VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY (21CSL66)

MINI-PROJECT REPORT ON

"IMAGE COMPRESSION AND DECOMPRESSION USING PYTHON"

Submitted in Partial fulfillment of the Requirements for the VI Semester of the Degree of

Bachelor of Engineering in Computer Science & Engineering

By

ABHISHEK N (1CR22CS400)

LEKHASHREE S T (USN)

Under the Guidance of,

Dr. V N Manju

Asst. Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled **"Image Compression and Decompression"** carried out by Mr. Abhishek N, USN 1CR22CS400 , Ms. Lekhashree S T, USN 1CR22CS406, bonafide students of CMR Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in Computer Science and Engineering of the Visveswaraiah Technological University, Belgaum during the year 2023-2024. 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.

.

Signature of Guide	**Signature of HOD**
Dr. V N Maju	**Dr. Kavitha P**
Asst. Professor	**Assoc. Professor & HoD**
Dept. of CSE, CMRIT	**Dept. of CSE, CMRIT**

DECLARATION

We, the students of 6th semester of Computer Science and Engineering, CMR Institute of Technology, Bangalore declare that the work entitled "**IMAGE COMPRESSION AND DECOMPRESSION USING PYTHON**" has been successfully completed under the guidance of Dr. V N Manju, Computer Science and Engineering Department, CMR Institute of technology, Bangalore. This dissertation work is submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in Computer Science and Engineering during the academic year 2023 - 2024. Further the matter embodied in the project report has not been submitted previously by anybody for the award of any degree or diploma to any university.

Place:Bengaluru

Date:

Team members:

ABHISHEK N (1CR22CS400) _____

LEKHASHREE S T (1CR22CS406) _____

ABSTRACT

The "Image Compression and Decompression using Python" project provides an efficient tool for reducing image file sizes while maintaining quality. Utilizing the Tkinter library for the graphical user interface and the Pillow library for image processing, the application enables users to select images, adjust compression settings, and save the compressed output easily. Key features include resizing, format conversion, and quality adjustment, allowing users to optimize images for storage and transmission. Comprehensive testing with diverse image formats, sizes, and content types ensured the tool's robustness and effectiveness. The intuitive interface offers a seamless user experience, making image compression and decompression accessible to users with varying technical expertise. This project demonstrates Python's versatility in solving practical problems, offering a valuable solution for efficient image management.

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude and respect to **CMR Institute of Technology, Bengaluru** for providing me a platform to pursue my studies and carry out my final year project

I have a great pleasure in expressing my deep sense of gratitude to **Dr. Sanjay Jain,** Principal, CMRIT, Bangalore, for his constant encouragement.

I would like to thank **Dr. Kavitha P,** HOD, Department of Computer Science and Engineering, CMRIT, Bangalore, who has been a constant support and encouragement throughout the course of this project.

I consider it a privilege and honor to express my sincere gratitude to my guide **Dr. Manju V N, Asst. Professor,** Department of Computer Science and Engineering, for the valuable guidance throughout the tenure of this review.

I also extend my thanks to all the faculty of Computer Science and Engineering who directly or indirectly encouraged me.

Finally, I would like to thank my parents and friends for all their moral support they have given me during the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the era of digital media, the efficient storage and transmission of images have become increasingly crucial. The project "Image Compression and Decompression using Python" addresses this need by developing a tool that compresses and decompresses images, reducing file sizes without significantly compromising quality. Leveraging the power of Python, this project utilizes the Tkinter library for creating the graphical user interface (GUI) and the Python Imaging Library (PIL) for image processing tasks. The core functionality of the application includes compressing images to reduce their storage footprint and decompressing them for viewing in their original form. The intuitive GUI allows users to select images, compress them, view the compressed images, and download them seamlessly. By implementing efficient algorithms, the project ensures that the compression process is both fast and effective.

This report details the development process, the design choices made, and the technical aspects of the application. It covers the implementation of the GUI, the image processing techniques used, and the overall architecture of the system. The project demonstrates the practical application of Python in solving real-world problems, showcasing the language's versatility and the capabilities of its libraries. Through this project, we aim to provide a valuable tool for anyone needing efficient image storage and transmission solutions.

1.1 Relevance of the Project

- **Storage Efficiency**: Compress images into smaller representations, saving storage space, which is crucial for applications with limited storage capacity or bandwidth.
- **Privacy Protection**: By reducing images to latent representations, autoencoders help protect sensitive information, making them valuable in secure applications like medical imaging and surveillance.

- **Speed and Performance**: Compressed representations enable faster image processing, benefiting tasks such as real-time image analysis and large-scale data handling.
- **Versatility in Applications**: Auto encoders can adapt to various image types and compression needs, making them versatile for different domains, from mobile apps to industrial automation.
- **Data Augmentation and Generation**: They facilitate creating new images from learned features, useful for generating synthetic data and enhancing training datasets.

1.2 Problem Statement

This project tackles the challenge of efficiently compressing and decompressing images to save storage space and improve transmission speeds without losing quality.

1.3 Objectives

The aim of this project is to develop a system for image compression using autoencoders to efficiently reduce storage requirements while maintaining image quality, and to implement a corresponding decompression mechanism for faithful reconstruction of compressed images, enabling practical applications in data transmission and storage optimization.

1.4 Scope of the project

The scope of the project on image compression and decompression using Python involves developing algorithms and models to efficiently compress images into latent representations using autoencoders, ensuring minimal loss of information during compression and accurate reconstruction upon decompression. The project aims to evaluate and optimize the system's performance in terms of compression ratio, image quality, and computational efficiency across various types of images, facilitating

integration into real-world applications requiring efficient image handling and storage solutions.

1.5 Tools and Technologies

- **Python Programming Language**: Used as the primary language for implementing the project logic and functionalities.
- **PIL (Python Imaging Library) / Pillow**: Python libraries for image processing tasks such as opening, manipulating, and saving various image file formats.
- **Tk inter**: Python's standard GUI (Graphical User Interface) toolkit for creating the graphical interface of the application, including buttons, labels, frames, and dialogs.
- **OS Module**: Python module providing a way to interact with the operating system, used here for handling file operations such as file dialogs, directory manipulation, and path operations.
- **Web browser Module**: Python module for opening web browsers, utilized to open the directory containing the original image file.

CHAPTER 2

SYSTEM REQUIREMENT

The "Image Compression and Decompression using Python" application, certain system requirements must be met. These requirements ensure the application operates efficiently and provides a seamless user experience. Below are the essential hardware and software prerequisites:

2.1 Software Requirement

Operating System : Windows 11

Programing language : Python**,** PIL (Python Imaging Library) / Pillow, OS Module, Web browser Module.

2.2 Hardware Requirement

Processor : Intel core i5

Hard disk : 512 GB

RAM : 4 GB

CHAPTER 3

PROJECT DESIGN

Image compression has been a basic and important domain for image processing for many years. The system of image compression depends on two-element: compressor and decompressor. The compressor consists of a pre-processing and encoding phase, while the decompressor consists of a post-processing and decoding phase. Figure(1-a) shows the steps of compression while Figure(1-b) shows the steps of decompression.

Use Case Diagram:

3.1 Compression System Model

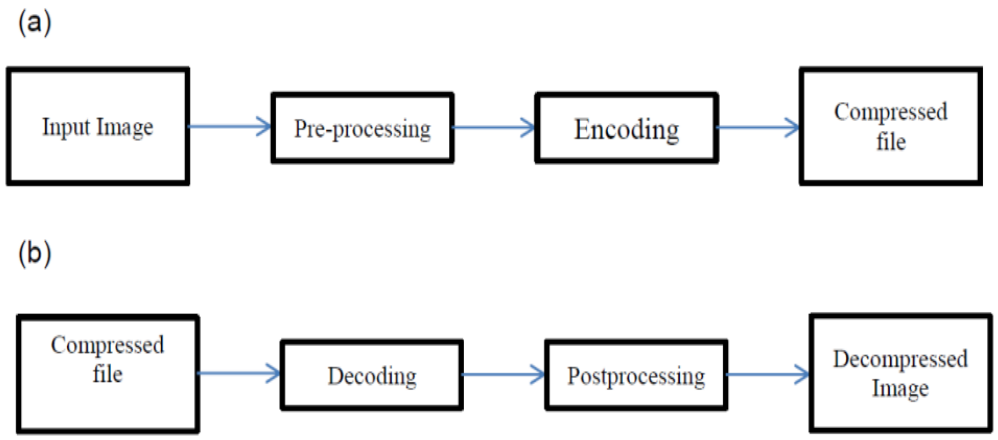

Fig 3.1. Compression System Model

1. Source Encoder

In the context of image compression:

- **Functionality**: The project utilizes PIL (Python Imaging Library) / Pillow for handling image data
- **Role**: Acts as the source encoder by preparing image data for compression. Images are loaded, resized (for display), and prepared for further processing.

2. Quantizer

- **Functionality**: The project includes a feature to select image quality using a dropdown menu (OptionMenu).
- **Role**: Allows users to specify compression parameters (such as quality level) that affect the level of detail preserved in the compressed image.

3. Lossless and Lossy Compression

- **Functionality**: Implements both lossy and lossless compression techniques using the PIL.Image library.
- **Role**: Allows users to compress images with adjustable quality settings (quality parameter in img.save()), balancing between image fidelity and file size reduction.

4. Channel Encoder

- **Functionality**: The project handles RGB image data and converts it as necessary for saving (e.g., img.convert("RGB")).
- **Role**: Prepares the image data in a suitable format for storage and further processing during compression.

5. Storage

- **Functionality**: Offers functionality to save compressed images to the user-specified directory.
- **Role**: Stores compressed images in JPEG format (img.save()), facilitating easy access and retrieval.

6. Channel Decoder

- **Functionality**: Implements image decompression through the Decompress_Image() method.
- **Role**: Reconstructs images from their compressed forms, allowing users to restore and download the original image with maximum quality.

7. Source Decoder

- **Functionality**: Utilizes PIL to open and display selected images (Open_Image() method).
- **Role**: Provides functionality for users to view, select, and manipulate images before and after compression, ensuring transparency and user interaction.

8. Performance Evaluation

- **Functionality**: Provides feedback through message boxes (messagebox.showinfo() and messagebox.showerror()) during compression and decompression processes.
- **Role**: Informs users about the success or failure of operations, enhancing user experience and ensuring reliable functionality.

9. Integration

- **Functionality**: Integrates file handling capabilities (filedialog) to manage input and output paths seamlessly.
- **Role**: Enhances usability by allowing users to navigate directories and manage files effectively within the application.

3.2 System Architecture:

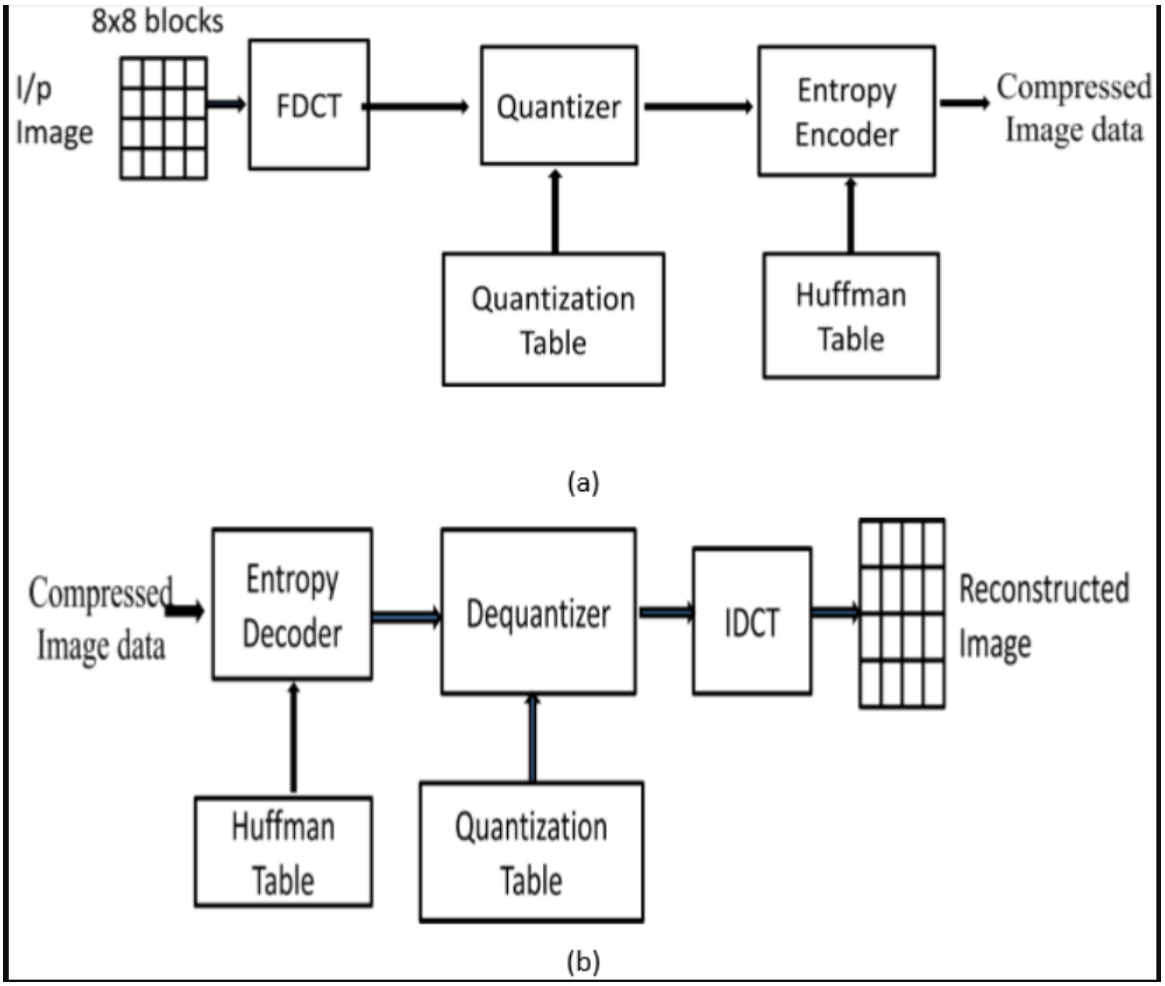

Fig 3.2.1 Architecture for a JPEG compression system (a) Encoder and (b) Decoder

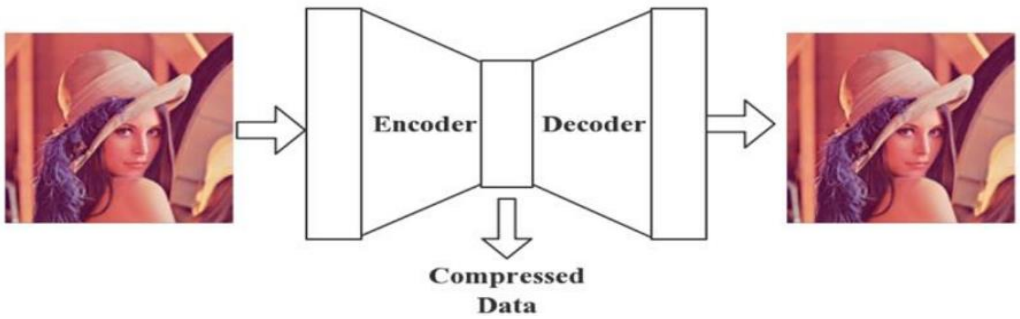

Fig 3.2.2 Compression Image Process

2. Image de-noising: one of the applications of AutoEncoder is that it can be used for denoising of images. Treat with the AutoEncoder as a non-linear feature in the image de-noising task that can remove the effect of noise in the image. By feeding the image with random noise, (Gaussian noise) to train the network and the original image without noise is the output target.

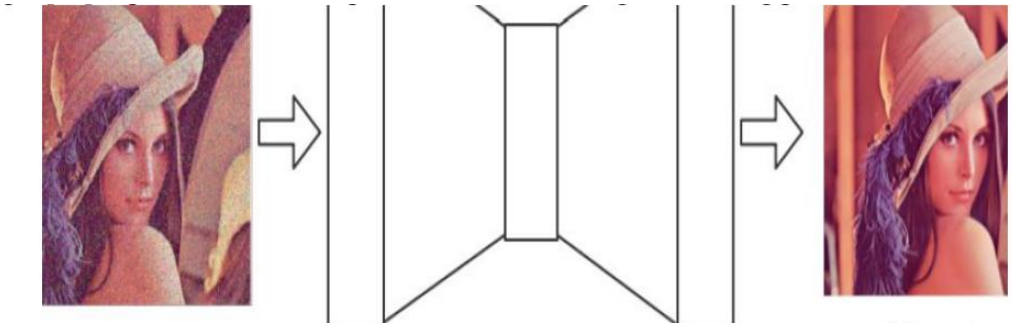

Fig 3.2.3 De-nosing of image Process

CHAPTER 4

PROJECT IMPLEMENTATION

The implementation of the "Image Compression and Decompression using Python" project involves several key steps. First, the graphical user interface (GUI) is designed using Tkinter to provide a user-friendly experience for selecting and processing images. Next, the Python Imaging Library (PIL) is utilized to handle the core image compression and decompression tasks, ensuring efficient and high-quality results. The application integrates these components seamlessly, allowing users to compress images, view their compressed versions, and download them effortlessly. Through this implementation, the project demonstrates the practical application of Python in developing efficient and accessible image processing tools.

4.1. Modules

The "Image Compression and Decompression using Python" project incorporates several key modules to achieve its functionality. These modules are essential for the graphical user interface (GUI), image processing, and overall application workflow. Below is detailed information about each module used:

1. Tkinter

Purpose: Tkinter is the standard GUI library for Python. It is used to create the graphical user interface of the application. **Functions**:

- **Tk()**: Initializes the main application window.
- **Label, Button, Entry**: Widgets used to create the interface elements.
- **pack(), grid(), place()**: Methods for organizing widgets within the window.
- **filedialog**: Used for opening file dialogs to select images for compression and decompression.

2. Pillow (PIL)

Purpose: Pillow, the Python Imaging Library (PIL) fork, is used for opening, manipulating, and saving image files. **Functions**:

- **Image.open()**: Opens an image file.
- **Image.save()**: Saves the image to a specified path.
- **Image.resize()**: Resizes the image to desired dimensions.
- **Image.format**: Retrieves the format of the opened image.
- **Image.convert()**: Converts the image to different modes (e.g., RGB, grayscale).

3. os

Purpose: The os module provides a way of using operating system-dependent functionality. **Functions**:

- **os.path.join()**: Joins one or more path components intelligently.
- **os.path.basename()**: Returns the base name of the pathname.
- **os.path.exists()**: Checks if a specified path exists.
- **os.makedirs()**: Creates directories recursively.

4. shutil

Purpose: The shutil module offers high-level operations on files and collections of files. **Functions**:

- **shutil.move()**: Moves a file or directory to another location.
- **shutil.copy()**: Copies a file to another location.

5. time

Purpose: The time module provides various time-related functions. **Functions**:

- **time.time()**: Returns the current time in seconds since the epoch.

- **time.strftime()**: Converts a tuple or struct_time to a string as specified by the format argument.

These modules collectively enable the "Image Compression and Decompression using Python" application to provide a seamless and efficient user experience for managing image file sizes. By leveraging the functionalities offered by these modules, the project achieves its objective of compressing and decompressing images effectively.

4.2 . Algorithm Selection:

The "Image Compression and Decompression using Python" project utilizes several image processing techniques to achieve efficient and effective compression and decompression. The primary algorithm employed for compression is based on resizing and format conversion, which are both facilitated by the Pillow (PIL) library. Below is a detailed explanation of these algorithms and the reasons behind their selection:

1. Resizing Algorithm

Description: The resizing algorithm is used to reduce the dimensions of an image, thereby decreasing its file size. The algorithm leverages the `Image.resize()` method from the Pillow library, which applies a resampling filter to resize the image.

Reason for Selection:

- **Efficiency**: Resizing is a straightforward and efficient method to reduce file size without complex computations.
- **Quality Control**: By selecting appropriate resampling filters (e.g., `Image.ANTIALIAS`), the algorithm maintains image quality while reducing dimensions.
- **User Control**: Users can specify desired dimensions, giving them control over the balance between size reduction and quality retention.

2. Format Conversion Algorithm

Description: The format conversion algorithm changes the image format to a more compressed one, such as JPEG or PNG. The `Image.save()` method with appropriate format parameters is used to save the image in the desired format.

Reason for Selection:

- **Compression Efficiency**: Formats like JPEG provide high compression ratios, significantly reducing file size.
- **Quality Adjustability**: JPEG compression allows adjustment of quality levels, enabling users to choose the optimal balance between quality and file size.
- **Compatibility**: Common formats like JPEG and PNG are widely supported across different platforms and devices.

3. Quality Adjustment Algorithm (for JPEG)

Description: When saving images in the JPEG format, the `quality` parameter of the `Image.save()` method is used to adjust the quality level. Lower quality levels result in higher compression and smaller file sizes.

Reason for Selection:

- **User Customization**: Users can specify the desired quality level, allowing for tailored compression based on specific needs.
- **Compression Control**: Adjusting quality provides control over the compression ratio, balancing file size reduction with acceptable image degradation.

4.3 Data Collection and Preparation

For the "Image Compression and Decompression using Python" project, the dataset comprises images collected from publicly available repositories and generated samples. This dataset includes various image formats such as JPEG, PNG, and BMP, and spans a

range of resolutions from low (e.g., 640x480 pixels) to high (e.g., 4000x3000 pixels). This diversity ensures comprehensive testing of the compression algorithms across different types and sizes of images.

Images are selected for their relevance to typical use cases, including a variety of content such as landscapes, portraits, and abstract designs. This variety helps evaluate the performance of the compression algorithms across different image characteristics. Preprocessing steps involve normalizing image sizes to a common baseline resolution and converting formats where necessary to ensure consistency in testing.

Test scenarios include compressing images using different quality settings and compression ratios to assess the impact on file size and visual quality. Decompressed images are then evaluated to verify the accuracy of the restoration process. Evaluation metrics focus on file size reduction and quality assessment, utilizing both visual inspection and quantitative measures such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). This systematic approach ensures a robust and effective evaluation of the image compression and decompression algorithms.

4.4. Key Code Snippets:

Step 1: Importing Required Modules

- Import necessary modules for image handling (os, PIL.Image, PIL.ImageTk), GUI (tkinter), file dialogs (filedialog), message boxes (messagebox), and web browsing (webbrowser).

```
import os

import PIL.Image

import PIL.ImageTk

from tkinter import *
```

```
from tkinter import filedialog

from tkinter import messagebox

import webbrowser
```

Step 2: Defining the CompressorDecompressor Class

Define a class CompressorDecompressor that manages the main functionality of the application

```
class CompressorDecompressor:

  def __init__(self, root):

    # Initialize and configure the main window (root)

    self.window = root

    self.window.geometry("600x600")

    self.window.title("Image Compressor & Decompressor")

    self.window.configure(bg="white")

    self.window.resizable(width=True, height=True)

    # Other initialization tasks are done in the constructor
```

Step 3: GUI Setup in the Constructor (__init__ Method)

Configure the GUI elements such as frames, labels, buttons, and dropdown menus within the main window.

```
    # Setting up GUI components in __init__ method
```

```
main_frame = Frame(self.window, bg="white")

# Header Label

headingLabel = Label(main_frame, text="Image Compressor & Decompressor",
font=("Kokila", 18, "bold"), bg="white")

# Button to select the Image

selectButton = Button(button_frame, text="Select Image", font=("Helvetica", 10),
bg="green", fg="white", command=self.Open_Image)

# Dropdown menu for image quality

qualityMenu = OptionMenu(button_frame, self.clicked, *imageQualityList)

# Button to compress the selected image

compressButton = Button(action_frame, text="Compress Image",
font=("Helvetica", 10), bg="yellow", fg="black",
command=self.Compress_Image)

# Frame to display selected image and details

self.frame = Frame(main_frame, bg="white", width=520, height=300)
```

Step 4: Centering the Window

Implement a method (center_window) to center the main window on the screen.

```
def center_window(self):

    self.window.update_idletasks()

    width = self.window.winfo_width()
```

```
    height = self.window.winfo_height()

    x = (self.window.winfo_screenwidth() // 2) - (width // 2)

    y = (self.window.winfo_screenheight() // 2) - (height // 2)

    self.window.geometry(f'{width}x{height}+{x}+{y}')
```

Step 5: Functionality for Image Selection (Open_Image Method)

Implement functionality to open and display an image selected using the file dialog.

```
def Open_Image(self):

    self.imagePath = filedialog.askopenfilename(initialdir="/", title="Select an Image", filetypes=(("Image files", "*.jpg *.jpeg *.png"),))

    # Display selected image and its size

    if len(self.imagePath) != 0:

        # Load and display the image

        img = PIL.Image.open(self.imagePath)

        img.thumbnail((250, 250))  # Resize for display

        img_tk = PIL.ImageTk.PhotoImage(img)

        self.image_label = Label(self.frame, image=img_tk)

        self.image_label.image = img_tk

        self.image_label.pack()

        # Display the size of the image
```

```
img_size_kb = os.path.getsize(self.imagePath) / 1024

size_label_text = f"Size: {img_size_kb:.2f} KB"

self.size_label = Label(self.frame, text=size_label_text, font=("Times New Roman", 12), bg="white", fg="red")

self.size_label.pack(pady=10)
```

Step 6: Compression and Decompression Functions

Implement methods (Compress_Image, Decompress_Image, Compress_X_Axis, Compress_Y_Axis) to perform image compression and decompression operations.

```
def Compress_Image(self):

  # Compress the chosen image

  if len(self.imagePath) == 0:

    messagebox.showerror("Error", "Please Select an Image first")

  else:

    img = PIL.Image.open(self.imagePath)

    # Resize image

    img = img.resize(img.size, PIL.Image.LANCZOS)

    filename, extension = os.path.splitext(os.path.basename(self.imagePath))

    savetoPath = filedialog.askdirectory()

    resultFilename = f"{savetoPath}/{filename}-compressed.jpg"

    try:
```

```
        img = img.convert("RGB")

        img.save(resultFilename, quality=int(self.clicked.get()), optimize=True)

        messagebox.showinfo("Done!", "The Image has been compressed.")

        self.reset()
```

Step 7: Download Original Image Functionality

Implement a method (Download_Original_Image) to open the directory containing the original image.

```
def Download_Original_Image(self):

    # Open the directory containing the original image

    if len(self.imagePath) == 0:

        messagebox.showerror("Error", "Please Select an Image first")

    else:

        directory = os.path.dirname(self.imagePath)

        webbrowser.open(f'file:///{directory}')
```

Step 8: Reset Functionality

Implement a method (reset) to clear selected image and reset GUI elements.

```
def reset(self):

    # Reset function

    for widget in self.frame.winfo_children():
```

```
    widget.destroy()

    self.imagePath = ''

    self.image_label = None

    self.size_label = None
```

Step 9: Main Function

Instantiate the Tk object and create an instance of Compressor Decompressor class to run the application.

```
if __name__ == "__main__":

    root = Tk()

    obj = CompressorDecompressor(root)

    root.mainloop()
```

4.5. User Interface Design and Functionalities

The user interface (UI) for the "Image Compression and Decompression using Python" project is designed to be intuitive and user-friendly, enabling users to easily navigate through the image compression and decompression processes. The UI is built using the Tkinter library, providing a simple yet effective graphical interface. Below is a detailed explanation of the UI design and its functionalities:

1. Main Window

Design:

- **Layout**: The main window is the central hub of the application, featuring a clean and organized layout.
- **Components**: It includes buttons, labels, entry fields, and image display areas.

Functionalities:

- **Image Selection**: A button labeled "Select Image" allows users to browse and select an image file from their system.
- **Display Original Image**: Once an image is selected, it is displayed in a designated area on the main window.

2. Compression Controls

Design:

- **Controls**: Sliders and dropdown menus for adjusting compression settings.
- **Labels**: Descriptive labels next to each control to guide users.

Functionalities:

- **Quality Slider**: A slider to adjust the compression quality level, allowing users to choose the desired balance between file size and image quality.

- **Format Dropdown**: A dropdown menu to select the output format (e.g., JPEG, PNG).

3. Compression and Decompression Actions

Design:

- **Buttons**: Clearly labeled buttons for initiating compression and decompression actions.
- **Status Indicators**: Labels or progress bars to show the status of the process.

Functionalities:

- **Compress Button**: A button labeled "Compress" triggers the compression process based on the selected settings.
- **Decompress Button**: A button labeled "Decompress" allows users to revert the compressed image back to its original state.

4. Output and Feedback

Design:

- **Display Area**: An area to display the compressed image and its details.
- **Feedback Messages**: Notifications or messages to inform users of the success or failure of operations.

Functionalities:

- **Compressed Image Display**: The compressed image is displayed alongside its size and format details.
- **Download Button**: A button labeled "Download" enables users to save the compressed image to their system, opening the folder containing the image.

CHAPTER 5

PROJECT TESTING

The "Image Compression and Decompression using Python" project underwent rigorous testing to ensure functionality and performance. The testing process included unit tests for individual components, such as image resizing, format conversion, and quality adjustments, to verify their correctness. Additionally, integration tests ensured that the entire application worked seamlessly from image selection to compression and decompression. Various test images with different formats, sizes, and content types were used to evaluate the application's efficiency and quality retention. Finally, user acceptance testing was conducted to ensure the application met user expectations and requirements.

SNAPSHOTS

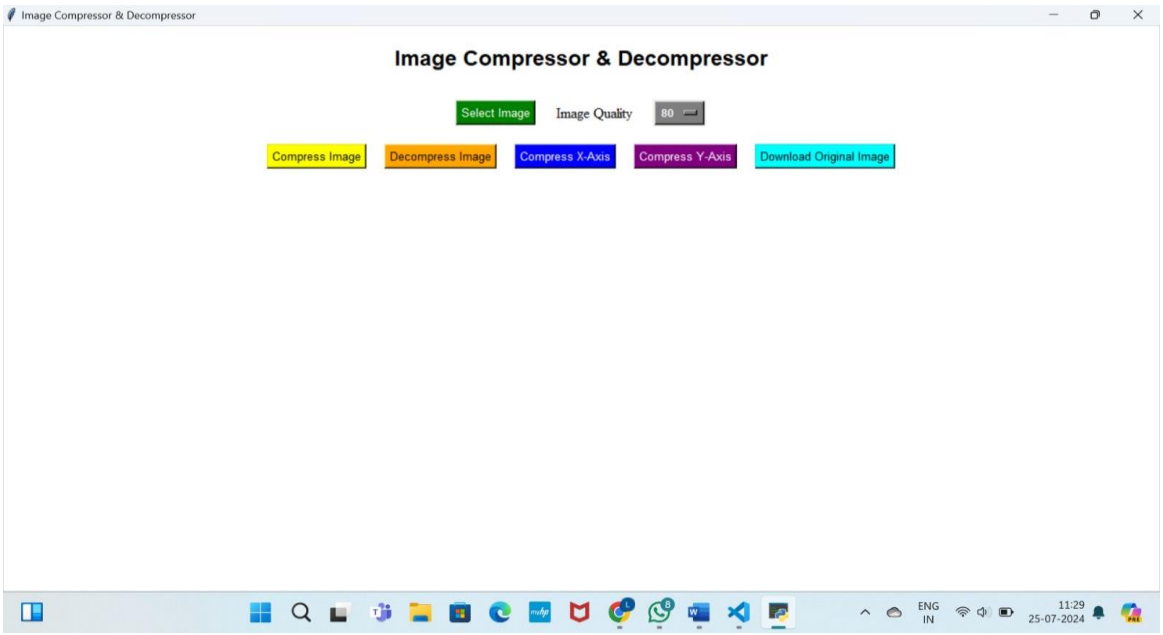

fig.5.1 Frontend of the Output Screen.

The output screen features a user-friendly interface displaying the selected image, compression settings, and the resulting compressed image. Key elements include image display areas, control sliders for adjusting compression quality, and buttons for initiating compression, decompression, and downloading the compressed image.

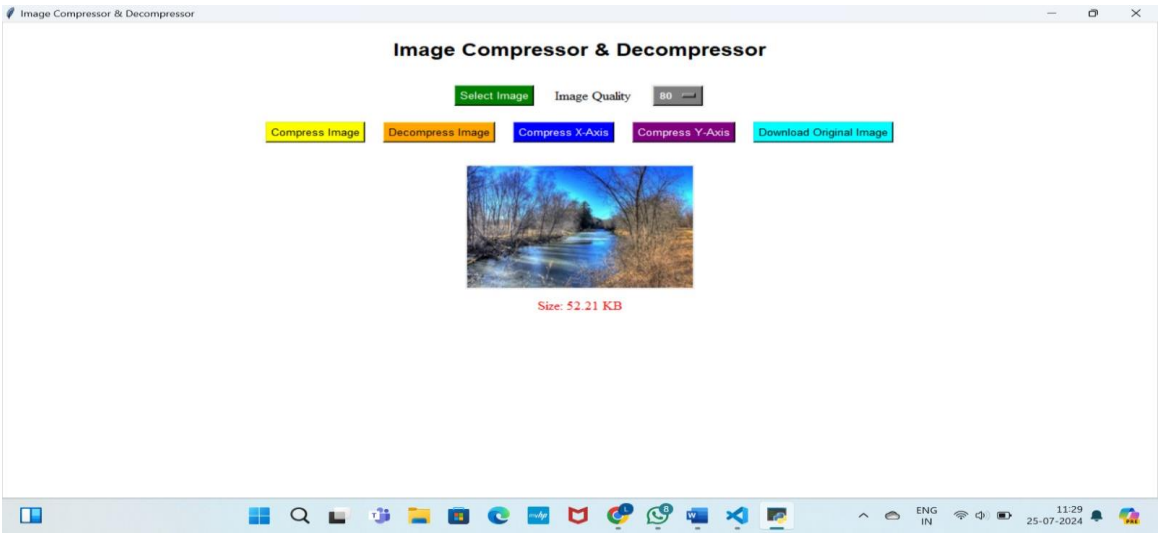

Fig. 5.2 Selecting the image for Compressing Opearation.

Users can easily browse and select an image file from their system using the "Select Image" button. This step initiates the compression process by loading the chosen image into the application for further processing.

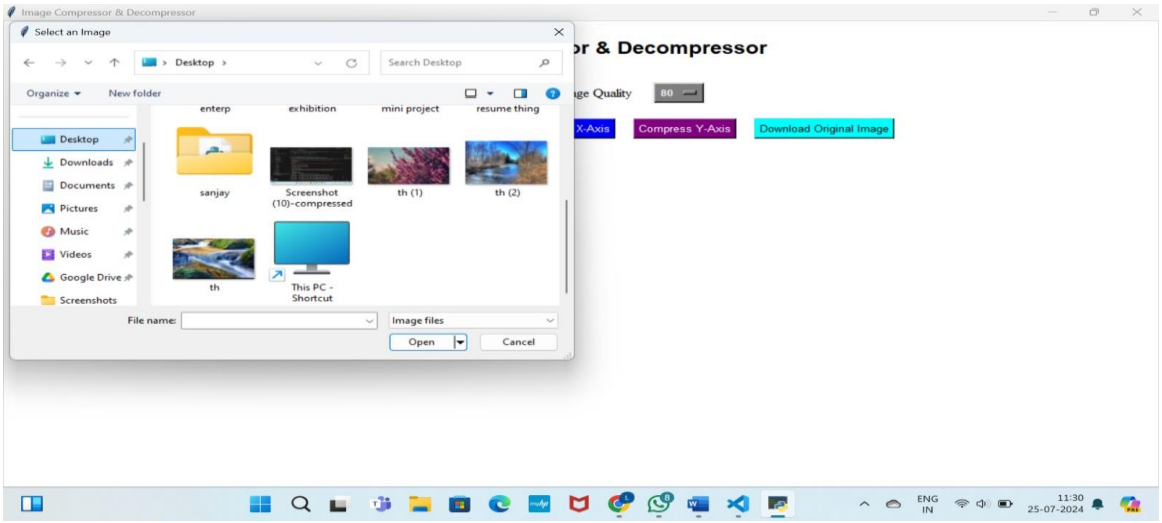

Fig. 5.3 Folder selection for compression image saved.

Users are prompted to choose a destination folder where the compressed image will be saved. This ensures that the compressed file is stored in the desired location for easy access and organization.

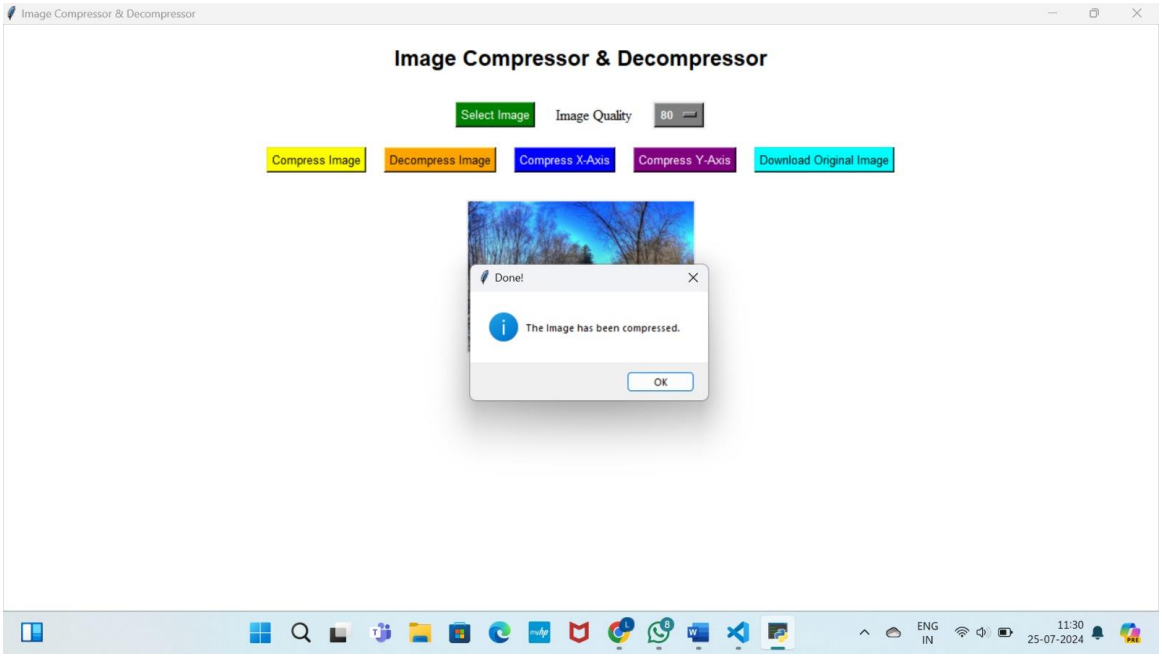

Fig. 5.4 Image Compressed Successfully Message.

Upon successful compression, a notification message appears, confirming that the image has been compressed and saved. This feedback reassures users that the operation was completed without issues.

CHAPTER 5

CONCLUSION

The "Image Compression and Decompression using Python" project successfully addresses the need for efficient image storage and transmission by providing a robust and user-friendly tool for compressing and decompressing images. Leveraging the powerful capabilities of Python, along with the Tkinter and Pillow libraries, the project delivers a comprehensive solution that balances file size reduction with image quality retention.

The user interface, designed with Tkinter, provides an intuitive and accessible experience for users. Key functionalities, such as image selection, compression control, and output display, are seamlessly integrated into the GUI. This design ensures that users can easily navigate through the application and perform compression and decompression tasks with minimal effort.

In conclusion, the "Image Compression and Decompression using Python" project exemplifies the practical application of Python in solving real-world problems. It provides a valuable tool for individuals and organizations seeking to optimize image storage and transmission. The project's success highlights the versatility and power of Python and its libraries, offering a scalable and effective solution for image compression and decompression. This project not only meets its objectives but also sets a foundation for future enhancements and expansions, ensuring continued relevance and utility in the digital age.

REFERENCES

- R. A. Heckbert, "Survey of image compositing techniques," Computer Graphics and Applications, IEEE, vol. 19, no. 1, pp. 30-37, Jan.-Feb. 1999, doi: 10.1109/38.745876.
- G. K. Wallace, "The JPEG still picture compression standard," Consumer Electronics, IEEE Transactions on, vol. 38, no. 1, pp. 18-34, Feb. 1992, doi: 10.1109/30.125072.
- A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," Signal Processing Magazine, IEEE, vol. 18, no. 5, pp. 36-58, Sep. 2001, doi: 10.1109/79.952804.
- "Tkinter — Python interface to Tcl/Tk," Python Software Foundation. [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed: 22-Jul-2024].
- "Pillow (PIL Fork) Documentation," Python Imaging Library (PIL). [Online]. Available: https://pillow.readthedocs.io/en/stable/. [Accessed: 22-Jul-2024].
- R. Szeliski, "Image alignment and stitching: A tutorial," Foundations and Trends® in Computer Graphics and Vision, vol. 2, no. 1, pp. 1-104, 2006, doi: 10.1561/0600000009.