# Mini-Photoshop Application

Kajori Roy
301549849

Submission Date: April 5, 2024
CMPT 820 Multimedia Systems

Simon Fraser University

**Abstract**

This report documents the development process, challenges, and outcomes of the Mini-Photoshop application, a project undertaken as part of the CMPT 820 Multimedia Systems course. The primary objective was to create a simplified version of Adobe Photoshop with a user-friendly graphical user interface (GUI) that supports basic image manipulation functionalities. The project was designed to enhance our understanding of GUI development, media file formats, and fundamental image processing operations. The Mini-Photoshop application successfully implements the following core operations, accessible through a Core Operations: Open File, Exit, Grayscale conversion, Ordered Dithering, Auto Level adjustment, and Huffman encoding analyses. These features laid the groundwork for the application, demonstrating essential capabilities in image processing and analysis. In addition to these mandatory features, the application was enriched with optional operations that explored beyond the basic requirements, adding significant value and functionality to the project. Sharpness Adjustment, Blur Effect, Image Rotation, Hue and Saturation Adjustment, Saving an Image. This report includes screenshots of the operation results for sample images provided ,discussions on the implementation details, challenges encountered, and interesting findings. Through the Mini-Photoshop project, we demonstrated a comprehensive understanding of multimedia systems and their applications in software development, contributing to our preparedness for tackling future projects in the domain of multimedia and image processing.

# Contents


**1 Introduction** **5**
- 1.1 Project Overview . . . 5
- 1.2 Objectives . . . 5

**2 Background** **5**

**3 Methodology** **6**
- 3.1 Tools and Technologies Used . . . 6
- 3.2 GUI and Interaction . . . 7
- 3.3 Core Feature Implementation . . . 7
  - 3.3.1 Open File . . . 7
  - 3.3.2 Grayscale Conversion . . . 8
  - 3.3.3 Ordered Dithering . . . 8
  - 3.3.4 Auto Level . . . 10
  - 3.3.5 Huffman Coding . . . 12
  - 3.3.6 Exit . . . 13
- 3.4 Optional Features Implementation . . . 14
  - 3.4.1 Blurring an Image . . . 14
  - 3.4.2 Sharpening an Image . . . 15
- 3.5 Image Rotation . . . 16
  - 3.5.1 Resize Image . . . 17
  - 3.5.2 Hue Saturation Adjustment . . . 18
  - 3.5.3 Saving Process . . . 20

**4 Challenges and Solutions** **20**
- 4.1 Challenges and Solutions . . . 21
- 4.2 Error Handling Strategies . . . 22

**5 Results and Discussion** **24**
- 5.1 Core Operations . . . 24
  - 5.1.1 Open File . . . 24
  - 5.1.2 Grayscale Conversion . . . 26
  - 5.1.3 Auto Level . . . 28
  - 5.1.4 Ordered Dithering . . . 34
  - 5.1.5 Huffman Coding . . . 36
- 5.2 Optional Features . . . 38

5.2.1 Blur . . . . . 38
5.2.2 Sharpen . . . . . 39
5.2.3 Image Resizing . . . . . 41
5.2.4 Image Rotation . . . . . 47
5.2.5 Hue and Saturation Adjustment . . . . . 50
5.2.6 Saving an Image . . . . . 54

**6 Conclusion** **56**

# 1 Introduction

## 1.1 Project Overview

This project serves as a practical exploration into the world of multimedia systems, focusing on the application of image processing techniques within a user-friendly graphical interface. The significance of the project lies not only in its technical challenge but also in its potential to provide a foundational tool for creative image manipulation, catering to users who require a lightweight yet powerful alternative to full-fledged image editing software.

## 1.2 Objectives

The primary objectives of the Mini-Photoshop project were to:

1. **Implement Core Image Manipulation Operations**: To create a functional application capable of performing essential image editing tasks such as opening image files, converting images to grayscale, applying ordered dithering, auto leveling images, and calculating Huffman encoding.

2. **Develop a User-Friendly GUI**: To design and implement an intuitive graphical user interface that allows users to easily navigate and utilize the application's features.

3. **Incorporate Advanced Image Processing Features**: Beyond the core requirements, to enhance the application's capabilities by adding optional advanced features like sharpness adjustment, blur effect, image rotation, hue and saturation adjustment, and the ability to save edited images.

# 2 Background

The realm of image editing software encompasses a wide range from industry giants like Adobe Photoshop to more accessible platforms such as Canva and Pixlr, each designed to offer users diverse capabilities in image manipulation through intuitive graphical interfaces. The Mini-Photoshop project taps into this rich landscape, leveraging advanced programming languages and libraries like Python, OpenCV, and Tkinter to deliver a simplified yet potent image

editing tool. This initiative aims not only to replicate basic functionalities such as photo editing and graphic design but also to make image processing accessible to a broader audience.

Embedded within the broader discourse on multimedia systems and GUI importance in contemporary software, the Mini-Photoshop project underscores the essential role of user-friendly interfaces in democratizing access to sophisticated image editing tools. As digital media continues to pervade all aspects of professional and personal communication, the demand for such tools is ever-increasing. By marrying intuitive GUI design with fundamental image processing operations, Mini-Photoshop contributes to the ongoing evolution of multimedia applications, ensuring users can efficiently manipulate and enhance digital images without extensive technical know-how.

# 3 Methodology

## 3.1 Tools and Technologies Used

The development of the Mini-Photoshop application involved a selection of programming languages and libraries chosen for their robustness, flexibility, and suitability for handling complex image processing and GUI tasks. The core technologies used in this project include:

- **Python**: Selected as the primary programming language due to its readability, extensive support libraries, and widespread use in both academic and industry settings for multimedia applications.
- **OpenCV (Open Source Computer Vision Library)**: A powerful library for computer vision and image processing tasks. In this project, OpenCV was utilized for its comprehensive set of features designed to handle image manipulations such as converting images to grayscale, applying dithering effects, and performing advanced operations like sharpness adjustment and blur effects.
- **Tkinter**: Python's standard GUI toolkit, chosen for its simplicity and effectiveness in creating graphical user interfaces. Tkinter provides a straightforward way to create windows, dialogs, buttons, and other GUI elements, making it ideal for this project's requirements.

- **NumPy**: A fundamental package for scientific computing with Python. NumPy was used extensively for its efficient handling of arrays and matrices, which are crucial for image data manipulation and processing tasks in the project.

These tools and technologies were selected not only for their specific capabilities but also for their compatibility and ability to work together seamlessly, providing a solid foundation for developing the Mini-Photoshop application.

## 3.2 GUI and Interaction

The GUI, central to the user experience, is meticulously designed to facilitate easy access to all features. The main window, crafted with Tkinter, organizes interactive elements in a clear and intuitive layout. A dedicated panel on the left side contains buttons for each core feature, allowing for straightforward operation. The image display area is dynamically adjusted to accommodate the loaded images, ensuring optimal visibility for editing and comparison. This user interface design prioritizes usability, aiming to make advanced image processing accessible to users regardless of their technical expertise.

The GUI was carefully crafted to ensure that users could easily navigate through the application and access its features without prior training. Emphasis was placed on creating a clean and organized interface that encourages exploration and experimentation with the image manipulation tools provided.

## 3.3 Core Feature Implementation

### 3.3.1 Open File

The "Open File" functionality allows users to interactively select and open image files for editing. Utilizing the Tkinter `filedialog.askopenfilename` method, the application presents a file dialog box, guiding users to navigate their file system and choose an image file. Once a file is selected, it's loaded into the application using the Pillow library, which supports handling image files. The image is then resized to fit the application window and displayed, ready for further processing. This feature is the entry point for all image editing operations, establishing the groundwork for user interaction with their images.

### 3.3.2 Grayscale Conversion

Converting color images to grayscale, a fundamental process in many image editing and processing tasks. This conversion is critical for various applications, such as simplifying image analysis, reducing computational complexity for further processing, and preparing images for certain types of visual output where color is not required or desired. We have used it in our other modules in this project for Huffman Coding and Dithering.

**Grayscale Conversion Process** The process of converting an image to grayscale involves reducing the color information in the image so that each pixel represents a shade of gray corresponding to the perceived luminance of the original colors. The specific steps in this implementation include:

- **Reading the Image**: The original image is loaded into memory using `cv2.imread`. It's important to note that OpenCV reads images in BGR (Blue, Green, Red) format by default.
- **Conversion to Grayscale**:
  - The algorithm iterates over each pixel in the image, extracting its BGR components.
  - A weighted sum of these components is calculated using the formula $Y = 0.114B + 0.587G + 0.299R$. This formula is derived from the standard definition of luminance, which takes into account the human eye's different sensitivities to various colors (it is more sensitive to green, for example).
  - The calculated luminance value $Y$ replaces the original color value, effectively converting that pixel to a shade of gray.
- **Saving the Grayscale Image**: The processed image is saved as a new file, storing the grayscale version of the original image for subsequent use or display.

### 3.3.3 Ordered Dithering

This feature illustrates an implementation of ordered dithering, a technique used to simulate grayscale images on displays that only support a limited

number of colors, typically binary (black and white). This method is particularly useful for creating the illusion of depth and detail in such restricted color spaces. The implementation specifically utilizes a Bayer matrix for the dithering process, a popular choice for its simplicity and effectiveness.

**Ordered Dithering Process** Ordered dithering works by comparing each pixel's intensity in the image against a threshold defined by a repeating pattern (in this case, the Bayer matrix). The outcome determines whether the pixel will be turned black or white in the dithered image.

- **Bayer Matrix Normalization**: The Bayer matrix used in the dithering process is normalized by dividing all its values by the maximum value in the matrix, scaling the range from 0 to 255. This step ensures that the matrix can be applied to grayscale images with pixel values in the standard 8-bit range.
- **Threshold Matrix Creation**: The normalized Bayer matrix is tiled to match the dimensions of the input image. This threshold matrix is used to compare against the original image pixel by pixel.
- **Dithering Decision**: For each pixel, if its value is greater than the corresponding threshold, it's set to white (255); otherwise, it's set to black (0). This binary decision based on the Bayer matrix pattern creates the dithered effect.
- **Combining Images**: The final step involves combining the original grayscale image and the dithered image side by side for comparison, showcasing the effect of the dithering process.

**Advantages of Using the Bayer Matrix**

- **Uniform Distribution of Error**: The Bayer matrix ensures a more uniform distribution of quantization error across the image, which can reduce the appearance of banding and provide a smoother gradient effect in the dithered image.
- **Simple Implementation**: Despite its effectiveness, the Bayer matrix involves a straightforward implementation that does not require complex computations, making it an efficient choice for real-time applications.

- **Versatility**: The Bayer matrix can be easily scaled to different sizes (e.g., 2x2, 4x4, 8x8) to accommodate varying levels of detail and contrast in the output image. This implementation uses a 4x4 matrix as a balance between detail and computational efficiency.
- **Perceptual Quality**: The pattern created by the Bayer matrix in the dithered image is less likely to interfere with the perceptual details of the original image, preserving essential visual information while reducing the color depth.

This implementation of ordered dithering in the Mini-Photoshop application demonstrates a practical application of digital image processing principles, enhancing the application's capabilities to simulate grayscale images on binary displays effectively. By leveraging the Bayer matrix, the module provides a reliable and efficient means to create visually appealing dithered images, suitable for various applications from digital art to print media.

#### 3.3.4 Auto Level

For implementing the Auto Levelling, Histogram Stretching is implemented. This technique is designed to improve the contrast in an image by adjusting the intensity range of its histogram to span the entire available scale. Here's a generalized overview of how this operation is performed and its significance in image processing:

**Auto-Leveling for Color Images** Auto-leveling, or histogram stretching, is a fundamental image enhancement technique that redistributes the pixel intensity values of an image. The primary goal is to expand the dynamic range of the intensities, particularly when the original image's histogram is narrowly concentrated in a specific intensity range.

- **Process Overview**:
    1. **Histogram Plotting (Before)**: Initially, the histogram of the original image for each color channel (Blue, Green, Red) is plotted using `matplotlib.pyplot`. This step visualizes the distribution of pixel intensities before the auto-leveling process.
    2. **Reading the Image**: The input image is loaded, and its color channels are separated using `cv2.split`.

3. **Histogram Stretching**: For each channel, the minimum and maximum intensity values are identified. The pixel intensities are then linearly transformed so that the minimum and maximum values map to 0 and 255, respectively. This transformation stretches the histogram to cover the entire intensity range, effectively enhancing the contrast of the image.
4. **Recombination and Saving**: The stretched channels are merged back into a single image, which is then saved to disk. This output image exhibits enhanced contrast and visibility.
5. **Histogram Plotting (After)**: Finally, the histogram of the enhanced image is plotted for each color channel, demonstrating the effect of histogram stretching.

**Significance** Auto-leveling is particularly useful in scenarios where images are captured under less-than-ideal lighting conditions, resulting in low contrast. By expanding the range of intensities, details that were previously obscured in darker or lighter regions become more visible. This technique does not alter the content or structure of the image but rather improves its visual quality, making it easier to interpret or analyze.

**Advantages of This Implementation**

- **Channel-wise Processing**: The method processes each color channel independently, ensuring that the color balance of the image is preserved while enhancing contrast.
- **Visualization**: By plotting histograms before and after the operation, users can visually assess the impact of auto-leveling on the image's intensity distribution.
- **Flexibility**: The implementation is flexible enough to handle images where no enhancement is necessary (i.e., when the histogram already spans the full intensity range), leaving such images unchanged.

This approach to auto-leveling in color images underscores the power of simple yet effective image processing techniques to significantly enhance the visual quality of digital images. By employing histogram stretching, the module contributes to the broader goal of the Mini-Photoshop application:

to provide users with intuitive and effective tools for image manipulation and enhancement.

### 3.3.5 Huffman Coding

This feature demonstrates the principles of entropy calculation and Huffman coding for image compression within the context of the Mini-Photoshop application. This module not only calculates the entropy of an image—a measure of the randomness or unpredictability in the image data—but also constructs a Huffman tree to determine the average length of the Huffman codes for the image's pixels. Here's how the implementation is generalized:

**Entropy Calculation** Entropy is a statistical measure of randomness that can be used to characterize the texture of an image. In the context of image processing, it is used to quantify the amount of information contained in an image.

- **Process Overview**: The `calculate_entropy` function computes the entropy of an image by first determining the frequency of each pixel value. It then calculates the probabilities of these pixel values by dividing their frequencies by the total number of pixels. The entropy is calculated using the formula:

  $$Entropy = -\sum(p(x)\log_2 p(x))$$

  where $p(x)$ is the probability of occurrence of pixel value $x$. This gives a measure of the information content of the image.

**Huffman Tree Construction** Huffman coding is a method of lossless data compression that is widely used for creating compressed representations of data (e.g., for file compression or video encoding). The Huffman coding process involves building a Huffman tree, where the most frequent pieces of data are assigned the shortest codes and less frequent data are assigned longer codes.

- **Process Overview**: The `huffman_tree` function constructs the Huffman tree using a priority queue (min-heap), where each node contains a symbol and its frequency. The tree is built by repeatedly removing the two nodes of lowest frequency from the queue, creating a new node

as their parent with a frequency equal to the sum of their frequencies. This new node is then added back into the queue. This process is repeated until there is only one node left, which becomes the root of the Huffman tree. The function then assigns binary codes to each symbol based on their position in the tree, with '0' and '1' representing left and right branches, respectively.

**Main Function Workflow** The `huffman_main` function serves as the entry point for the entropy and Huffman coding calculation.

1. **Grayscale Conversion**: Initially, the image is converted to grayscale using the `grayscale` module. This step simplifies the data, focusing the entropy and Huffman coding calculation on luminance rather than color information.

2. **Entropy Calculation**: The entropy of the grayscale image is calculated to assess its information content.

3. **Huffman Coding**: The frequency of occurrence of each grayscale level in the image is computed, and a Huffman tree is constructed. The function then calculates the average Huffman code length by weighting the length of each symbol's code by its frequency of occurrence.

4. **Output**: The entropy and average Huffman code length are printed out, offering insights into the image's compressibility and complexity.

**Advantages** This implementation showcases foundational concepts in information theory and coding, specifically applied to the context of image processing. Calculating the entropy provides a measure of the image's information content, while Huffman coding demonstrates a practical application of these principles in data compression.

#### 3.3.6 Exit

The "Exit" feature in this application provides users with the ability to safely close the application. This function is crucial for ensuring that the application terminates correctly, without leaving any processes running in the background. Additionally, it integrates a cleanup process that removes

temporary files created during the application's use, maintaining the user's system cleanliness and efficiency.

Upon selecting "Exit," the application triggers the `exit_program` function. This function first calls `root.quit()` and `root.destroy()` methods provided by Tkinter to close the GUI window and then executes `root.quit()` again to ensure that the main event loop is terminated, effectively shutting down the application. This multi-step approach ensures that the application closes smoothly and without error.

To complement the exit process, the application employs an `atexit` register, which calls the `delete_images` function upon program termination. The `delete_images` function scans the 'output_images' directory, which is used to store temporary image files created during the application's runtime, such as processed images awaiting to be saved permanently by the user. Each file within this directory is removed, ensuring that no residual data is left behind. This cleanup process is crucial for preserving disk space and preventing the accumulation of unnecessary files on the user's system.

By incorporating these mechanisms, the "Exit" feature not only provides a straightforward way for users to close the application but also upholds best practices in software design by ensuring resource cleanliness and application stability.

## 3.4 Optional Features Implementation

### 3.4.1 Blurring an Image

Blurring is achieved by averaging the pixel values in the vicinity of each target pixel. This process tends to reduce image noise and detail, resulting in a softer appearance. The Gaussian filter is a low-pass filter that is used to blur the image. The filter is applied to each pixel in the image to create a blurred version of the image.The filter size is 49x49 and the padding is done using the reflect mode.

#### Process Overview

1. **Reading the Image**: The original image is loaded into memory.

2. **Padding**: To ensure the blurring operation affects the entire image uniformly, including edges, the image is padded. This means extra rows

and columns are temporarily added around the edges, often mirroring the edge pixels.

3. **Applying the Blur**: For each pixel, a 49X49 area around it (including the pixel itself) is considered. The average color value of these 25 pixels is calculated and assigned to the target pixel, effectively blurring the image.

4. **Output**: The blurred image is saved, illustrating the blurring effect.

### 3.4.2 Sharpening an Image

Sharpening is conducted via the unsharp mask technique, where a blurred copy of the original image is subtracted from the original to create a mask highlighting edges and details. Adding this mask back to the original image enhances its sharpness.

**Process Overview**

1. **Creating a Blurred Copy**: Initially, a blurred version of the original image is created using the aforementioned blurring technique.

2. **Generating the Mask**: The mask is produced by subtracting the blurred image from the original. This mask represents the high-frequency components (edges) of the original image.

3. **Enhancing the Original**: By adding the mask back to the original image, edges and details are accentuated, making the image appear sharper.

4. **Output**: The sharpened image is saved, showcasing the enhanced edges and textures.

This approach to blurring and sharpening images manually, without reliance on high-level library functions, demonstrates a fundamental understanding of image processing techniques. It allows for deeper insights into how images can be manipulated at a pixel level, offering a foundational perspective on digital image editing.

## 3.5 Image Rotation

Here implementation designed for rotating images by 90 degrees clockwise without utilizing high-level OpenCV functions for rotation. Instead, it applies a fundamental understanding of image processing techniques, specifically matrix transformations, to accomplish the task. Here's an overview of how this implementation works:

- **Import Necessary Libraries**: The script begins by importing `cv2` for basic image handling, `os` for file path operations, and `numpy` for numerical calculations, which are essential for the image rotation process.
- **Rotation Function**: The core of the script is the `rotate(original_image_path)` function, which takes the path of an original image as input.
- **Determine Output Path**: It checks if a rotated image already exists in the `output_images` directory. If it does, that image is used; otherwise, it proceeds with the original image.
- **Image Loading**: The script attempts to load the image using `cv2.imread`. If the image cannot be loaded, it exits with an error message.
- **Calculate New Dimensions**: To accommodate the rotated image without clipping, it calculates the new image dimensions. These dimensions ensure that the rotated image can fit entirely within the new canvas size, taking into account the rotation's effect on the image's width and height.
- **Create a New Image Canvas**: A new, empty image canvas is initialized with the calculated dimensions. This canvas will hold the rotated image.
- **Calculate Centers**: The centers of the original and new images are calculated to help map coordinates from the original image to the rotated image.
- **Pixel Mapping**: For each pixel in the new canvas, the script calculates the corresponding pixel position in the original image using rotation matrices. This involves translating the pixel coordinates to rotate around the image center and then mapping these coordinates back to the original image's space.

- **Copy Pixels**: If the calculated original coordinates are within the bounds of the original image, the pixel value is copied to the corresponding position in the new canvas. This step effectively rotates the image.
- **Save Rotated Image**: The rotated image is saved to a specified path using `cv2.imwrite`.

This approach highlights the intricate process of manually handling image rotation, offering insights into the underlying principles of digital image processing and matrix manipulation.

### 3.5.1 Resize Image

The resize module implemented provides a manual approach to resizing an image to specified dimensions, leveraging fundamental principles of digital image processing without relying on high-level OpenCV resizing functions. This process, known as image resampling, is crucial for various applications, including web optimization, image analysis, and adapting visuals to different screen sizes. Here's a generalized explanation of the resizing process implemented in this module:

**Overview of Manual Image Resizing**

1. **Reading the Original Image**: The process initiates by loading the image from a specified path using OpenCV's `cv2.imread`. Ensuring the image is successfully loaded is critical before proceeding with the resizing operation.
2. **Calculating Scaling Factors**: The core of resizing lies in determining how much each dimension of the image needs to be scaled. This is done by dividing the desired dimensions by the original dimensions, yielding scaling factors for both width and height. These factors dictate how the coordinates in the original image map to the new, resized image.
3. **Creating the Resized Image**: A new image array is initialized with zeros to match the desired dimensions. This array will be populated with pixel values from the original image but adjusted according to the scaling factors.

4. **Mapping Pixels**: For each pixel in the target (resized) image, the corresponding pixel position in the original image is calculated using the scaling factors. This step involves a nearest neighbor approach, where the closest pixel in the original image is selected to represent each pixel in the resized image.

5. **Handling Edge Cases**: To ensure that the mapped coordinates do not exceed the boundaries of the original image, the calculated source coordinates are clamped to the maximum allowable values. This precaution prevents indexing errors that could occur due to rounding during the scaling calculation.

6. **Saving the Resized Image**: Once all pixels in the new image have been assigned values from the original image, the resized image is saved to a specified path using `cv2.imwrite`. This step finalizes the resizing process, making the altered image available for further use or analysis.

### 3.5.2 Hue Saturation Adjustment

The `adjust_hue_saturation.py` and `hue_saturation_window.py` modules work together to provide an interactive interface for adjusting the hue and saturation of an image within the Mini-Photoshop application. These adjustments allow users to modify the color tone and intensity of their images, offering a broad range of creative possibilities. Here's a generalized overview of how these features are implemented and function:

#### Hue and Saturation Adjustment Process

1. **Conversion to HSV**: The core of the hue and saturation adjustment lies in converting the image from the BGR color space (used by OpenCV) to HSV (Hue, Saturation, Value). This color space conversion facilitates easier manipulation of color properties.

2. **Adjustment of Hue and Saturation**:

    - **Hue Adjustment**: The hue component (H channel) of each pixel in the HSV image is modified based on a user-specified shift value. This shift can change the overall color tone of the image.

- **Saturation Adjustment**: The saturation component (S channel) is altered by a specified amount, increasing or decreasing the intensity of colors. Saturation is clipped to ensure values remain within the valid range of 0 to 255.

3. **Conversion Back to BGR**: After adjustments, the image is converted back to the BGR color space for display and saving.

4. **Saving the Adjusted Image**: The modified image is saved to a specified path, making the changes persistent.

**Interactive GUI for Adjustments** The `hue_saturation_window.py` module implements a GUI window for interactive hue and saturation adjustments using Tkinter. It allows users to visualize changes in real time and provides a more intuitive and user-friendly approach to image editing.

- **Sliders for Adjustments**: Two sliders are provided for users to adjust the hue and saturation values dynamically. These sliders send the updated values to the adjustment function, which applies the changes to the image.

- **Image Display**: The GUI displays both the original and adjusted images, enabling users to compare the before and after effects of their adjustments directly.

- **Immediate Feedback**: Changes to hue and saturation are applied in real-time, allowing users to experiment with different settings and immediately see the results.

**Advantages of This Implementation**

- **User-Friendly Interface**: The GUI makes it easy for users, regardless of their technical expertise, to modify the hue and saturation of images. The immediate visual feedback helps in achieving the desired look more efficiently.

- **Flexibility in Image Editing**: By adjusting hue and saturation, users can correct color imbalances, enhance specific tones, or creatively alter the mood and atmosphere of their images.

- **Direct Control Over Color Properties**: Operating in the HSV color space allows for more intuitive control over color adjustments compared to direct manipulations in the RGB space.

This combination of backend processing for hue and saturation adjustments with an interactive frontend interface exemplifies a practical application of image processing techniques. It enhances the Mini-Photoshop application's capabilities, providing users with powerful tools for creative image manipulation.

### 3.5.3 Saving Process

The `save_image.py` module is designed with a straightforward purpose: to save an edited image to a specified destination. This functionality is essential in any image processing or editing software, allowing users to preserve their modifications and use the edited images outside the application. Here's a generalized overview of how the saving process is implemented in this context:

**Overview of the Saving Process**

1. **Load the Edited Image**: The module begins by loading the edited image from its temporary storage location. This step is typically done using a function like `cv2.imread`, which reads the image into memory as an array of pixel values.

2. **Specify Destination**: The function `save_file` accepts two parameters: the path to the temporary file (`output_file`) and the destination path (`destination_filepath`) where the user wants to save the edited image. The destination path includes both the directory and the filename, providing flexibility in naming and organizing saved files.

3. **Save the Image**: Using the `cv2.imwrite` function, the image is written to the specified destination. This step effectively copies the edited image from its temporary location to a more permanent location chosen by the user, in the specified file format.

# 4 Challenges and Solutions

Throughout the development of the Mini-Photoshop application, several challenges were encountered, primarily in implementing the functionality without

relying on OpenCV's built-in functions and ensuring a robust, user-friendly GUI. Here's a look at these challenges and the strategies and error handling measures adopted to address them:

## 4.1 Challenges and Solutions

1. **Manual Implementation of Image Processing Functions**:
    - *Challenge*: Developing custom functions for tasks like resizing, rotation, and hue/saturation adjustments without relying on OpenCV's built-in functions required a deep understanding of image processing algorithms.
    - *Solution*: By breaking down each task into fundamental steps (e.g., calculating scaling factors for resizing, applying matrix transformations for rotation), and utilizing basic NumPy operations, the functionality was replicated manually. This approach also involved extensive testing to ensure accuracy and performance.
2. **Memory Leak**:
    - *Challenge*: Upon pressing the exit button the window was closing but the process was still running in the backend
    - *Solution*: Graceful exit of the UI and the process through root.quit() and sys.exit()
3. **GUI Responsiveness and Usability**:
    - *Challenge*: Ensuring the GUI remains responsive during image processing operations and intuitive for users to navigate.
    - *Solution*: Operations that could potentially take longer to execute were optimized for efficiency. Moreover, feedback mechanisms (e.g., status messages) were incorporated to keep the user informed about ongoing processes. The layout was carefully designed to be intuitive, with clear labels and organized controls.

## 4.2 Error Handling Strategies

Throughout development, various challenges were encountered, particularly in error handling and ensuring a smooth user experience. Below is a description of the error handling mechanisms used across the application:

1. **Validating Image Load Operations**: Before proceeding with any image processing operation, the script checks whether the image was successfully loaded. If `cv2.imread` fails to load an image, functions like `rotate_image`, `grayscale_image`, and others display an error message urging the user to open an image first. This prevents the application from attempting to process a non-existent image, which could lead to errors.

2. **Handling File Dialog Operations**: Functions that involve saving (`save_image`) and opening (`open_image`) files use file dialog boxes. The application checks if the user has selected a file (or specified a save path) before proceeding. If the user cancels the operation, the application safely returns without attempting further actions, preventing unnecessary errors.

3. **Ensuring Valid User Input**: The `resize_image_gui` function invokes a custom dialog box for the user to input new dimensions for image resizing. It performs checks on the provided values to ensure they are within reasonable limits and displays an error message if invalid input is detected. This prevents the application from attempting to resize the image to nonsensical dimensions.

4. **Managing Temporary Files**: The application uses a designated directory (`output_images`) for storing temporary files generated during image processing operations. The `atexit.register(delete_images)` call ensures that all temporary files are cleaned up when the application exits, regardless of the exit method. This prevents the accumulation of unused files, helping maintain system cleanliness.

5. **Resource Path Handling for PyInstaller**: The `resource_path` function is designed to handle resource paths in both development and bundled (PyInstaller) environments. This ensures that resources like the application icon can be correctly accessed after the application

is compiled into an executable, preventing errors related to missing resources.

**Generalized Error Handling Considerations**

- Feedback for User Actions: The application consistently provides feedback for user actions, especially in scenarios where operations cannot be completed due to missing inputs or errors. This helps in maintaining a transparent communication channel between the application and the user.
- Graceful Handling of Exceptional Scenarios: By checking conditions and handling exceptions where operations might fail (such as file I/O actions or invalid user inputs), the application enhances its robustness and reliability.
- Cleanup of Resources: The proactive cleanup of temporary files and resources upon application exit minimizes potential issues related to disk space usage and ensures that the application's environment remains tidy.

Through these error handling mechanisms, the Mini-Photoshop application aims to provide a stable and user-friendly experience, reducing the likelihood of crashes or unexpected behavior due to unhandled errors. “

# 5 Results and Discussion

## 5.1 Core Operations

### 5.1.1 Open File

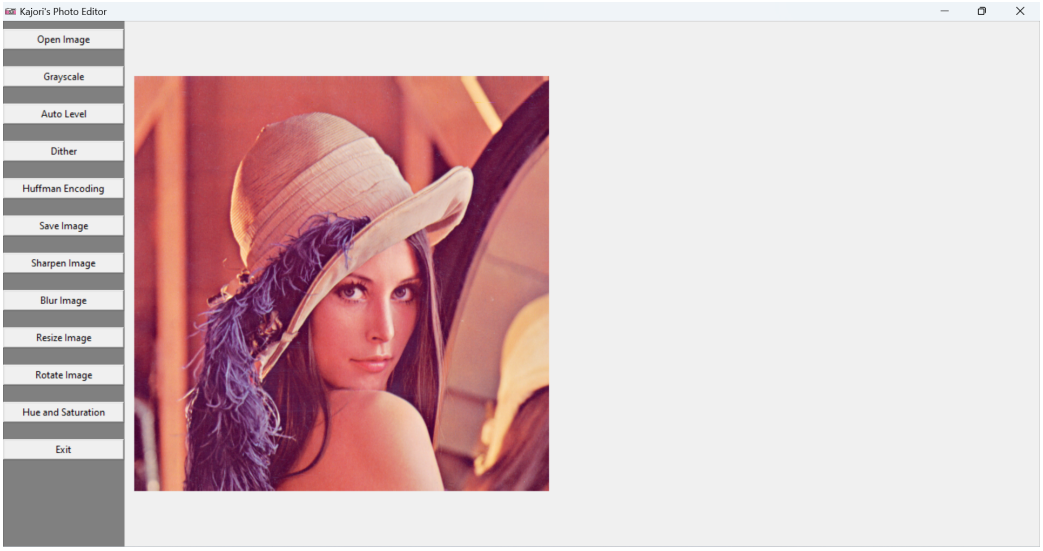


Figure 1: Open image for the first example image

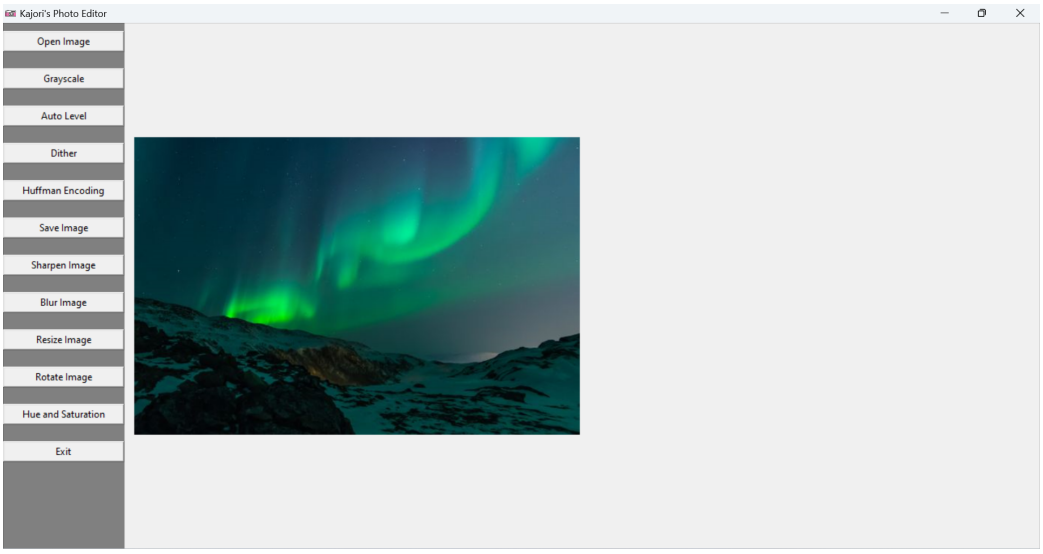


Figure 2: Open image for the second example image


Figure 3: Error when the image file is too large

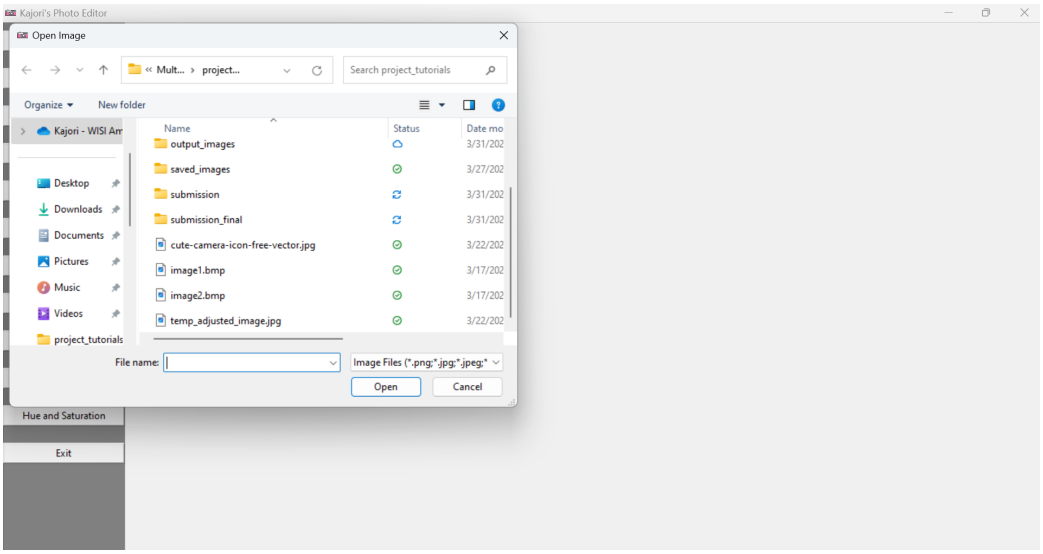


Figure 4: Error handling
This demonstrates the error handling mechanism where an already opened image disappears upon opening a new image.

### 5.1.2 Grayscale Conversion

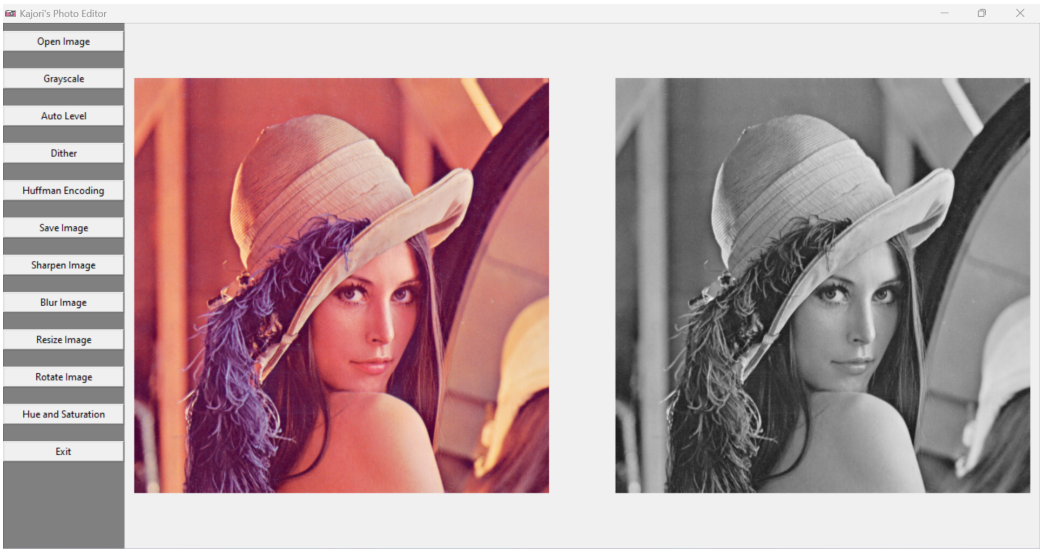


Figure 5: Gray scale Conversion for the first example image

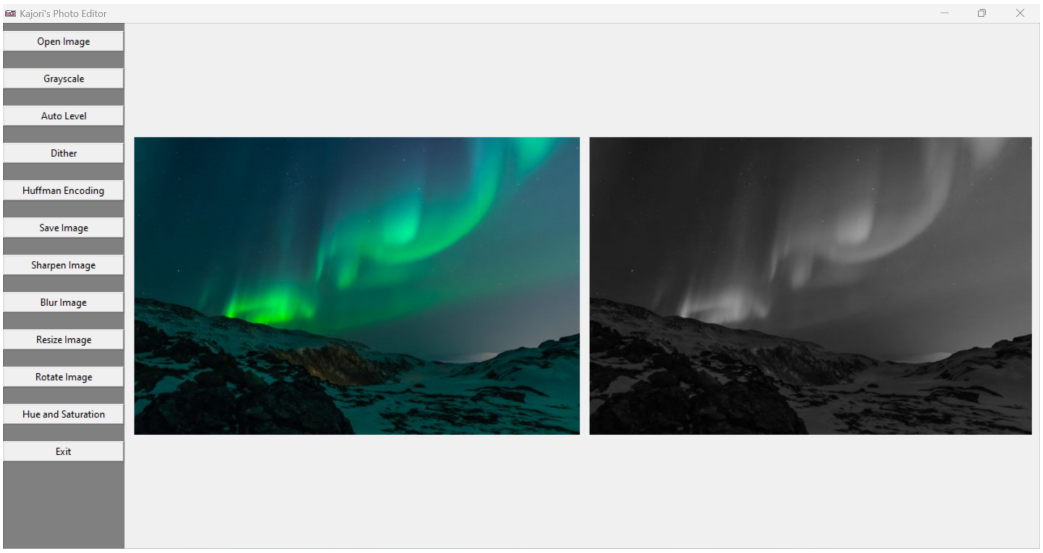


Figure 6: Gray scale Conversion for the second example image

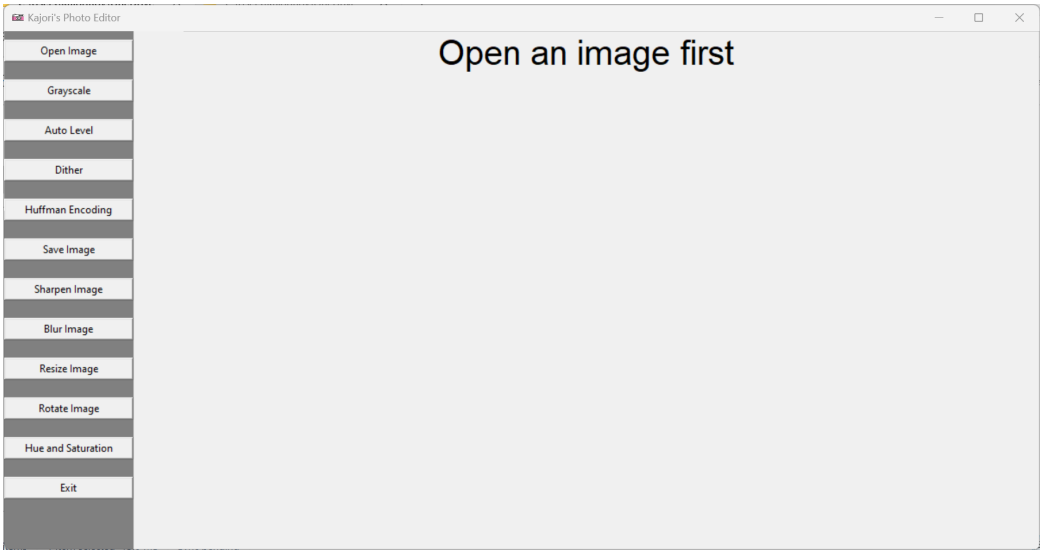


Figure 7: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Grayscale button when no image is opened

### 5.1.3 Auto Level

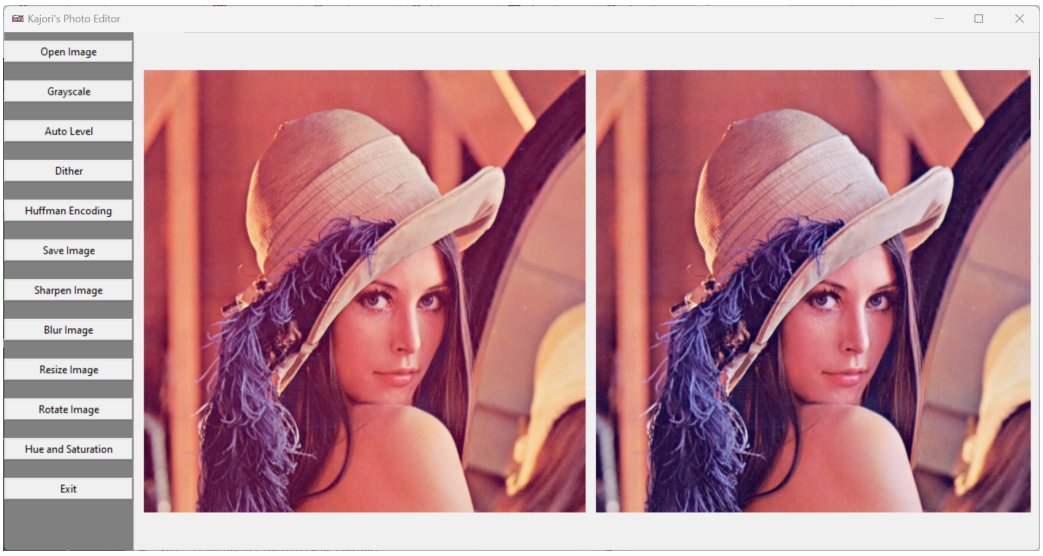


Figure 8: Auto Leveling - Histogram Stretching for the first example image

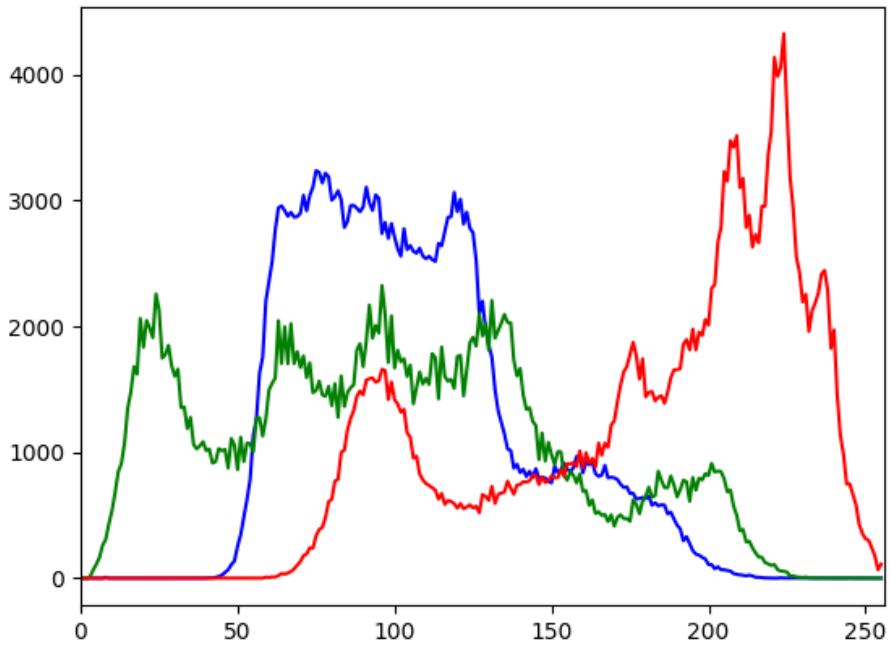


Figure 9: Original Histogram for the first example image

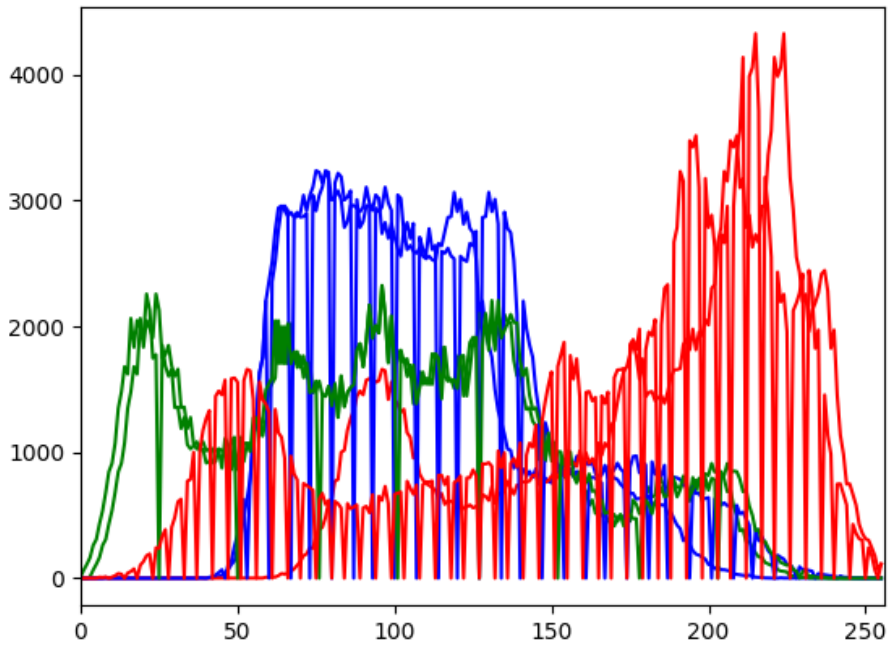


Figure 10: Histogram Stretching for the first example image

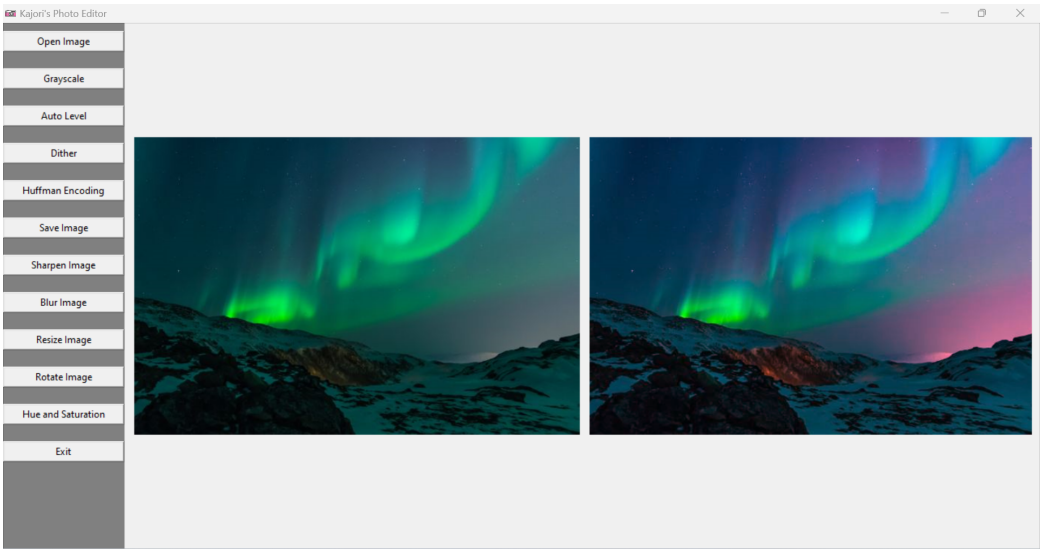


Figure 11: Auto Leveling - Histogram Stretching for the second example image

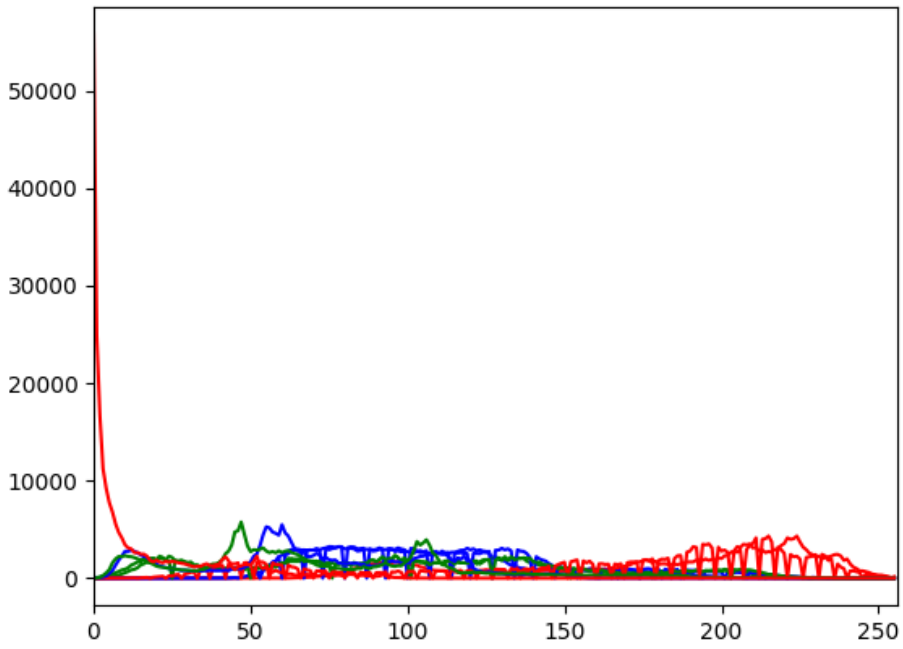


Figure 12: Original Histogram Stretching for the second example image

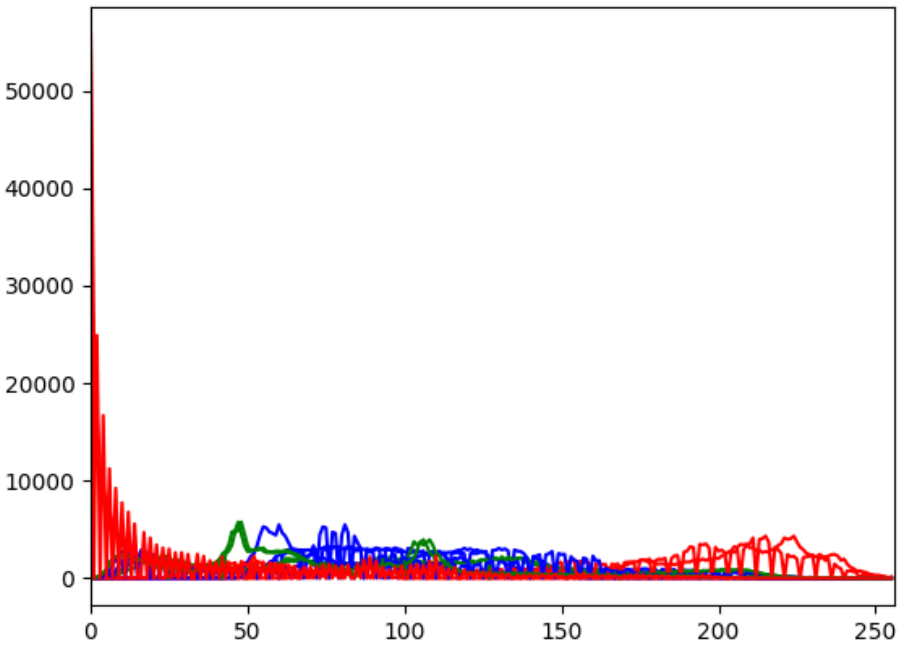


Figure 13: Histogram Stretching for the second example image

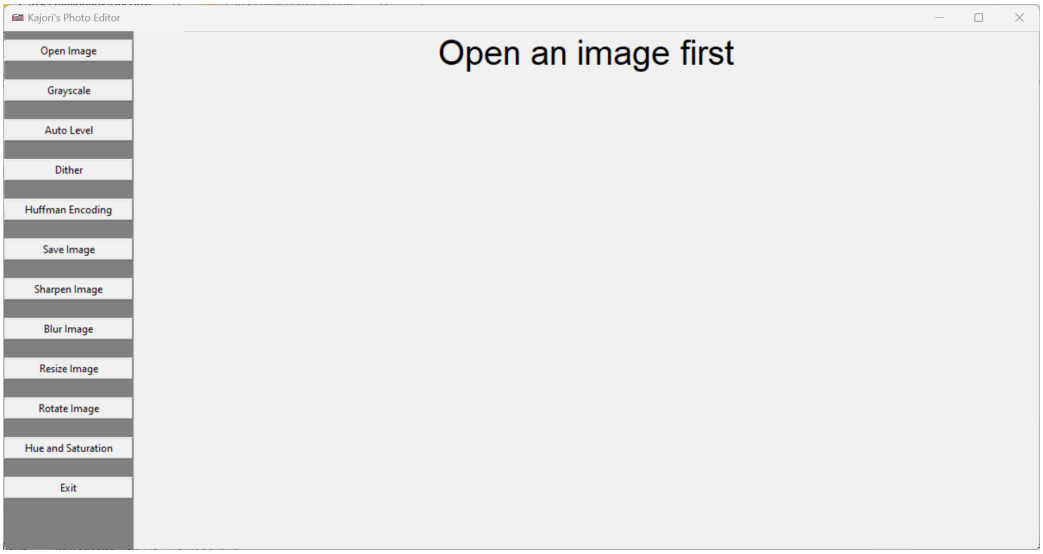


Figure 14: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Auto Level button when no image is opened.

### 5.1.4 Ordered Dithering

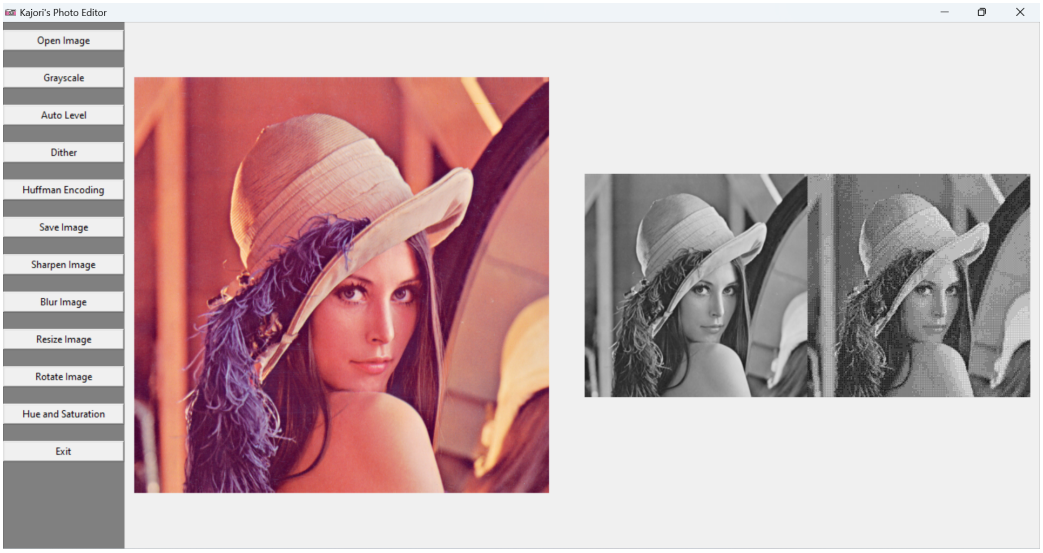


Figure 15: Dithering for the first example image

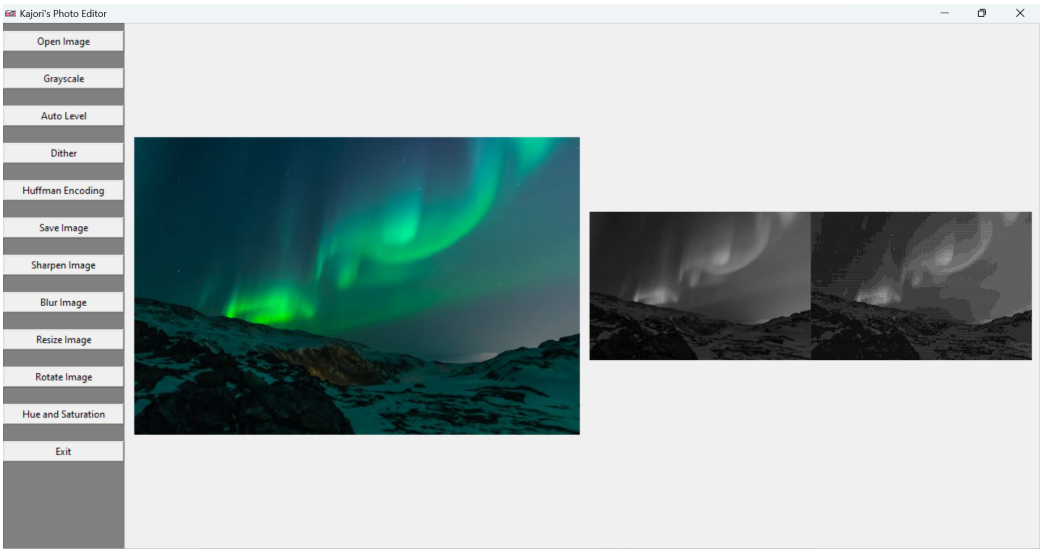


Figure 16: Dithering for the second example image

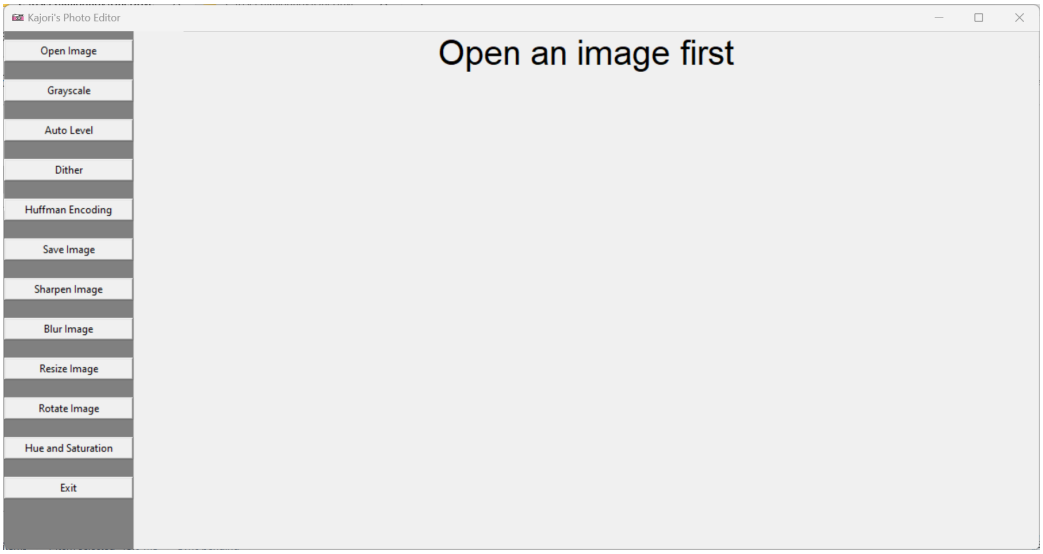


Figure 17: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Dither button when no image is opened

### 5.1.5 Huffman Coding

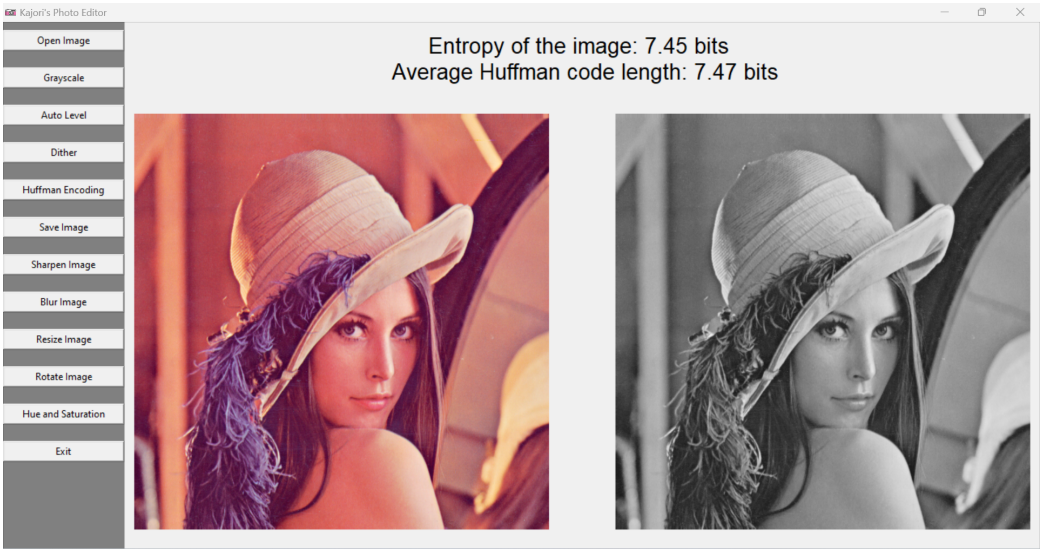


Figure 18: Huffman coding for the first example image

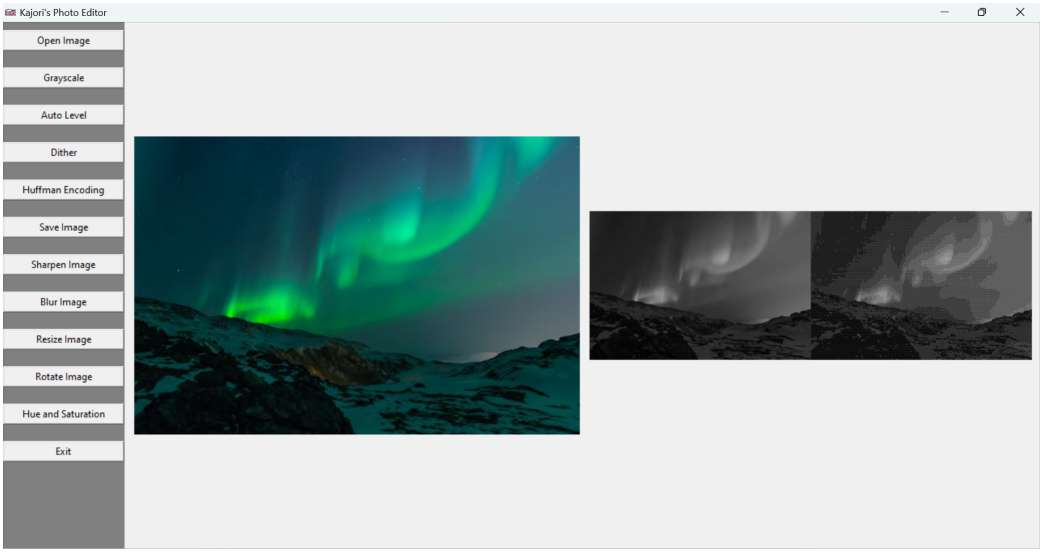


Figure 19: Huffman coding for the second example image

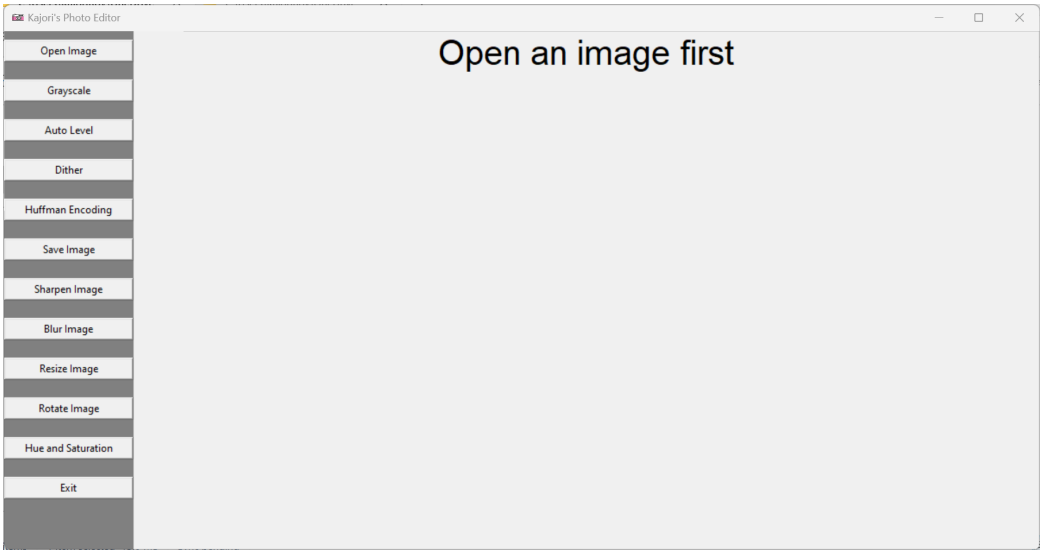


Figure 20: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Huffman coding button when no image is opened

## 5.2 Optional Features

### 5.2.1 Blur

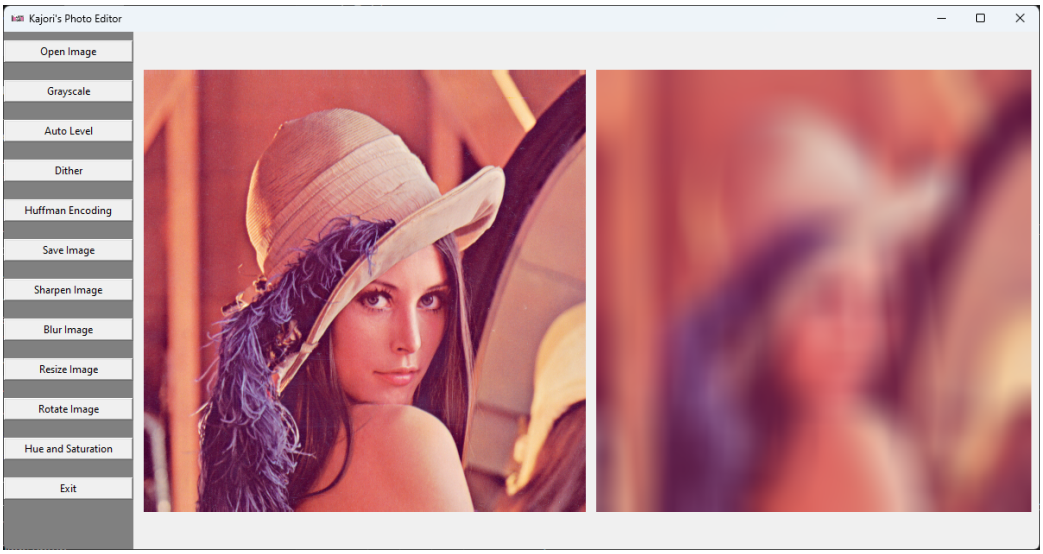


Figure 21: Blurring for the first example image

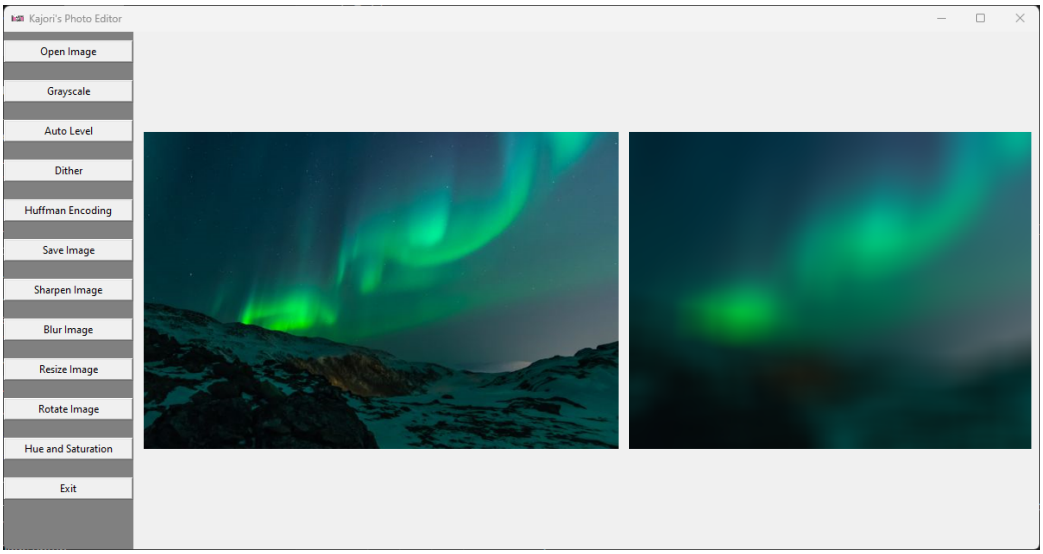


Figure 22: Blurring for the second example image

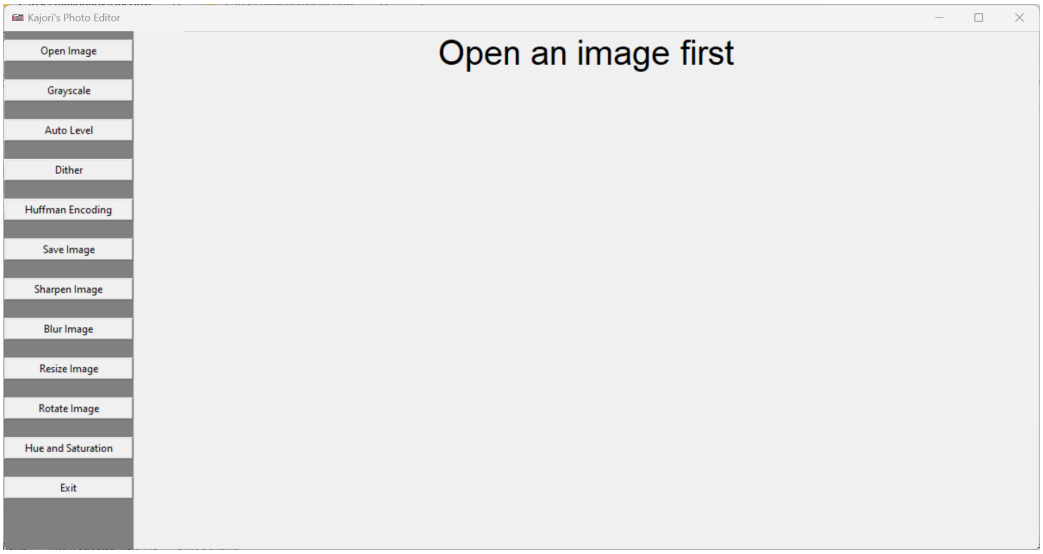


Figure 23: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Blur button when no image is opened

### 5.2.2 Sharpen


Figure 24: Sharpening for the first example image

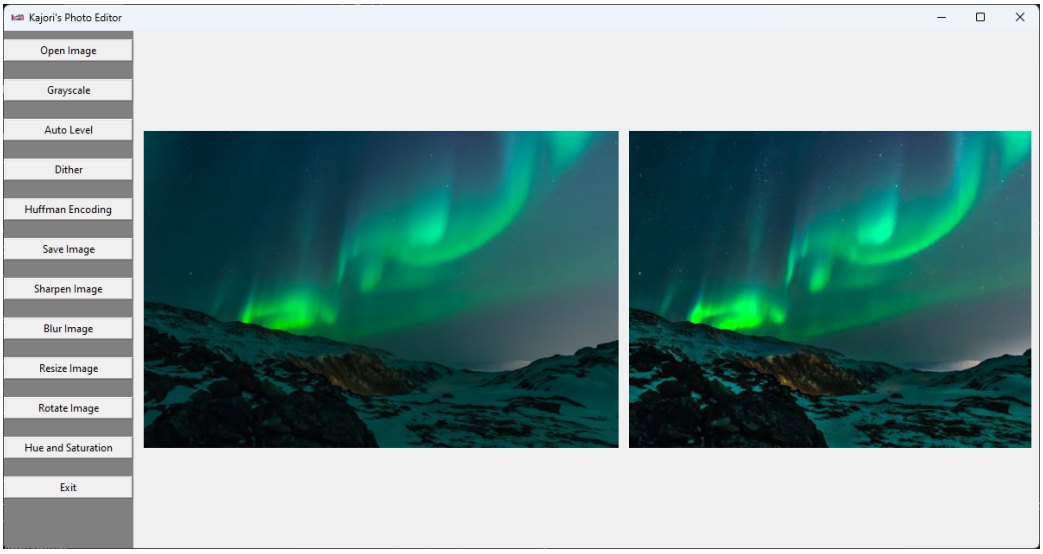


Figure 25: Sharpening for the second example image

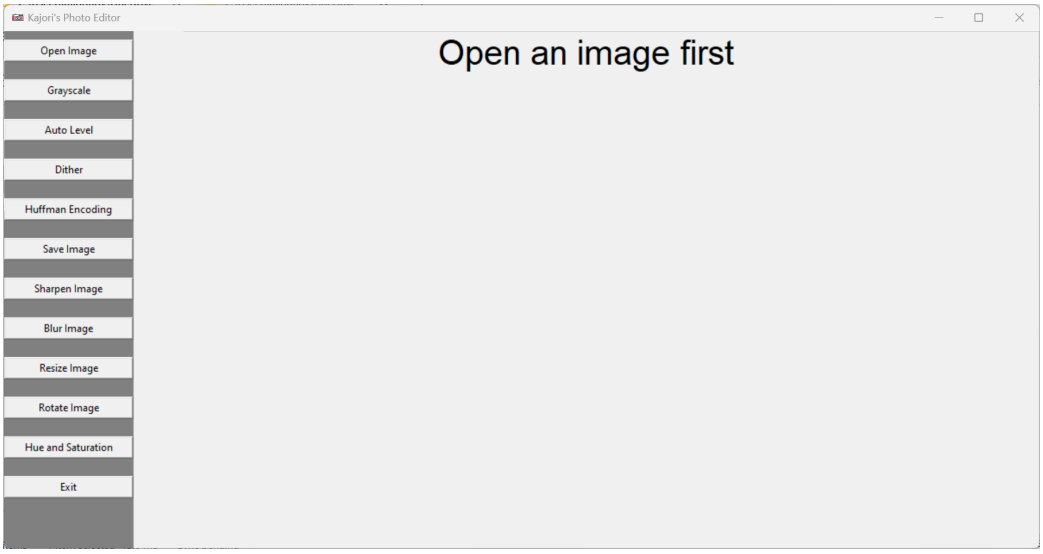


Figure 26: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Sharpen button when no image is opened

### 5.2.3 Image Resizing

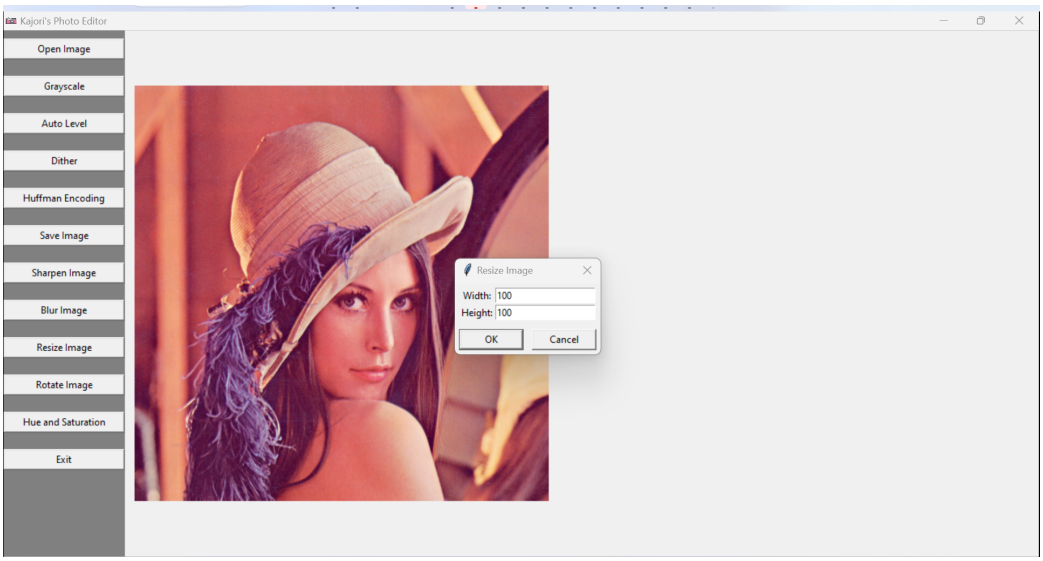


Figure 27: Resizing the first example image : 100X100

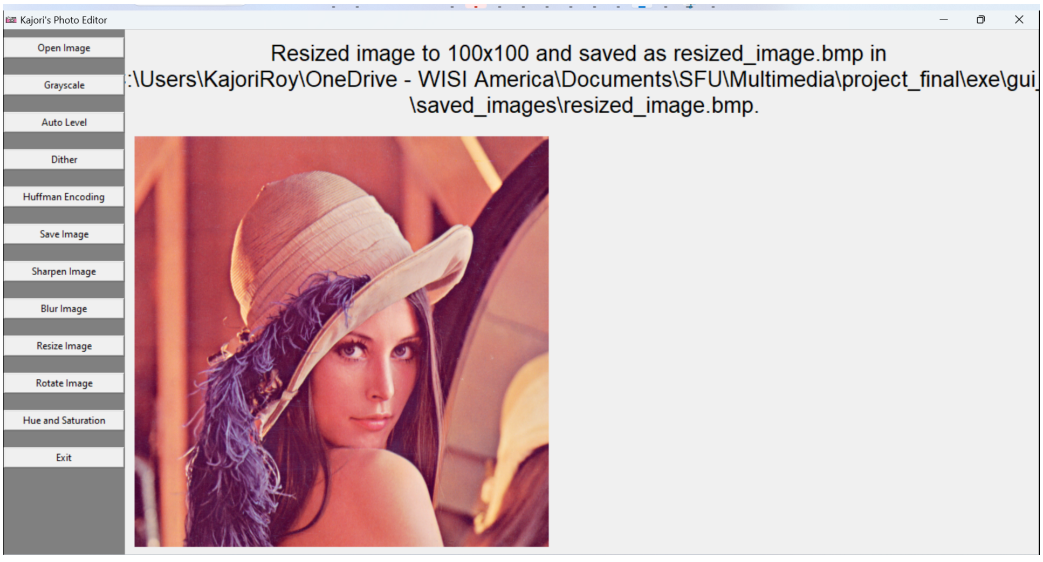


Figure 28: UI Output after resizing the first example image

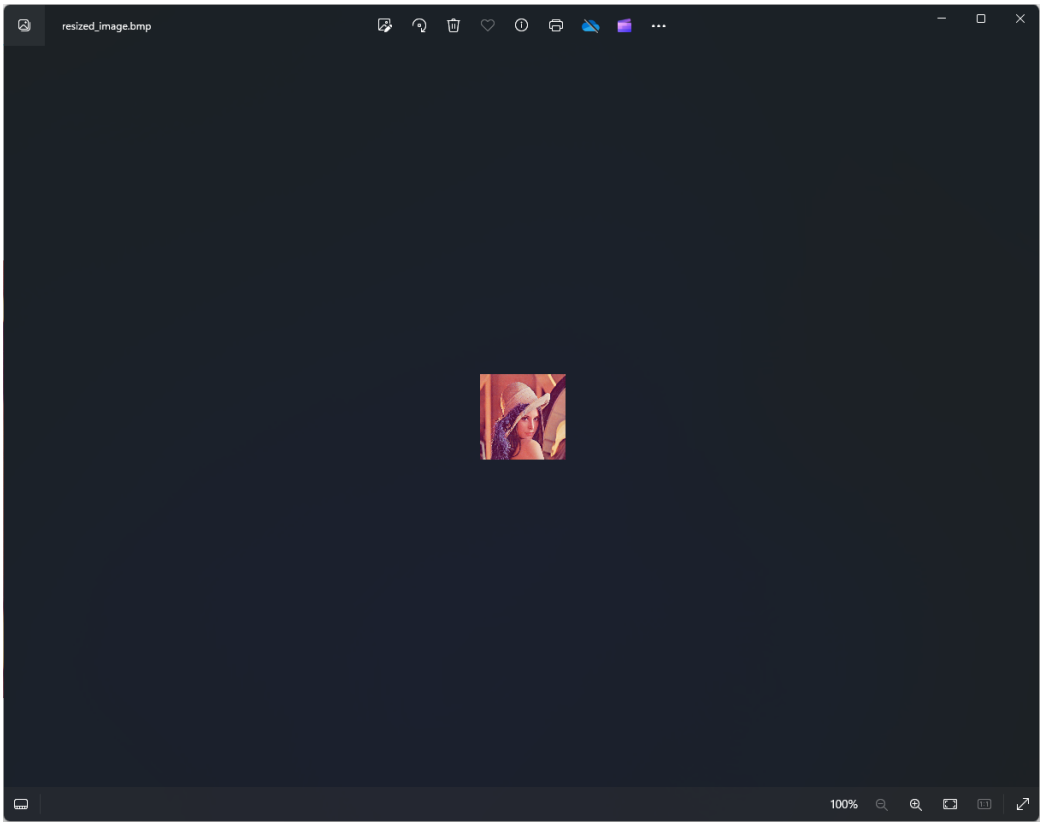


Figure 29: Image after resizing

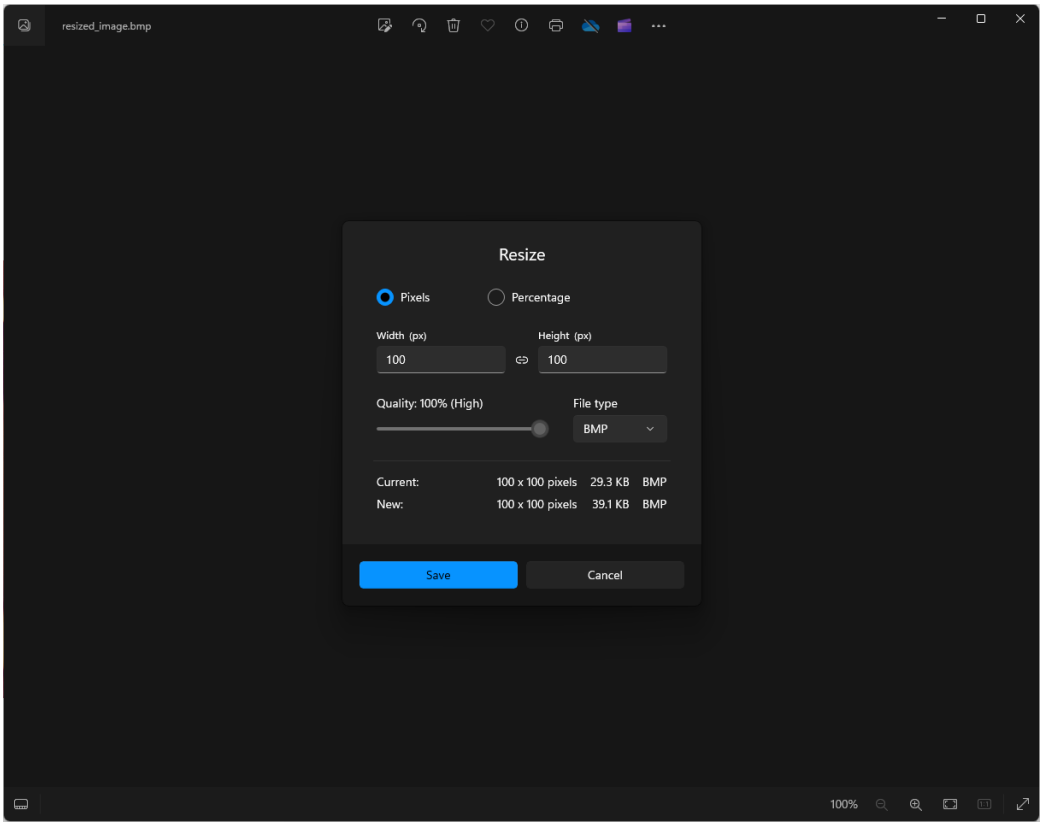


Figure 30: Size of the image after resizing
*We can see that it is 100X100 now*

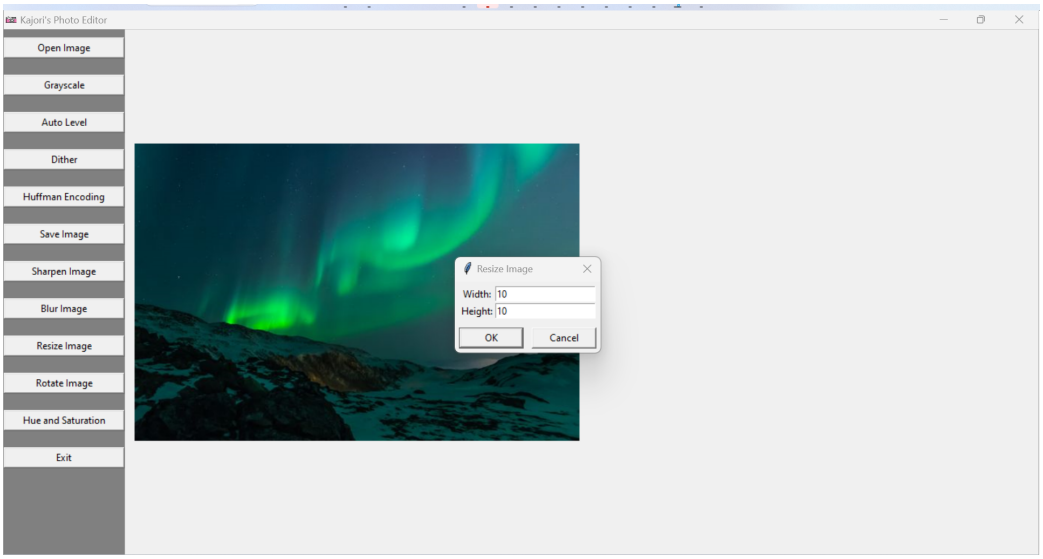


Figure 31: Resizing the second example image: 10X10

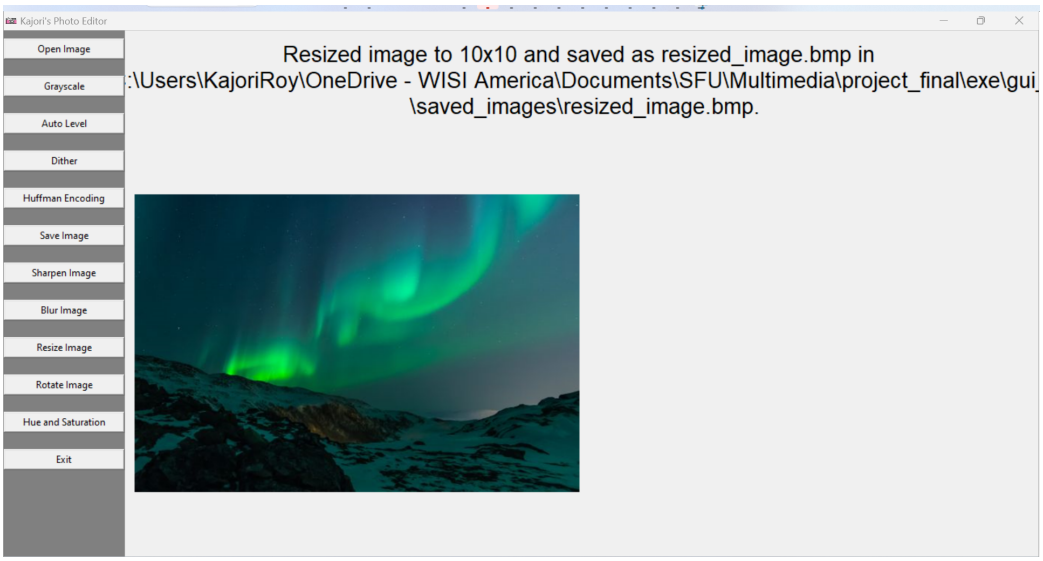


Figure 32: UI Output after resizing the second example image

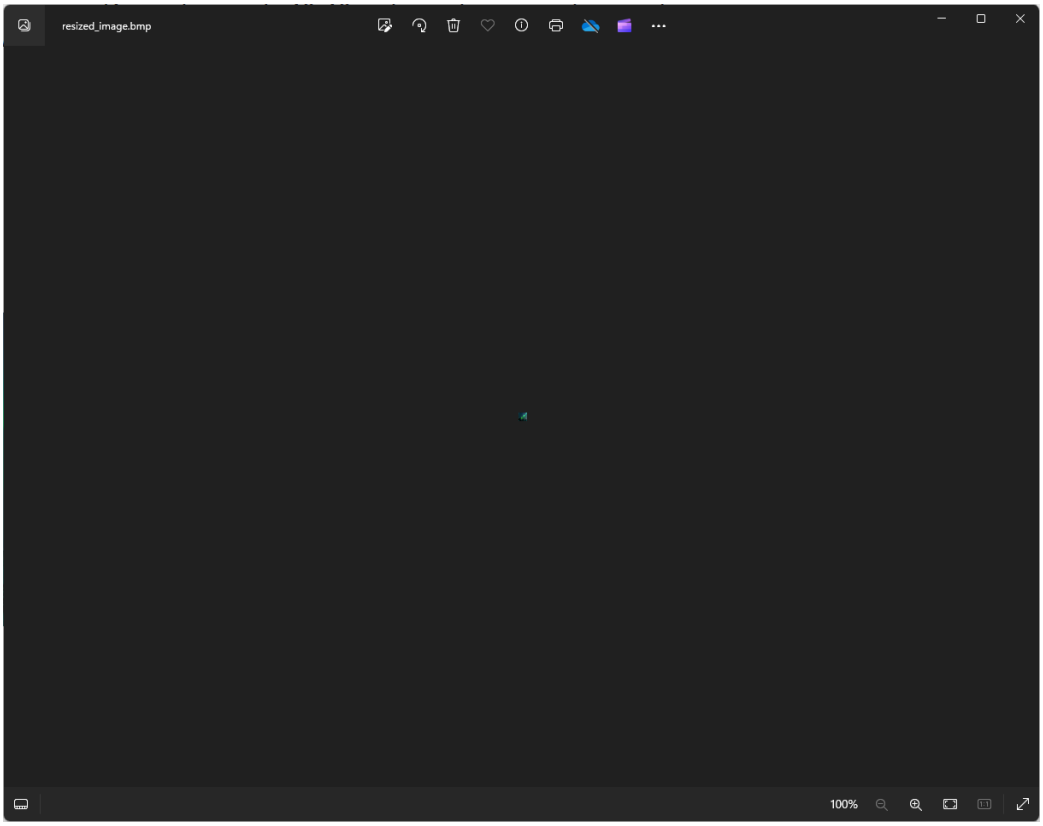


Figure 33: Image after resizing

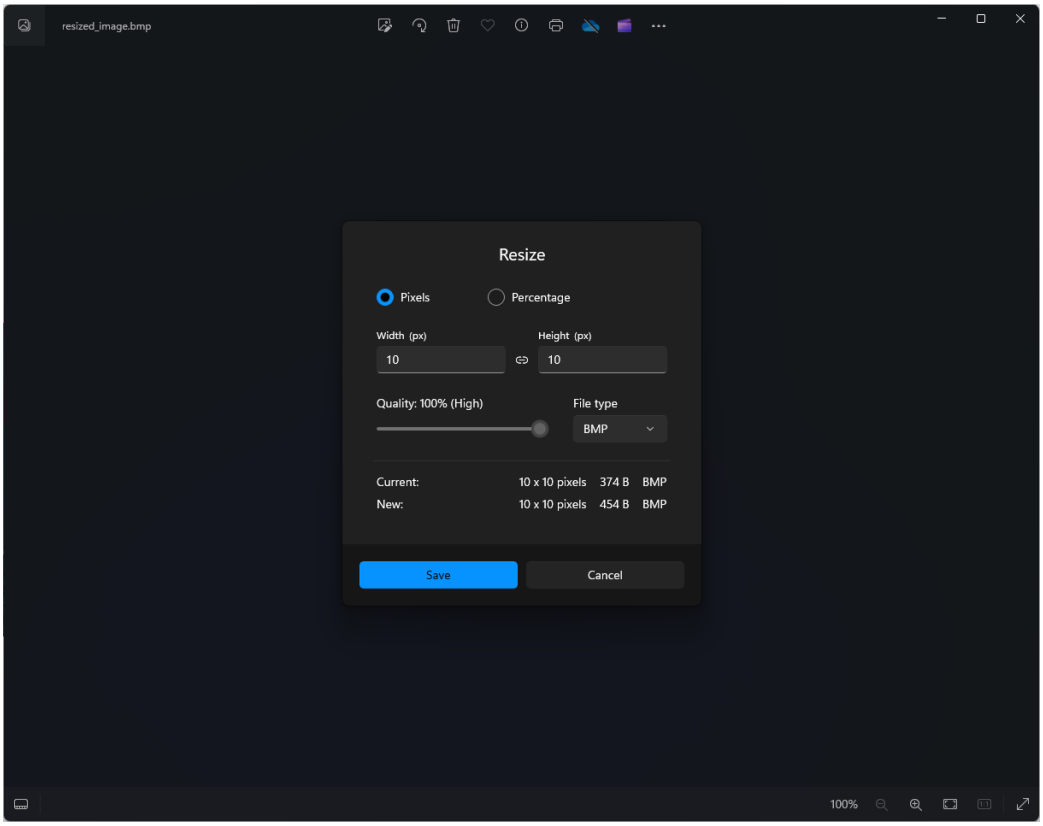


Figure 34: Size of the image after resizing
*We can see that it is 10X10 now*

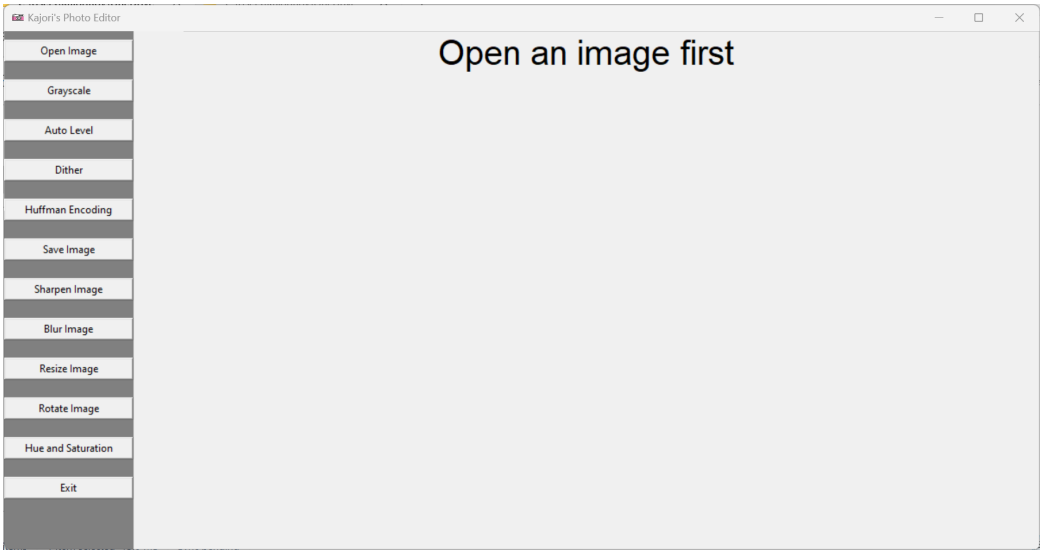


Figure 35: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Resize button when no image is opened

### 5.2.4 Image Rotation


Figure 36: Rotating the first example image

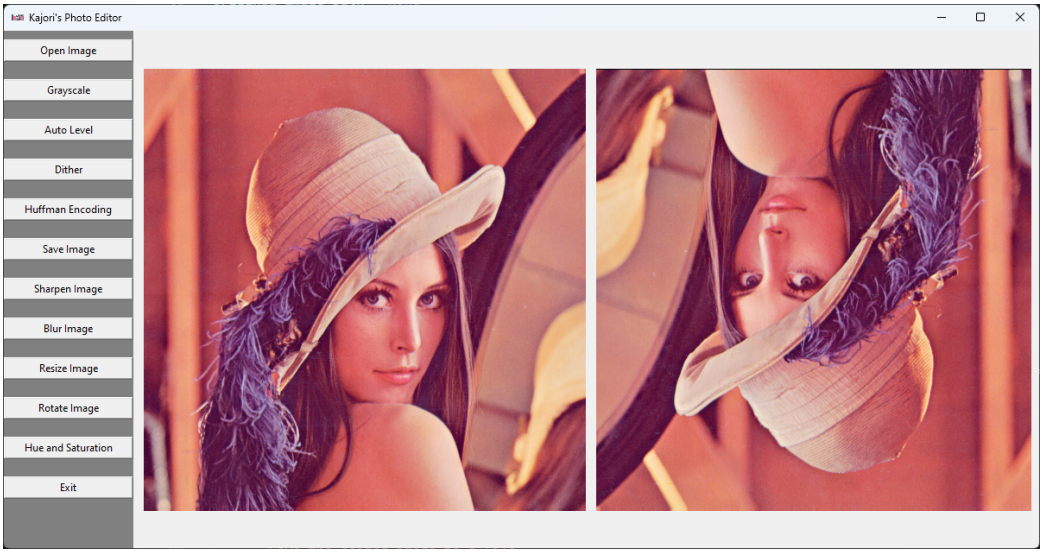


Figure 37: Rotating the first example image a second time

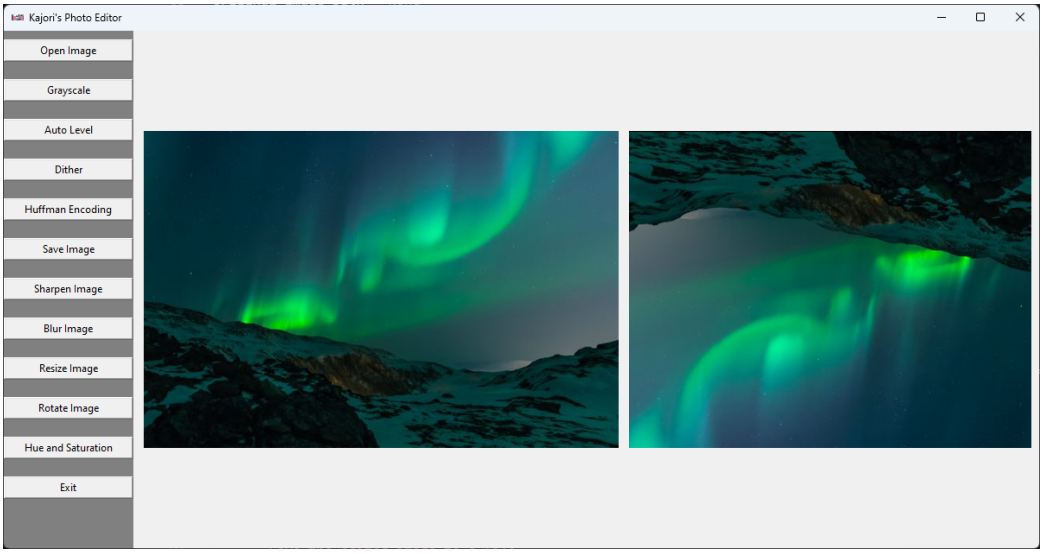


Figure 38: Rotating the second example image

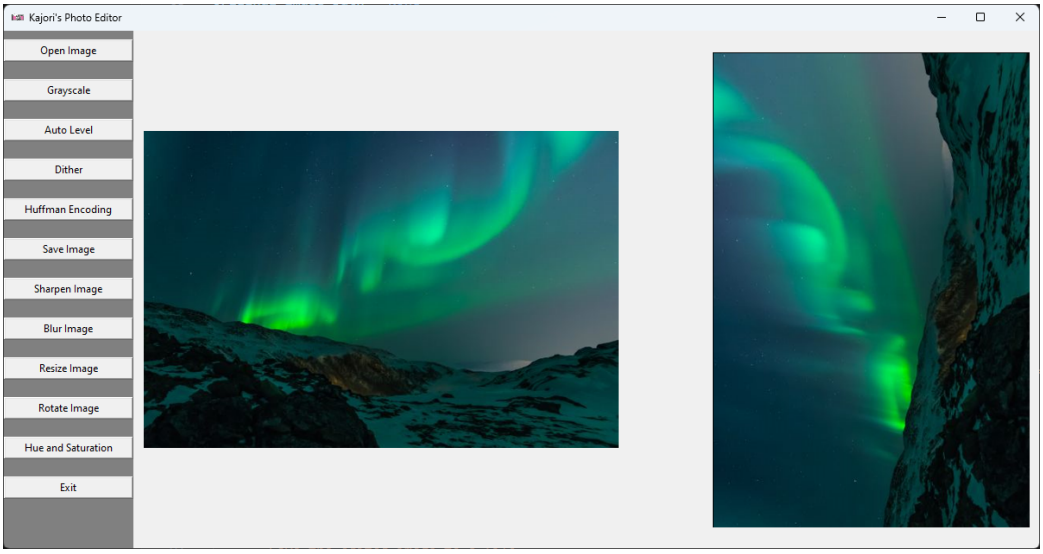


Figure 39: Rotating the second example image a second time

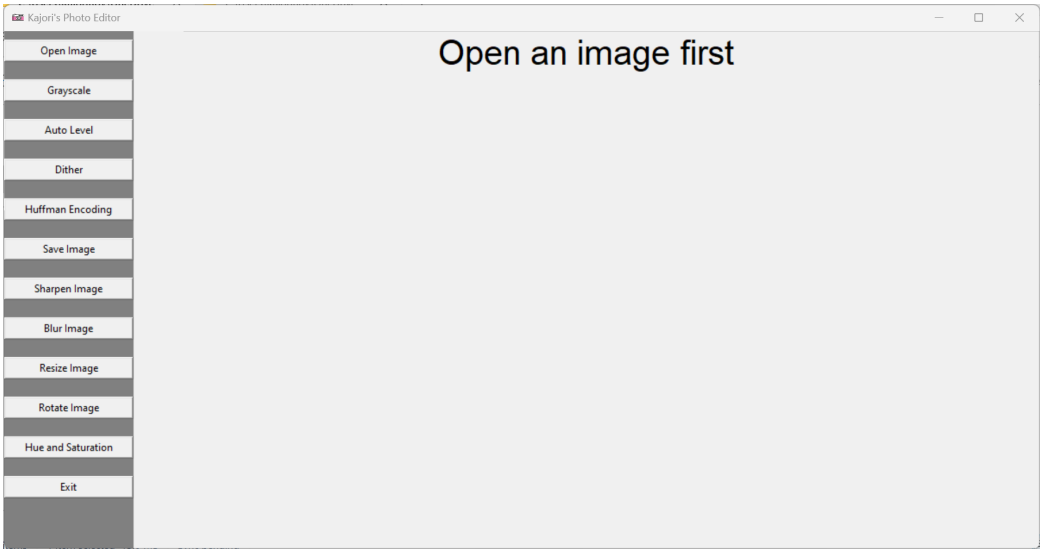


Figure 40: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Rotate button when no image is opened

### 5.2.5 Hue and Saturation Adjustment

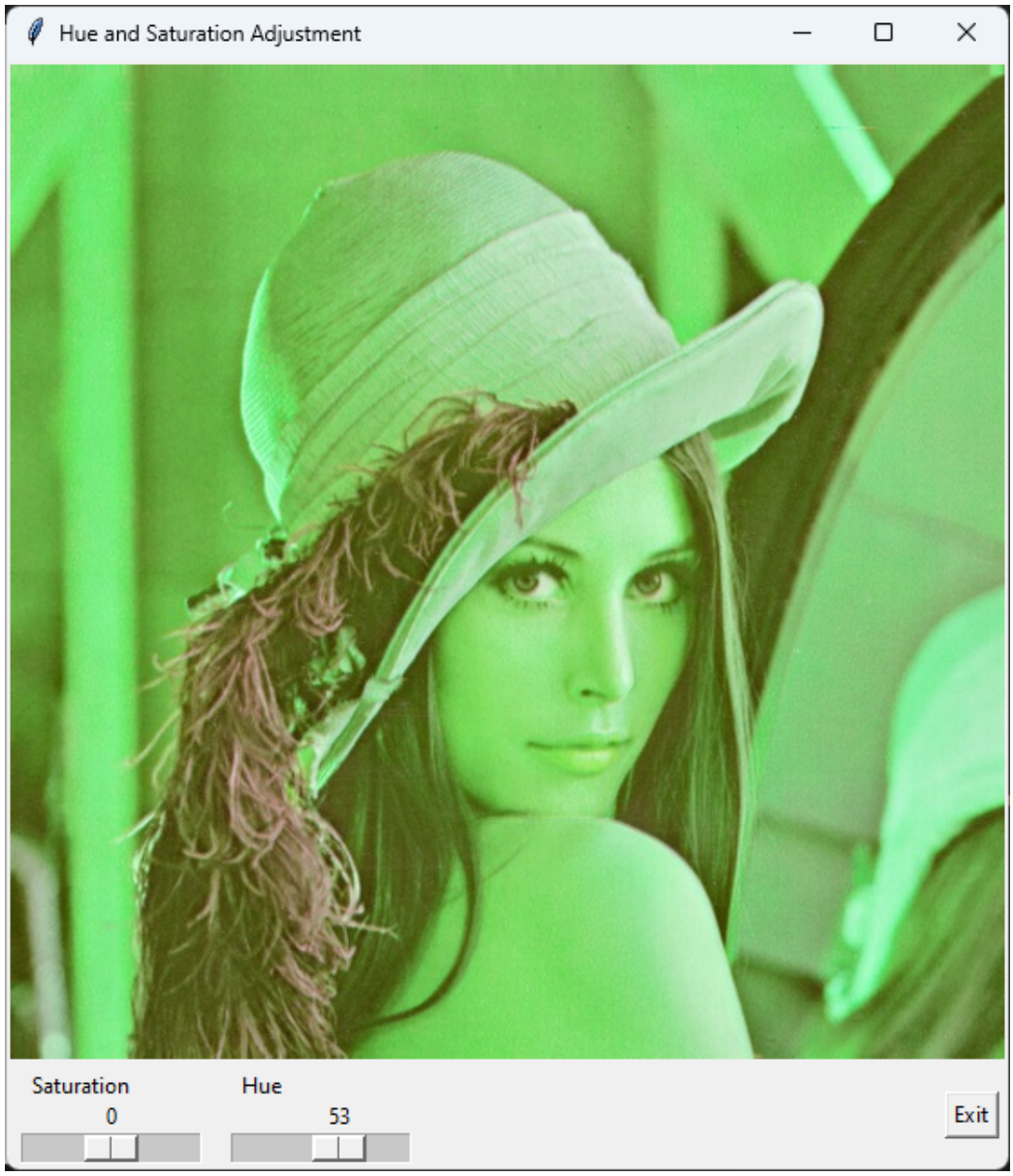


Figure 41: Separate Hue Saturation Window for the first example image

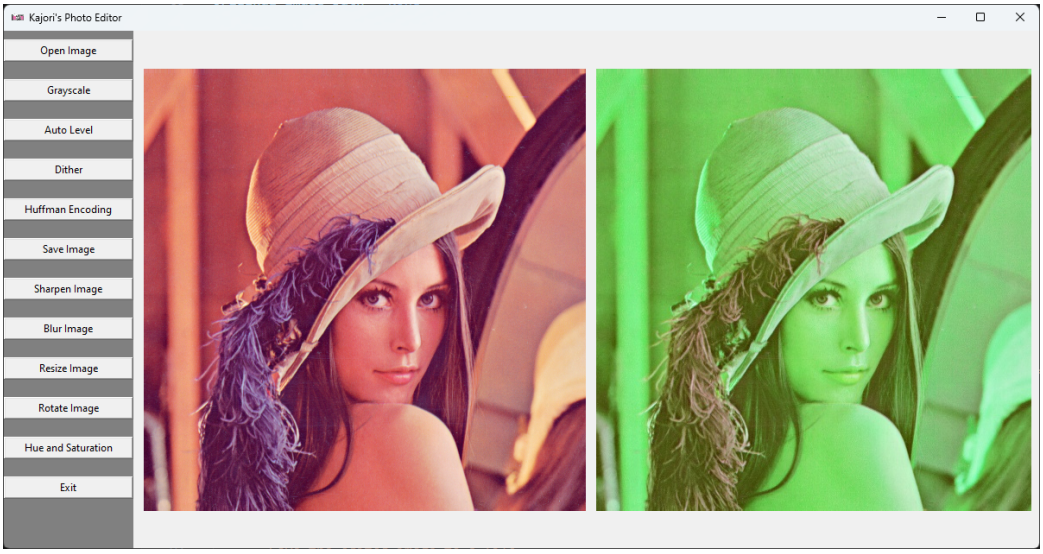


Figure 42: Output after adjusting the hue and saturation for first example image

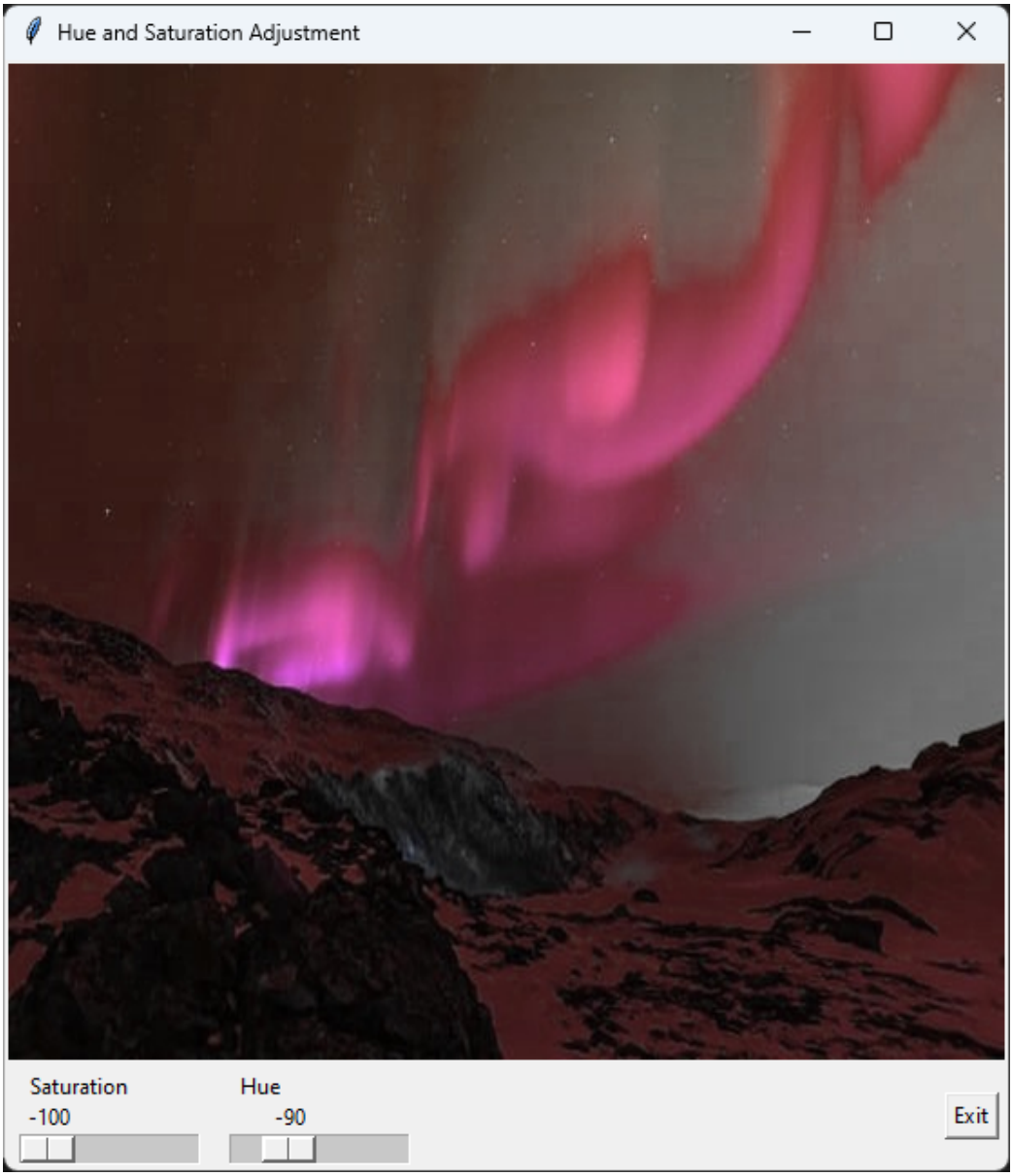


Figure 43: Separate Hue Saturation Window for the second example image

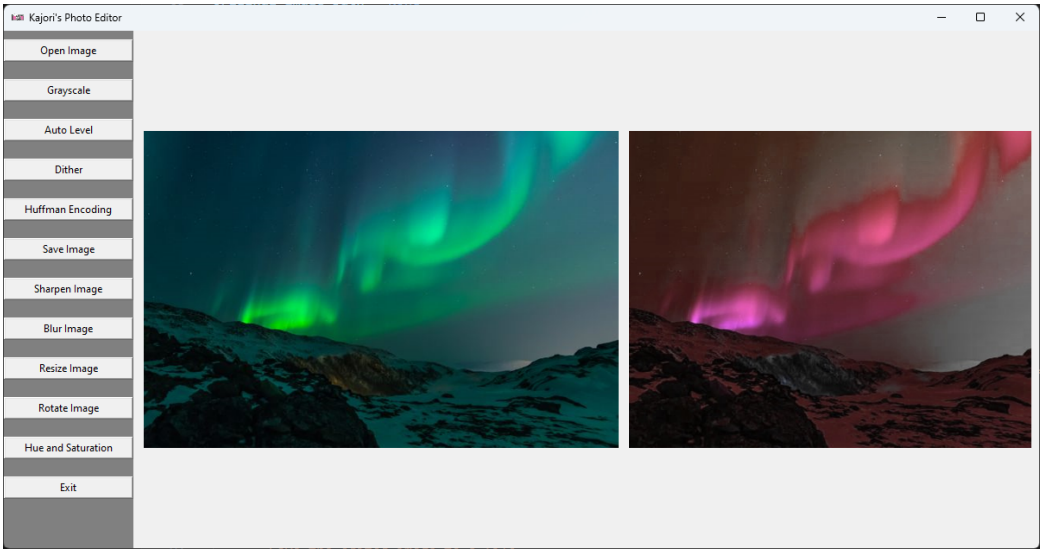


Figure 44: Output after adjusting the hue and saturation for the second example image

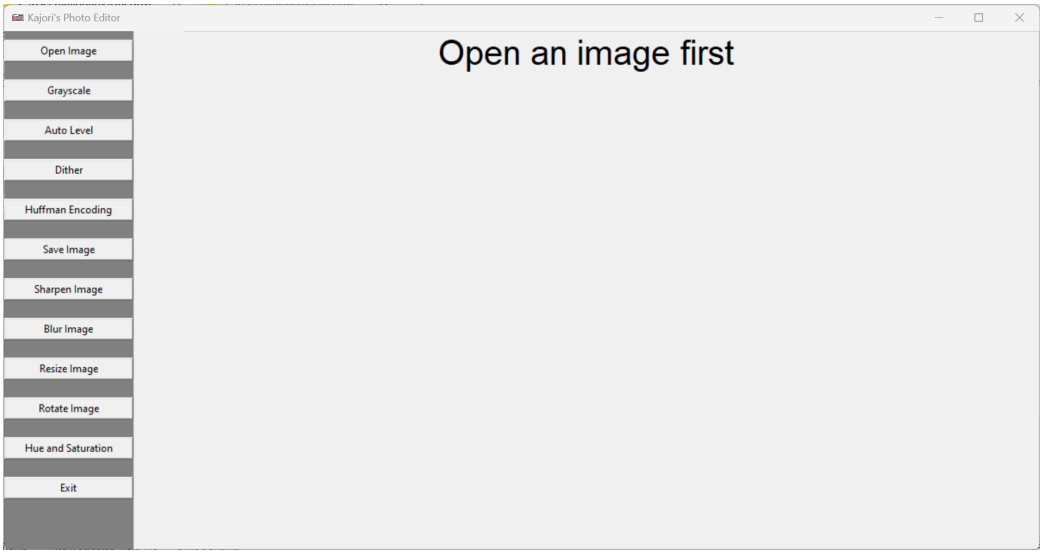


Figure 45: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Rotate button when no image is opened

### 5.2.6 Saving an Image

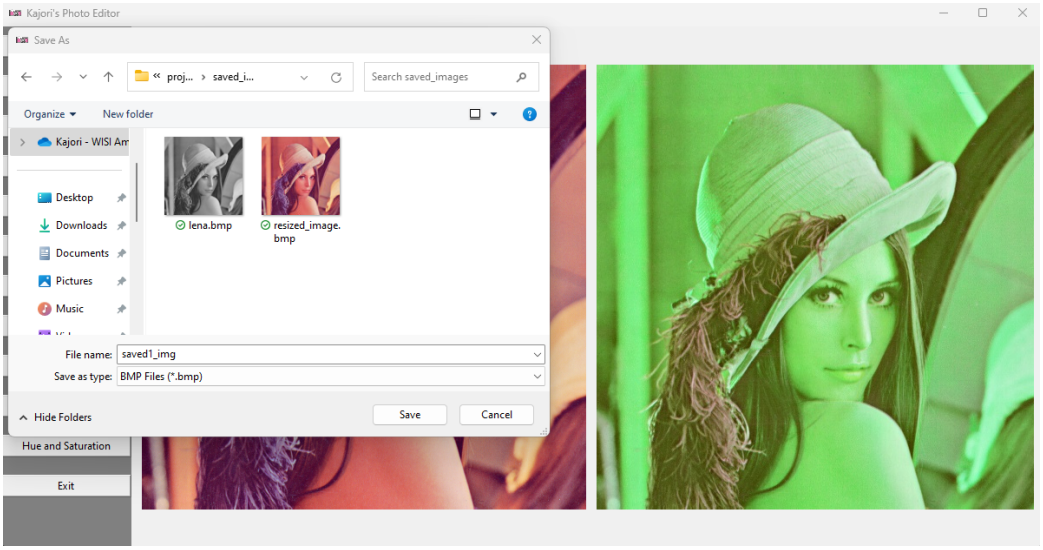


Figure 46: Saving the Hue Saturation output for the first example image

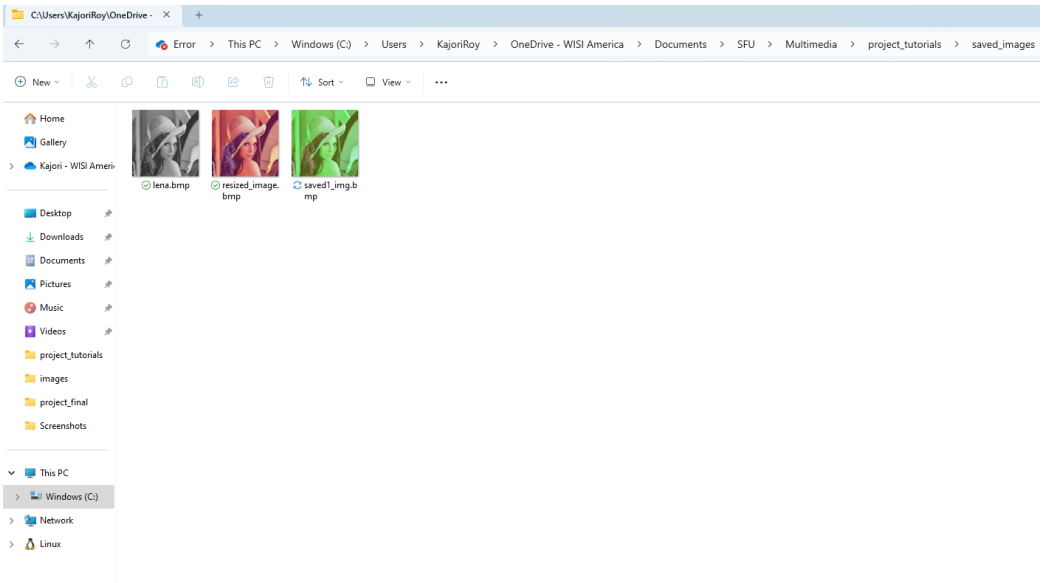


Figure 47: Saved Image

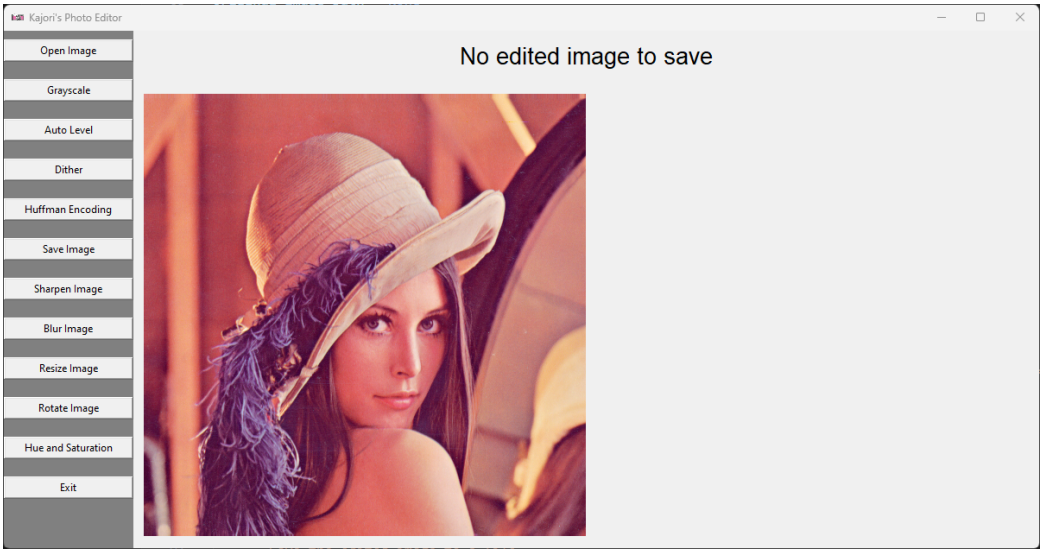


Figure 48: Error Handling
*Only an edited image will be saved*

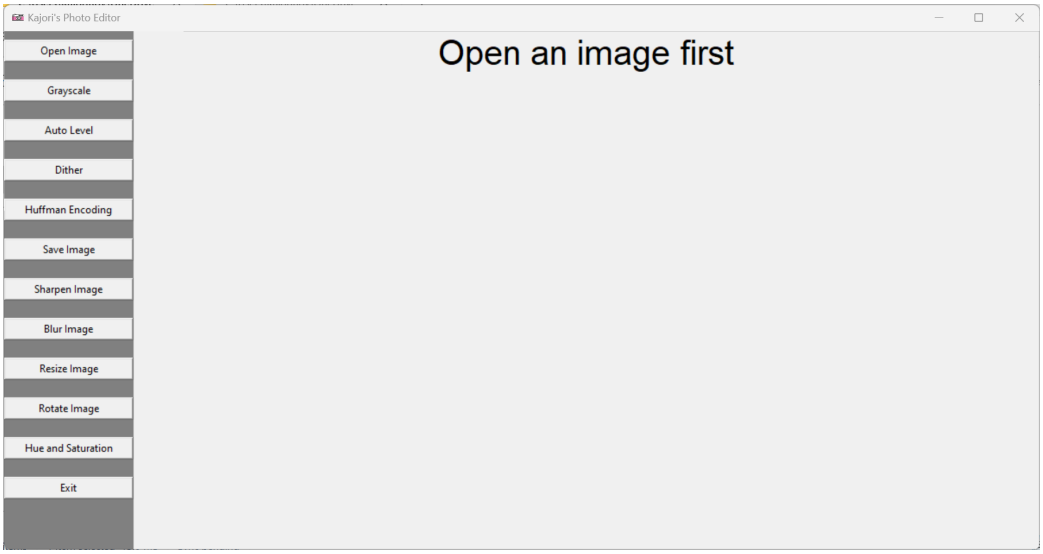


Figure 49: Error handling if no image is opened
This demonstrates the error handling mechanism where we press the Save Image button when no image is opened

# 6 Conclusion

In conclusion, the Mini-Photoshop project represents a meaningful confluence of image processing theory and software engineering practice. It not only provides a functional tool for image editing but also serves as a platform for ongoing learning and development in the exciting field of digital media. As technology advances, the potential for further enriching this application and exploring new frontiers in image processing and user interface design remains vast, promising a continually evolving landscape of opportunities for developers and users alike.