Visvesvaraya Technological University
Belagavi-590 018, Karnataka

A Mini Project Report on

"Image Compression using K-Means Algorithm"

Submitted in partial fulfillment of the requirement for the File Structures Lab [17ISL68]

Bachelor of Engineering

In

Information Science and Engineering

Submitted by

Prajna [1JT17IS028]

Under the Guidance of

Mr.Vadiraja A

Asst. Professor

Dept. Of ISE

Department of Information Science and Engineering

Jyothy Institute of Technology

Tataguni, Bengaluru-560082

2020-21

Jyothy Institute of Technology

Tataguni, Bengaluru-560082

Department of Information Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **"IMAGE COMPRESSION USING K-MEANS ALGORITHM on e-books dataset"** carried out **by Prajna [1JT17IS028]** bonfire student of Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering** in **Information Science and Engineering** department of the **Visvesvaraya Technological University, Belagavi** during the year **2020-2021**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Mr .Vadiraja A
Guide,Asst.Professor
Dept.Of ISE

Dr. HarshvardhanTiwari
Associate. Professor and HOD
Dept.Of ISE

External Viva Examiner

1.
2.

Signature with Date:

ACKNOWLEDGEMENT

Firstly, I'm very grateful to this esteemed institution **"Jyothy Institute of Technology**" for providing me an opportunity to complete my project.

I express my sincere thanks to our Principal **Dr. Gopalakrishna K, for** providing me with adequate facilities to undertake this project.

I would like to thank **Dr. Harshvardhan Tiwari, Associate Prof. and Head** of Information Science and Engineering Department for providing for his valuable support.

I would like to thank my guide **Mr.Vadiraja A, Asst. Prof.** for his keen interest and guidance in preparing this work.

Finally, I would thank all my friends who have helped me directly or indirectly in this project.

Prajna[1JT17IS028]

ABSTRACT

In recent years, the development and demand of multimedia product grows increasingly fast, contributing to insufficient bandwidth of network and storage of memory device. Therefore, the theory of data compression becomes more and more significant for reducing the data redundancy to save more hardware space and transmission bandwidth. In computer science and information theory, data compression or source coding is the process of encoding information using fewer bits or other information-bearing units than an unencoded representation. Compression is useful because it helps reduce the consumption of expensive resources such as hard disk space or transmission bandwidth. There are several different ways in which images can be compressed. In this project, I have built an image compressor using Python, Numpy and Pillow. We'll be using machine learning, the unsupervised K-means algorithm to be precise. The user is asked to input an image, which is used to demonstrate the compression. The K-means algorithm can be used to compress the image. Unlike lossless compression, K-means uses lossy compression, so it is not possible to recover the original image from the compressed image. Larger the compression ratio, the larger the difference between the compressed image and the original image.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 Introduction to File Structure

- A disk's relatively slow access time and the enormous, nonvolatile capacity is the driving force behind FILE STRUCTURE design!!
- FS should give access to all the capacity without making the application spend a lot of time waiting for the disk.
- FS is a combination of representation for data in files and of operations for accessing the data.
- It allows applications to read, write and modify data, finding the data, or reading the data in a particular order
- Efficiency of FS design for a particular application is decided based on details of the representation of the data and implementation of the operations
- A large variety in the types of data and in the needs of application makes FS design important.
- What is best for one situation may be terrible for other.

1.2 Introduction to Python

- Python is a general purpose, high-level programming language which is widely used in meteorology and elsewhere. The language facilitates a rapid development cycle with its interpreted and interactive nature. A focus on code readability makes the language easy to learn, and the expressive syntax leads to short, clear programs.
- Additionally, Python has excellent support for interfacing with legacy code written in C, C++ and Fortran making it an excellent tool for integrating existing software.
- Python has an extensive and comprehensive collection of freely available packages covering a variety of topics. Scientific Python libraries such as NumPy, SciPy, and pandas provide efficient implementation of numerical operations and tasks common in science and engineering. These libraries provide a strong base from which more advanced scientific software can be built without needing to worry about low-level algorithms. Additionally, many domain specific packages exist which address the scientific needs of the meteorological community.

- Writing software to address the challenges in meteorology is simple using the Python programming language due to the language design, available libraries and community culture.
- I have built an image compressor using Python.
- Numpy is a python library used for working with arrays.
- Pillow used for opening, manipulating and saving many different image file formats.

1.3 Introduction to Image Compression

- Image compression is a type of data compression applied to digital images, to reduce their cost for storage or transmission. Algorithms may take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data.
- Image compression may be lossy or lossless. Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics. Lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate. Lossy compression that produces negligible differences may be called visually lossless. There are different methods.
- In this project, image compression is implemented using K-Means Algorithm, a lossy compression.

1.4 GUI with Tkinter

- Python offers multiple options for developing GUI (Graphical User Interface).
- Out of all the GUI methods, Tkinter is the most commonly used method.
- It is a standard Python interface to the Tk GUI toolkit shipped with Python.
- Python with Tkinter is the fastest and easiest way to create the GUI applications.
- Creating a GUI using Tkinter is an easy task.

CHAPTER 2

METHODOLOGY

2.1 The K-means algorithm

- K means algorithm is an unsupervised machine learning algorithm. Simplified, the major difference between unsupervised and supervised machine learning algorithms is that supervised learning algorithms learn by example (labels are in the dataset), and unsupervised learning algorithms learn by trial and error; you label the data yourself.
- K is the number of clusters we want to have, hence the name K means.
- We start by randomly getting K training examples and in μ so that the points are μ1, μ2, ... , μk and $\mu \in \mathbb{R}^n$ where n is the number of features. At initialization, the points might be very close to one another, so we have to check if our result looks like how we want it to look because it might stuck be in a local optimum.
- Then for a certain number of iterations, execute the following steps:
- For every training example, assign $c^{(i)}$ to the closest centroid.
- For every centroid, set the location to be the average of examples assigned to it.
- As the algorithm progresses, generally the centroids will move to the center of the clusters and the overall distance of the examples to the clusters gets smaller.

2.2 K-means for image compression

- This technique can also be used for image compression. Each pixel of the image consists of three values, R(ed), B(lue) and G(reen). Those can be seen as the points on the grid, in 3D in case of RGB. The objective of image compression in this case is to reduce the number of colours by taking the average K colours that look closest to the original image. An image with fever colours takes up less disk space, which is what we want.
- Compressing the image can also be a preprocessing step to another algorithm. Reducing the size of the input data generally speeds up learning.

2.3 Fundamentals of Tkinter

- Tkinter is actually an inbuilt Python module used to create simple GUI apps. It is the most commonly used module for GUI apps in the Python.
- We first import the Tkinter model. Followed by that, we create the main window. It is in this window that we are performing operations and displaying visuals and

everything basically. Later, we add the widgets and lastly we enter the main event loop.

- An event loop is basically telling the code to keep displaying the window until we manually close it. It runs in an infinite loop in the back-end.
- Widgets are something like elements in the HTML. You will find different types of widgets to the different types of elements in the Tkinter. I have used the following in this project:
- Button – Button widget is used to place the buttons in the Tkinter.

```
btn1 = tk.Button(window, text="SHOW", fg='black', command=hello)
btn1.place(x=390, y=310)
```

- Entry – Entry widget is used to create input fields in the GUI.

```
txtfld1 = Entry(window, text="", bg='white', fg='black', bd=5, width=80)
txtfld1.place(x=170, y=70)
```

- Label – Label is used to create a single line widgets like text, images etc.

```
lbl = tk.Label(window, text="PLEASE ENTER THE ABSOLUTE PATH OF THE IMAGE TO
BE COMPRESSED", fg='black',font=("Helvetica", 10))
lbl.place(x=175, y=50)
```

CHAPTER 3
IMPLEMENTATION

3.1 Implementing K-means

- We start by implementing a function that creates initial points for the centroids. This function takes as input X, the training examples, and chooses K distinct points at random.
- Then we write a function to find the closest centroid for each training example. This is the first step of the algorithm. We take X and the centroids as input and return the the index of the closest centroid for every example in c, an m-dimensional vector.
- For the second step of the algorithm, we compute the distance of each example to 'its' centroid and take the average of distance for every centroid μk. Because we're looping over the rows, we have to transpose the examples.
- Finally, we got all the ingredients to complete the K-means algorithm. The maximum number of iterations is also used which is later inputed by the user . Note that if the centroids aren't moving anymore, we return the results because we cannot optimize any further.

3.2 Getting the image

- I have used the Pillow api.
- The user is supposed to specify the image by entering its absolute path.
- Because we want to get the raw image data, eg the rgb values for each pixel, we create a Numpy.
- Then we get our feature matrix X. We're reshaping the image because each pixel has the same meaning (colour), so they don't have to be presented as a grid.
- Finally we can make use of our algorithm and get the K colours. These colours are chosen by the algorithm.
- Because the indexes returned by the function are 1 iteration behind the colours, we compute the indexes for the current colours. Each pixel has a value in 0...K corresponding, of course, to its colour.
- To get a better intuition, we ask the user to enter the values for K and maximum number of iteration. Maximum number of iteration can be same as K because one iteration for one colour. You can also reduce the value of maximum number of iteration but it should not be greater than the value of K.
- Once we have all the data required we reconstruct the image by substituting the colour index with the colour and reshaping the image back to its original dimensions. Then

using the Pillow function Image.fromarray we convert the raw numbers back to an image. We also convert the indexes to integers because Numpy only accepts those as indexes for matrices.

- Finally, we save the image back to the disk.

CHAPTER 4
RESULTS AND SCREENSHOTS

4.1 Final GUI

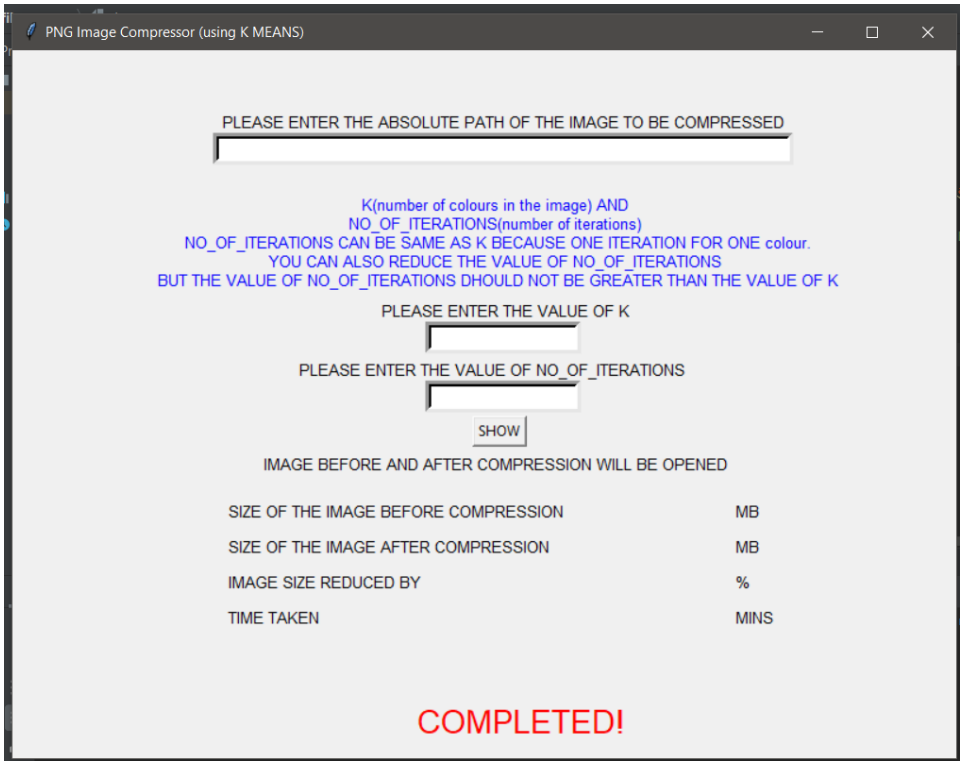


Fig 4.1.1 Final GUI

- We have the Window above created using Tkinter. There are three entry fields to enter the absolute path of the image, value of K and the number of iterations.
- I have also used button, on pressing it the program runs. Others are the labels.

4.2 Testing the result

- You can choose any (png) image you want by entering the absolute path of the image. Here is an image that I have taken using C:/Users/Pragna/Pictures/FS/pets.png, the value ok K and the number of iterations. Number of iterations can be same as K because one iteration for one colour. You can also reduce the value of number of iteration but it should not be greater than the value of K.
- And then click on the “SHOW” button.

Fig 4.2.1 Entering all the details

- As soon as the "SHOW" button is clicked, the image you have specified appears and after few minutes, the code is executed and the compressed image appears along with all the details in the Window.

Fig 4.2.2.1 Image before compression

Fig 4.2.2.2 Image after compression

- The image before compression(**Fig** 4.2.2.1) and the image after compression (**Fig** 4.2.2.2) both are displayed as and all the details of the image like size of image before compression and after compression, image size reduced by and time taken are displayed on the window as shown in **Fig** 4.2.2.3

Fig 4.2.2.3 All details

- Size of image before compression: 899 KB
- Size of image after compression: 112 KB
- Image size reduced by: 87.53 %

- Time taken: 3.1396 MINS

4.3 Analysis

- Now lets us analyse the results using three different cases.
- **CASE 1:**

  K = NO_OF_ITERATIONS = 20 for 7 different image files.

  Image1:

Fig 4.3.1.1 Image1 before compression

Fig 4.3.1.2 Image1 after compression

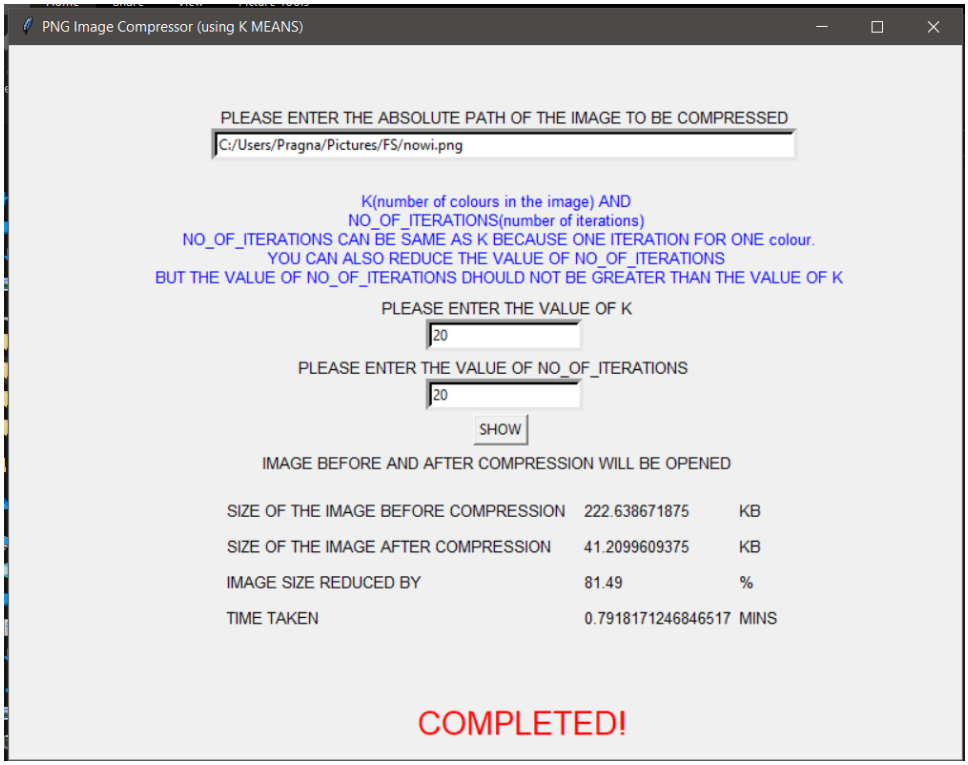


Fig 4.3.1.3 All details

- Image1 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 222 KB
- Size of image after compression: 41 KB
- Image size reduced by: 81.49 %
- Time taken: 0.7918 MINS

Image2:

Fig 4.3.2.1 Image2 before compression

Fig 4.3.2.2 Image2 after compression

Fig 4.3.2.3 All details

- Image2 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 333 KB
- Size of image after compression: 90 KB
- Image size reduced by: 72.91 %
- Time taken: 2.1505 MINS

Image3:

Fig 4.3.3.1 Image3 before compression

Fig 4.3.3.2 Image3 after compression

Fig 4.3.3.3 All details

- Image3 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 341 KB
- Size of image after compression: 74 KB
- Image size reduced by: 78.08 %
- Time taken: 2.5044 MINS

Image4:

Fig 4.3.4.1 Image4 before compression

Fig 4.3.4.2 Image4 after compression

Fig 4.3.4.3 All details

- Image4 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 112 KB
- Image size reduced by: 87.53 %
- Time taken: 3.1396 MINS

Image5:

Fig 4.3.5.1 Image5 before compression

Fig 4.3.5.2 Image5 after compression

Fig 4.3.5.3 All details

- Image5 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 740 KB
- Size of image after compression: 156 KB
- Image size reduced by: 78.85 %
- Time taken: 3.5146 MINS

Image6:

Fig 4.3.6.1 Image6 before compression

Fig 4.3.6.2 Image6 after compression

Fig 4.3.6.3 All details

- Image6 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 1230 KB
- Size of image after compression: 175 KB
- Image size reduced by: 85.71 %
- Time taken: 6.6305 MINS

Image7:

Fig 4.3.7.1 Image7 before compression

Fig 4.3.7.2 Image7 after compression

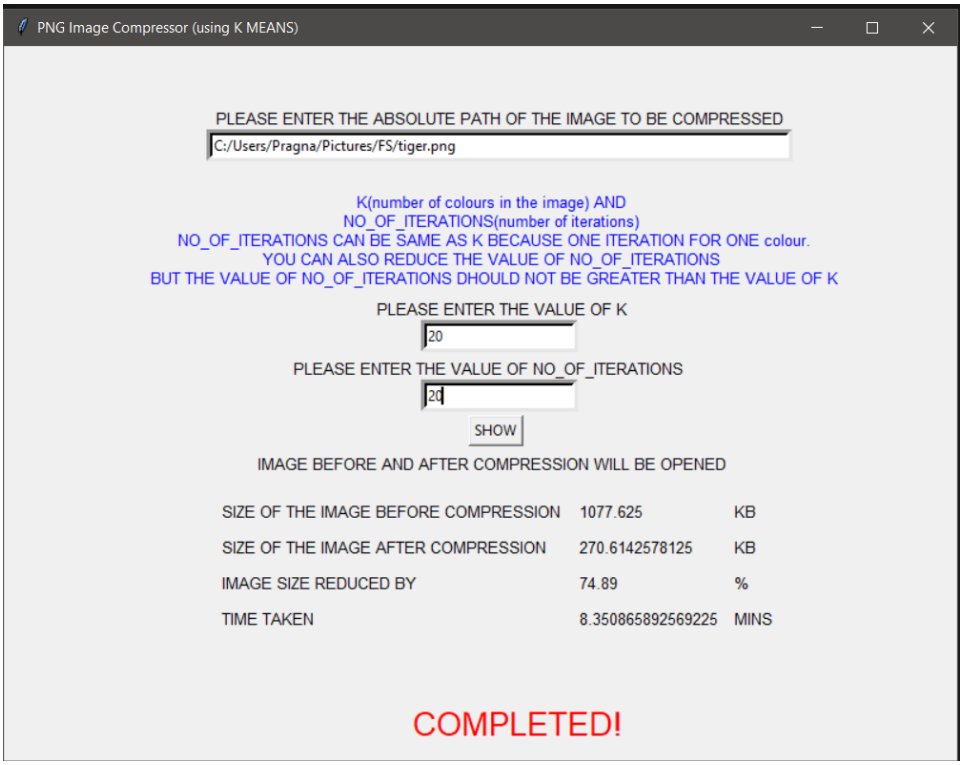


Fig 4.3.7.3 All details

- Image7 is compressed where K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 1077 KB
- Size of image after compression: 270 KB
- Image size reduced by: 74.89 %
- Time taken: 8.3508 MINS

- We have 2 graphs for **CASE 1** (of above 7 different images) as shown below in **Fig** 4.3.8 and **Fig** 4.8.9

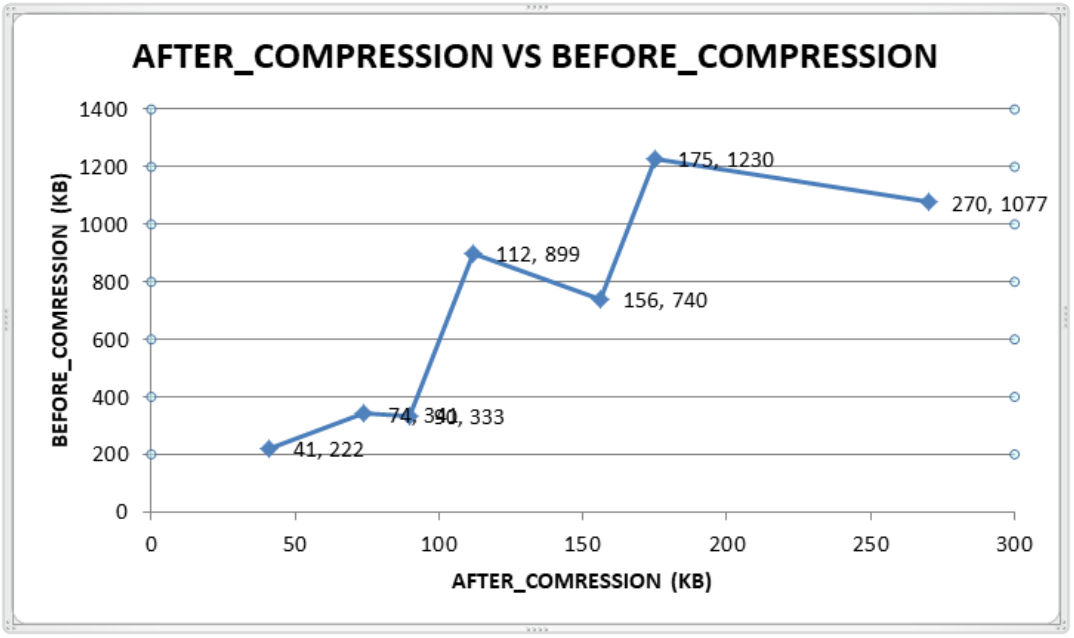


Fig 4.3.8 AFTER_COMPRESSION VS BEFORE_COMPRESSION

- In this graph, we can see that image size before compression is reduced from 222KB, 333KB, 341KB, 740KB, 899KB, 1077KB, 1230KB to 41KB, 74KB, 90KB, 156KB, 112KB, 270KB, 175KB respectively after compression.
- Images of when K = NO_OF_ITERATIONS = 20 is reduced by an average of 79.98 %

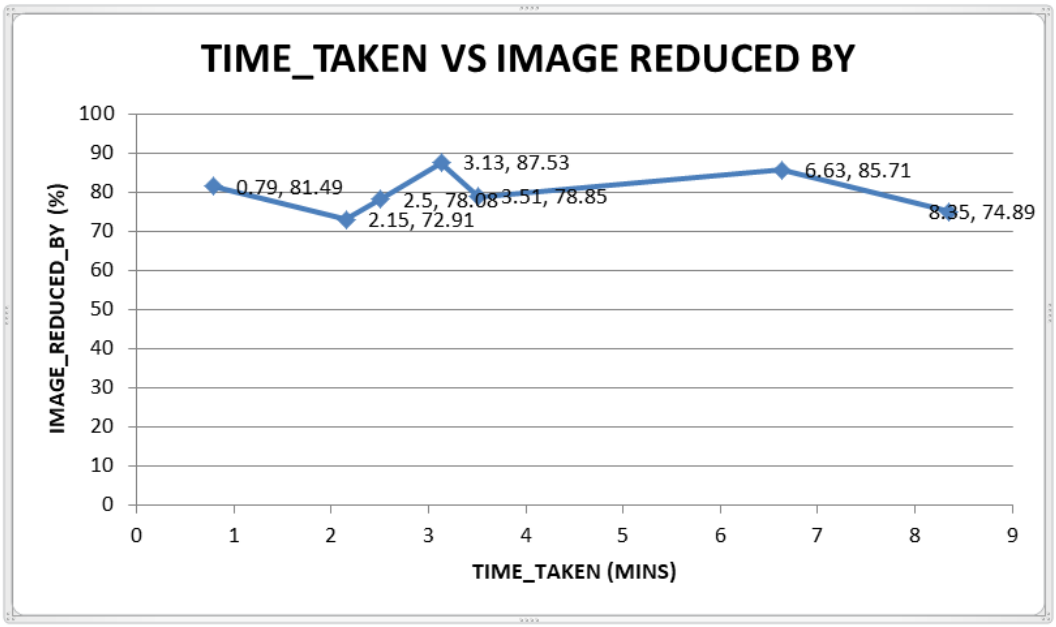


Fig 4.3.9 TIME_TAKEN VS IMAGE_REDUCED_BY

- In this graph, we can see the time taken to reduce the image by 81.49%, 72.91%, 78.08%, 87.53%, 78.85%, 85.71%, 874.89% in 0.79MINS, 2.15MINS, 2.50MINS, 3.13MINS, 3.51MINS, 6.63MINS, 8.35MINS respectively.
- Images when K = NO_OF_ITERATIONS = 20 is reduced by an average of 79.98 % in an average time of 3.86MINS.

- **CASE 2:**

  K = NO_OF_ITERATIONS for the same image file but for different K values.

Image before compression:

Fig 4.3.10.1 Image before compression

Image after compression and K = 50:

Fig 4.3.10.2 Image after compression

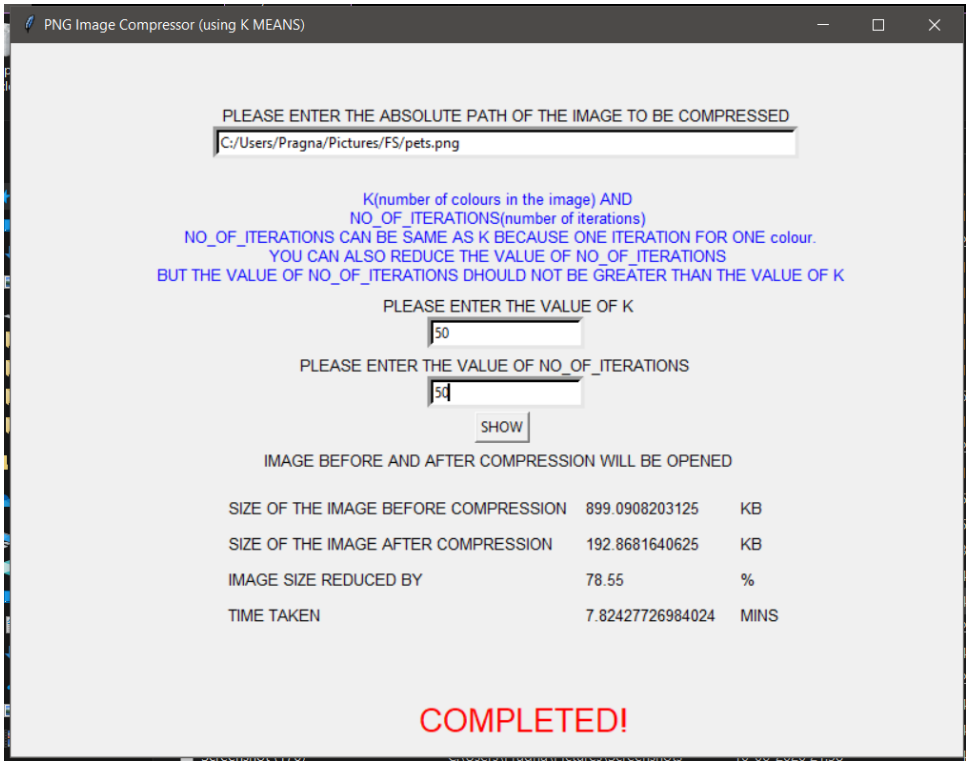


Fig 4.3.10.2.1 All details

- K = NO_OF_ITERATIONS = 50 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 192 KB
- Image size reduced by: 78.55 %
- Time taken: 7.8242 MINS

Image after compression and K = 45:

Fig 4.3.10.3 Image after compression

Fig 4.3.10.3.1 All details

- K = NO_OF_ITERATIONS = 45 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 181 KB
- Image size reduced by: 79.81 %
- Time taken: 7.2556 MINS

Image after compression and K = 40:

Fig 4.3.10.4 Image after compression

Fig 4.3.10.4.1 All details

- K = NO_OF_ITERATIONS = 40 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 163 KB
- Image size reduced by: 81.84 %
- Time taken: 3.6721 MINS

Image after compression and K = 35:

Fig 4.3.10.5 Image after compression

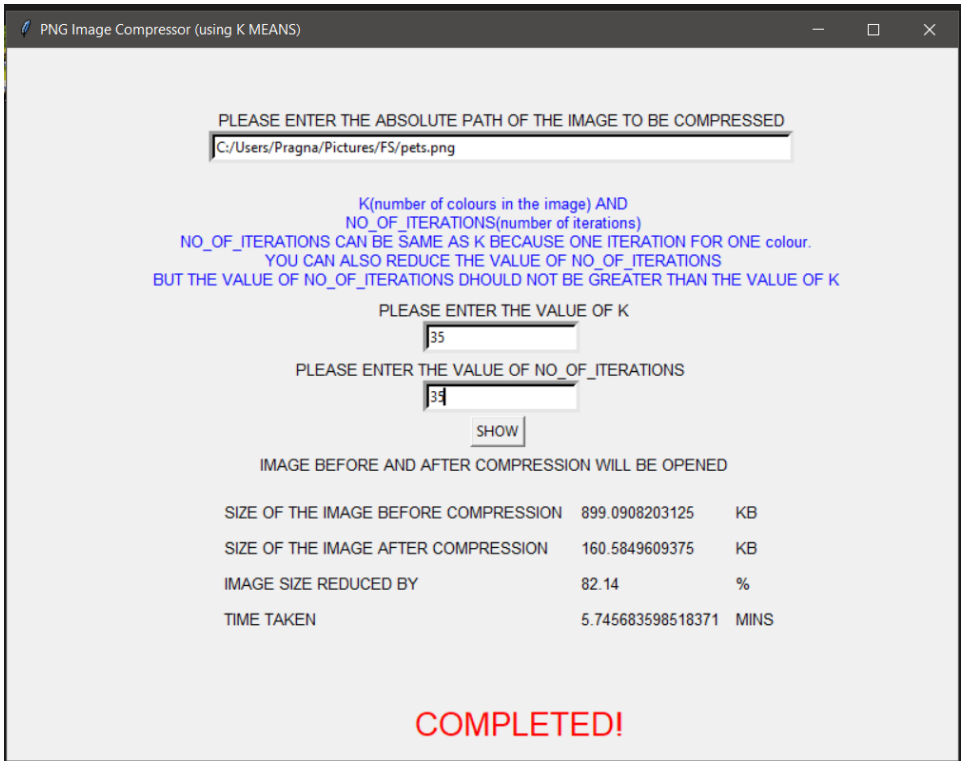


Fig 4.3.10.5.1 All details

- K = NO_OF_ITERATIONS = 35 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 160 KB
- Image size reduced by: 82.14 %
- Time taken: 5.7456 MINS

Image after compression and K = 30:

Fig 4.3.10.6 Image after compression

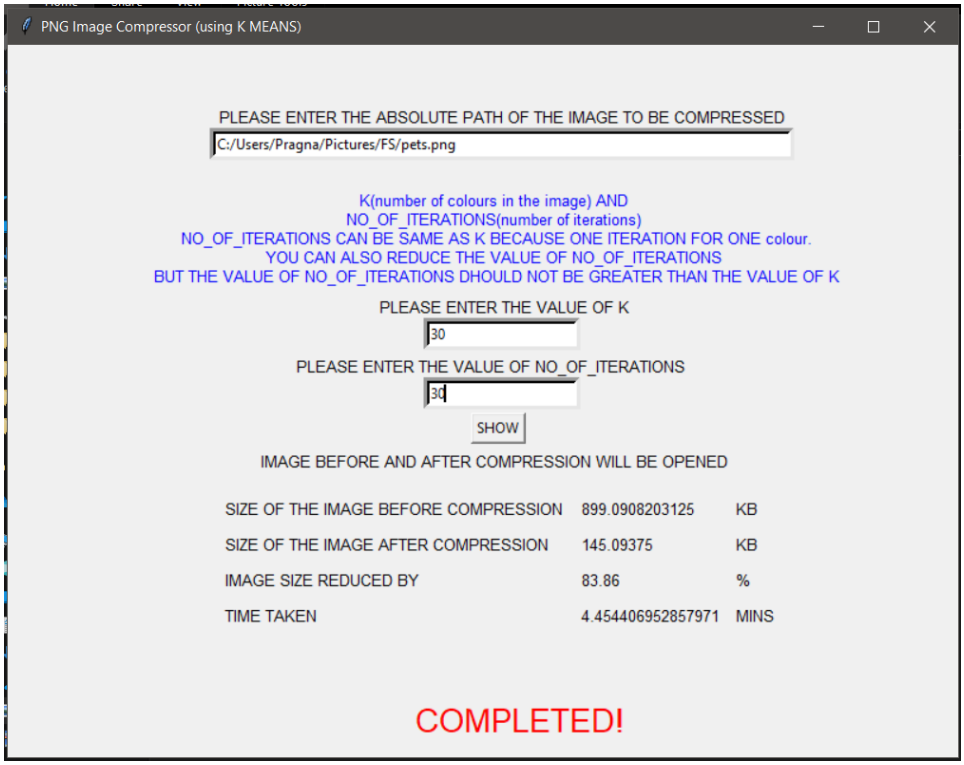


Fig 4.3.10.6.1 All details

- K = NO_OF_ITERATIONS = 30 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 145 KB
- Image size reduced by: 83.86 %
- Time taken: 4.4544 MINS

Image after compression and K = 25:

Fig 4.3.10.7 Image after compression

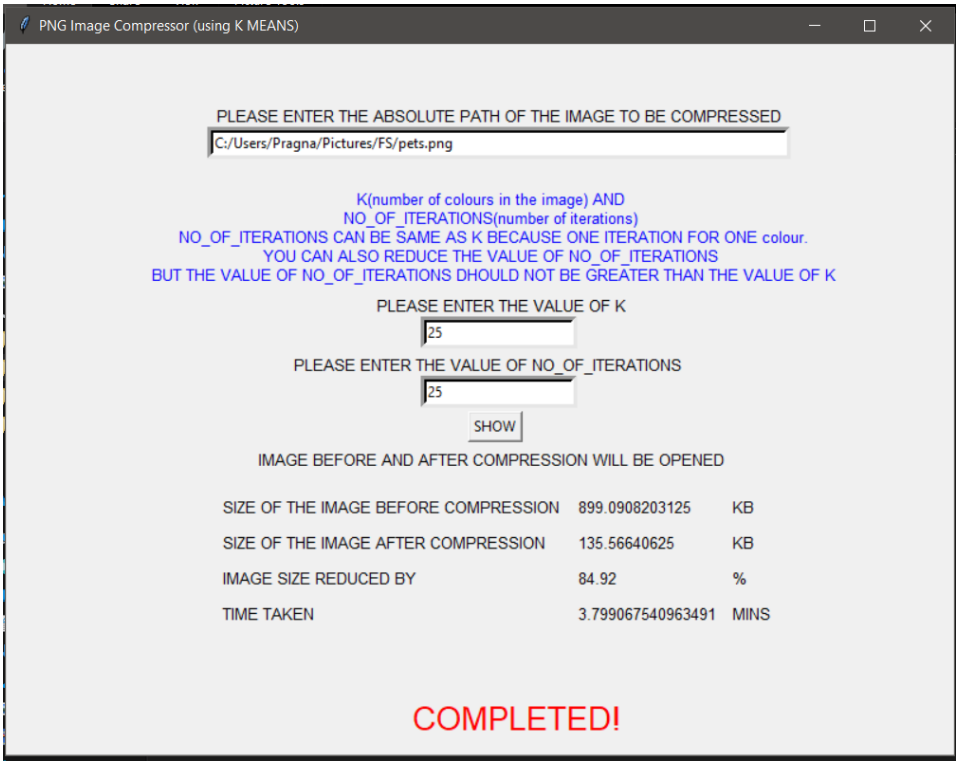


Fig 4.3.10.7.1 All details

- K = NO_OF_ITERATIONS = 25 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 135 KB
- Image size reduced by: 84.92 %
- Time taken: 3.7990 MINS

Image after compression and K = 20:

Fig 4.3.10.8 Image after compression

Fig 4.3.10.8.1 All details

- K = NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 112 KB
- Image size reduced by: 87.53 %
- Time taken: 3.1396 MINS

Image after compression and K = 15:

Fig 4.3.10.9 Image after compression

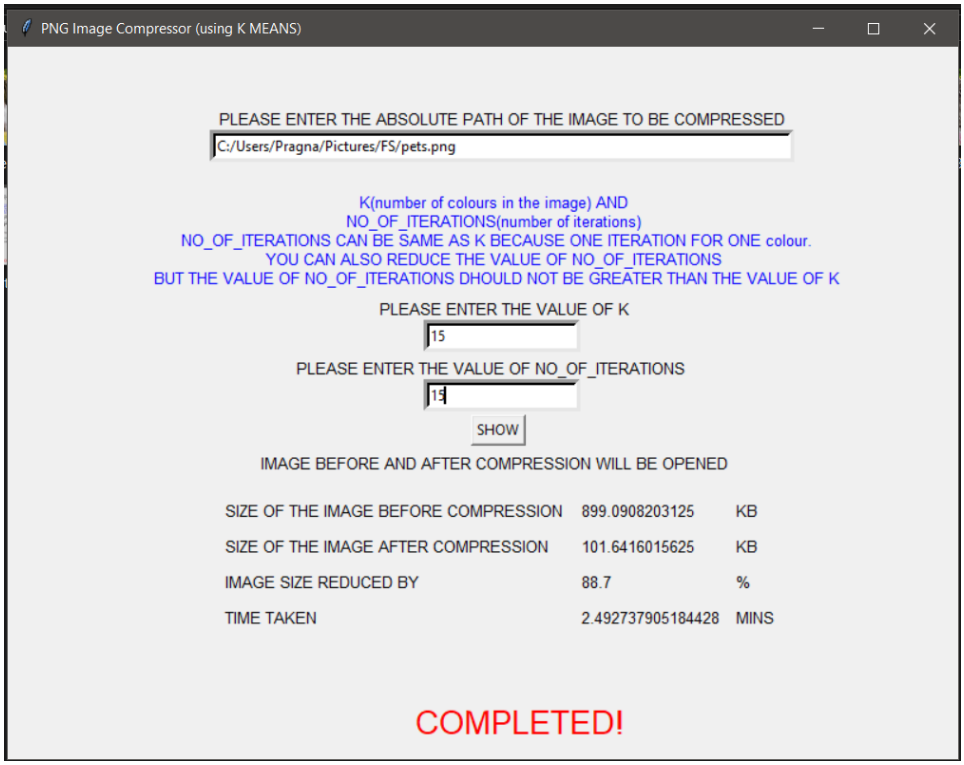


Fig 4.3.10.9.1 All details

- K = NO_OF_ITERATIONS = 15 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 101 KB
- Image size reduced by: 88.7 %
- Time taken: 2.4927 MINS

Image after compression and K = 10:

Fig 4.3.10.10 Image after compression

Fig 4.3.10.10.1 All details

- K = NO_OF_ITERATIONS = 10 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 80 KB
- Image size reduced by: 91.03 %
- Time taken: 1.7306 MINS

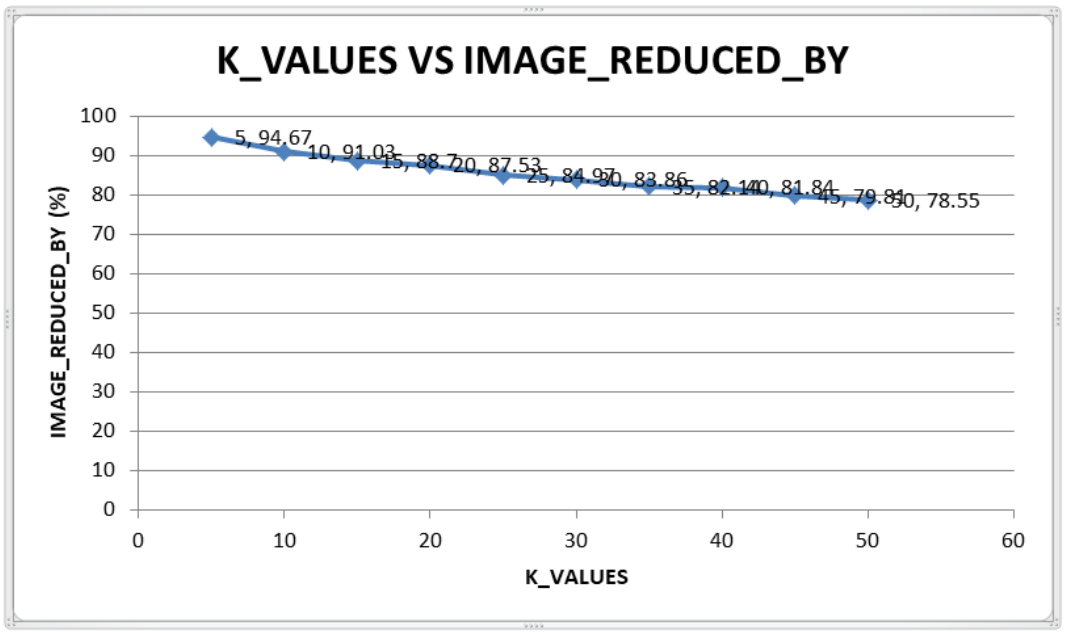
Image after compression and K = 5:

Fig 4.3.10.11 Image after compression

Fig 4.3.10.11.1 All details

- K = NO_OF_ITERATIONS = 5 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 47 KB
- Image size reduced by: 94.67 %
- Time taken: 1.1080 MINS

- We have 3 graphs for **CASE 2** (different values of K for the same image file) as shown below in **Fig** 4.3.10.12 , **Fig** 4.3.10.13 and **Fig** 4.3.10.14



K_VALUE = NO_OF_ITERATIONS

Fig 4.3.10.12 K_VALUES VS IMAGE_REDUCED_BY

- In this graph, we can see the image reduced by 78.55%, 79.81%, 81.84%, 82.14%, 83.86%, 84.97%, 87.53%, 88.7%, 91.03%, 94.67% as K is 50, 45, 40, 35, 30, 25, 20, 15, 10, 5 respectively.
- Here as the value of K decreases, the compression ratio increases.

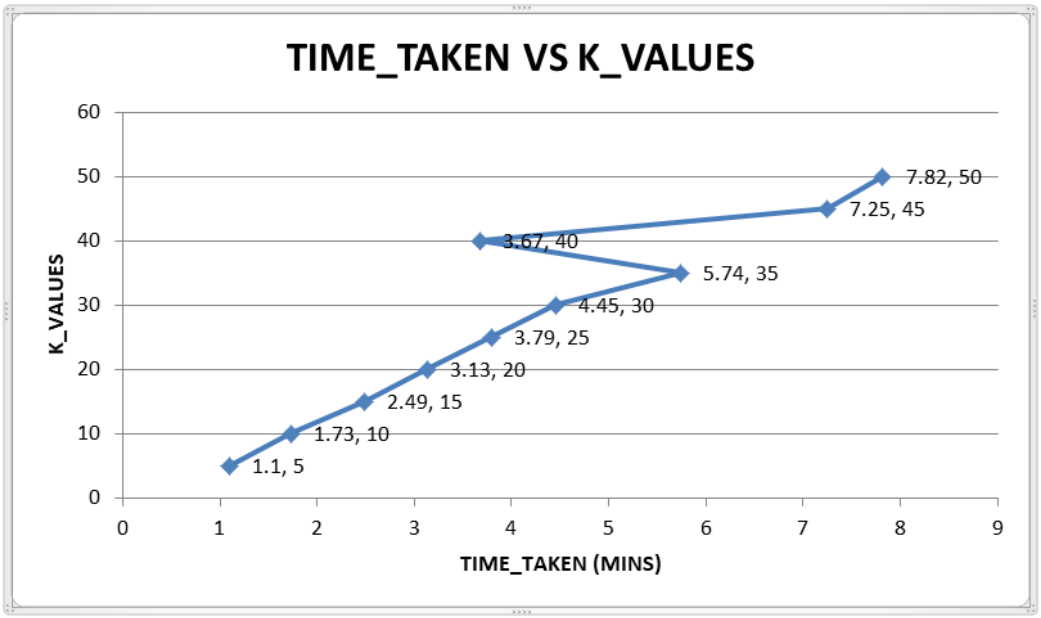


Fig 4.3.10.13 TIME_TAKEN VS K_VALUES

- In this graph, we can see the time taken for K value 50, 45, 40, 35, 30, 25, 20, 15, 10, 5 is 7.82MINS, 7.25MINS, 3.67MINS, 5.74MINS, 4.45MINS, 3.79MINS, 3.13MINS, 2.49MINS, 1.73MINS, 1.10MINS respectively.
- On an average, as value of K decreases, the time taken to compress the image also reduces.

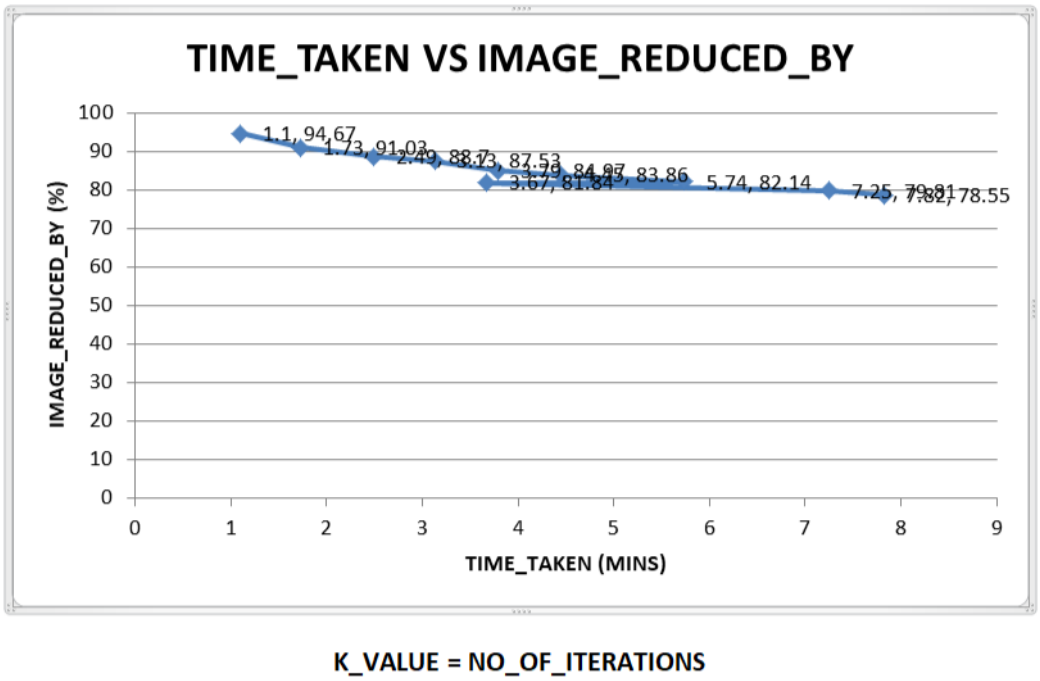


Fig 4.3.10.14 TIME_TAKEN VS IMAGE_REDUCED_BY

- In this graph, we can see the time taken to reduce the image by 81.49%, 72.91%, 78.08%, 87.53%, 78.85%, 85.71%, 874.89% in 0.79MINS, 2.15MINS, 2.50MINS, 3.13MINS, 3.51MINS, 6.63MINS, 8.35MINS respectively.
- Images when K = NO_OF_ITERATIONS = 20 is reduced by an average of 79.98 % in an average time of 3.86MINS.

- **CASE 3:**

  K=40 and K ≠ NO_OF_ITERATIONS for the same image file but for different values of NO_OF_ITERATIONS.

Image before compression:

Fig 4.3.10.15 Image before compression

Image after compression and NO_OF_ITERATIONS = 40:

Fig 4.3.10.16 Image after compression

Fig 4.3.10.16.1 All details

- NO_OF_ITERATIONS = 40 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 163 KB
- Image size reduced by: 81.84 %
- Time taken: 3.6721 MINS

Image after compression and NO_OF_ITERATIONS = 35:

Fig 4.3.10.17 Image after compression

Fig 4.3.10.17.1 All details

- NO_OF_ITERATIONS = 35 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 167 KB
- Image size reduced by: 81.33 %
- Time taken: 3.1667 MINS

Image after compression and NO_OF_ITERATIONS = 30:

Fig 4.3.10.18 Image after compression

Fig 4.3.10.18.1 All details

- NO_OF_ITERATIONS = 30 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 172 KB
- Image size reduced by: 80.78 %
- Time taken: 2.6005 MINS

Image after compression and NO_OF_ITERATIONS = 25:

Fig 4.3.10.19 Image after compression

Fig 4.3.10.19.1 All details

- NO_OF_ITERATIONS = 25 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 170 KB
- Image size reduced by: 81.05 %
- Time taken: 2.1411 MINS

Image after compression and NO_OF_ITERATIONS = 20:

Fig 4.3.10.20 Image after compression

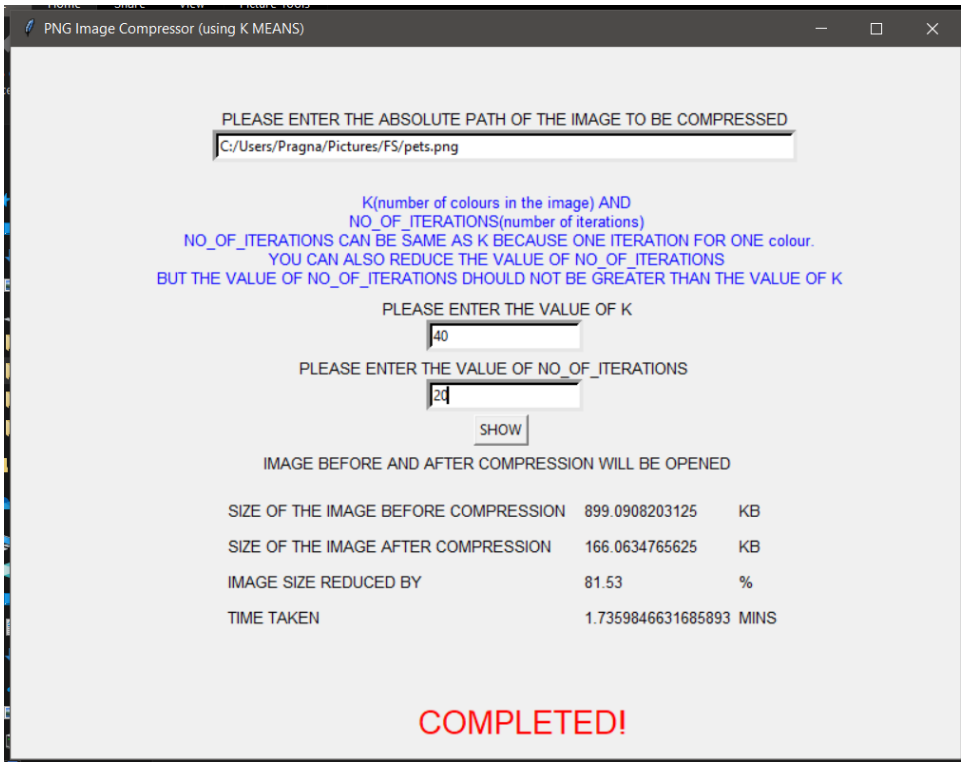


Fig 4.3.10.20.1 All details

- NO_OF_ITERATIONS = 20 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 166 KB
- Image size reduced by: 81.53 %
- Time taken: 1.7359 MINS

Image after compression and NO_OF_ITERATIONS = 15:

Fig 4.3.10.21 Image after compression

Fig 4.3.10.21.1 All details

- NO_OF_ITERATIONS = 15 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 162 KB
- Image size reduced by: 81.97 %
- Time taken: 1.43860 MINS

Image after compression and NO_OF_ITERATIONS = 10:

Fig 4.3.10.22 Image after compression

Fig 4.3.10.22.1 All details

- NO_OF_ITERATIONS = 10 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 170 KB
- Image size reduced by: 81.06 %
- Time taken: 1.0382 MINS

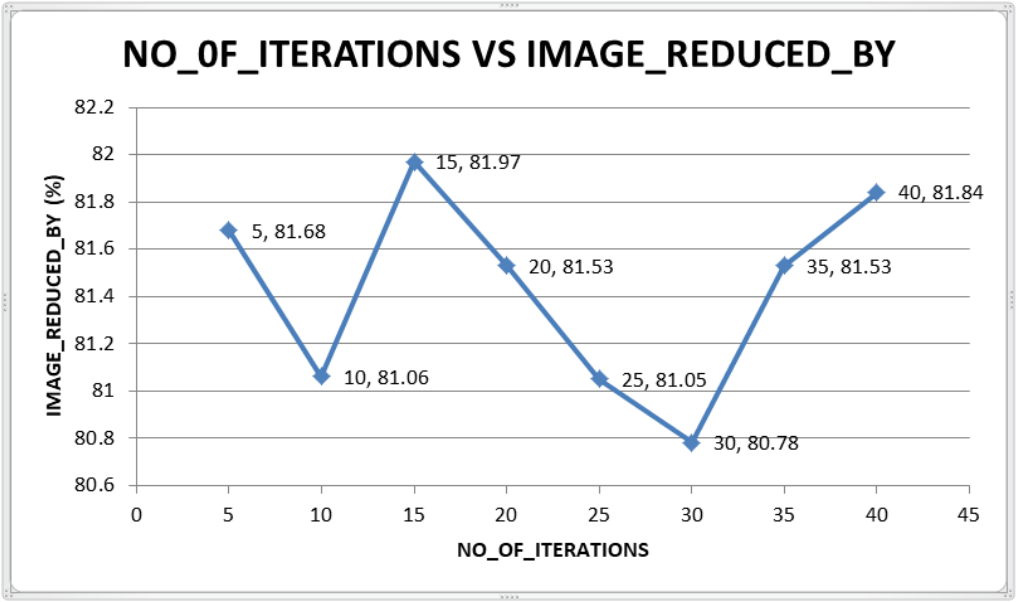
Image after compression and NO_OF_ITERATIONS = 5:

Fig 4.3.10.23 Image after compression

Fig 4.3.10.23.1 All details

- NO_OF_ITERATIONS = 5 and all the details are given:
- Size of image before compression: 899 KB
- Size of image after compression: 164 KB
- Image size reduced by: 81.68 %
- Time taken: 0.6597 MINS

We have 3 graphs for **CASE 3** (K=40 and K ≠ NO_OF_ITERATIONS for the same image file but for different values of NO_OF_ITERATIONS) as shown below in **Fig** 4.3.10.24, **Fig** 4.3.10.25 and **Fig** 4.3.10.26



K_VALUE ≠ NO_OF_ITERATIONS

K_VALUE= 40

Fig 4.3.10.24 NO_OF_ITERATIONS VS IMAGE_REDUCED_BY

- In this graph, we can see the image reduced by 81.84%, 81.53%, 80.78%, 81.05%, 81.53%, 81.97%, 81.06%, 81.68% as the number of iterations are 40, 35, 30, 25, 20, 15, 10, 5 respectively.
- Here there is a almost constant change in compression ratio as the number of iterations changes.

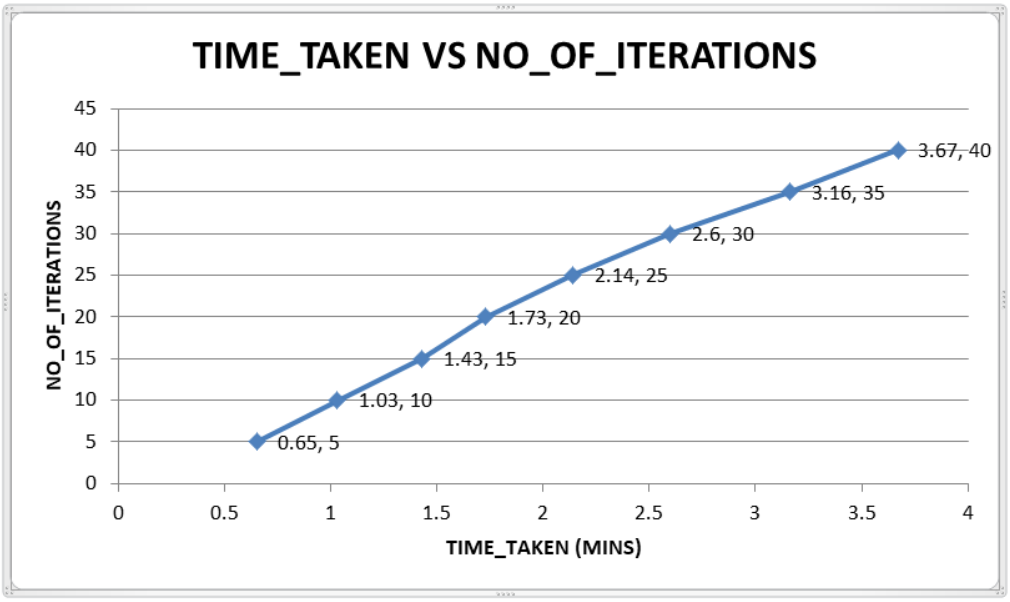
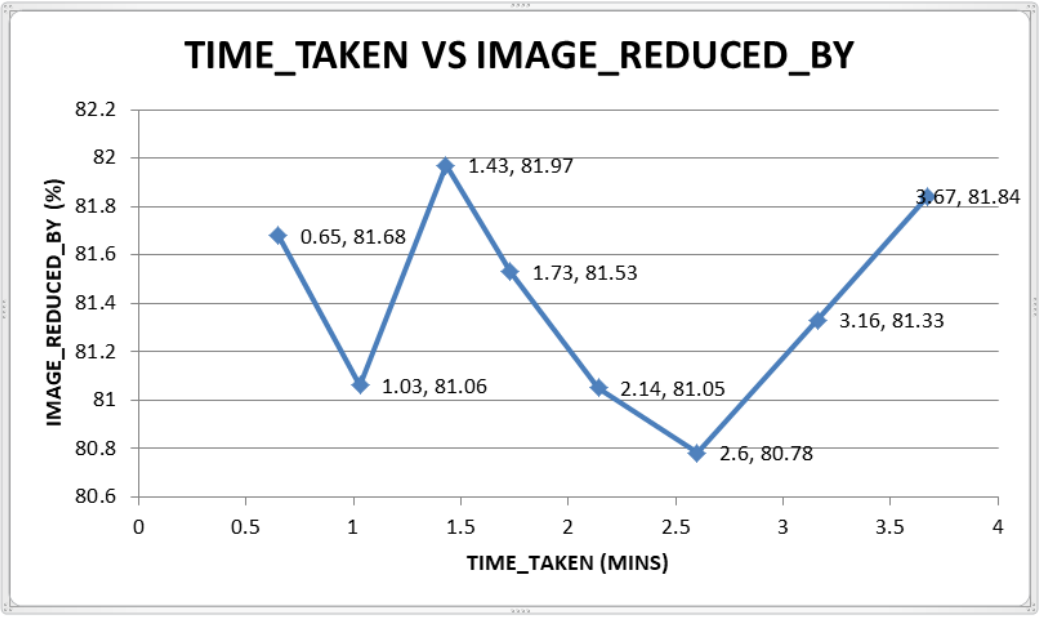


Fig 4.3.10.25 TIME_TAKEN VS NO_OF_ITERARTIONS

- In this graph, we can see the taken taken when number of iterations is 40, 35, 30, 25, 20, 15, 10, 5 is 3.67MINS, 3.16MINS, 2.6MINS, 2.14MINS, 1.73MINS, 1.43MINS, 1.03MINS, 0.65MINS respectively.
- Here as number of iterations decreases, the time taken to compress the image also decreases.



K_VALUE ≠ NO_OF_ITERATIONS

K_VALUE= 40

Fig 4.3.10.26 TIME_TAKEN VS IMAGE_REDUCED_BY

- In this graph, we can see the image reduced is by 81.84%, 81.53%, 80.78%, 81.05%, 81.53%, 81.97%, 81.06%, 81.68% in 3.67MINS, 3.16MINS, 2.6MINS, 2.14MINS, 1.73MINS, 1.43MINS, 1.03MINS, 0.65MINS respectively.
- Here the time taken to compress the image decreases as the compression ratio increases very slowly.

CONCLUSION

We have successfully implemented image compression using K-means algorithm which is used to shrink the size of an image while maintaining the resolution.

It is easy to reduce the file size of an image using this algorithm. But it takes more time for larger files. Larger the compression ratio, the larger the difference between the compressed image and the original image. Though number of iterations were reduced while keeping K constant, the time taken was reasonable and also the loss was less and compressed file size was also acceptable.

K-means is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of K-means is to group data points into distinct non-overlapping subgroups. It does a very good job when the clusters have a kind of spherical shapes. However, it suffers as the geometric shapes of clusters deviates from spherical shapes. Moreover, it also doesn't learn the number of clusters from the data and requires it to be pre-defined. To be a good practitioner, it's good to know the assumptions behind algorithms/methods so that you would have a pretty good idea about the strength and weakness of each method. This will help to decide when to use each method and under what circumstances.

REFERENCES

[1]https://rickwierenga.com/blog/machine%20learning/image-compressor-in-Python.html

[2] https://www.kaggle.com/stefanjaro/image-compression-with-k-means-clustering

[3] https://stackoverflow.com/

[4] https://www.w3schools.in/python-tutorial/gui-programming/

[5] https://www.geeksforgeeks.org/k-means-clustering-introduction/

[6] https://iopscience.iop.org/article/10.1088/1757-899X/563/5/052042/pdf

[7]https://www.researchgate.net/publication/335080736_Application_of_K-means_Algorithm_in_Image_Compression

[8] https://github.com/preetmishra/image-compression-kmeans

[9] https://appliedmachinelearning.blog/2017/03/08/image-compression-using-k-means-clustering/

[10] https://www.tutorialspoint.com/python/python_gui_programming.htm

[11] https://www.youtube.com/watch?v=4b5d3muPQmA&t=96s

[12] https://www.youtube.com/watch?v=D8-snVfekto&t=1316s

[13]https://www.youtube.com/watch?v=F5PfbC5ld-Q&t=949s

[14]Michael J.Folk,Bill Zoeclick,Greg Riccardi:File Structures-An Object Oriented Approach with C++,3rd Edition,Pearson Education ,1998.

[15]Scott Robert Ladd:C++ Components and Algorithms,BPB Publications,1993.