no trend found in how long it took to compress an image versus the number of coefficients used. The team speculated that this could be caused by Mathematica's SVD algorithm, which might calculate every singular value and then pick the ones requested. There might be some other reason behind it including the CPU performance of a specific computer,

[2]

other tasks going on behind the scene (for example, virus scan), the strength of internet connection on the specific place etc. Thus it is difficult for the team to exactly pinpoint the reason for the abnormality of the graph.

The team concluded that SVD is very good at reducing file sizes, but for very low file sizes (the compression ratio is high), it often looks worse than JPEG compressed images. As a compression algorithm, it is not often used for this reason. Another reason for this algorithm not being implemented in practice is the long time of computation.

Paper - 2 :[3] Image Compression using Singular Value Decomposition (SVD)

This research paper helped infer that all an image is, is data represented on a matrix being visually displayed through pixels of red, green and blue on a computer. This data can be manipulated through the use of the SVD theorem to calculate a level of precision close to the original without storing as much data. The SVD allows us to store (#*modes*)($m + n$) information instead of ($m \times n$) information where the size of the image is $m \times n$, or 3(#*modes*)($m + n$) when the image is in color instead of 3($m \times n$).

Paper - 3 :[4] SVD Based Image Compression

This Paper was helpful in concluding that Singular Value Decomposition (SVD) is a simple, robust and reliable technique. This SVD technique provides a stable and effective method of splitting an image matrix into a set of linearly independent matrices. SVD provides a good compression ratio and also a practical solution to image compression problems. The selection of k values plays a crucial role in the SVD based image compression technique.

3

4

REPORT ON PRESENT INVESTIGATION

An image can be represented as data in a matrix. The elements of the matrix are numbers that specify the intensity of the corresponding pixel. Singular Value Decomposition will split an m*n data matrix A into two orthogonal matrices (U, V) and a diagonal matrix (∑).

$A = U \Sigma V^T$

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

Where $u_1, u_2, \ldots u_m$ are m *1 column vectors or U

$v_1, v_2, \ldots v_n$ are 1*n row vectors of V

σ_i for i=1, 2, . . n are the singular values arranged along the diagonal of Σ

$\mathbf{AA^T = U \Sigma \Sigma^T U^T}$

$\mathbf{A^TA = V \Sigma^T \Sigma V^T}$

Thus the column vectors of U are eigenvectors of $\mathbf{AA^T}$ and the row vectors of V are the eigenvectors of $\mathbf{A^TA}$.[5]

Applying SVD alone does not compress the image. To compress an image, after applying SVD, only a few singular values have to be retained while other singular values have to be discarded. All the singular values are arranged in descending order on the diagonal of ∑ matrix. The discarding of the singular values follows the fact that the first singular value on the diagonal of ∑ contains the greatest amount of information and subsequent singular values contain

5

decreasing amounts of image information. Thus, a negligible amount of information is contained in the lower singular values. So, they can be positively discarded after performing SVD, simultaneously avoiding significant image distortion.

Furthermore, one of the properties of SVD is that 'the number of non-zero singular values of A is equal to the rank of A'. In cases where the lower order singular values after the rank of the matrix are not zero, the discarding can still be done since they have negligible values and are treated as noise.

SVD image compression process can be illustrated by implementing the following algorithm, where a given matrix A is expressed as follows:

$$A = U \Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

Now the image matrix 'A' can be represented by the outer product expansion:

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_n u_n v_n^T$$

When performing image compression, the sum is not performed to the very last Singular Values(SV's); the SV's with small enough values are dropped. The values falling outside the required rank are equated to zero. The closest matrix of rank k is obtained by truncating those sums after the first k terms:

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_k u_k v_k^T$$

The total storage for A_k will be:

$$A_k = k * (m + n + 1)$$

The value of integer k can be chosen less than n. This won't bring any significant change in the image under consideration. Hence the digital image corresponding to A_k will still have very close resemblance to the original image. As different values of k are chosen, it is observed that each value of k pertains to each corresponding image with their corresponding storage capacities. Further approximation in the image matrix can be achieved by dropping more singular terms of

the matrix A, thereby reducing the storage space of the image on the computer and achieving disk space optimization.

SVD Image Compression Measures

To compare the results of different compression techniques and also to measure the degree to which an image is compressed, many performance measures are available such as :

1. Compression Ratio (C_R):[6]

Compression Ratio is defined as the ratio of file sizes of the uncompressed image to that of the compressed image:

$$C_R = m * n / (k * (m + n + 1))$$

2. Mean Square Error (MSE) :

MSE is defined as the square of the difference between the pixel value of the original image and the corresponding pixel value of the compressed image averaged over the entire image.

Mean Square Error (MSE) is computed to measure the quality difference between the original image A and the compressed image A_k, using the following formula:

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

6

Image compression using SVD was performed on coloured and grayscale images. The procedure is as follows :

1. Compressing a grayscale image :
    - ❏ The coloured image is represented by m*n*3 data matrix. It is first converted to a grayscale image which is represented by m*n data matrix.
    - ❏ The SVD of the data matrix if found
    - ❏ The image is compressed for various values of k (the singular values beyond this are truncated) and displayed
    - ❏ Appropriate graphs are plotted

```
In [1]: from matplotlib.image import imread
        import matplotlib.pyplot as plt
        import numpy as np
        import os

In [2]: # Coloured Images
        colour_images={"Nature" :'nature.jpeg',"Image" :'image.jpeg'}

In [3]: #plotting the graphs
        def disp_graph(S):
            print(" Graphs :")
            plt.figure(1)
            plt.semilogy(np.diag(S))
            plt.title("Singular Values")
            plt.show()
            plt.figure(2)
            plt.plot(np.cumsum(np.diag(S))/np.sum(np.diag(S)))
            plt.title("Singular Values: Cumulative Sum")
            plt.show()
```

Image c.1

In [4]:

```
#Compressing a grayscale image

def compress_grayscale(image_path):
    A = imread(image_path)
    print("The coloured Image :")
    plt.imshow(A)

    width, height, channel = A.shape
    print("Width :", width)
    print("Height :", height)
    print("Channels :", channel)
    print("Number of pixels :", width * height * channel)
    plt.show()

#Converting RBG to grayscale image

    X = np.mean(A,-1)
    print("Grayscale Image :")

    width,height=X.shape

    print("Width :", width)
    print("Height :", height)
    print("Number of pixels :", width * height)

    image = plt.imshow(X)
    image.set_cmap('gray')
    plt.axis('off')
    plt.show()
    U, S, V = np.linalg.svd(X, full_matrices=False)
    S = np.diag(S)
    j = 0;
    for k in (5, 20, 25, 40, 100):
        approx_img = U[:,:k] @ S[0:k,:k] @ V[:k,:]
        plt.figure(j+1)
        j+=1
        c_image = plt.imshow(approx_img)
        c_image.set_cmap('gray')
        plt.title("For k value :"+str(k))
        print("For k value :"+str(k))
        pixel = k*(width+height+1)

        print("Number of pixels ",pixel)
        print("Compression Ratio :",round((width*height)/pixel,2))
        plt.axis('off')

        plt.show()
    disp_graph(S)
```

Image c.2

2. Compressing a coloured image :

- Coloured image is represented by m*n*3 data matrix.But SVD is applicable only on 2D matrices.Thus the layer method is used to convert the 3D matrix to 2D, compress it and then reconstruct the 3D matrix.
- In the layer method, the colour image is treated as a stack of 3 separate 2D images(red, blue and green). Truncated SVD reconstruction is applied on each of the 2D layers. The reconstructed layers are then put back together.
- The above technique is applied for various values of k(singular values beyond this are truncated) and the corresponding compressed images are displayed.
- Appropriate graphs are plotted.

In [7]:
```
#Compressing a coloured Image
def compress_svd(image,k):
    U,S,V=np.linalg.svd(image,full_matrices=False)
    recon_matrix=np.dot(U[:,:k],np.dot(np.diag(S[:k]),V[:k,:]))
    return recon_matrix,S
```

In [8]:
```
def compress_coloured_image(image_path):
    image=imread(image_path)
    w,h,c=image.shape

    print("Original Image :")
    print("Number of pixels:",w*h*c)
    plt.axis('off')
    plt.imshow(image)
    plt.show()

    original_shape=image.shape
    for k in (5,20,25,40,60,100,125,175):

        # layer method used here
        image_layers=[compress_svd(image[:,:,i],k)[0] for i in range(3)]
        image_reconst=np.zeros(image.shape)
        for i in range(3):
            image_reconst[:,:,i]=image_layers[i]
        pixel=k*(h+w+1)*3

        print("For k value :",str(k))
        print("Number of pixels :",pixel)
        print("Compression Ratio :",round((w*h*3)/pixel,2))

        image_reconst=np.array(image_reconst,np.int32)
        plt.axis('off')
        plt.imshow(image_reconst)
        plt.title("For k value :"+str(k))
        plt.show()
```

In [9]:
```
for img in colour_images:
    compress_coloured_image(colour_images[img])
```

Image c.3

RESULTS AND DISCUSSIONS

1. Compressing grayscale image :

Image 1:

Coloured Image :

Width: 179 Height :282

Channels : 3

Number of Pixels :151434

Image i.1

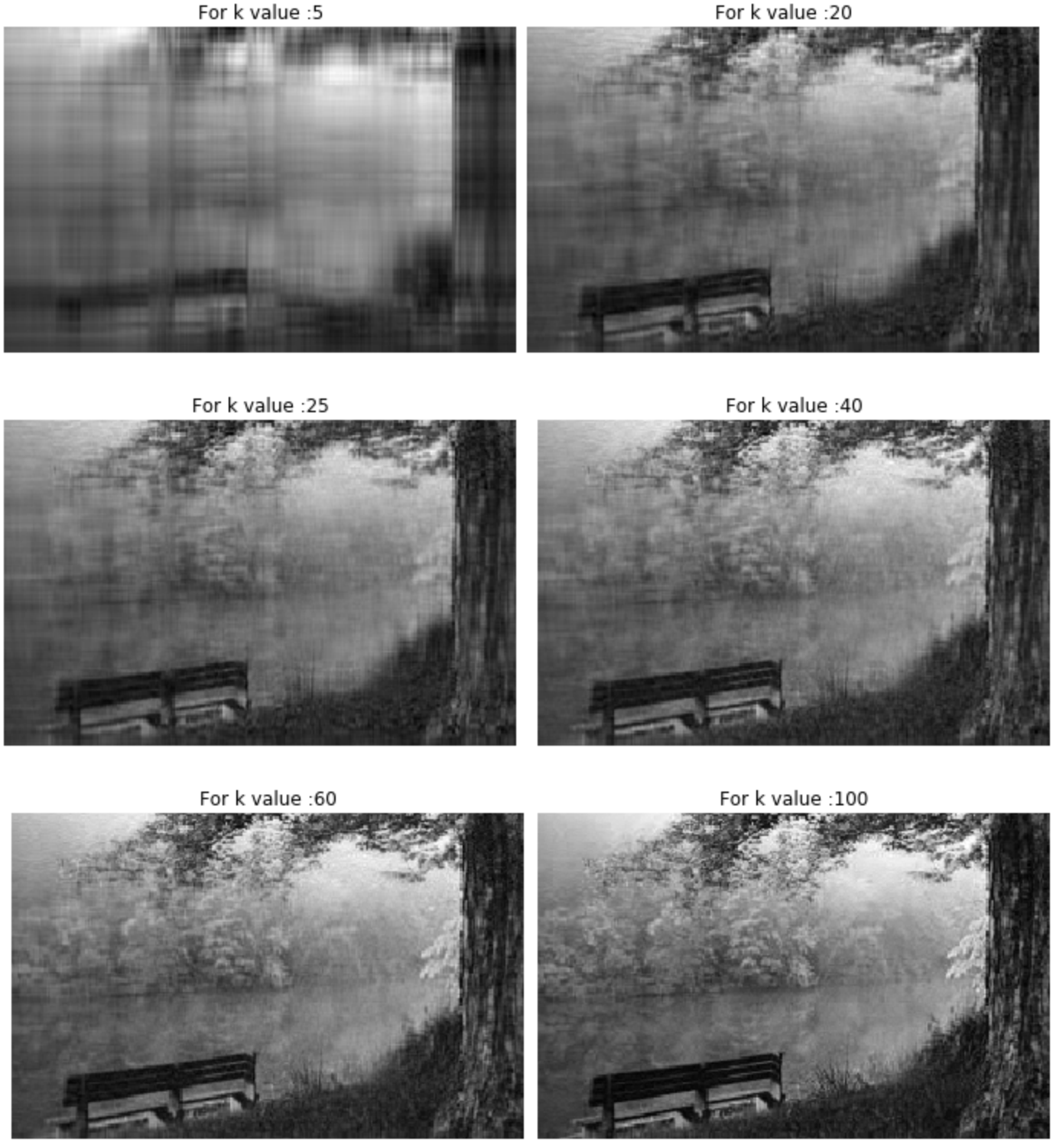
Grayscale Image(Original Image) :

Width : 179

Height : 282

Number of pixels : 50478

Image i.2



Images i.3 to i.8

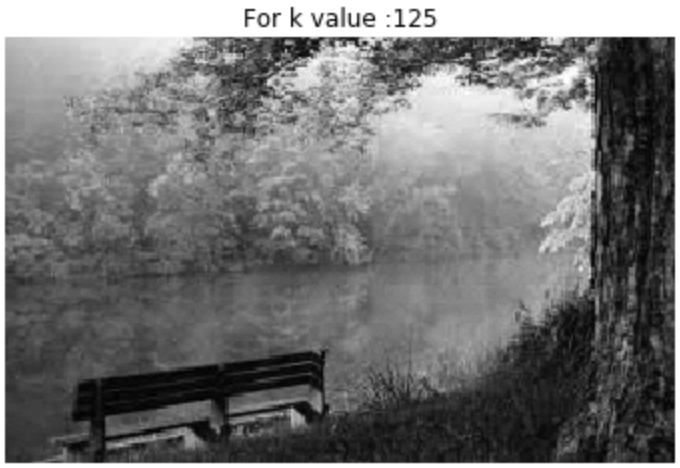


Image i.9

Image i.10

Graphical representation of Singular Values and Cumulative Singular Values describing the Σ matrix:

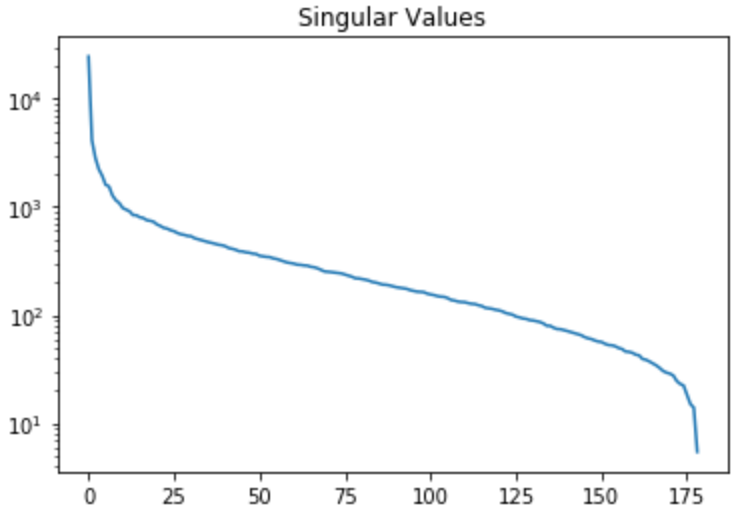


Image g.1

Legend

X axis : j

Y axis : $log\ \sigma_j$

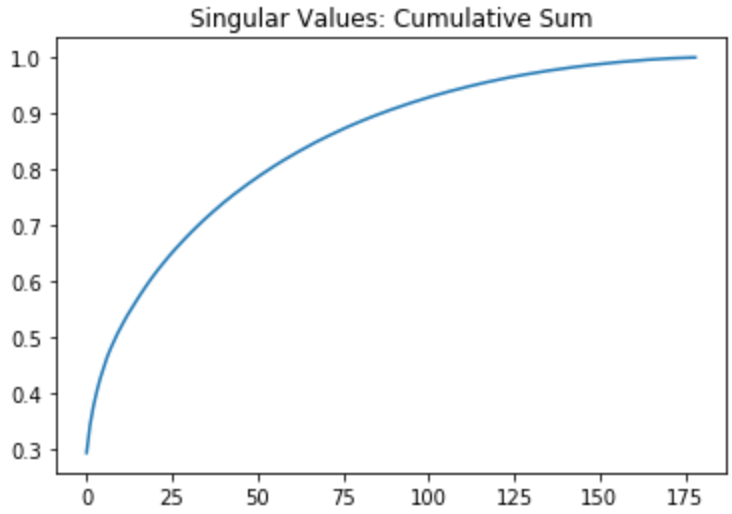


Image g.2

Legend

X axis : j

Y axis : $\sum_{j=1}^{n} \sigma_j \div \sum_{j=1}^{k} \sigma_j$

Image 2:

Coloured Image :

Width : 184 Height : 274

Channels : 3

Number of pixels : 151248

Image i.11

Grayscale Image (Original Image) :

Width : 184

Height : 274

Number of pixels : 50416

Image i.12

Image i.13 to i.18

Image i.19

Image i.20

Graphical representation of Singular Values and Cumulative Singular Values describing the Σ matrix:

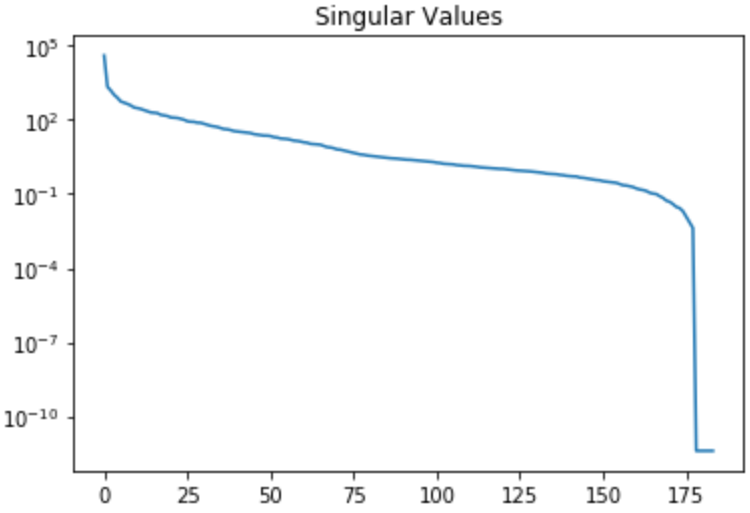


Image g.3

Legend

X axis : j

Y axis : $log\ \sigma_j$

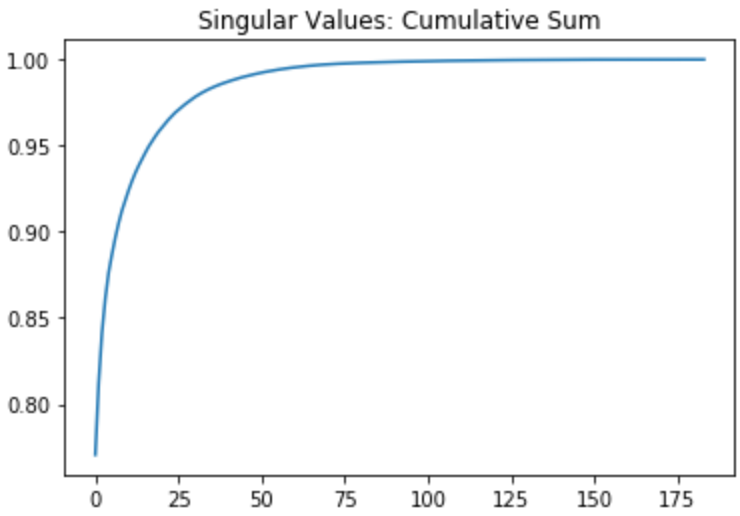


Image g.4

Legend

X axis : j

Y axis : $\sum_{j=1}^{n} \sigma_j \div \sum_{j=1}^{k} \sigma_j$

2. Compressing coloured image

Image 1:

Original Image

Width : 179 Height : 282

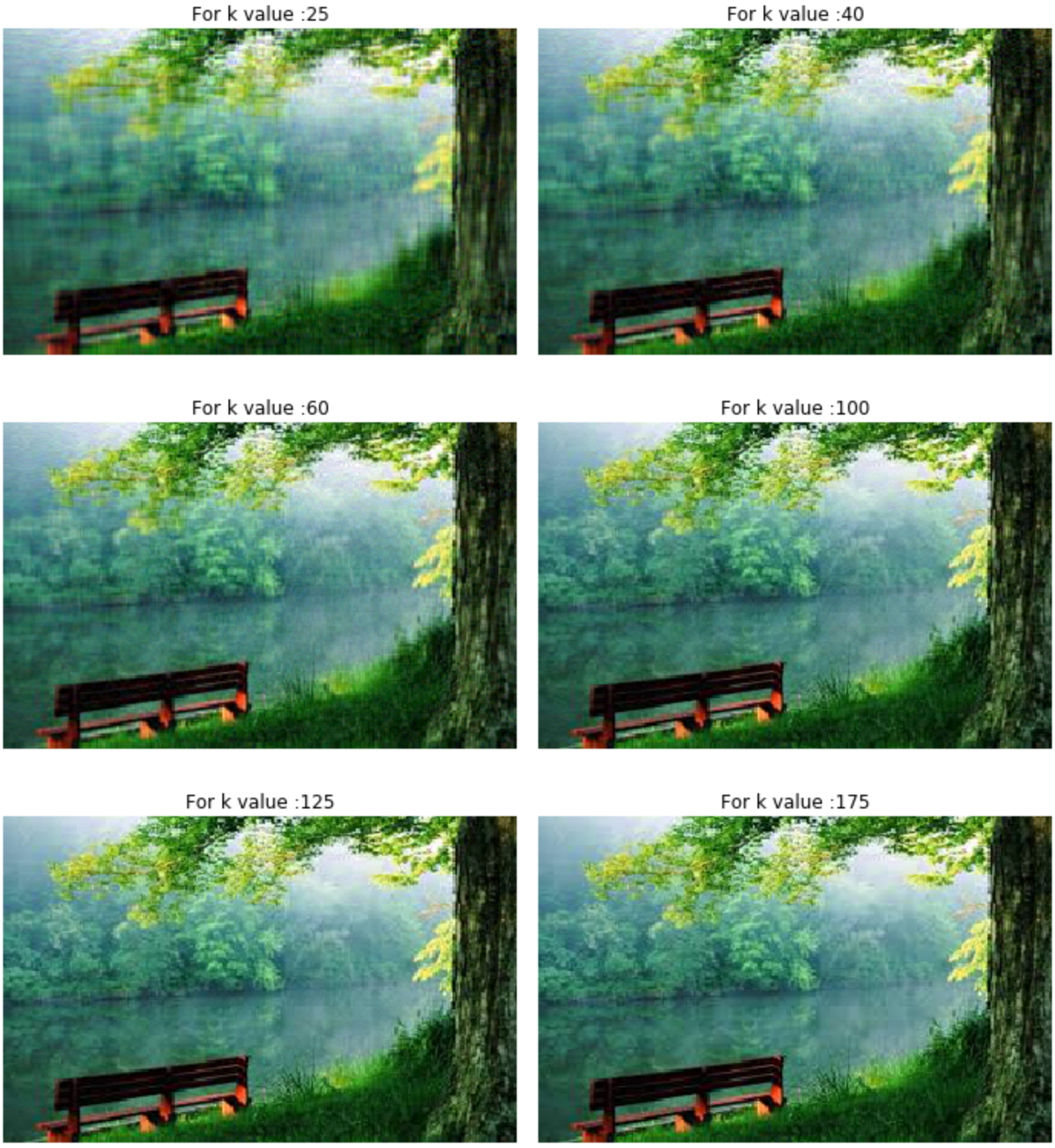
Channels : 3

Number of pixels : 151434

Image i.20

Image i.21

Image i.22



Images i.23 to i.28

Image 2:

Width : 184 Height : 274

Channels : 3

Number of pixels : 151248

Image i.29

Image i.30

Image i.31

Image i.32 to i.37

Tabular Records of k v/ s Memory Occupancy in Pixels

Image 1

Image i.38

Value of k	Compression Ratio	Number of Pixels
5	21.85	6930
20	5.46	27720
25	4.37	34650
40	2.73	55440
60	1.82	83160
100	1.09	138600
125	0.87	173250
175	0.62	242550

Table t.1

Graphical representation of relation between k and Memory occupancy in Pixels :

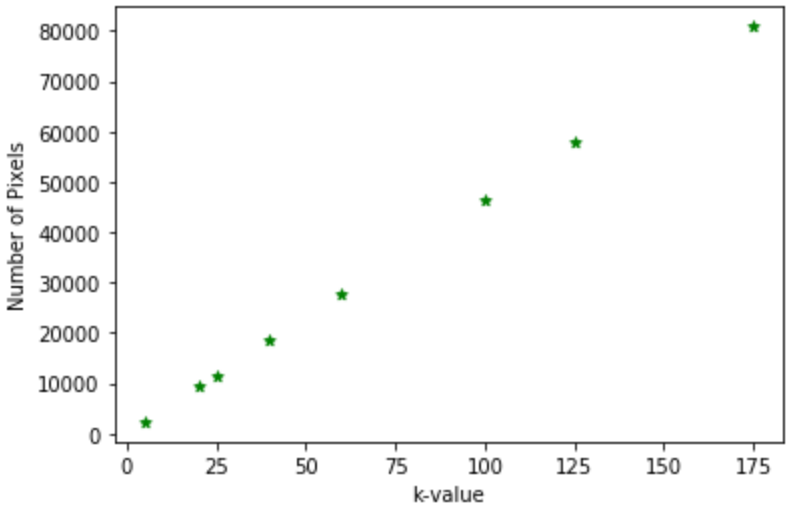


Image g.5

Image 2

Image i.39

Value of k	Compression Ratio	Number of Pixels
5	2295	21.97
20	9180	5.49
25	11475	4.39
40	18360	2.75
60	27540	1.83
100	45900	1.1
125	57375	0.88
175	80325	0.63

Table t.2

Graphical representation of relation between k and Memory occupancy in Pixels :

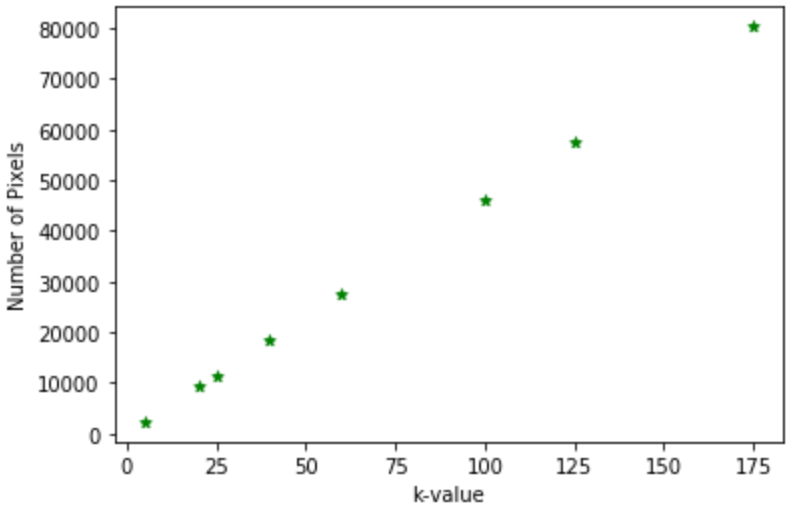


Image g.6

SUMMARY AND CONCLUSION

We have summarized below the conclusions of our report as Conceptual Conclusions that we have drawn from studying and understanding the concepts in focus, and Factual Conclusions drawn from analyzing the instances chosen for implementation.

Conceptual Conclusions :

1. SVD gives good compression results with less computational complexity compared to other compression techniques.

2. A certain degree of compression as required by an application can be achieved by choosing an appropriate value of k (i.e. the number of Eigenvalues). The degree of compression can be varied by varying the value of k.

3. To achieve a high value of compression ratio, image quality has to be sacrificed. Therefore it is required to select the proper k value to choose between compression ratio and image quality. Once the value of k is selected for specific application or for specific video the same benchmark can be used for all the frames.

   Choice of k depends on the application. In some applications, if image quality is more important than the memory consumed, higher values of k are chosen. However, sometimes, storage space is more important than image quality, in which case lower values of k are adopted.

4. When the value of k is equal to the **rank** of the image matrix, the reconstructed image is not very different from the original image. This indicates that there is very negligible decline in the image quality.

Factual Conclusions :

1. From tables t.1 and t.2, it is evident that there is a **decreasing exponential trend** in the compression ratio (original file size divided by compressed file size) as the number of singular values increased. This makes sense because when all the singular values are used, the resulting matrix is identical to the original matrix, and the resulting image is the same as the original image. Thus, they would have the same file size.

2. After analyzing the compressed images for different values of k , we can conclude that the threshold k value (any value below this will result in a distorted image) depends on the image.

   For example, for image 1 depicting a nature landscape, the k value is 60. Whereas for image 2 depicting a living room with a standing lamp, the k value is 20. It is observed that the image constructed considering only 20 singular values is similar to the original image. Further increasing the magnitude of k does not have any noticeable effect on the image quality. This might be due to the fact that a large portion of image 2 is the plain wall, having several pixels of the same RGB composition. The corresponding matrix, as a result, will contain several of the same entries. Thus, the effect of a relatively smaller k value is not as drastic as in image 1, which is composed of intricate details, demanding higher resolution.

3. From images g.5 and g.6, it can be observed clearly that the memory space occupied is directly proportional to the k value chosen during compression.

As k increases, the number of pixels in the image - the resolution of the image - increases. As the resolution increases, the memory required to store increases.

This has been established by a linearly sloped graph.

SCOPE FOR FUTURE WORK

Image processing technology extracts information from images and integrates it for a wide range of applications. Here, we have outlined the most prominent fields where image processing could bring significant benefits

1. In production automation:

   Image processing applications can make it possible for machines to act as more self-sufficient and ensure the quality of products. Damaged parts can be replaced or corrected, which would lead to more efficacies of production facilities.
2. In agricultural landscape:

   Irrigation monitoring and providing information can be made possible by tracking satellite imaging of the fields. This analysis can then be utilized in pre-harvesting operations. Growth of weeds can also be detected by using a combination of machine learning and image processing algorithms and techniques**.**
3. Biomedical and other healthcare applications :

   3D imaging creates an optical illusion of depth, assisting doctors to see extremely high quality 3D images of organs that they couldn't have seen otherwise which help them carry out delicate surgeries and make accurate diagnoses.
4. Disaster management

   Image processing can help save lives during natural disasters like flood, earthquake, wildfires, etc.

BIBLIOGRAPHY

1. Mathematics Behind Image Compression - Semantic Scholar." https://pdfs.semanticscholar.org/47e4/0066116c5e9720016a2b032e91221e9a2cc2.pdf.

2. "SVD - U of U Math - University of Utah." 12 Dec. 2014, http://www.math.utah.edu/~goller/F15_M2270/BradyMathews_SVDImage.pdf.

3. "175. svd based image compression." http://pnrsolution.org/Datacenter/Vol3/Issue2/175.pdf.

4. "Difference between Lossy Compression and Lossless ...." https://www.geeksforgeeks.org/difference-between-lossy-compression-and-lossless-compression/.

5. "Image Processing by Linear Algebra - MIT Math." https://math.mit.edu/~gs/linearalgebra/linearalgebra5_7-1.pdf

6. "Data compression ratio - Wikipedia." https://en.wikipedia.org/wiki/Data_compression_ratio.

7. **Linear Algebra and Its Applications**, Fourth Edition by Gilbert Strang, published in the year 2006, Chapter 6, Section 6.3 - SVD, Page Number 367

8. **Linear Algebra** by David Cherney, Tom Denton, Rohit Thomas and Andrew Waldron, 1st Edition, published in the year 2013, in California, 2013, Chapter 17, Sections 17.1 - 17.3 - Least Squares and Singular Values, Page Number 303