A Project Report on

Huffman Based LZW Lossless Image Compression

Using Retinex Algorithm

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

In

Computer Science and Engineering

Submitted by

R.YESHWANTH RAJU
(20H51A0546)
M.PAVAN KUMAR
(20H51A05H6)

Under the esteemed guidance of
Mr. BK .CHINNA MADDILETI
(Assistant Professor)

Department of Computer Science and Engineering
CMR COLLEGE OF ENGINEERING & TECHNOLOGY
(UGC Autonomous)
*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A^+ Grade
KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2020- 2024

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Major Project Phase I report entitled " **Huffman Based LZW Lossless Image Compression Using Retinex Algorithm** " being submitted R. Yeshwanth Raju (20H51A0546), M. Pavan Kumar (20H51A05H6) in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

Mr. BK Chinna Maddileti	**Dr.Siva Skandha Sanagala**	
Assistant Professor	**Associate Professor and HOD**	**EXTERNAL EXAMINER**
Department of CSE	**Department of CSE**	

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. BK Chinna Maddileti**, Assistant Professor Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. Siva Skandha Sanagala,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Ghanta Devadasu**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary and **Shri. Ch.Abhinav Reddy**, CEO of CMR Group of Institutions, for their continuous care.

Finally, We extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

R.Yeshwanth Raju 20H51A0546
M.Pavan Kumar 20H51A05H6

TABLE OF CONTENTS

List of Figures

ABSTRACT

Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce irrelevance and redundancy of the image data in order to be able to store or transmit data in an efficient form. So image compression can reduce the transmit time over the network and increase the speed of transmission. In Lossless image compression no data loss when the compression Technique is done. In this research, a new lossless compression scheme is presented and named as Huffman Based LZW Lossless Image Compression using Retinex Algorithm which consists of three stages: In the first stage, a Huffman coding is used to compress the image. In the second stage all Huffman code words are concatenated together and then compressed with LZW coding and decoding. In the third stage the Retinex algorithm are used on compressed image for enhance the contrast of image and improve the quality of image. This Proposed Technique is used to increase the compression ratio (CR), Peak signal of Noise Ratio (PSNR), and Mean Square Error (MSE) in the MATLAB Software.

CHAPTER 1
INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

The study aims to identify the specific challenges and objectives of this research or project. It is essential for applications that require the faithful reconstruction of the original images. The problem at hand is to develop an efficient and effective image compression method that combines Huffman coding with LZW to achieve lossless compression while integrating the Retinex algorithm for image enhancement.

1.2 Research Objective

A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information. The foremost task then is to find less correlated representation of the image. Two fundamental components of compression are redundancy and irrelevancy reduction .Redundancies reduction aims at removing duplication from the signal source (image/video). Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System.

1.3 Project Scope and Limitations

Scope

- The scope of the project encompasses the development of lossless.
- Image compression method that integrates Huffman coding , LZW compression.
- It includes research, experimentation, evaluation, and may involve the Development of a prototype implementation.
- The primary focus is on improving compression efficiency while enhancing.
- Image quality, with an eye towards real-world applications.

Limitations

- Large Data Requirements: U-Net models, like many deep learning models, require large amounts of labeled data for training. Gathering and annotating medical images can be a time-consuming and expensive process, and in some cases, there may be limited access to such data due to privacy and legal restrictions.

CHAPTER 2
BACKGROUND WORK

CHAPTER 2
BACKGROUND WORK

2.1 Lossless-JPEG compression method

2.1.1 Introduction

The problem is to develop a lossless image compression method that reduces image file sizes without any loss of image data. [1]The objective is to create a compression technique that can store images more efficiently while enabling exact reconstruction.

2.1.2 Merits

1. Lossless Compression: JPEG-LS is primarily designed for lossless compression, which means it can compress images without any loss of image quality. This makes it suitable for applications where preserving every detail of the image is crucial.

2. Efficient Compression: It often achieves high compression ratios while maintaining lossless quality, making it an effective choice for reducing file sizes of images with minimal degradation.

2.1.3 Demerits

1. Limited Support: JPEG-LS is not as widely supported as other image compression formats, such as JPEG or PNG. This means that it may not be compatible with all software and devices.

2. Complexity: The encoding and decoding processes of JPEG-LS can be more complex compared to some other lossless compression methods. This complexity can result in higher computational requirements and slower processing times.

3. Not Suitable for All Types of Images: JPEG-LS is most effective for images with smooth areas, natural gradients, and fine details. It may not perform as well on images with sharp edges or high-frequency patterns.

Challenges

Lossless JPEG, or JPEG-LS, is a valuable image compression method that maintains image quality while reducing file size. It's well-suited for applications where preserving image fidelity is critical.

2.1.4 Implementation

JPEG-LS was developed with the aim of providing a low-complexity lossless and near-lossless image compression standard that could offer better compression efficiency than lossless JPEG. It was developed because at the time,[2] the Huffman coding-based JPEG lossless standard and other standards were limited in their compression performance.[3] Total decorrelation cannot be achieved by first order entropy of the prediction residuals employed by these inferior standards. JPEGLS, on the other hand, can obtain good decorrelation. Part 1 of this standard was finalized in 1999. [4]Part 2, released in 2003, introduced extensions such as arithmetic coding.

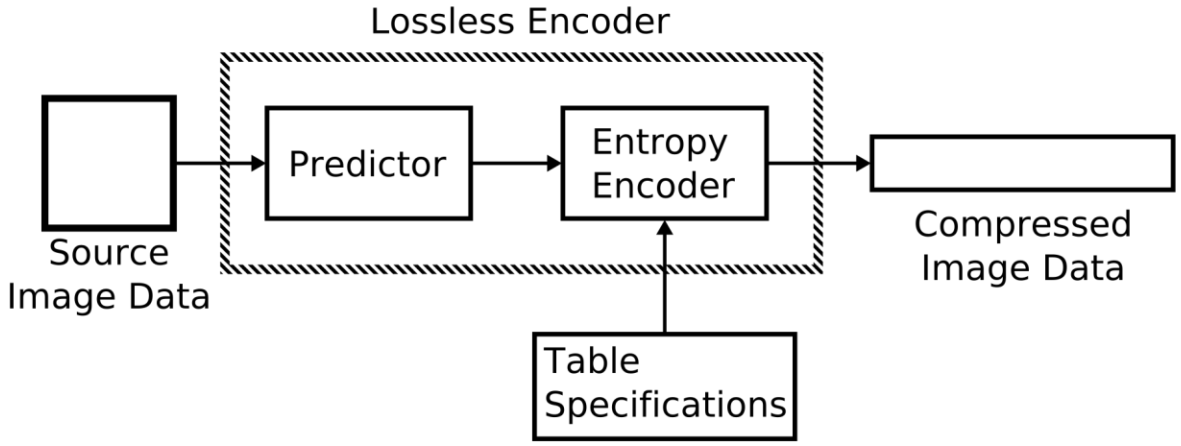


Fig 2.1.3: Lossless-JPEG Compression

2.2 Run-Length Encoding (RLE)

2.2.1 Introduction

Run-Length Encoding (RLE) is a simple form of data compression that is used to reduce the size of files or data by encoding consecutive repeated characters or values as a single value followed by a count. [5]It's often employed in scenarios where there are long sequences of the same data, such as simple graphics, black and white images, or certain types of binary data.

2.2.2 Merits:

1. Simplicity: RLE is easy to understand and implement. It's a basic compression technique that doesn't require complex algorithms or substantial computational resources.

2. Fast Encoding and Decoding: Both encoding and decoding operations are typically very fast since they involve straightforward processes of counting and replacing.

3. Efficient for Certain Data: RLE works particularly well with data that has long sequences of repeated values, making it efficient for certain types of data, such as simple graphics or binary data.

2.2.3 Demerits:

1. Inefficient for Complex Data: RLE is not efficient for data with little or no repetitive patterns. In such cases, the compression might not be significant or could even result in larger files.

2. Limited Compression: RLE doesn't offer the same level of compression as more advanced algorithms like ZIP or GZIP. It's not suitable for highly compressible data.

3. Implementation Compatibility: Compatibility issues can arise when transferring RLE-encoded data between different systems or software that may not support RLE.

Challenges:

1. Choosing the Right Threshold: Determining when to use RLE can be challenging. [6]It's critical to identify situations where long runs of the same value exist, and RLE can be beneficial.

2. Lossless vs. Lossy: RLE is typically used for lossless compression, but in some cases, you may have to decide whether to use lossy compression (where some data is sacrificed for greater compression) or lossless RLE.

3. Implementation Compatibility: Compatibility issues can arise when transferring RLE-encoded data between different systems or software that may not support RLE.

2.2.4 Implementation

Run-Length Encoding is a basic compression algorithm that works by replacing sequences of identical values with a single value followed by a count of the number of times that value occurs.[7] For example, [8] the string "AAABBBCCDAA" could be encoded as "3A3B2C1D2A." It's simple to implement and can be useful in scenarios where there are long runs of identical data.

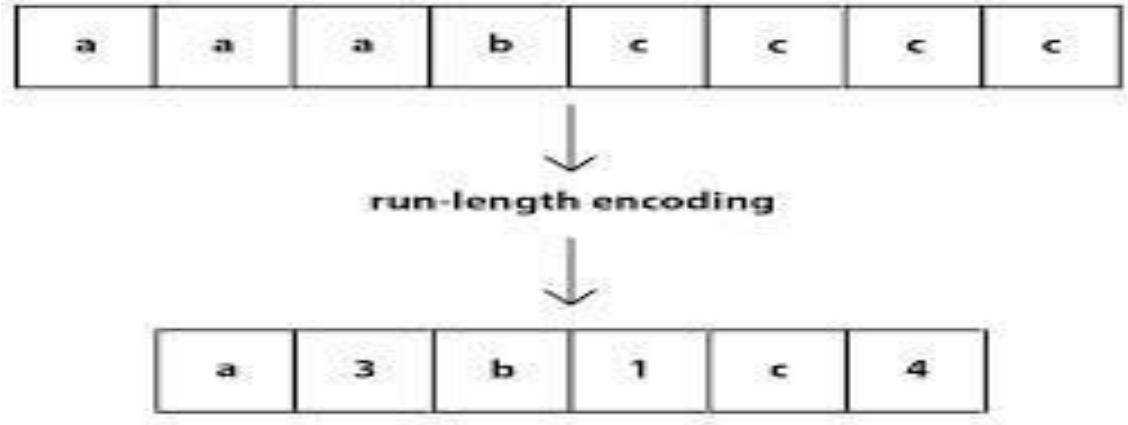


Fig 2.2.3 **Run-Length Encoding (RLE)**

2.3 Wavelet Compression

2.3.1 Introduction

Wavelet compression is a method of data compression that employs wavelet transformations to analyze and reduce the data.[9] It is particularly useful for handling data with varying levels of detail, such as images, audio, and other types of signals. The fundamental idea is to transform the data into a different domain where it can be represented more efficiently.[10] Wavelet compression techniques aim to capture the most important features of the data while discarding redundant or less significant information.

2.3.2 Merits:

1. Multi-Resolution Analysis: Wavelet compression supports multi-resolution analysis, which allows it to represent data at different levels of detail. This makes it suitable for compressing data with both fine and coarse features.

2. Good Image Compression: Wavelet compression is commonly used in image compression, and it often outperforms other methods like JPEG in terms of image quality at high compression ratios.

3. Lossless or Lossy Compression: Wavelet compression can be applied in both lossless and lossy modes, giving flexibility depending on the application's requirements.

4. Progressive Transmission: Wavelet compression allows for progressive transmission, meaning that an incomplete compressed image or signal can be viewed, and more detail is added as the transmission continues.

5. Robustness: Wavelet compression is relatively robust to noise and can maintain the quality of the reconstructed data even in the presence of some data corruption.

Demerits:

1. Complexity: Implementing wavelet compression algorithms can be complex and computationally intensive, especially for real-time applications or large datasets.

2. Artifact Generation: In lossy compression, wavelet-based methods can generate compression artifacts, especially at high compression ratios.

3. Non-Uniform Quality: Depending on the chosen wavelet, the quality of compression can be non-uniform across different types of data.

4. Limited to Specific Data Types: Wavelet compression is not a one-size-fitsall solution and may not perform well on all types of data.

Challenges:

1. Choice of Wavelet: Selecting the right wavelet function is critical for achieving optimal compression results. Different types of data may require different wavelets.

2. Parameter Tuning: Many wavelet compression algorithms have various parameters that need to be tuned to optimize compression quality and speed.

3. Data Dependency: Handling data dependencies, especially in video compression, can be a significant challenge.

2.3.3 Implementation

Wavelet compression is a technique used to compress data by employing the mathematical tool of wavelet transform. [11] It's particularly effective in signal and image compression. The fundamental idea behind wavelet compression involves breaking down the data into different frequency components using wavelets, which are small waves of varying frequency and limited duration.

The process typically involves:

1. **Decomposition:** The data is decomposed into its high and low-frequency components using a wavelet transform. [12] This step creates a multiresolution representation, where high frequencies (fine details) are separated from the low frequencies (coarse details).

2. Quantization and Compression: The coefficients obtained from the wavelet transform are quantized, and the less important coefficients (usually smaller in magnitude) are discarded or set to zero. [13] This reduction in data results in compression.

3. Reconstruction: The compressed data is then reconstructed by applying the inverse wavelet transform. [14] This process allows the approximate original data to be reconstructed, often with some loss of fine details that were discarded during compression.

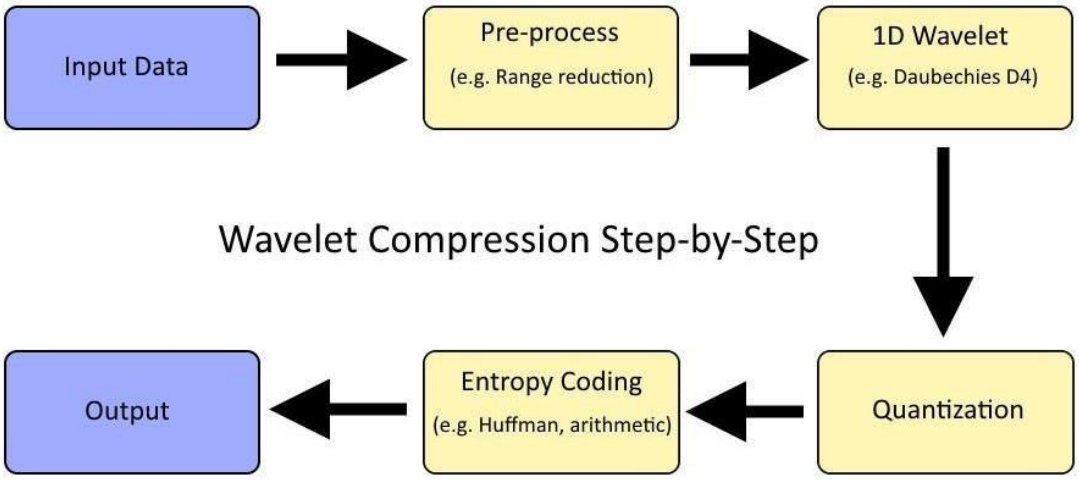


Fig 2.3.3: Wavelet Compression

CHAPTER 3
PROPOSED SYSTEM

CHAPTER 3
PROPOSED SYSTEM

3.1. Objective of Proposed Model

This paper presents an extensive survey on image compression techniques and standards; it is applicable to various fields of image processing. Image compression is very important for efficient transmission and storage of image. On the basis of evaluating and analysing the current image compression techniques this paper presents the Principal Component Analysis approach applied to image compression. There are some techniques that perform this compression in different ways; some are Lossless and keep the same information as the original image using entropy coding some others lossy compression which losses information when compressing the image. A part from these techniques, JPEG, JPEG2000, MPEG, H.26x are the different existing standards in still and moving Image Compression. My aim with this paper is to make a comparison of some of the most used image compression technique on a set of images.

Huge data system applications require storage of large volumes of data set, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. At the same time, the proliferation of communication networks is resulting in massive transfer of data over communication links. Compressing data to be stored or transmitted reduces storage and communication costs. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Here efficient method for decoding the compressed data is proposed. This paper aims toward the implementation of a high speed Huffman decoding system. This proposed model enhances the speed of decoding operation. The model is implemented using VHDL language, simulated on Active HDL 5.1, synthesized, placed and routed and floorplaned using Xilinx tools. The availability of images in different applications are augmented owing to the technological advancement which cannot impacts on several image operation, on availability of sophisticated software tools that manipulate the image and on image management. Despite the technological advances in storage and transmission, the demand placed on the storage capacities and on bandwidth of communication exceeds the availability. Hence, image compression has proved to be a valuable technique as one solution.

3.2. Algorithms Used for Proposed Model

1. Single-scale Retinex

- **Illumination Estimation:**

The process begins with estimating the illumination component of the image. This is achieved by convolving the image with a Gaussian filter, which effectively smooths out the image and captures the overall intensity variations caused by illumination. The Gaussian filter helps in eliminating noise and capturing the global illumination characteristics.

- **Reflectance Estimation:**

Once the illumination is estimated, the next step involves obtaining the reflectance component. This is done by dividing the original image by the estimated illumination. The rationale behind this step is to remove the effects of illumination variations, thereby revealing the intrinsic reflectance properties of the objects in the scene. The reflectance image represents the true colors and textures of the objects, independent of illumination.

- **Color Restoration:**

In the final step, the reflectance image is normalized to maintain consistency in color representation. This normalized reflectance image is then multiplied by the original illumination estimate to obtain the final color-corrected image. By combining the reflectance with the illumination, the algorithm ensures that the colors appear natural and consistent across different lighting conditions.

2. Multi-scale Retinex:

- **Illumination Estimation:**

To address the limitations of the single-scale approach, the multi-scale Retinex algorithm employs multiple Gaussian filters of different scales. Each Gaussian filter captures illumination variations at a different level of granularity, allowing for better handling of images with varying illumination levels and textures. The illumination estimates obtained from different scales provide a more comprehensive representation of the overall illumination distribution.

- **Reflectance Estimation:**

Reflectance components are estimated at each scale by dividing the original image by the respective illumination estimates obtained from different scales. This multi-scale approach ensures that both fine and coarse details are preserved in the reflectance estimation process. By considering illumination variations at multiple scales, the algorithm can effectively separate reflectance from illumination across a wide range of image textures and lighting conditions.

- **Color Restoration:**

The reflectance images obtained from different scales are then combined to produce the final color-corrected image. This fusion process involves blending the reflectance components while preserving important image details. By incorporating information from multiple scales, the algorithm achieves better color constancy and perception, resulting in visually pleasing images with enhanced color fidelity and texture detail.

3. Dynamic Range Compression:

In addition to the basic Retinex algorithm, dynamic range compression techniques can be applied to further enhance the visual quality of the image. These techniques aim to compress the dynamic range of pixel intensities to fit within the displayable range while preserving important image details. By adjusting the contrast and brightness levels of the image, dynamic range compression can improve visibility and enhance overall image appearance.

Overall, the Retinex algorithm and its variations offer powerful tools for image enhancement and color constancy. By separating illumination and reflectance components, these algorithms can significantly improve the visual quality of images, making them more vibrant, natural, and appealing to viewers. While the primary focus of Retinex is on image enhancement rather than compression, its techniques can complement compression algorithms by preserving important image details and enhancing overall perceptual quality.

3.3. Designing

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake

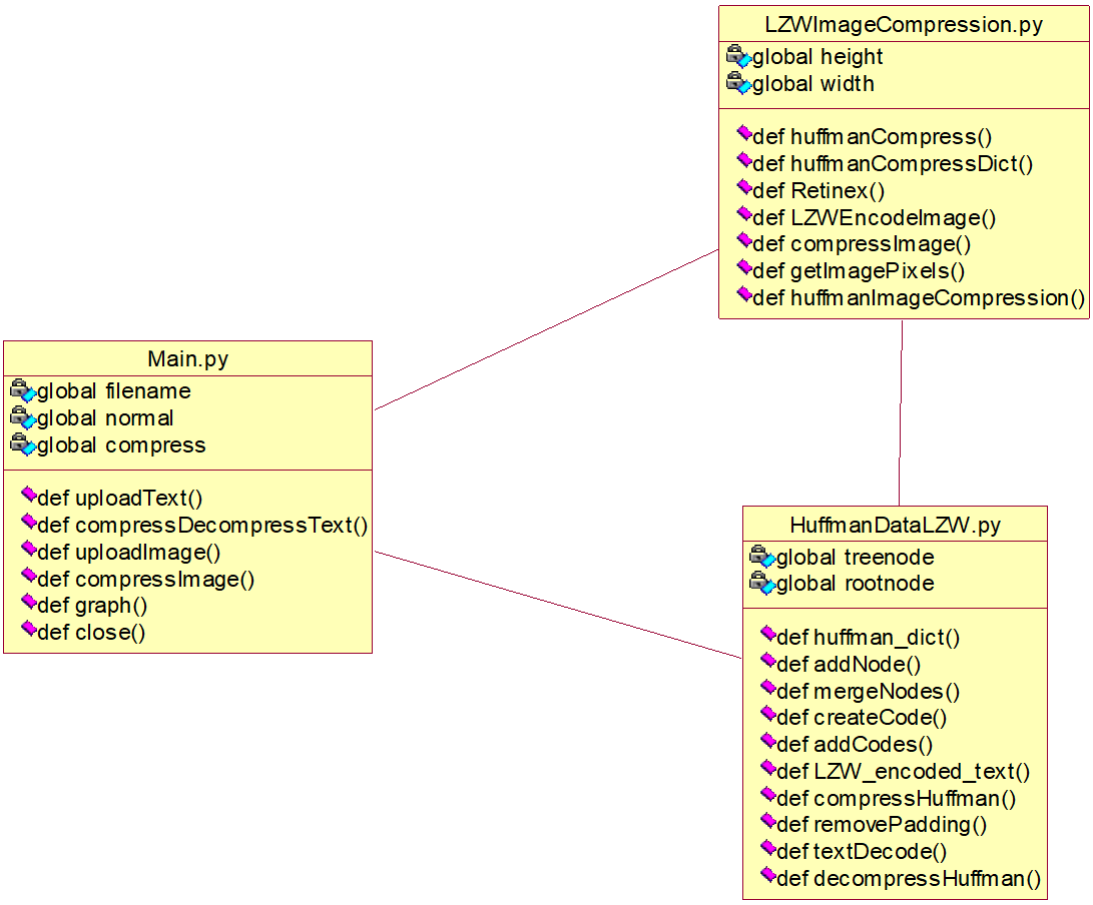


Fig – 3.3 Class Diagram

3.3.1 Architecture

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as we.

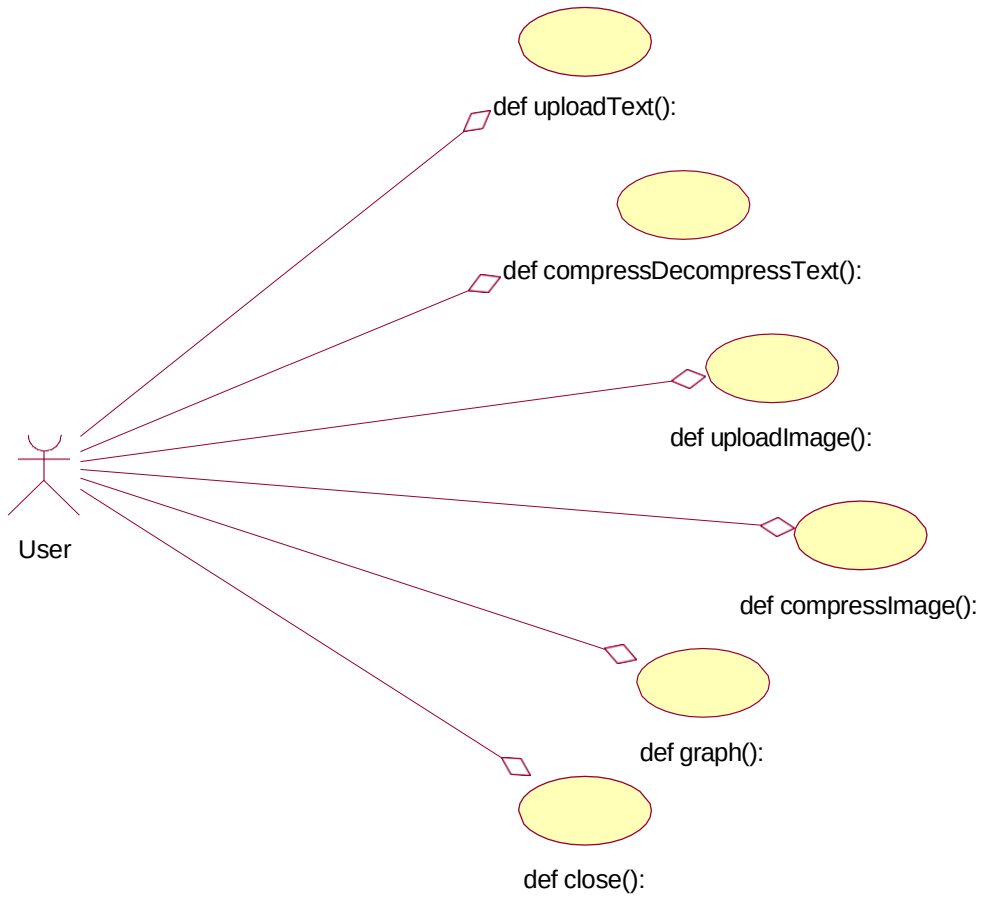


Fig – 3.3.1 Architecture

3.3.2. Sequence Diagram

A **sequence diagram** is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams**, **event scenarios**, and timing diagrams.

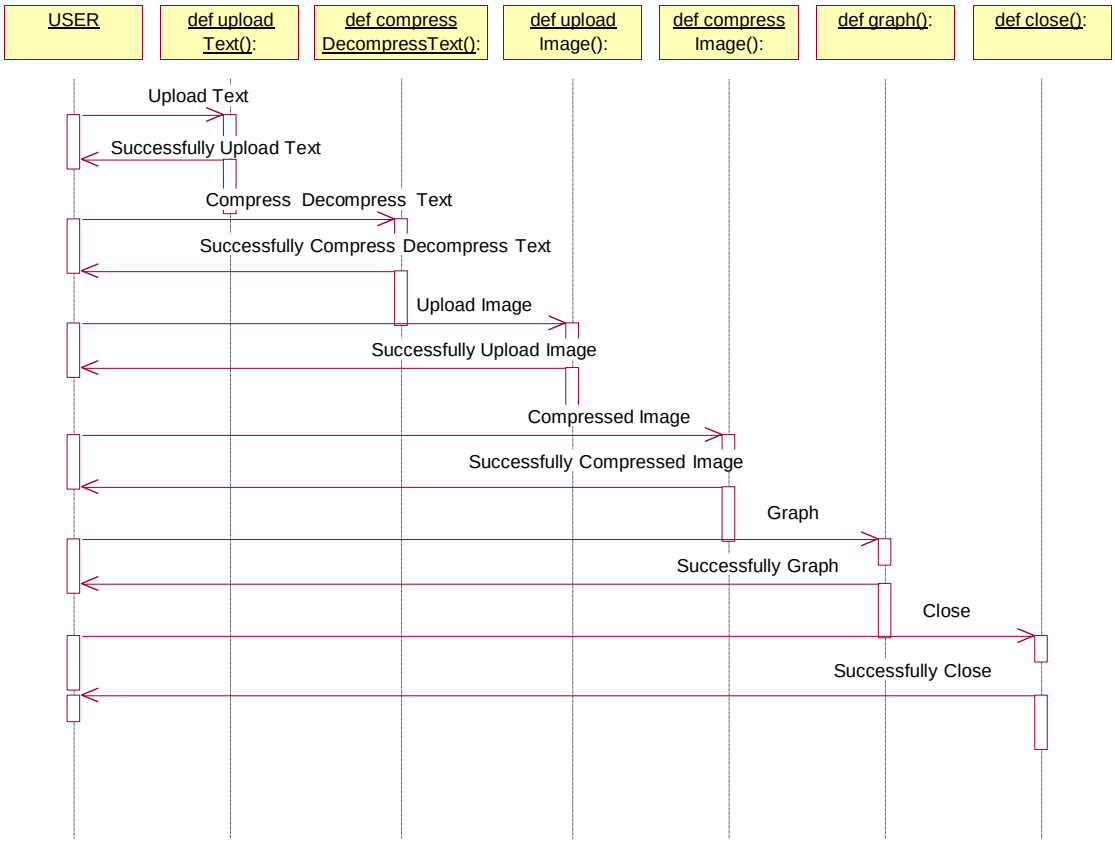


Fig – 3.3.2 Sequence diagram

3.3.3 COLLABORATION DIAGRAM:

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

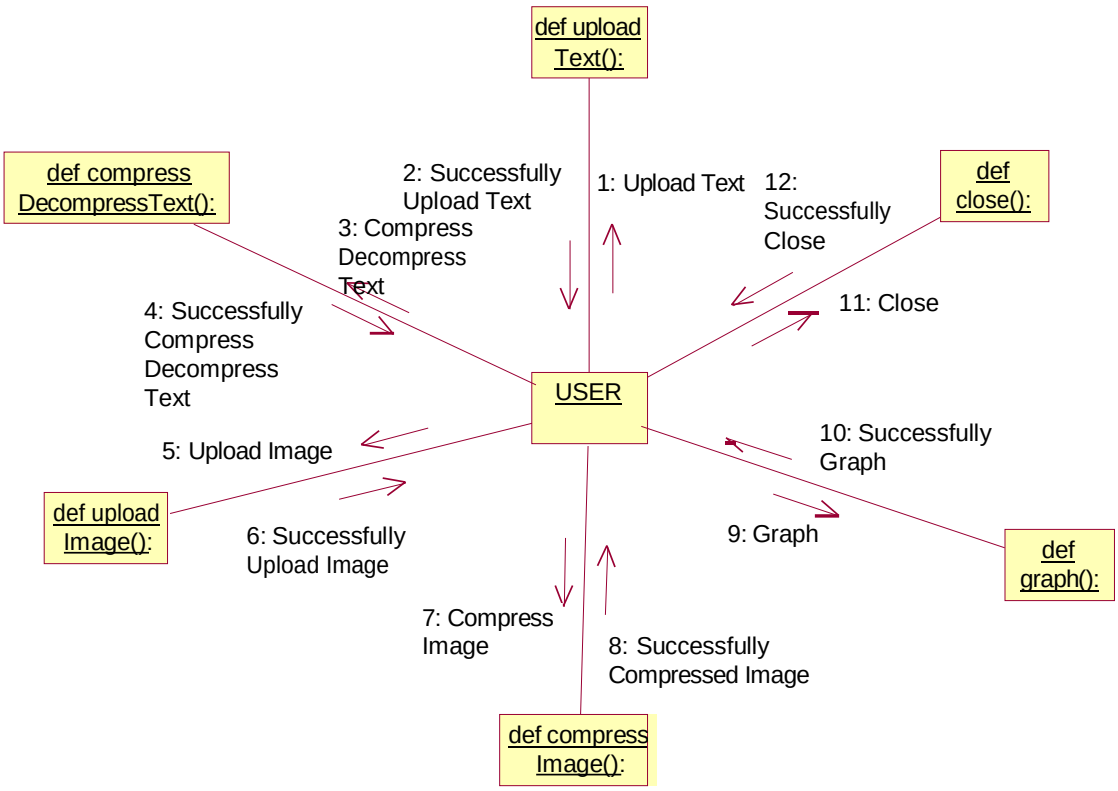


Fig – 3.3.3 Collaboration Diagram

3.4. Stepwise Implementation

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to use the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

Module Testing

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

Integration Testing

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

Acceptance Testing

When the user find no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

5.2 Sample Code:

Main.py

```
from matplotlib import pyplot as plt
from tkinter import *
from tkinter import simpledialog
from tkinter.filedialog import askopenfilename
import tkinter
from tkinter import filedialog
from HuffmanDataLZW import HuffmanLZWCoding
import os
import LZWImageCompression
from LZWImageCompression import *
import cv2
root = tkinter.Tk()
root.title("Huffman Based LZW Lossless Image Compression Using Retinex Algorithm")
root.geometry("1000x500")
global filename
global normal
global compress
def uploadText():
    global filename
    filename = askopenfilename(initialdir = "TextFiles")
```

```
 trainpath.config(text=filename+' loaded')

def compressDecompressText():
    global normal
    global compress
    h = HuffmanLZWCoding(filename)
    output_path = h.compressHuffman()
    h.decompressHuffman(output_path)
    compress = os.stat(output_path).st_size
    trainpath.config(text=filename+' original size : '+str(normal)+' & its compress size : '+str(compress))

def uploadImage():
    global filename
    filename = askopenfilename(initialdir = "ImageFiles")
    trainpath.config(text=filename+' loaded')

def compressImage():
    global normal
    global compress
    LZWImageCompression.huffmanImageCompression(filename)
    normal = os.stat(filename).st_size
    compress = os.stat('compress/Compress.jpg').st_size
    original_image =  cv2.imread(filename)
    original_image = cv2.resize(original_image,(500,500))
    compress_image = cv2.imread('compress/Compress.jpg')
    compress_image = cv2.resize(compress_image,(500,500))
```

```
retinex_compress_image = cv2.imread('compress/retinex_Compress.jpg')
retinex_compress_image = cv2.resize(retinex_compress_image,(500,500))
    cv2.imshow("Original Image & its Size : "+str(normal),original_image);
    cv2.imshow("Compress Image & its Size : "+str(compress),compress_image);
    cv2.imshow("Retinex Compress Image &its Size :
"+str(compress), retinex_compress_image);
    cv2.waitKey();
def graph():
    height = [normal,compress]
    bars = ('Normal File Size','Compress File Size')
    y_pos = numpy.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()
def close():
    root.destroy()
ttt = Label(root)
ttt.grid(row=0)
ttt1 = Label(root)
ttt1.grid(row=3)
font = ('times', 14, 'bold')
uploadTextButton = Button(root, text="Upload Text File", command=uploadText)
uploadTextButton.grid(row=0)
uploadTextButton.config(font=font)
```

```
trainpath = Label(root)

trainpath.grid(row=6)

trainpath.config(font=font)

textButton = Button(root, text="Compress & Decompress Text File",
command=compressDecompressText)

textButton.grid(row=9)

textButton.config(font=font)

tt = Label(root)

tt.grid(row=12)

uploadimageButton = Button(root, text="Upload Image", command=uploadImage)

uploadimageButton.grid(row=15)

uploadimageButton.config(font=font)

tt1 = Label(root)
tt1.grid(row=18)
imageButton = Button(root, text="Compress Image & Apply Retinex",
command=compressImage)
imageButton.grid(row=21)
imageButton.config(font=font)
tt2 = Label(root)
tt2.grid(row=24)
graphButton = Button(root, text="Comparison Graph", command=graph)
graphButton.grid(row=27)
graphButton.config(font=font)


tt3 = Label(root)

tt3.grid(row=30)

closeButton = Button(root, text="Close Application", command=close)
```

```
closeButton.grid(row=33)
closeButton.config(font=font)
root.mainloop()
```

LZWCompression.py

```
import io
from PIL import Image
import numpy
import matplotlib.pyplot as plt
import cv2
from collections import defaultdict
import os
img_pixels = defaultdict(list)
global height
global width
def huffmanCompress(compressed_image):
    # building dictionary here.
    dictionary_size = 256
    dictionary_arr = dict((i, chr(i)) for i in range(dictionary_size))
    results = io.StringIO()
    pixel = chr(compressed_image.pop(0))
    results.write(pixel)
for m in compressed_image:
        if m in dictionary_arr:
```

```
            entry_pixel = dictionary_arr[m]
        elif m == dictionary_size:
            entry_pixel = pixel + pixel[0]
        else:
            raise ValueError('Bad compression m: %s' % m)
        results.write(entry_pixel)
        # Adding pixel to the dictionary.
        dictionary_arr[dictionary_size] = pixel + entry_pixel[0]
        dictionary_size += 1
        pixel = entry_pixel
    return results.getvalue()
def huffmanCompressDict(uncompressed_image):
    dictionary_size = 256
    dictionary_arr = dict((chr(i), i) for i in range(dictionary_size))
    pixel = ""
    results = []
    for chars in uncompressed_image:
        pixel_chars = pixel + chars
        if pixel_chars in dictionary_arr:
            pixel = pixel_chars

        else:
            results.append(dictionary_arr[pixel])
            dictionary_arr[pixel_chars] = dictionary_size
            dictionary_size += 1
            pixel = chars
```

```
new_dictionary = list(dictionary_arr.items())
    if pixel:
        results.append(dictionary_arr[pixel])
    return results
def Retinex(image, gamma_value=0.5):
 inv_Gamma = 1.0 / gamma_value
    lookup_colour_improve_table = numpy.array([((i / 255.0) ** inv_Gamma) * 255 for i in numpy.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, lookup_colour_improve_table)
def LZWEncodeImage(codes,image_name,retinex_image):
    pixel_list = []
    for code in codes:
        pixel_list.append(ord(code))
    arr = numpy.asarray(pixel_list)
    img = arr.reshape(height,width)
    print(img.shape)
    orig = numpy.zeros((height,width,3))
    for i in range(width):
        for j in range(height):
            value = int(img[j,i])
            values = img_pixels.get(str(i)+","+str(j))
            if values is not None:
                values = values[0]
                r = values[0]
```

```
g = values[1]
            b = values[2]
            orig[j][i][0] = b
            orig[j][i][1] = g
            orig[j][i][2] = r
    cv2.imwrite(image_name, orig,[cv2.IMWRITE_JPEG_QUALITY, 35])
    img = cv2.imread(image_name)
    img = Retinex(img)
    cv2.imwrite(retinex_image, img,[cv2.IMWRITE_JPEG_QUALITY, 45])
def compressImage(uncompress_image):
    compress_image = huffmanCompressDict(uncompress_image)
    arr = numpy.asarray(compress_image)
    arr1 = numpy.zeros(10000)
    for i in range(len(arr1)):
        arr1[i] = arr[i]
    compress_image = huffmanCompress(compress_image)
LZWEncodeImage(compress_image,'compress/Compress.jpg','compress/retinex_Compress.jp
g')
def getImagePixels(image_path):
    global height
    global width
    input_image = Image.open(image_path)
    pixels = input_image.load()
    widths, heights = input_image.size

 width = widths
```

```
        height = heights
    pixels_arr = []
    for i in range(widths):
        for j in range(heights):
            color_pixel = pixels[i, j]
            gray_value = int(round(sum(color_pixel) / float(len(color_pixel))))
            img_pixels[str(i)+","+str(j)].append(color_pixel)
pixels_arr.append(gray_value)
     return pixels_arr
def huffmanImageCompression(input_image):
    img_pixels.clear()
    pixel_values = []
    for p in getImagePixels(input_image):
        pixel_values.append(p)
    pixelString = []
    String_pixel = ""
    for f in pixel_values:
        pixelString.append(chr(f))
    for ps in pixelString:
        String_pixel +=str(ps)
    compressImage(String_pixel)
#huffmanImageCompression('images.jpg')
#original_size = os.stat('images.jpg').st_size
```

```
#compress_size = os.stat('Compress.jpg').st_size

#original_image = cv2.imread('images.jpg')

#original_image = cv2.resize(original_image,(500,500))

#compress_image = cv2.imread('Compress.jpg')

#compress_image = cv2.resize(compress_image,(500,500))

#cv2.imshow("Original Image & its Size : "+str(original_size),original_image);

#cv2.imshow("Compress Image & its Size : "+str(compress_size),compress_image);

#cv2.waitKey();
```

HuffmanDataLZW.py

```
import heapq

import os

from functools import total_ordering

@total_ordering

class TreeNode:

    def__init (self, chars, frequency):

        self.chars = chars

        self.frequency = frequency

        self.leftNode = None
        self.rightNode = None

    def__lt__(self, othernode):

        return self.frequency < othernode.frequency

    def__eq_(self, othernode):

       if(othernode == None):

            return False

        if(not isinstance(othernode, TreeNode)):
```

```
    return False
            return self.frequency == other.frequency
class HuffmanLZWCoding:
def___init__(self, filepath):
        self.filepath = filepath
        self.heap_arr = []
        self.codes_arr = {}
        self.reverse_mapping_arr = {}
# functions for compression:
def huffman_dict(self, text_char):
        frequency_arr = {}
        for characters in text_char:
                if not characters in frequency_arr:
                        frequency_arr[characters] = 0
                frequency_arr[characters] += 1
        return frequency_arr
def addNode(self, frequency_arr):
        for keys in frequency_arr:
                treenode = TreeNode(keys, frequency_arr[keys])
                heapq.heappush(self.heap_arr, treenode)
def mergeNodes(self):
        while(len(self.heap_arr)>1):
                node1 = heapq.heappop(self.heap_arr)
```

```
                node2 = heapq.heappop(self.heap_arr)
                mergednode = TreeNode(None, node1.frequency + node2.frequency)
                mergednode.leftnode = node1
                mergednode.rightnode = node2
                heapq.heappush(self.heap_arr, mergednode)
def createCode(self, rootnode, currentcode):
        if(rootnode == None):
                return
        if(rootnode.chars != None):
                self.codes_arr[rootnode.chars] = currentcode
                self.reverse_mapping_arr[currentcode] = rootnode.chars
                return
        self.createCode(rootnode.leftnode, currentcode + "0")
        self.createCode(rootnode.rightnode, currentcode + "1")
      def addCodes(self):
        rootnode = heapq.heappop(self.heap_arr)
        currentcode = ""
        self.createCode(rootnode, currentcode)
     def LZW_encoded_text(self, textdata):
        encodedtext = ""
        for characters in textdata:
encodedtext += self.codes_arr[characters]
        return encodedtext
def LZWpad_encoded_text(self, encodedtext)

extrapadding = 8 - len(encodedtext) % 8
```

```
for i in range(extrapadding):
                encodedtext += "0"
        paddedinfo = "{0:08b}".format(extrapadding)
        encodedtext = paddedinfo + encodedtext
        return encodedtext
def LZW_byte_array(self, paddedencoded_text):
        if(len(paddedencoded_text) % 8 != 0):
                print("Encoding textdata not properly padded")
                exit(0)
        b_arr = bytearray()
for i in range(0, len(paddedencoded_text), 8)
byte = paddedencoded_text[i:i+8]
b_arr.append(int(byte, 2))
        return b_arr
def compressHuffman(self):
        input_file, file_extension = os.path.splitext(self.filepath)
        outputpath = "compress/compress.bin"
        with open(self.filepath, 'r+') as file, open(outputpath, 'wb') as output:
                textdata = file.read()
textdata = textdata.rstrip()
                frequency_arr = self.huffman_dict(textdata)
                self.addNode(frequency_arr)
                self.mergeNodes()
```

```
                self.addCodes()
                encodedText = self.LZW_encoded_text(textdata)
                paddedencoded_text = self.LZWpad_encoded_text(encodedText)
                b_arr = self.LZW_byte_array(paddedencoded_text)
                output.write(bytes(b_arr))
        print("Compression Completed")
        return outputpath
""" functions for decompression: """
def removePadding(self, paddedencoded_text):
        paddedInfo = paddedencoded_text[:8]
        extraPadding = int(paddedInfo, 2)
        paddedencoded_text = paddedencoded_text[8:]
        encodedText = paddedencoded_text[:-1*extraPadding]
        return encodedText
def textDecode(self, encodedText):
        currentCode = ""
        decodedText = ""
        for bits in encodedText:
                currentCode += bits
                if(currentCode in self.reverse_mapping_arr):
characters = self.reverse_mapping_arr[currentCode]
                        decodedText += characters
                        currentCode = ""
        return decodedText
        def decompressHuffman(self, inputPath):
```

```
        inputfile, file_extension = os.path.splitext(self.filepath)
        outputPath = "compress/decompress.txt"
        print(outputPath)
        with open(inputPath, 'rb') as file, open(outputPath, 'w') as output:
            bitStrings = ""
            bytes = file.read(1)
            while(len(bytes) > 0):
                bytes = ord(bytes)
                bit = bin(bytes)[2:].rjust(8, '0')
                bitStrings += bit
                bytes = file.read(1)
            encodedText = self.removePadding(bitStrings)
decompressedText = self.textDecode(encodedText)
              output.write(decompressedText)
print("Decompression Process Completed")
        return outputPath
```

CHAPTER 4
RESULTS AND DISCUSSION

CHAPTER 4
RESULTS AND DISCUSSION

- Using Huffman coding, LZW (Lempel-Ziv-Welch) compression, and the Retinex algorithm for image processing can provide effective lossless image compression with enhanced image quality.
- The Retinex algorithm is applied for image enhancement. Retinex is a color and brightness perception model that aims to improve the visual appearance of images by enhancing details, reducing shadows, and correcting for uneven illumination.
- Finally the combination of Huffman and LZW compression, along with the Retinex enhancement, ensures that you achieve high compression efficiency without compromising image quality.

RESULT

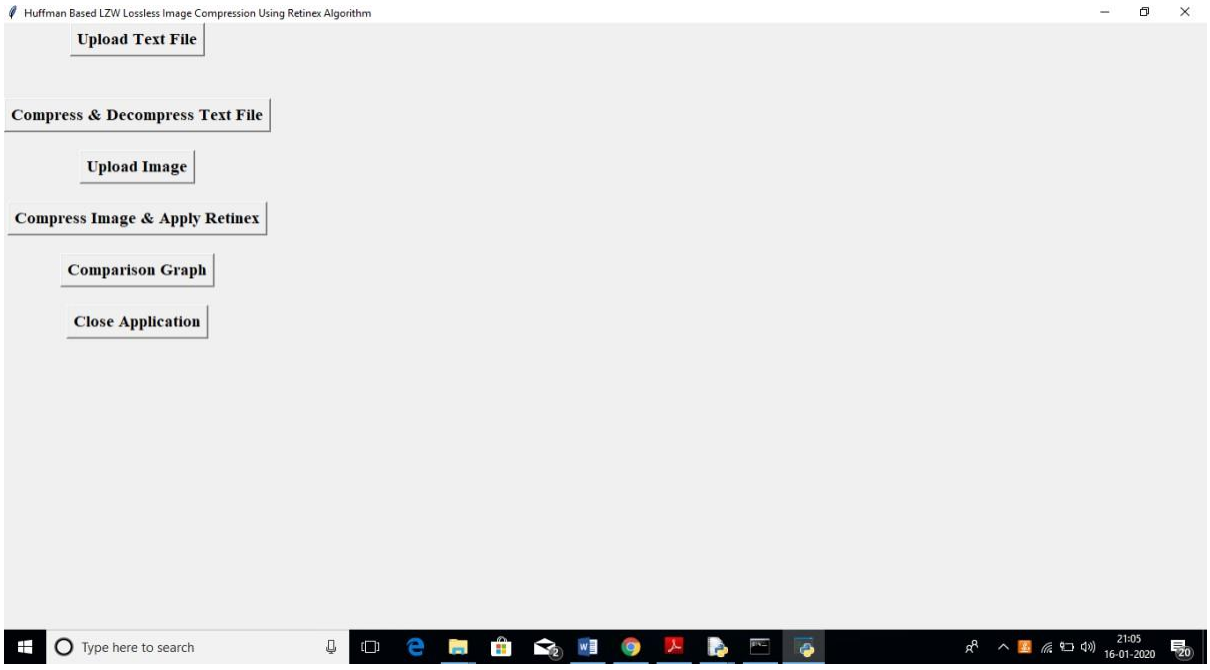


Fig – 4.1 Upload Text File

- In above screen if you want to compress text file then click on 'Upload Text File' button and upload text files.

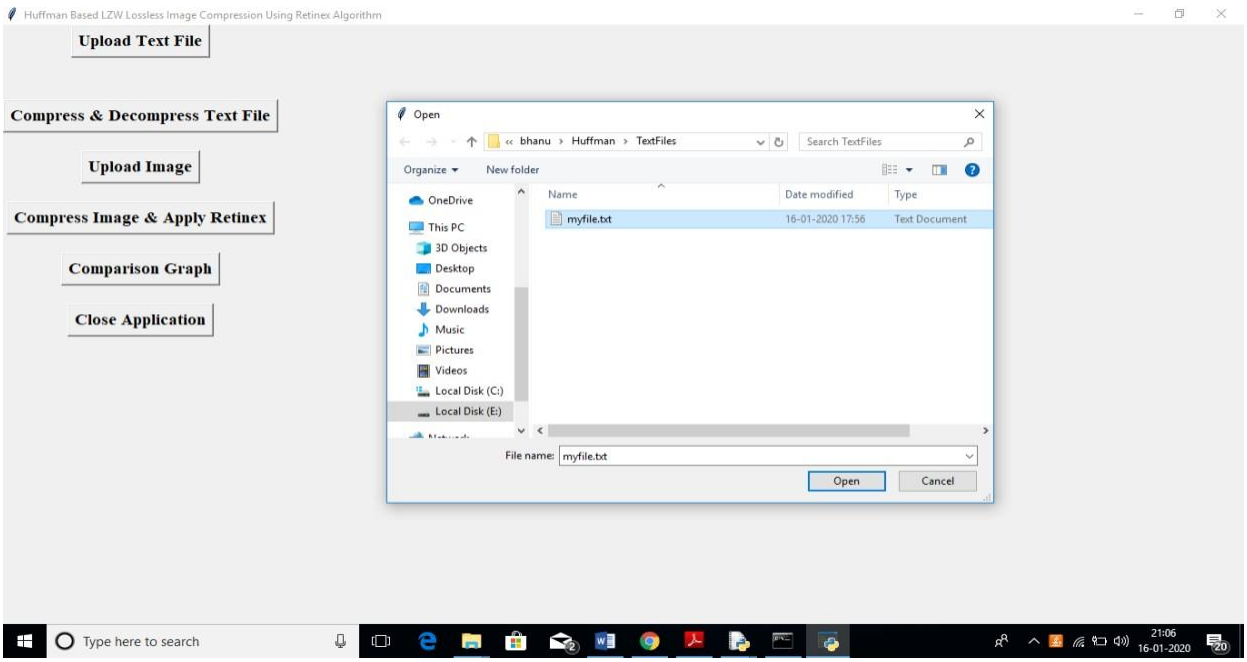


Fig – 4.2 Uploading Text File

- In above screen I am uploading 'myfile.txt' file and after upload will get below screen

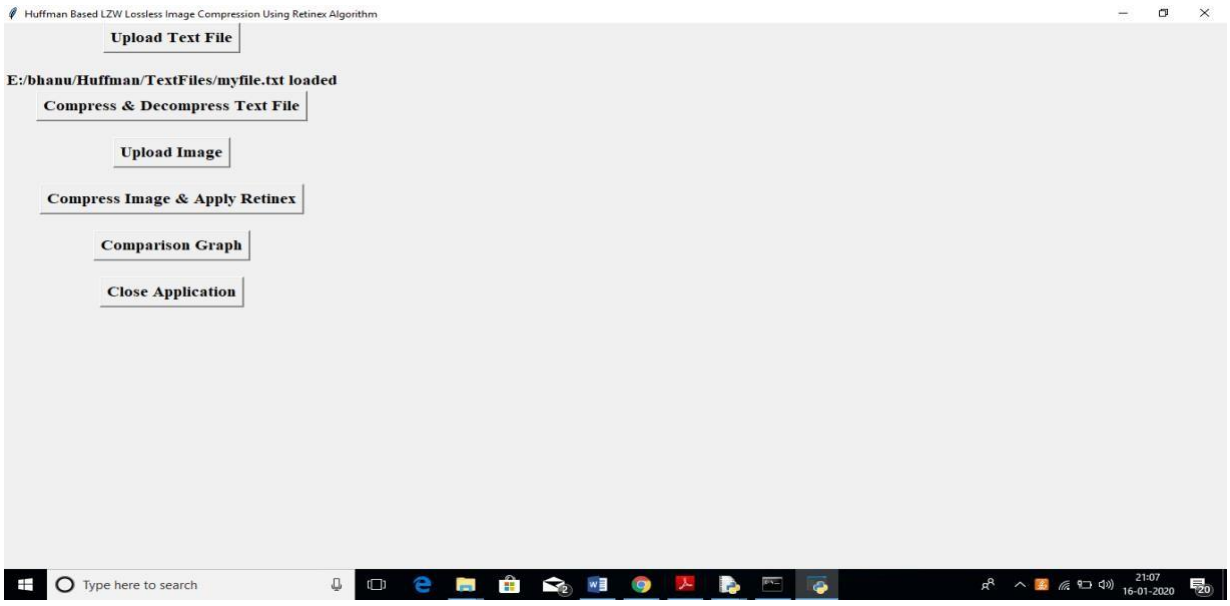


Fig – 4.3 Compress and Decompress Text File

- Now click on 'Compress & Decompress Text File' button to get compress and decompress text file.

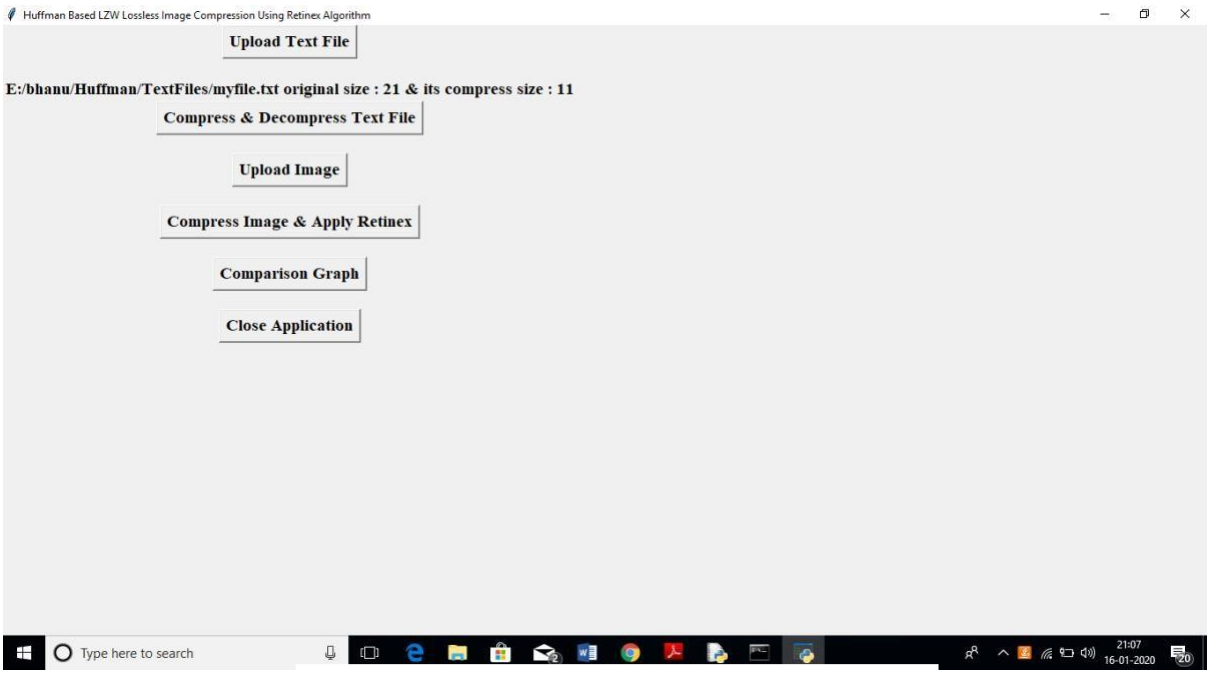


Fig – 4.4 Text File Size Is Compressed

- In above screen we can see message as original file size is 21 Bytes and after compress file size is 11 Bytes. Now you can see original file size inside 'TextFiles' folder.

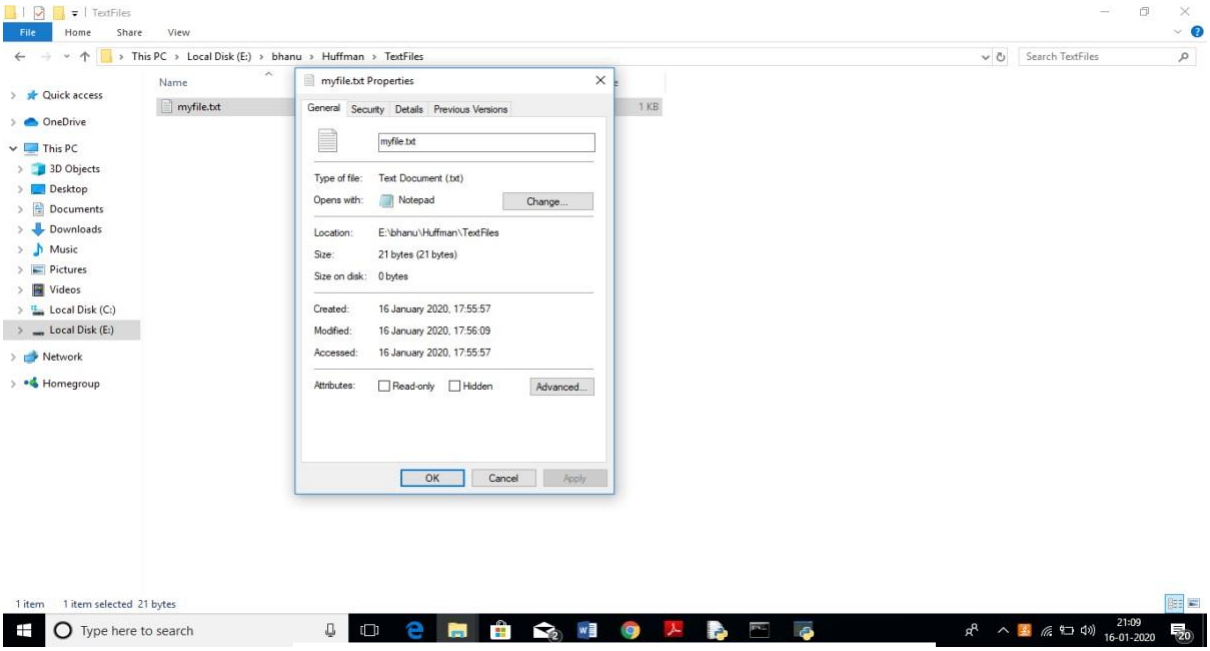


Fig – 4.5 Compressed Text File

- In above screen we can see original file size in directory also, now see compress file size inside 'compress' folder.

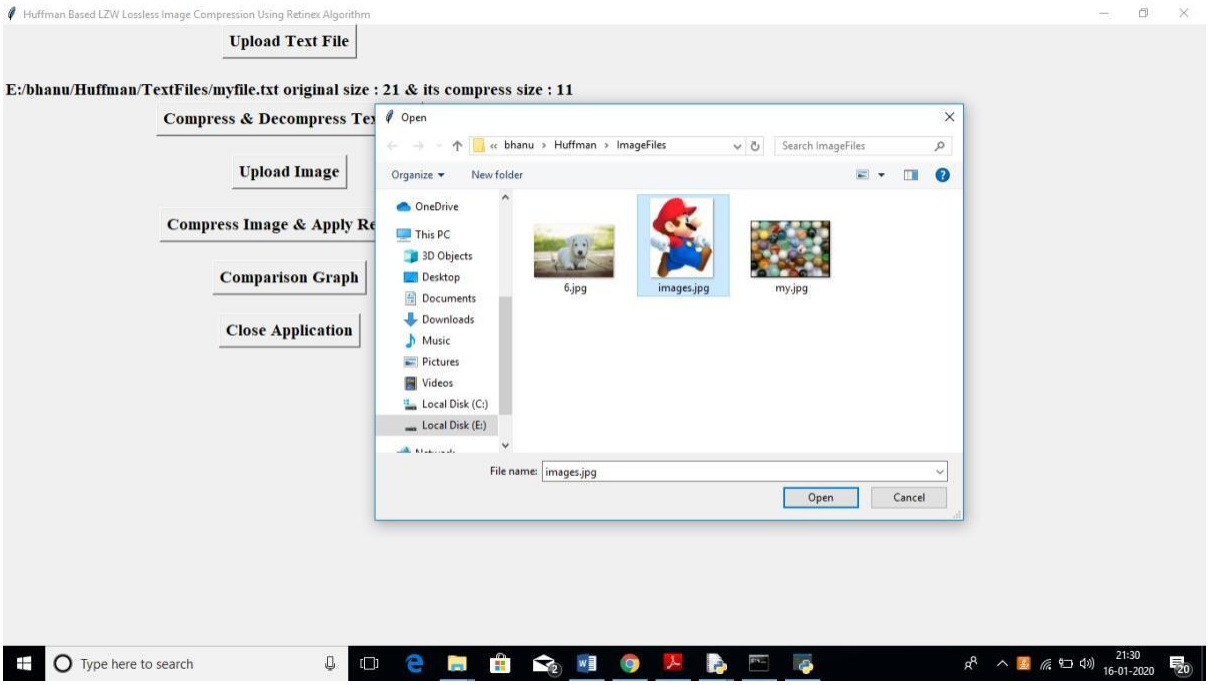


Fig – 4.6 Upload Image

- In above screen uploading 'images.jpg' file and after uploading image will get below screen

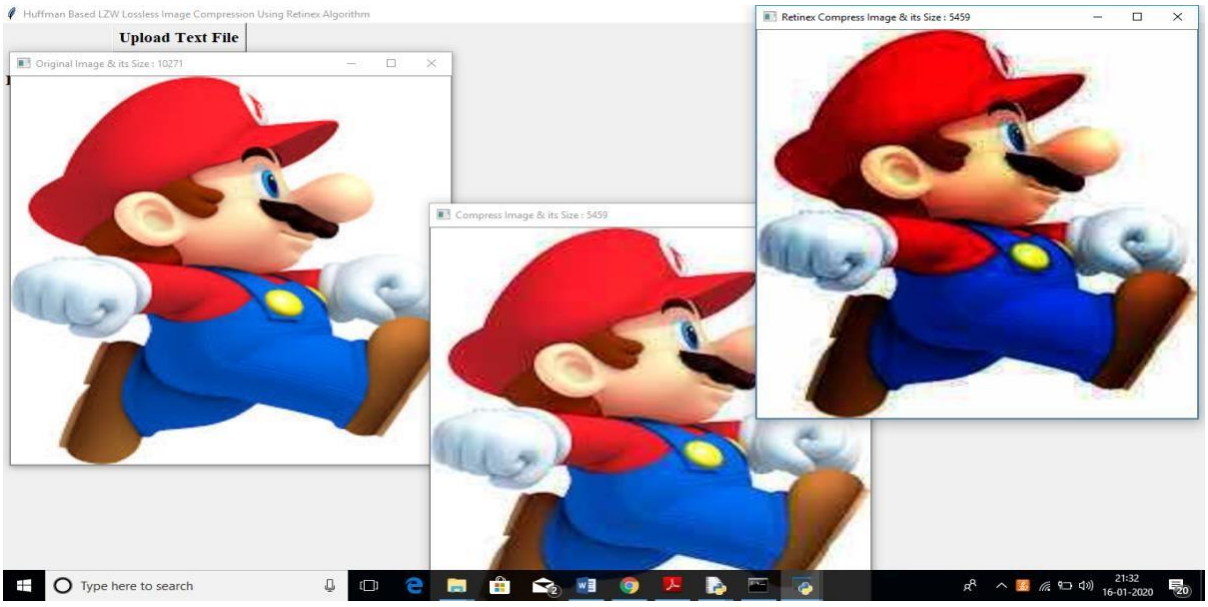


Fig – 4.7 Compressed Image And Retinex Compress Image

- In above screen first image is the original image and you can see its name and size in title bar of the image, first image is the original image and its size is 10271 bytes and second image is the compress image and its size also u can see in title bar and compress image size is 5459 bytes and third image is Retinex algorithm applied compress image and its size also same and you can see after applying Retinex algorithm third image is looking little clean and bright. Now click on 'Comparison Graph' button to get below graph.

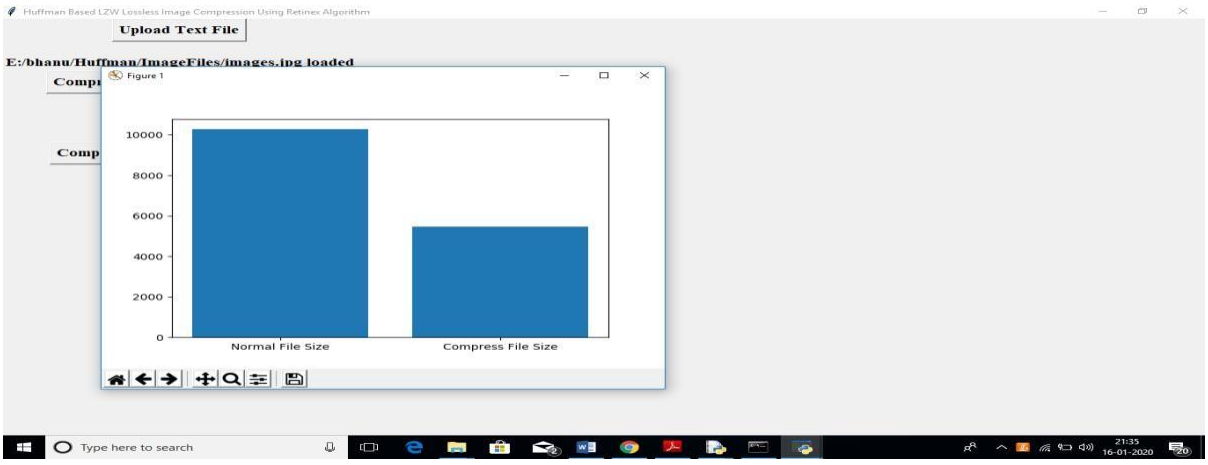


Fig – 4.8 Comparison Graph

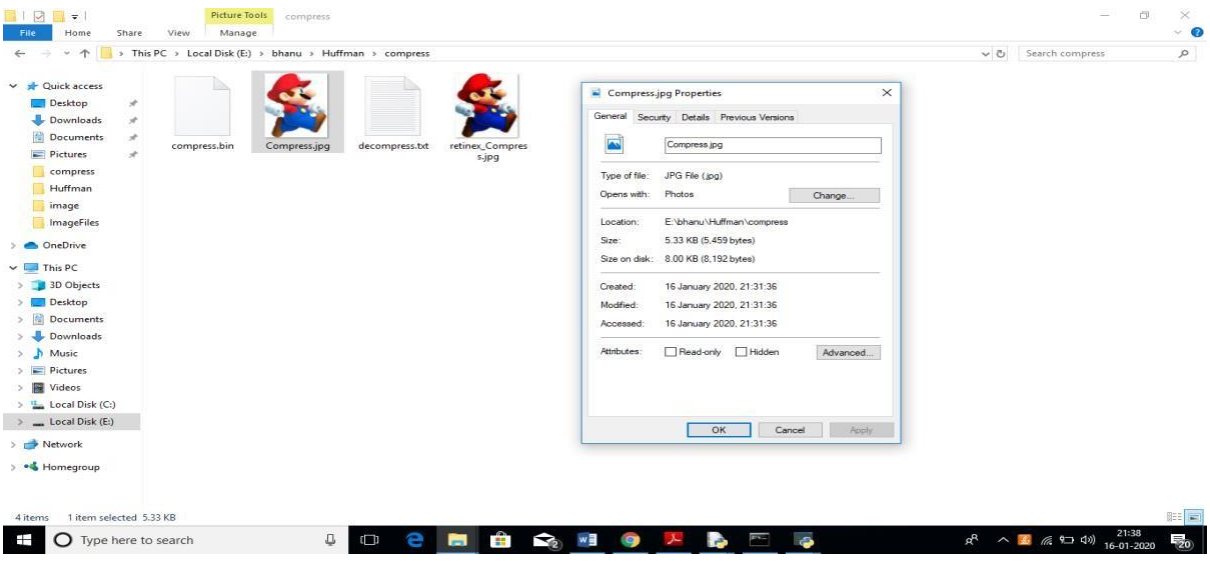


Fig – 4.9 Comparing The Image Size

- In above screen compress.jpg file size is 5.33 KB and we can see Retinex applied image also along with compress image.

CHAPTER 4
CONCLUSION

CHAPTER 4
CONCLUSION

Compression is a topic of much importance and many applications. This thesis presents the lossless image compression on different images. Different algorithms have been evaluated in terms of the amount of compression they provide, algorithm efficiency, and susceptibility to error. While algorithm efficiency and susceptibility to error are relatively independent of the characteristics of the source ensemble, the amount of compression achieved depends upon the characteristics of the source to a great extent. It is concluded that the higher data redundancy helps to achieve more compression. Reproduced image and the original image are equal in quality by using Retinex Algorithm, as it enhances the image contrast using MSR.

FUTURE SCOPE

The future scope of your project on lossless image compression is multifaceted, offering several avenues for exploration and development. First, there's room for enhancing compression algorithms, either through refining existing techniques or devising entirely new approaches, with a focus on achieving better compression ratios while maintaining lossless quality. Second, the exploration of adaptive compression techniques, which dynamically adjust compression methods based on image characteristics, holds promise for optimizing compression for specific types of images or scenes. Additionally, the development of real-time compression solutions tailored for applications like medical imaging or video streaming could significantly impact various industries Lastly, cross-domain collaboration with researchers from related fields could provide valuable insights for advancing lossless image compression technology

REFERENCES

REFERENCES

1. Mamta Sharmap "Compression Using Huffman Coding*" IJCSNS International Journalof Computer Science and Network Security, VOL.10 No.5, May 2010.*

2. Sindhu M, Rajkamal R"Images and Its Compression Techniques A Review"*International Journal of Recent Trends in Engineering, Vol 2, No. 4, November 2009.*

3. C.Saravanan,R. Ponalagusamy "Lossless Grey-scaleImag Compression using Source Symbols Reduction and Huffman Coding".

4. Mayur Nandihalli,Vishwanath Baligar "lossless gray scale images using dynamic array for predction and applied to higher bitplan" *International Journal of Scientific and Research Publications, Volume 3, Issue 1, January 2013.*

5. R.S.Aarthi, D. Muralidharan, P. Swaminathan "DOUBLE COMPRESSION OF TEST DATA USING HUFFMAN CODE *" Journal of Theoretical and Applied Information Technology15 May 2012.*

6. Alarabeyyat, S. Al-Hashemi1, T. Khdour1, M. Hjouj Btoush1, S. Bani-Ahmad1, R. Al- Hashemi "Lossless Image Compression Technique Using Combination Methods "*Journal of Software Engineering and Applications, 2012.*

7. Md. Rubaiyat Hasan "Data Compression using Huffman based LZW Encoding Technique" *International Journal of Scientific & Engineering Research, Volume 2, Issue11, November-2011.*

8. Mr.D.V.Patil, Mr.S.G.SutarMrs.A.N.Mulla.. "Automated Improvement of Image Clarity through Retinex Method for Enhanced Visualization" published in the International Journal of Scientific and Research Publications, Volume 3, Issue 6, June 2013.

9. JiangXing-fanam) , "Enhancing Color Images with an Advanced Multi-Scale Retinex Algorithm" presented at the International Symposium on Photoelectronic Detection andImaging in 2007 by Tao Chun-kan.

10. Examining the correlation between image improvement and image size reduction within the ramework of multi-scale Retinex. (Rahman, Z., Jobson, D.J., Woodall, G.A. 2011. Journal of Visual Communication and Image Representation, Volume 22, Pages 237– 250, ScienceDirect).

11. C. Saravanan and M. Surender's study titled "Improving Huffman Coding Efficiency with Lempel Zig Coding for Image Compression" published in the International Journalof Soft Computing and Engineering (IJSCE) in January 2013.

12. Asadollah Shabhahrami , MobinSabbaghiRostami , RaminBahrampour, Mostafa AyoubiMobarhan, "Assessment of Huffman and Arithmetic Algorithms for MultimediaCompression Norms".

13. Monika and Pragati Kapoor's study on "Implementing Image Compression through a Hybrid Huffman-based LZW Approach" in 2007.

14. A. Maliah, S. K. Shabbir and T. Subhashini's paper titled "Enhancing Image Compression and Quality through AnSpiht Algorithm with Huffman Encoder, Incorporating Retinex Algorithm" published in the June 2012 issue of the International Journal of Scientific & Technology Research, Volume 1, Issue 5.

Github Link:

https://github.com/RYESHWANTHRAJU

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Huffman Based LZW Lossless Image Compression Using Retinex Algorithm

[#1] R. Yeshwanth Raju, [#2] K. Nitin Reddy, [#3] M. Pavan Kumar, [#4] BK. Chinna Maddileti

[1,2,3] UG Student, Department of CSE, CMR College of Engineering & Technology, Hyderabad, Telangana

[4] Assistant Professor, Department of CSE, CMR College of Engineering & Technology, Hyderabad, Telangana

Abstract—Image Data compression finds application in picture compression, where it minimizes the number of bits required to represent an image compared to its original format. In order to store and transmit data efficiently, picture compression aims to decrease irrelevant and redundant image data. Therefore, picture compression may speed up transmission and decrease network latency.In employing a lossless compression technique, data remains intact throughout the compression process without any loss. This study introduces a novel three-stage lossless compression approach called Huffman Inspired LZW Lossless Compression of Images utilising Retinex Algorithm. The first step is to compress the picture using Huffman coding. The second step is to compress the data using LZW and then decode it after joining all of the Huffman code words. Stage three involves applying the Retinex algorithm to the compressed picture in order to boost contrast and overall image quality. Using this suggested method, you can improve MATLAB's compression rate (CR), or mean square error (MSE).

Keywords— Lossless LZW Compression, Huffman Coding, Compression Ratio, Retinex, Mean Square Error, LZW Coding and Decoding.

I. INTRODUCTION

By encoding the original picture with few bits, image compression applies data compression to images. The goal of picture compression is to make data storage and transmission more efficient by reducing irrelevant and redundant visual data. Compressing images entails retraining essential information while simultaneously decreasing the amount of the picture data. In mathematical terms, this entails converting a 2D array of pixels into a collection of data that is statically uncorrelated. The picture undergoes the alteration before it is stored or sent. In order to rebuild the picture or a close approximation of it, the image that was compressed is decompressed at a later time. Therefore, in order to reduce the amount of memory required to store a picture, image compression is used. It is impossible to transmit or store images that need an enormous amount of bits to be represented without lowering them in some way. Most photos have the trait of having correlated, and so redundant, information contained in nearby pixels. Finding a less associated representation of the picture is thus the primary objective. Redundancy & irrelevancy reduction are two essential components of compression. Eliminating duplicates in the signal source (picture or video) is the goal of redundancies reduction. Irrelevancy reduction removes information from the signal that is irrelevant to the receiver, in this case the eye of a human being.

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

II. RELATED WORK

A. Concise Overview of Common Methods and Standards for Image Compression:

With implications for many areas of image processing, this article provides a comprehensive overview of image compression methods and standards. In order to transmit and store images efficiently, image compression is crucial. This study introduces the Principal Component Analysis, a method for picture compression based on an analysis and evaluation of existing methods. There are a variety of methods for doing this compression; some use entropy coding to preserve all of the original image's information while others use lossy compression, which results in data loss. There are a number of other current standards for both still and moving picture compression outside of these methods, including as JPEG, JPEG2000, MPEG, and H.26x. In this research, I will compare and contrast some popular image compression techniques using a dataset of photos.

B. Compression through the use of Huffman coding:

As computer usage grows in new fields, more and more applications are relying on huge data systems, which in turn need the storage of massive data sets. Simultaneously, there is a tremendous increase in the transit of data via communication connections due to the growth of communication networks. One way to save money on storage and transmission is to compress data before storing or sending it. Decreases in the quantity of data that has to be communicated have the effect of making the communication channel more capable. In this paper, we provide a technique that efficiently decodes compressed data. An efficient Huffman decoding system is the focus of this work. The suggested model improves the decoding operation's speed. Xilinx tools are used for synthesis, placing and routing, floor planning, and simulation using Active HDL 5.1. The model is constructed using VHDL language.

C. Review of Images and Their compression Techniques:

Thanks to technical progress, which has no effect on picture management, advanced software tools that alter images, or the availability of many image operations, the accessibility of images in various applications is increased. There has been an outsized demand for storage capacity and communication bandwidth, even though both have been improved by technical advancements in transmission and storage. Because of this, picture compression has shown to be an effective method. Images and their compression methods are covered in this work. Our recommendations for the best picture compression algorithm are based on the review's overall criteria.

D. Lossless Compression of grayscale images utilizing the reduction of source symbols alongside Huffman coding:

Numerous apps have begun to make use of images, and their use has only grown. The ability to compress pictures before transferring them over a network is crucial for reducing file sizes and transmission times. The goal of this novel compressing method is to decrease the amount of source symbols while improving the compression proportion. Applying source symbols reductions and Huffman coding further reduces the source symbols to accomplish compression. The source symbols reduction method merges many source symbols into one new symbol, thereby reducing the amount of source symbols. So, it was also necessary to create fewer Huffman codes. With the decrease of Huffman code symbols, a greater compression ratio is achieved. Huffman coding and the suggested method were applied to standard pictures in the experiment. After looking at the data from the experiments, we can see that the newly suggested compression method attains a 10% higher compression ration than the standard Huffman coding.

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

E. The Huffman algorithm for double-compression of test data:

The quantity of test data is growing as a consequence of advancements in fabrication technologies and the complexity of designs. The main challenge in testing System-on-Chip (SoC) becomes the exponential growth of test size in relation to memory capacity. Multiple compression techniques have been proposed to reduce the volume of test data. One of these compression methods is code based approaches. When it comes to code-based compression, run-length coding is among the most used coding approaches. We may compress the test data and get a much higher compression ratio by using run length codes including Golomb codes, Frequency Directed Run Length Code (FDR code), Enhanced FDR, Adapted FDR, Shifted Alternate FDR, and OLEL coding techniques. A twofold compression strategy using the Huffman code is suggested for additional reduction of the input-test data. A comparison is made between the compression ratios achieved by various Run length coding and those obtained utilising the Double compression approach.

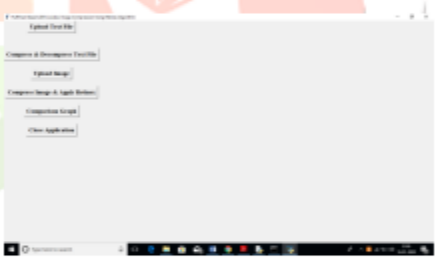
F. Image Compression with no Loss through combination:

The proliferation of digital photography and multimedia has resulted in an explosion in the amount of data needed to depict contemporary pictures. The storage space needed is substantial, and the time needed for transmission across computer networks is substantial, both of which are somewhat costly. All of these things point to the need of picture compression. To reduce the time and space needed for storing and transmitting digital images, image compression finds a solution by creating a smaller version of the original. The main point here is to decrease the size of a picture without losing any of the important information by removing redundant data from it. In this study, we focus on lossless picture compression. Our suggested method draws on a variety of established approaches. First, we subject the picture to the famous Lempel-Ziv-Welch (LZW) algorithm; this is the first step in our method. In the second stage, the results of the first stage are sent into the Bose, Chaudhuri, and Hocquenghem (BCH) algorithm for error detection and repair. The suggested method employs the BCH algorithms iteratively until "inflation" is identified in order to enhance the compression ratio. When tested against industry-standard compression methods, the experimental findings demonstrate that the suggested method can produce a superior compression ratio with no data loss.

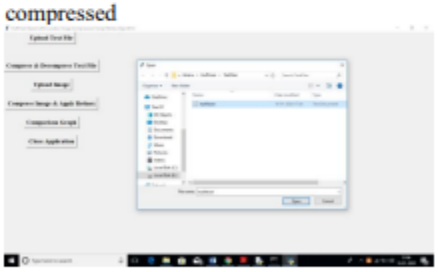
METHODOLOGY

1) **Upload Text**: We shall upload text using this module.
2) **Compress decompress Text:** This module will be used to compress and decompress text.
3) **Upload Image**: An picture will be uploaded using this module.
4) **Compress Image**: using this module, the image is compressed.
5) **Comparison graph**: We will compare the file sizes before and after compression using this module.
6) **Close application**: The application will be terminated when this module is used

III. RESULT AND DISCUSSION



Click the "Upload Text File" button on the previous page to submit text files that you would want compressed

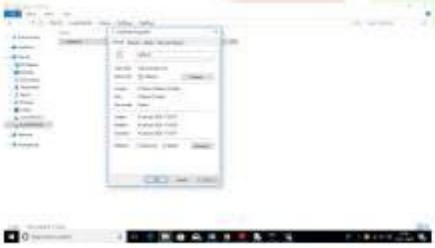


I am now uploading a file called "myfile.txt" to the screen up above; after it is uploaded, I will see the screen below.

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

To learn how to compress and decompress a text file, click the "Compress & Decompress Text File" button.

The notice is shown on the above screen as the compressed file size is 11 bytes, however the real file size is 21 bytes. The 'Text Files' folder now displays the original file size.



Now we can see the compressed file size within the 'compress' folder, as addition to the original file size in the directory, as seen in the previous screen.

On the screen up there, compress. The decompress file includes the decompress data, while the bin file is 11 bytes in size. To upload a picture, do the same thing and click the "Upload Image" button.

After you've uploaded the "images.jpg" file in the previous screen, you'll see the one below.

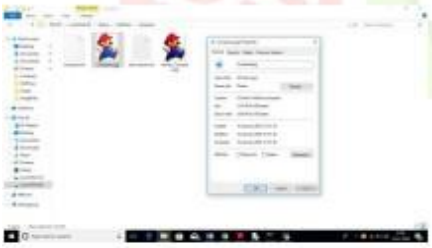
Then click on 'Compress Image & Apply Retinex' button to compress image

We can see the original image's name and size in the title bar of the first image; it's 10271 bytes in size. The second image is the compressed image; its size is 5459 bytes in size. The third image is the result of applying the Retinex algorithm to the compressed image; its size is also the same. After applying the algorithm, the third image looks slightly cleaner and brighter. The following graph will be appeared when you click the "Comparison Graph" button.

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

The y-axis in the following graph shows the size of the compressed picture, while the x-axis shows the regular image. We may lower the amount of the picture memory by applying compression, as seen plainly in the above graph. The directory now displays,the,picture-size

Pictured above are the originals. View the compressed file size in the compressed folder; the original jpg file is 10 KB in size.



Pictured above are the originals. View the compressed file size in the compressed folder; the original jpg file is 10 KB in size.

IV. CONCLUSION

With its wide range of uses, compression is an important and consequential subject. This thesis showcases the flawless image compressing method applied to various pictures. A number of methods have been tested for their compression levels, efficiency, and error proneness. The degree of compression accomplished is highly reliant on the source's qualities, in contrast to algorithm's efficiency and error susceptibility, which are somewhat independent of the trait of the source ensemble. Further compression is possible with additional data redundancy, it is determined. Since the Retinex Algorithm uses MSR to improve picture contrast, the quality of the reproduced image is same to that of the original.

VI. REFERENCES

[1] Vartika Singh "A Concise Overview of Common Methods and Standards for Image Compression" The 2013 Volume II edition of the International Journal of Technology and Research Advances.

[2] MamtaSharmap "Compression through the use of Huffman coding" The May 2010 issue of VOL.10 No.5 of the IJCSNS International Journal of Computer Science and Network Security.

[3] Sindhu M and Rajkamal R's paper titled "A Review of Image Compression Techniques" published in the International Journal of Recent Trends in Engineering, Volume 2, Issue 4, November 2009.

[4] C. Saravanan and R. Ponalagusamy's research on "Lossless Compression of Gray-scale Images via Reduction of Source Symbols Combined with Huffman Coding."

[5] MayurNandihalli, VishwanathBaligar "Lossless Compression of Gray Scale Images Utilizing Dynamic Arrays for Prediction, Focusing on Higher Bit Planes" published in the International Journal of Scientific and Research Publications, Volume 3, Issue 1, January 2013.

www.ijcrt.org © 2024 IJCRT | Volume 12, Issue 3 March 2024 | ISSN: 2320-2882

[6] R.S. Aarthi, D. Muralitharan, and P. Swaminathan's study on "Enhanced Compression of Test Data Employing Huffman Coding" published in the Journal of Theoretical and Applied Information Technology in May 2012.

[7]A.Al-Arabiya, S. Al-Hashemi, T. Khdour, M. Hjouj Btoush, S. Bani-Ahmad, and R. Al-Hashemi's research on "Lossless Image Compression Techniques through Combined Methods" published in the Journal of Software Engineering and Applications in 2012.

[8] Md. Rubaiyat Hasan's research on "Utilizing Huffman-based LZW Encoding Technique for Data Compression" published in the International Journal of Scientific & Engineering Research, Volume 2, Issue 11, November 2011.

[9] Monika and Pragati Kapoor's study on "Implementing Image Compression through a Hybrid Huffman-based LZW Approach" in 2007.

[10] A. Maliah, S. K. Shabbir and T. Subhashini's paper titled "Enhancing Image Compression and Quality through AnSpiht Algorithm with Huffman Encoder, Incorporating Retinex Algorithm" published in the June 2012 issue of the International Journal of Scientific & Technology Research, Volume 1, Issue 5.

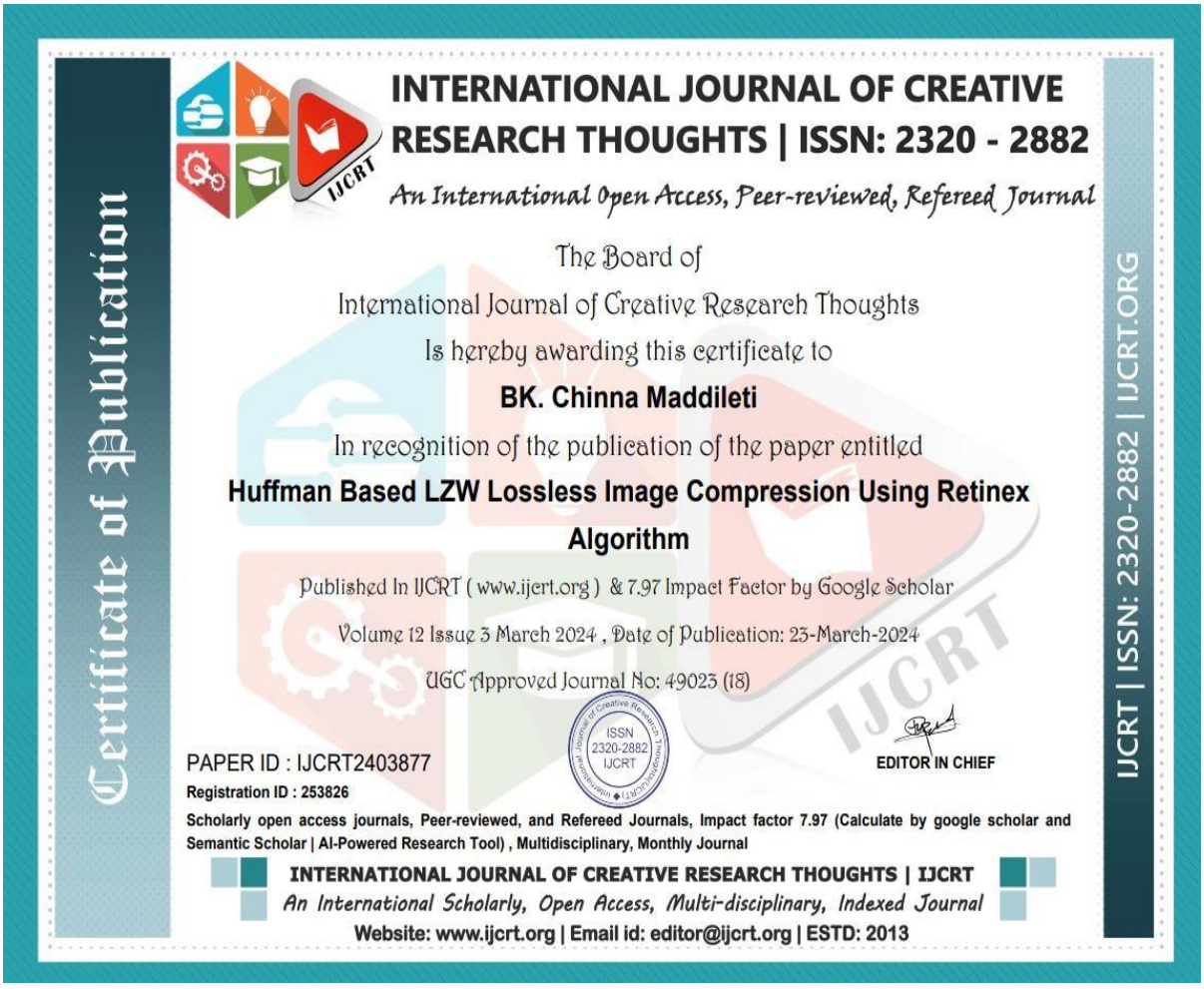
[11] C. Saravanan and M. Surender's study titled "Improving Huffman Coding Efficiency with Lempel Zig Coding for Image Compression" published in the International Journal of Soft Computing and Engineering (IJSCE) in January 2013.

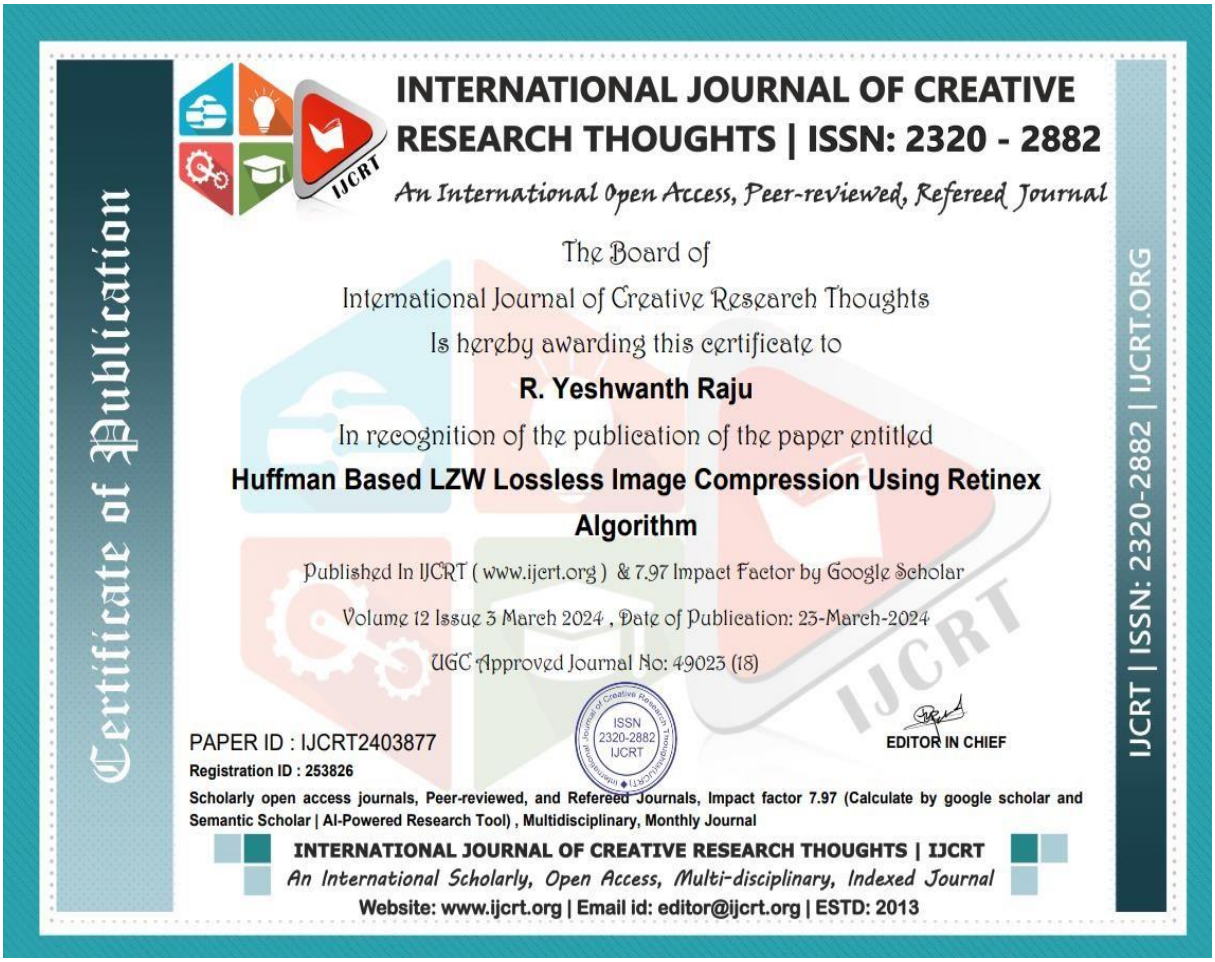
[12] Mr.D.V.Patil, Mr.S.G.SutarMrs.A.N.Mulla.. "Automated Improvement of Image Clarity through Retinex Method for Enhanced Visualization" published in the International Journal of Scientific and Research Publications, Volume 3, Issue 6, June 2013.

[13]JiangXing-fanam) , "Enhancing Color Images with an Advanced Multi-Scale Retinex Algorithm" presented at the International Symposium on Photoelectronic Detection and Imaging in 2007 by Tao Chun-kan.

[14] Examining the correlation between image improvement and image size reduction within the framework of multi-scale Retinex. (Rahman, Z., Jobson, D.J., Woodall, G.A. 2011. Journal of Visual Communication and Image Representation, Volume 22, Pages 237–250, ScienceDirect)

[15]AsadollahShabhahrami, RaminBahrampour, MobinSabbaghiRostami, Mostafa AyoubiMobarhan, "Assessment of Huffman and Arithmetic Algorithms for Multimedia Compression Norms"


Certificate of Publication
INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882
An International Open Access, Peer-reviewed, Refereed Journal
The Board of
International Journal of Creative Research Thoughts
Is hereby awarding this certificate to
BK. Chinna Maddileti
In recognition of the publication of the paper entitled
Huffman Based LZW Lossless Image Compression Using Retinex Algorithm
Published In IJCRT (www.ijcrt.org) & 7.97 Impact Factor by Google Scholar
Volume 12 Issue 3 March 2024 , Date of Publication: 23-March-2024
UGC Approved Journal No: 49023 (18)
PAPER ID : IJCRT2403877
Registration ID : 253826
EDITOR IN CHIEF
Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal
INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT
An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal
Website: www.ijcrt.org | Email id: editor@ijcrt.org | ESTD: 2013
IJCRT | ISSN: 2320-2882 | IJCRT.ORG


Certificate of Publication
INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882
An International Open Access, Peer-reviewed, Refereed Journal
The Board of
International Journal of Creative Research Thoughts
Is hereby awarding this certificate to
R. Yeshwanth Raju
In recognition of the publication of the paper entitled
Huffman Based LZW Lossless Image Compression Using Retinex Algorithm
Published In IJCRT (www.ijcrt.org) & 7.97 Impact Factor by Google Scholar
Volume 12 Issue 3 March 2024 , Date of Publication: 23-March-2024
UGC Approved Journal No: 49023 (18)
PAPER ID : IJCRT2403877
Registration ID : 253826
EDITOR IN CHIEF
Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal
INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT
An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal
Website: www.ijcrt.org | Email id: editor@ijcrt.org | ESTD: 2013
IJCRT | ISSN: 2320-2882 | IJCRT.ORG

Certificate of Publication

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of

International Journal of Creative Research Thoughts

Is hereby awarding this certificate to

M. Pavan Kumar

In recognition of the publication of the paper entitled

Huffman Based LZW Lossless Image Compression Using Retinex Algorithm

Published In IJCRT (www.ijcrt.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 3 March 2024 , Date of Publication: 23-March-2024

UGC Approved Journal No: 49023 (18)

PAPER ID : IJCRT2403877

Registration ID : 253826

ISSN 2320-2882 IJCRT

EDITOR IN CHIEF

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijcrt.org | Email id: editor@ijcrt.org | ESTD: 2013

IJCRT | ISSN: 2320-2882 | IJCRT.ORG