Digital Image Processing

Image Processing class project, acadimec year 2024 /2025

Student Name in English	Student Name in Arabic	Student ID	Section time as shown in zajel	Work percentage
Eman Abdalazeez	ايمان عبدالعزيز	12113148	11:00-12:30	%50
Ibrahim Asad	ابراهيم اسعد	12112090	11:00-12:30	%50

Abstract

The project showcases an interactive image processing application that was built from scratch in Python using OpenCV. Its main purpose is to allow users to upload their images and apply both basic and complex form of digital image processing techniques on them. The major features offered are grayscale conversion, brightness modification, histogram equalization, salt-and-pepper noise addition, and noise removal via mean and median filters. The system also allows sharpening, Gaussian blurring, adding watermarks, and visual comparison of results before and after processing. The entire application is user-friendly as the graphical interface has been created using Tkinter. This tool enhances the user's insight into understanding the basic concepts of image enhancement techniques and noise suppression methods.

Project

Main Idea:

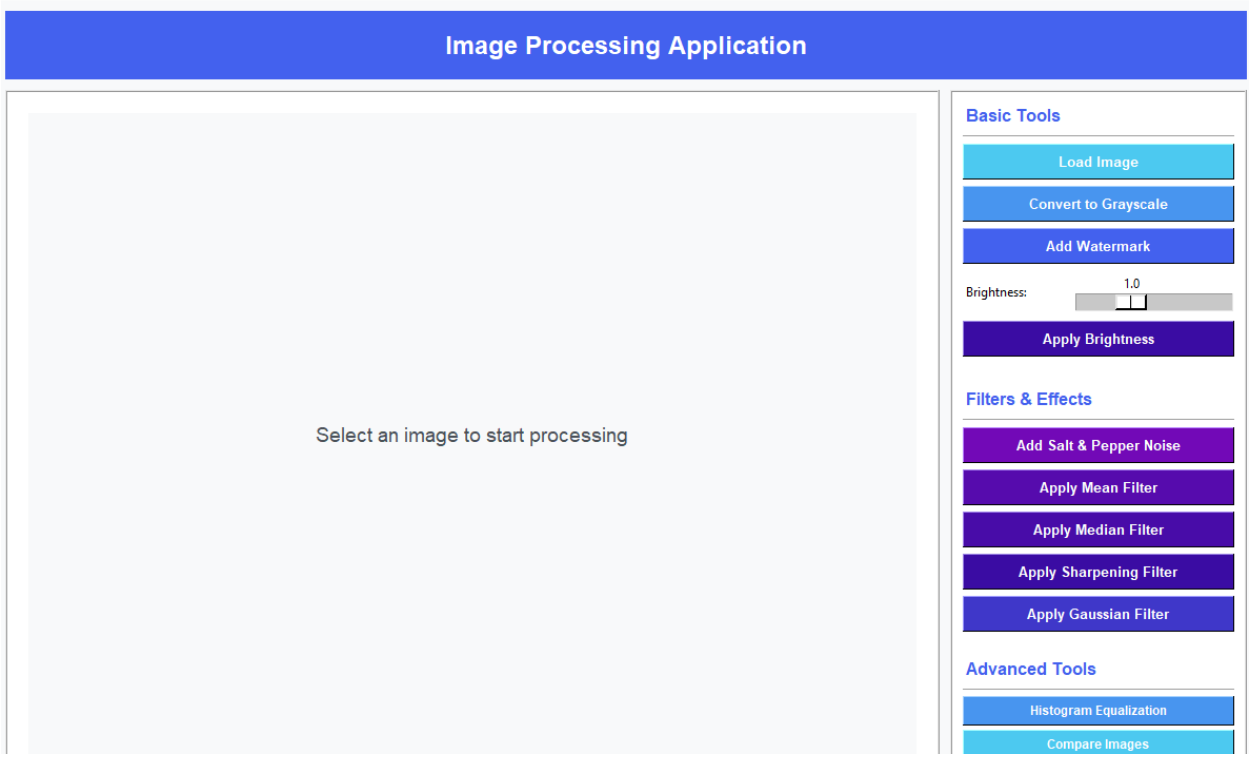
The project's simple and quick idea is to insert an image, randomly name it, and then apply specific filters according to the project requirements.

Libraries used:

We used the tkinter library defined in Python to create the GUI, and PIL.Image and ImageTk to convert images from the OpenCV format to a format suitable for display in Tkinter. We also used os and sys to handle files and paths.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
from tkinter import Tk, Button, Label, Frame, BOTH, LEFT, RIGHT, TOP, BOTTOM, filedialog, Scale, StringVar, OptionMenu
from tkinter import ttk, Canvas, PhotoImage, HORIZONTAL, DISABLED, NORMAL, RIDGE, GROOVE, RAISED, SUNKEN
from PIL import Image, ImageTk
import os
import sys
```

- When the program is run, this GUI appears.



The code and the function of each part:

Init() (initialization) function:

- Initializing the application and configuring the graphical interface:
- self.root: The application window.
- self.original_img: The original image (BGR).
- self.current_img: The image being modified.
- self.gray_img: A grayscale version of the image

- Frames are created and graphical widgets are loaded.

```
class ImageProcessingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Image Processing Application")
        self.root.geometry("1200x750")
        self.root.configure(bg="#f8f9fa")

        # Initialize variables
        self.original_img = None
        self.original_rgb = None
        self.gray_img = None
        self.noisy_img = None
        self.current_img = None
        self.current_operation = "No Operation"
        self.brightness_value = 1.0

        # Create main frames
        self.create_frames()

        # Create UI elements
        self.create_ui_elements()

        # Add status bar
        self.create_status_bar()

        # Set theme colors
        self.set_theme()
```

create_frames()

Creates the basic structure of the interface:

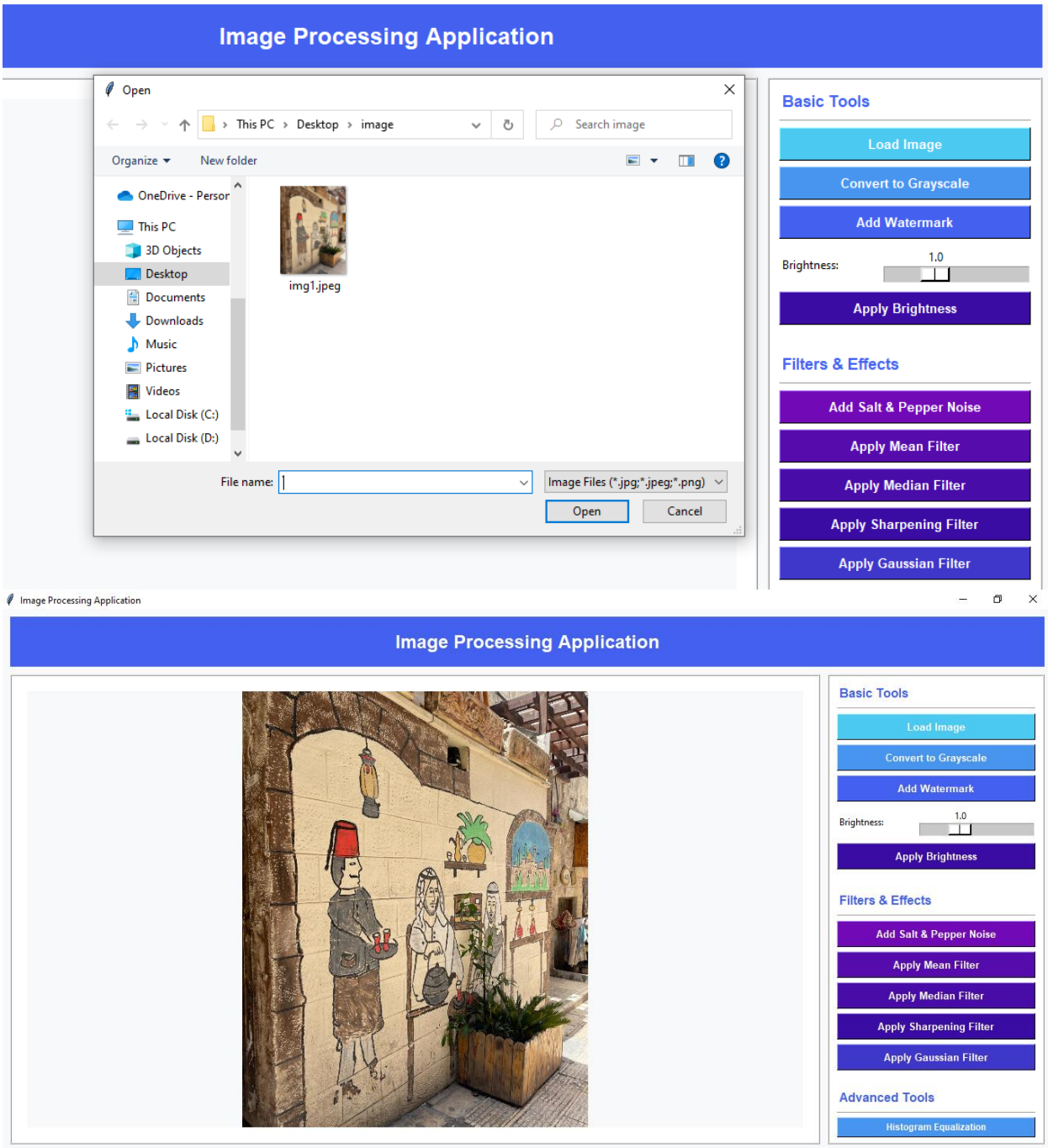
- main_frame: The main container.
- header_frame: To display the title.
- image_frame: To display images.
- control_frame: To display the controls, divided into:
- basic_controls: To upload an image and randomly assign names.
- filter_controls.
- advanced_controls: Contains the histogram equalization.

```
def create_frames(self):
    # Main frame
    self.main_frame = Frame(self.root, bg="#f8f9fa")
    self.main_frame.pack(fill=BOTH, expand=True, padx=10, pady=10)

    # Header frame
    self.header_frame = Frame(self.main_frame, bg="#436lee", height=60)
    self.header_frame.pack(fill="x", pady=(0, 10))

    # Image frame
    self.image_frame = Frame(self.main_frame, bg="#ffffff", bd=2, relief=RIDGE)
    self.image_frame.pack(side=LEFT, fill=BOTH, expand=True, padx=(0, 10))

    # Control frame
    self.control_frame = Frame(self.main_frame, bg="#ffffff", width=300, bd=2, relief=RIDGE)
    self.control_frame.pack(side=RIGHT, fill="y", padx=(0, 0))

    # Control sub-frames
    self.basic_controls = Frame(self.control_frame, bg="#ffffff")
    self.basic_controls.pack(fill="x", padx=10, pady=10)

    self.filter_controls = Frame(self.control_frame, bg="#ffffff")
    self.filter_controls.pack(fill="x", padx=10, pady=10)

    self.advanced_controls = Frame(self.control_frame, bg="#ffffff")
    self.advanced_controls.pack(fill="x", padx=10, pady=5)  # Reduced padding
```

load_image()

When you click on Load Image, you will be able to choose any image from your computer. The image will be displayed in its natural colors without changing it.

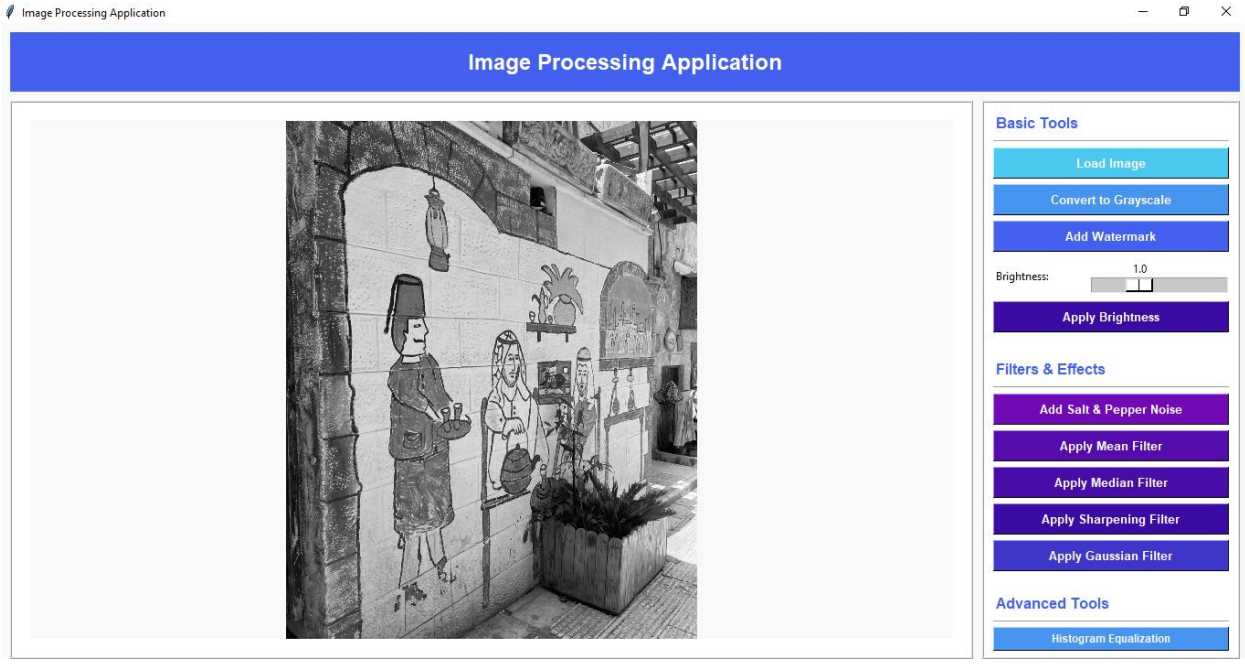
```
def load_image(self):
    """Load image from file"""
    file_path = filedialog.askopenfilename(
        filetypes=[("Image Files", "*.jpg;*.jpeg;*.png")]
    )
    if file_path:
        try:
            self.original_img = cv2.imread(file_path)
            if self.original_img is None:
                self.update_status("Failed to load image")
                return

            self.original_rgb = cv2.cvtColor(self.original_img, cv2.COLOR_BGR2RGB)
            self.gray_img = cv2.cvtColor(self.original_img, cv2.COLOR_BGR2GRAY)

            self.display_image(self.original_img)
            self.current_img = self.original_img.copy()

            self.update_status(f"Image loaded: {os.path.basename(file_path)}")
            self.update_operation("Original Image")
        except Exception as e:
            self.update_status(f"Error loading image: {str(e)}")
```



convert_to_grayscale()

Displays the grayscale image.
(Updates current_img with the grayscale version).

```python
def convert_to_grayscale(self):
    """Convert image to grayscale"""
    if self.original_img is None:
        self.update_status("Please load an image first")
        return

    self.display_image(self.gray_img, is_grayscale=True)
    self.current_img = self.gray_img.copy()
    self.update_status("Image converted to grayscale")
    self.update_operation("Grayscale")
```



add_watermark()

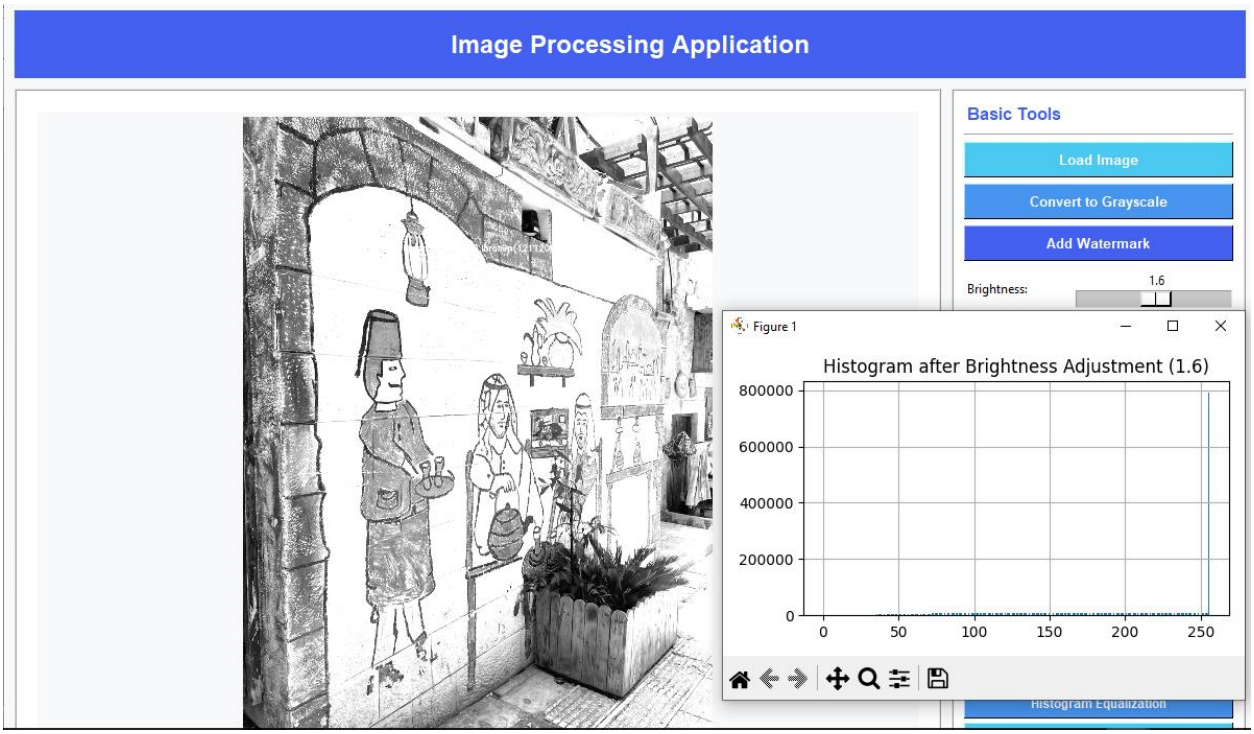
Add text: "Eman(12113148),Ibrahim(12112090)" in a random place inside the gray image. Each time you click on it, it will be added in a different place than the previous one.

```python
def add_watermark(self):
    """Add watermark to image"""
    if self.current_img is None:
        self.update_status("Please load an image first")
        return

    watermarked_img = self.current_img.copy()

    if len(watermarked_img.shape) == 3:
        watermarked_img = cv2.cvtColor(watermarked_img, cv2.COLOR_BGR2GRAY)

    x_pos = random.randint(0, watermarked_img.shape[1] - 100)
    y_pos = random.randint(50, watermarked_img.shape[0] - 10)

    cv2.putText(
        watermarked_img,
        "Eman(12113148),Ibrahim(12112090)",
        (x_pos, y_pos),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.7,
        255,
        2
    )

    self.display_image(watermarked_img, is_grayscale=True)
    self.current_img = watermarked_img.copy()
    self.update_status("Watermark added")
    self.update_operation("Watermark")
```

adjust_brightness()

- Multiply pixel values by brightness_value according to the equation given in the requirement.
- Use cv2.convertScaleAbs.
- Plot a histogram.

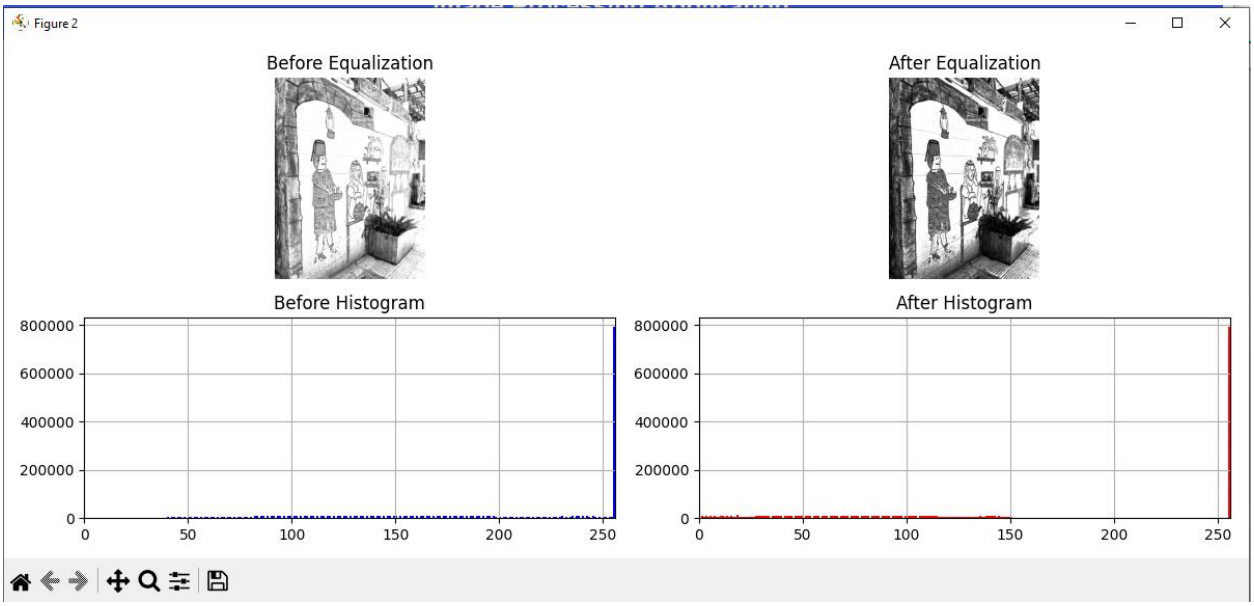
```python
def adjust_brightness(self):
    """Adjust image brightness"""
    if self.current_img is None:
        self.update_status("Please load an image first")
        return

    brightness_factor = self.brightness_value

    if len(self.current_img.shape) == 3:
        img_to_adjust = cv2.cvtColor(self.current_img, cv2.COLOR_BGR2GRAY)
    else:
        img_to_adjust = self.current_img.copy()

    bright_img = cv2.convertScaleAbs(img_to_adjust, alpha=brightness_factor, beta=0)
    self.display_image(bright_img, is_grayscale=True)
    self.current_img = bright_img.copy()

    self.update_status(f"Brightness adjusted to {brightness_factor:.1f}")
    self.update_operation(f"Brightness {brightness_factor:.1f}")

    plt.figure(figsize=(5, 3))
    plt.title(f"Histogram after Brightness Adjustment ({brightness_factor:.1f})")
    plt.hist(bright_img.ravel(), bins=256, range=[0, 256])
    plt.grid()
    plt.tight_layout()
    plt.show()
```



apply_equalization()

- Improves contrast in grayscale images.
- Uses cv2.equalizeHist().
- Displays before-and-after images and histograms.

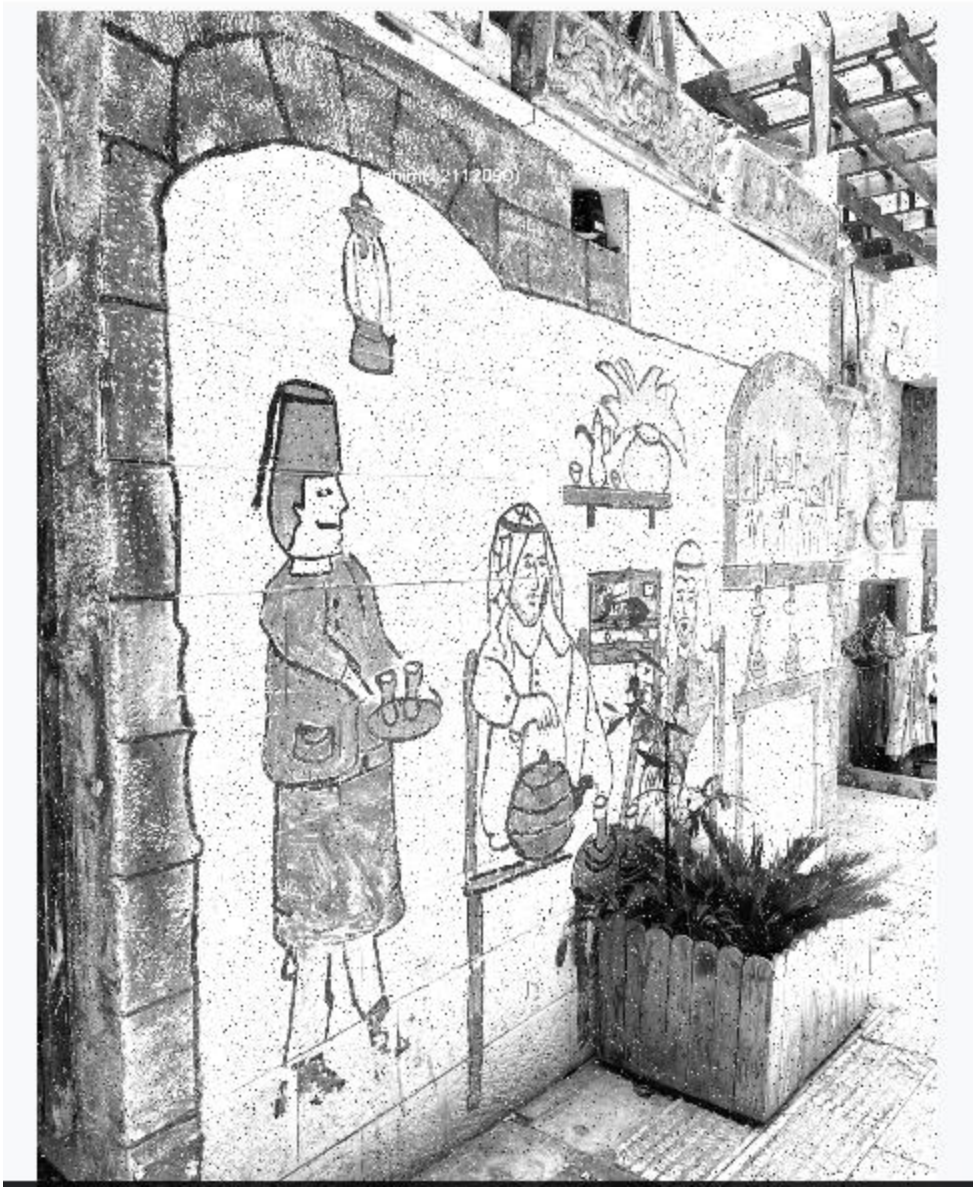
```
    def apply_equalization(self):
        """Apply histogram equalization"""
        if self.current_img is None:
            self.update_status("Please load an image first")
            return

        if len(self.current_img.shape) == 3:
            img_to_equalize = cv2.cvtColor(self.current_img, cv2.COLOR_BGR2GRAY)
        else:
            img_to_equalize = self.current_img.copy()

        equalized_img = cv2.equalizeHist(img_to_equalize)
        self.display_image(equalized_img, is_grayscale=True)
        self.current_img = equalized_img.copy()

        self.update_status("Histogram equalization applied")
        self.update_operation("Histogram Equalization")

        plt.figure(figsize=(12, 5))

        plt.subplot(2, 2, 1)
        plt.title("Before Equalization")
        plt.imshow(img_to_equalize, cmap='gray')
        plt.axis('off')

        plt.subplot(2, 2, 3)
        plt.title("Before Histogram")
        plt.hist(img_to_equalize.ravel(), bins=256, range=[0, 256], color='b')
        plt.xlim([0, 256])
        plt.grid(True)

        plt.subplot(2, 2, 2)
        plt.title("After Equalization")
        plt.imshow(equalized_img, cmap='gray')
        plt.axis('off')

        plt.subplot(2, 2, 4)
        plt.title("After Histogram")
        plt.hist(equalized_img.ravel(), bins=256, range=[0, 256], color='r')
        plt.xlim([0, 256])
        plt.grid(True)
```



add_salt_pepper_noise()

- Some pixels are randomly changed to 0 or 255.
- It simulates a specific type of noise common in images.

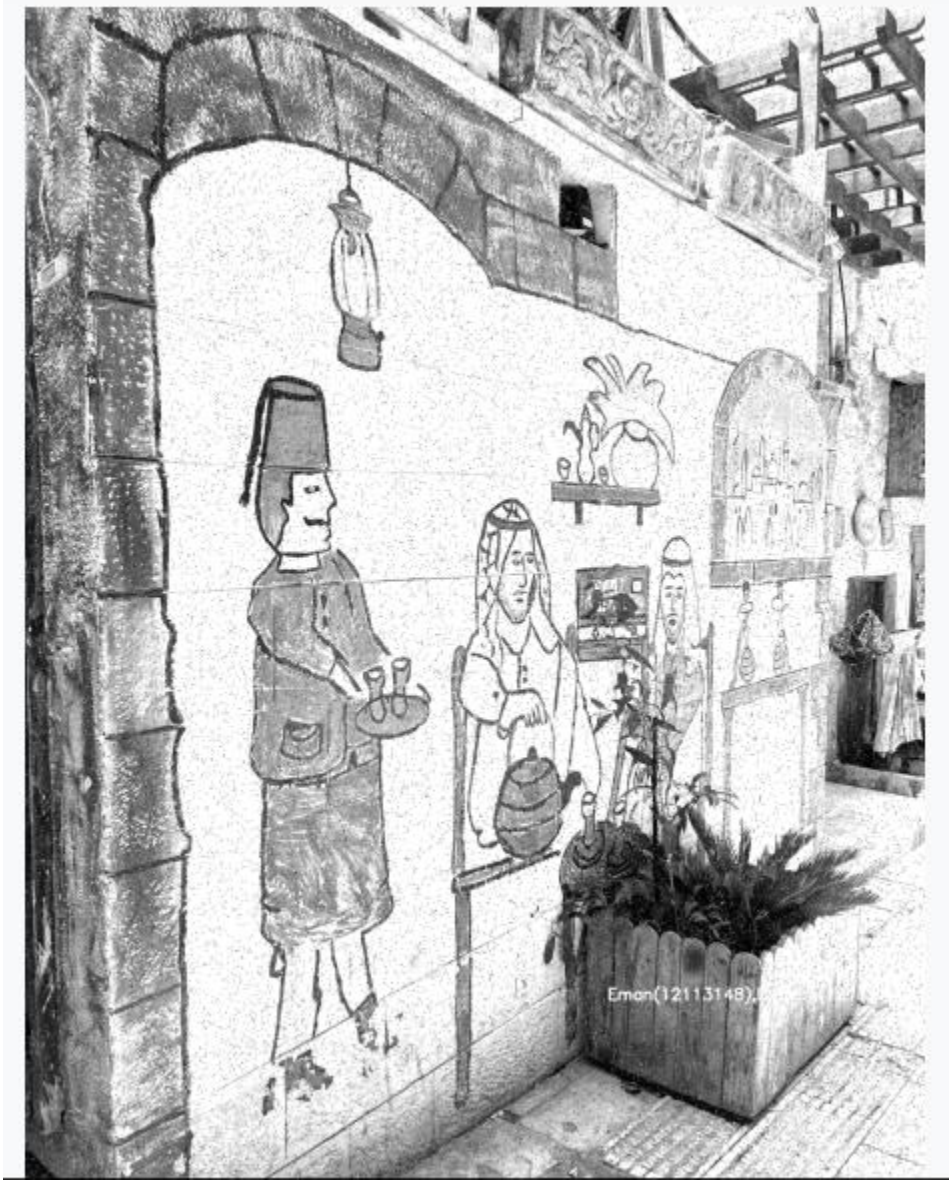
```
def add_salt_pepper_noise(self):
    """Add salt and pepper noise to image"""
    if self.current_img is None:
        self.update_status("Please load an image first")
        return

    if len(self.current_img.shape) == 3:
        img_for_noise = cv2.cvtColor(self.current_img, cv2.COLOR_BGR2GRAY)
    else:
        img_for_noise = self.current_img.copy()

    def apply_noise(img, noise_prob=0.02):
        noisy_output = np.copy(img)
        probabilities = np.random.rand(*img.shape)
        noisy_output[probabilities < noise_prob] = 0
        noisy_output[probabilities > 1 - noise_prob] = 255
        return noisy_output

    self.noisy_img = apply_noise(img_for_noise)
    self.display_image(self.noisy_img, is_grayscale=True)
    self.current_img = self.noisy_img.copy()

    self.update_status("Salt & pepper noise added")
    self.update_operation("Salt & Pepper Noise")
```



apply_mean_filter()

Median filter using cv2.blur.

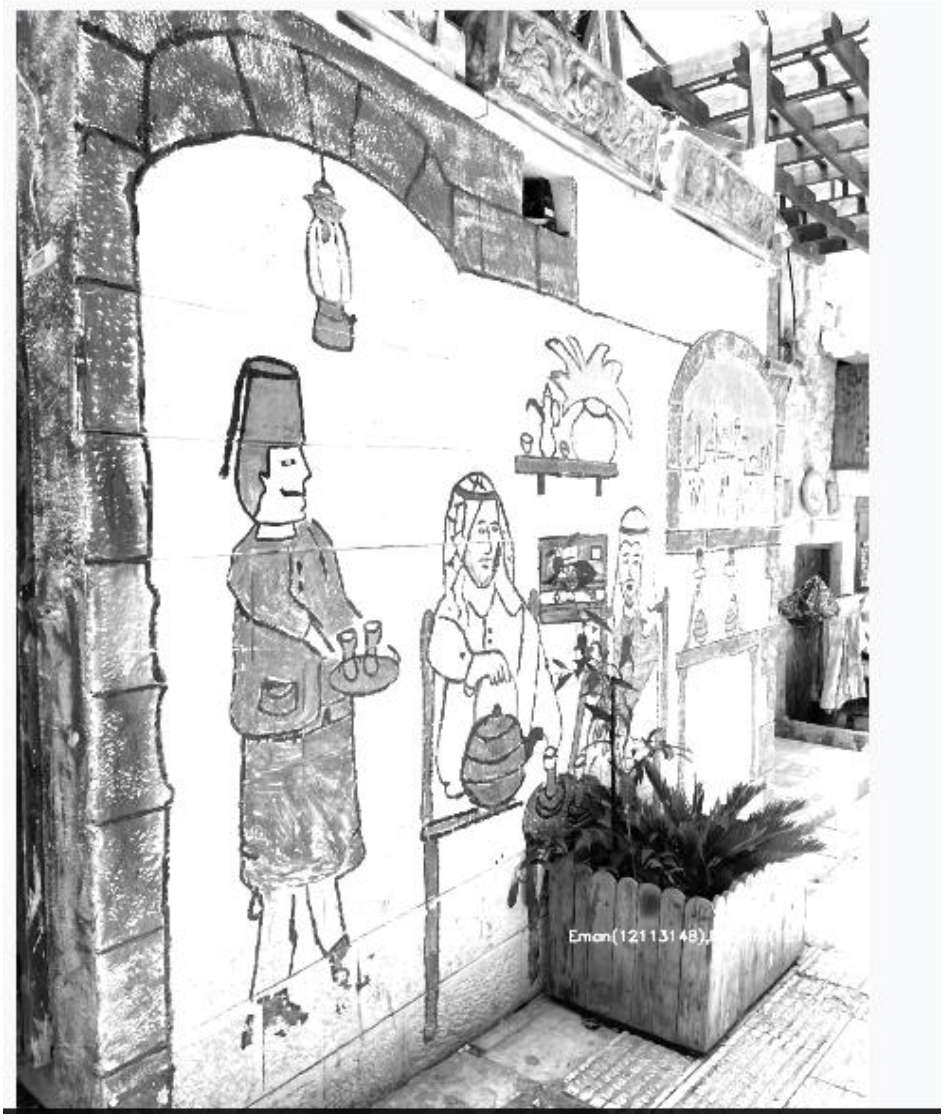
```python
def apply_mean_filter(self):
    """Apply mean filter"""
    if self.noisy_img is None:
        self.update_status("Please add noise first")
        return

    filtered_img = cv2.blur(self.noisy_img, (3, 3))
    self.display_image(filtered_img, is_grayscale=True)
    self.current_img = filtered_img.copy()

    self.update_status("Mean filter applied")
    self.update_operation("Mean Filter")
```



apply_median_filter()

Median filter using cv2.medianBlur.

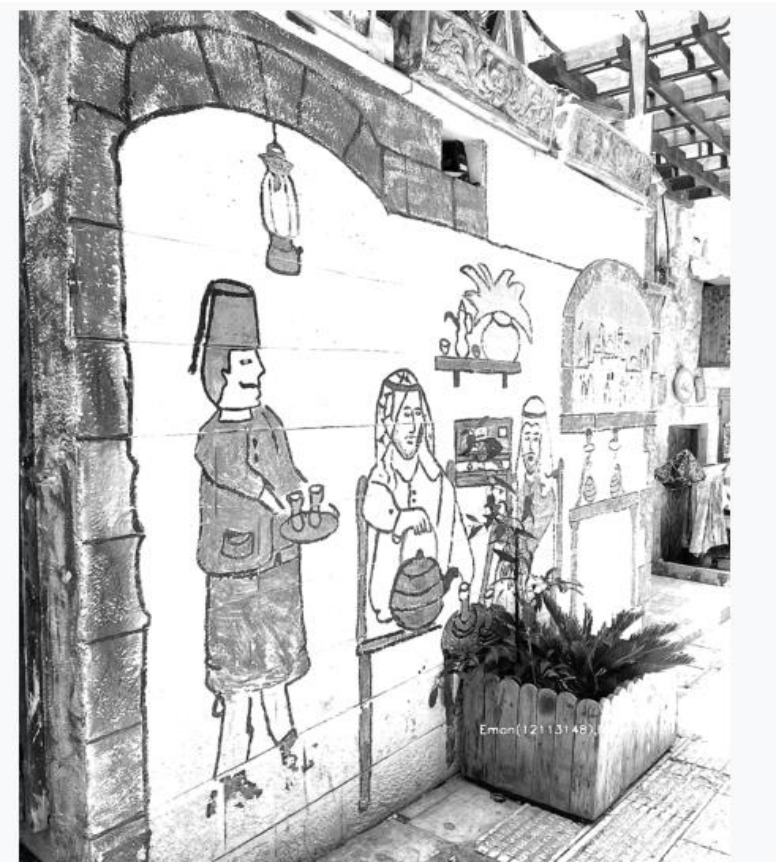
```
def apply_median_filter(self):
    """Apply median filter"""
    if self.noisy_img is None:
        self.update_status("Please add noise first")
        return

    filtered_img = cv2.medianBlur(self.noisy_img, 3)
    self.display_image(filtered_img, is_grayscale=True)
    self.current_img = filtered_img.copy()

    self.update_status("Median filter applied")
    self.update_operation("Median Filter")
```



apply_gaussian_filter()

Gaussian filter use cv2.GaussianBlur

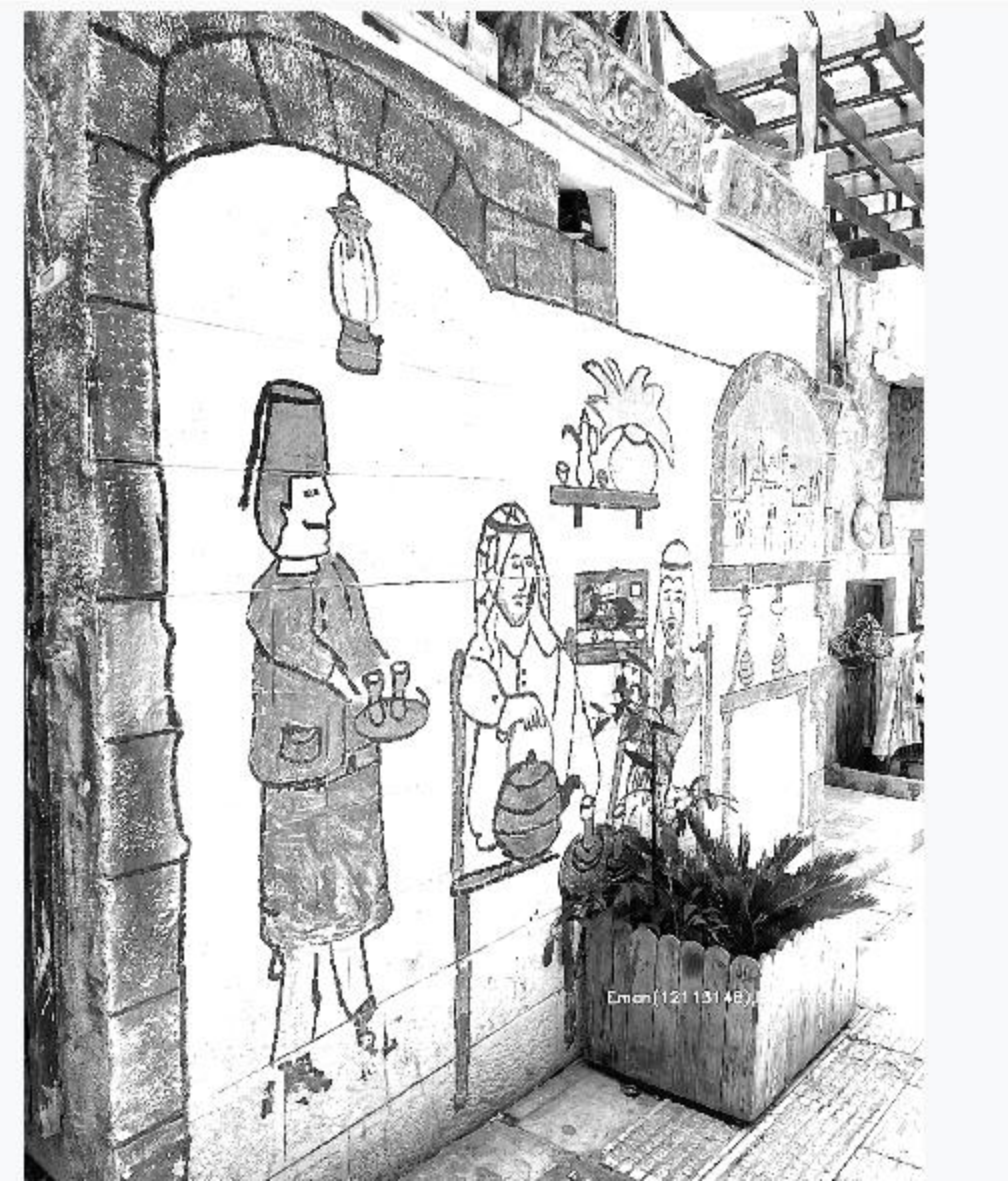
```
def apply_gaussian_filter(self):
    """Apply Gaussian filter"""
    if self.current_img is None:
        self.update_status("Please load an image first")
        return

    if len(self.current_img.shape) == 3:
        img_to_blur = cv2.cvtColor(self.current_img, cv2.COLOR_BGR2GRAY)
    else:
        img_to_blur = self.current_img.copy()

    blurred_img = cv2.GaussianBlur(img_to_blur, (5, 5), 0)
    self.display_image(blurred_img, is_grayscale=True)
    self.current_img = blurred_img.copy()

    self.update_status("Gaussian filter applied")
    self.update_operation("Gaussian Filter")
```



sharpen_image()

Uses a specific kernel to sharpen edges.

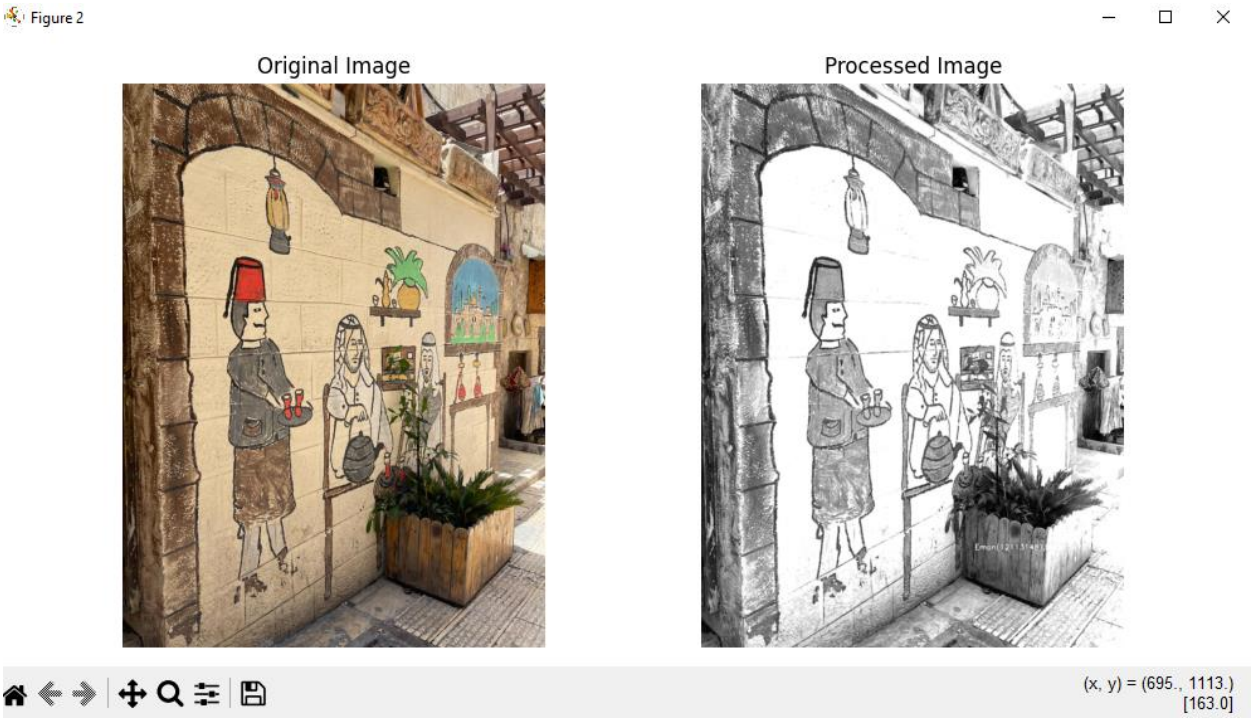
```
def sharpen_image(self):
    """Apply sharpening filter"""
    if self.current_img is None:
        self.update_status("Please load an image first")
        return

    if len(self.current_img.shape) == 3:
        img_to_sharpen = cv2.cvtColor(self.current_img, cv2.COLOR_BGR2GRAY)
    else:
        img_to_sharpen = self.current_img.copy()

    sharpening_kernel = np.array([
        [0, -1, 0],
        [-1, 5, -1],
        [0, -1, 0]
    ])

    sharpened_img = cv2.filter2D(img_to_sharpen, -1, sharpening_kernel)
    self.display_image(sharpened_img, is_grayscale=True)
    self.current_img = sharpened_img.copy()

    self.update_status("Sharpening filter applied")
    self.update_operation("Sharpening Filter")
```



compare_images()

The first image is placed without changes and next to it is the final image after modifications for comparison.

```
def compare_images(self):
    """Compare original and processed images"""
    if self.original_img is None or self.gray_img is None:
        self.update_status("Please load an image first")
        return

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(self.original_img, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    if len(self.current_img.shape) == 3:
        plt.imshow(cv2.cvtColor(self.current_img, cv2.COLOR_BGR2RGB))
    else:
        plt.imshow(self.current_img, cmap="gray")
    plt.title("Processed Image")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    self.update_status("Comparing images")
```



resize_image()

Make sure the image size covers the entire frame and appears harmonious and in its natural position.

```python
def resize_image(self, img, max_size=600):
    """Resize the image while maintaining aspect ratio if it exceeds max_size"""
    height, width = img.shape[:2]
    scale = min(max_size / width, max_size / height, 1.0)
    if scale < 1.0:
        img = cv2.resize(img, (int(width * scale), int(height * scale)))
    return img
```

The rest of the code is for formatting and arranging the page from buttons and frames to display the image and edit it.

```python
def create_ui_elements(self):
    # Application title
    title_label = Label(self.header_frame, text="Image Processing Application",
                        font=("Arial", 18, "bold"), bg="#4361ee", fg="white")
    title_label.pack(pady=15)

    # Image display area
    self.image_panel = Label(self.image_frame, bg="#f8f9fa")
    self.image_panel.pack(fill=BOTH, expand=True, padx=20, pady=20)

    # Initial message
    self.placeholder_text = Label(self.image_panel,
                                  text="Select an image to start processing",
                                  font=("Arial", 14), bg="#f8f9fa", fg="#495057")
    self.placeholder_text.pack(fill=BOTH, expand=True)

    # Basic tools section
    basic_title = Label(self.basic_controls, text="Basic Tools",
                        font=("Arial", 12, "bold"), bg="#ffffff", fg="#4361ee")
    basic_title.pack(anchor="w", pady=(0, 3))

    ttk.Separator(self.basic_controls).pack(fill="x", pady=3)

    # Basic tools buttons
    self.create_button(self.basic_controls, "Load Image", self.load_image, "#4cc9f0")
    self.create_button(self.basic_controls, "Convert to Grayscale", self.convert_to_grayscale, "#4895ef")
    self.create_button(self.basic_controls, "Add Watermark", self.add_watermark, "#4361ee")

    # Brightness control
    brightness_frame = Frame(self.basic_controls, bg="#ffffff")
    brightness_frame.pack(fill="x", pady=5)

    brightness_label = Label(brightness_frame, text="Brightness:", bg="#ffffff")
    brightness_label.pack(side=LEFT)

    self.brightness_scale = Scale(brightness_frame, from_=0.1, to=3.0, resolution=0.1,
                                  orient=HORIZONTAL, length=150, bg="#ffffff",
                                  highlightthickness=0, command=self.update_brightness)
    self.brightness_scale.set(1.0)
    self.brightness_scale.pack(side=RIGHT)
```

```
    # Filters section
    filter_title = Label(self.filter_controls, text="Filters & Effects",
                         font=("Arial", 12, "bold"), bg="#ffffff", fg="#436lee")
    filter_title.pack(anchor="w", pady=(5, 3))

    ttk.Separator(self.filter_controls).pack(fill="x", pady=3)

    # Filter buttons
    self.create_button(self.filter_controls, "Add Salt & Pepper Noise", self.add_salt_pepper_noise, "#7209b7")
    self.create_button(self.filter_controls, "Apply Mean Filter", self.apply_mean_filter, "#560bad")
    self.create_button(self.filter_controls, "Apply Median Filter", self.apply_median_filter, "#480ca8")
    self.create_button(self.filter_controls, "Apply Sharpening Filter", self.sharpen_image, "#3a0ca3")
    self.create_button(self.filter_controls, "Apply Gaussian Filter", self.apply_gaussian_filter, "#3f37c9")

    # Advanced tools section
    advanced_title = Label(self.advanced_controls, text="Advanced Tools",
                           font=("Arial", 12, "bold"), bg="#ffffff", fg="#436lee")
    advanced_title.pack(anchor="w", pady=(5, 3))

    ttk.Separator(self.advanced_controls).pack(fill="x", pady=3)

    # Advanced tools buttons with smaller padding
    self.create_button_small(self.advanced_controls, "Histogram Equalization", self.apply_equalization, "#4895ef")
    self.create_button_small(self.advanced_controls, "Compare Images", self.compare_images, "#4cc9f0")

    # Developer information
    dev_frame = Frame(self.control_frame, bg="#e9ecef", bd=1, relief=GROOVE)
    dev_frame.pack(fill="x", side=BOTTOM, padx=10, pady=10)

    dev_label = Label(dev_frame, text="Developed by: Eman(12113148), Ibrahim(12112090)",
                      font=("Arial", 8), bg="#e9ecef", fg="#495057")
    dev_label.pack(pady=5)

def create_button_small(self, parent, text, command, color="#436lee"):
    """Create buttons with smaller padding for advanced tools section"""
    btn = Button(parent, text=text, command=command,
                 bg=color, fg="white", font=("Arial", 9, "bold"),
                 relief=RAISED, borderwidth=1, padx=5, pady=3)
    btn.pack(fill="x", pady=2)
    return btn
```

```
    def create_status_bar(self):
        # Status bar
        self.status_frame = Frame(self.root, bg="#436lee", height=25)
        self.status_frame.pack(side=BOTTOM, fill="x")

        self.status_var = StringVar()
        self.status_var.set("Ready")

        self.status_label = Label(self.status_frame, textvariable=self.status_var,
                                  font=("Arial", 9), bg="#436lee", fg="white", anchor="w")
        self.status_label.pack(side=LEFT, padx=10)

        self.operation_var = StringVar()
        self.operation_var.set("Current Operation: None")

        self.operation_label = Label(self.status_frame, textvariable=self.operation_var,
                                     font=("Arial", 9), bg="#436lee", fg="white", anchor="e")
        self.operation_label.pack(side=RIGHT, padx=10)

    def set_theme(self):
        # Set button style
        style = ttk.Style()
        style.configure("TButton", font=("Arial", 10), borderwidth=1)
        style.configure("TScale", background="#ffffff")

    def create_button(self, parent, text, command, color="#436lee"):
        button_frame = Frame(parent, bg="#ffffff")
        button_frame.pack(fill="x", pady=3)

        button = Button(button_frame, text=text, command=command,
                        bg=color, fg="white", font=("Arial", 10, "bold"),
                        relief=RAISED, borderwidth=1, padx=5, pady=5)
        button.pack(fill="x")
        return button

# Run the application
if __name__ == "__main__":
    root = Tk()
    app = ImageProcessingApp(root)
    root.mainloop()
```