Speckle Noise Application

Presented to:

DR. Marwa

Presented by:

Abdelrahman Ghoniem Ghoniem **g4**

Mohamed Abdelmotleb Mohamed Shabaan **g7**

Mohamed Osama Abdelghany Abdallah **g7**

Table of Contents

Introduction

Speckle noise, also known as "graininess," is a type of noise commonly found in images captured by coherent imaging systems, such as ultrasound, synthetic aperture radar (SAR), or laser imaging. It appears as a granular pattern of bright and dark pixels, resembling the speckles seen in textured surfaces or on the wings of certain insects.

Speckle noise arises due to the coherent nature of the imaging process, where multiple coherent waves interfere with each other constructively or destructively. This interference leads to variations in the intensity of the received signal, resulting in the speckled appearance in the image.

Speckle noise can degrade image quality, making it difficult to visually interpret or analyze images accurately. Therefore, it is often desirable to reduce or remove speckle noise using various image processing techniques, such as filtering algorithms specifically designed to preserve image details while suppressing the noise.

Speckle noise presents several challenges in image processing and analysis, primarily due to its granular and multiplicative nature. Here are some of the problems associated with speckle noise and potential solutions:

1. **Loss of Image Detail**: Speckle noise can obscure fine details in the image, making it difficult to visually interpret or analyze.

Solution: Use speckle reduction techniques that aim to preserve important image features while suppressing noise. These techniques include filtering algorithms specifically designed to smooth out noise while retaining structural information.

2. **Impacts on Image Classification and Segmentation**: Speckle noise can interfere with image classification and segmentation tasks by introducing spurious texture patterns and altering pixel intensities.

   Solution: Preprocess images by applying speckle reduction filters before feeding them into classification or segmentation algorithms. These filters can help improve the accuracy of these tasks by reducing noise-induced artifacts.

3. **Degradation of Image Quality**: Speckle noise reduces the overall quality of images, affecting their usefulness for various applications.

   Solution: Employ advanced speckle reduction methods that utilize adaptive filtering or statistical modeling techniques to effectively suppress noise while preserving image quality. These methods often take into account the statistical properties of speckle noise to achieve better results.

4. **Increased Computational Complexity**: Some speckle reduction algorithms may be computationally intensive, especially for large or high-resolution images.

   Solution: Optimize algorithms for efficiency and implement parallel processing techniques to reduce computational overhead. Additionally, consider trade-offs between computational complexity and noise reduction effectiveness based on the specific requirements of the application.

5. **Artifact Introduction**: Improper speckle reduction techniques can introduce undesirable artifacts or blurring effects into the image.

   Solution: Evaluate different speckle reduction methods and parameters to find the optimal balance between noise suppression and preservation of image details. It may involve experimenting with various filter types, kernel sizes, and threshold values to achieve satisfactory results.

Overall, addressing speckle noise requires a careful balance between noise reduction and preservation of image content, considering the specific characteristics of the noise and the requirements of the application. Experimentation and adaptation of different techniques are often necessary to achieve the desired outcome.

Algorithm

The Python application follows a simple algorithm to add and remove speckle noise from images. Here's the algorithmic flow:

1. **Initialization**:
    - The application initializes a Tkinter GUI window with buttons, sliders, and labels for displaying images and adjusting noise strength.
2. **Image Selection**:
    - When the user clicks the "Choose Image" button, a file dialog opens, allowing them to select an image file.
    - The selected image is loaded using OpenCV and displayed in the GUI.
3. **Adding Speckle Noise**:
    - When the user clicks the "Add Noise" button, the application reads the noise strength value from the slider.
    - Speckle noise is generated using a random normal distribution, scaled by the specified strength, and added to the original image.
    - The noisy image is displayed in the GUI.

4. **Removing Speckle Noise**:
    - When the user clicks the "Remove Noise" button, the application reads the noise removal strength value from the slider.
    - A bilateral filter from OpenCV is applied to the noisy image to remove speckle noise while preserving image details.
    - The processed image is displayed in the GUI.

5. **Saving Images**:
    - If the user clicks the "Save Original Image" or "Save Processed Image" button, a file dialog opens, allowing them to specify the file name and location for saving the image in PNG format.
    - The respective image (original or processed) is saved to the specified file path.
6. **Error Handling**:
    - The application handles errors such as no image selected, errors during image processing, or issues with saving images.
    - Error messages are displayed using message boxes to alert the user.
7. **User Interaction**:
    - Throughout the process, the user can interact with the sliders to adjust the strength of noise addition and removal, allowing them to observe the effects on the image in real-time.

8. **GUI Update**:
    - After each step (image selection, noise addition, noise removal), the GUI updates to display the current state of the image (original, noisy, or processed).

Overall, the application provides a straightforward and interactive way for users to experiment with adding and removing speckle noise from images, observing the effects, and saving the results.

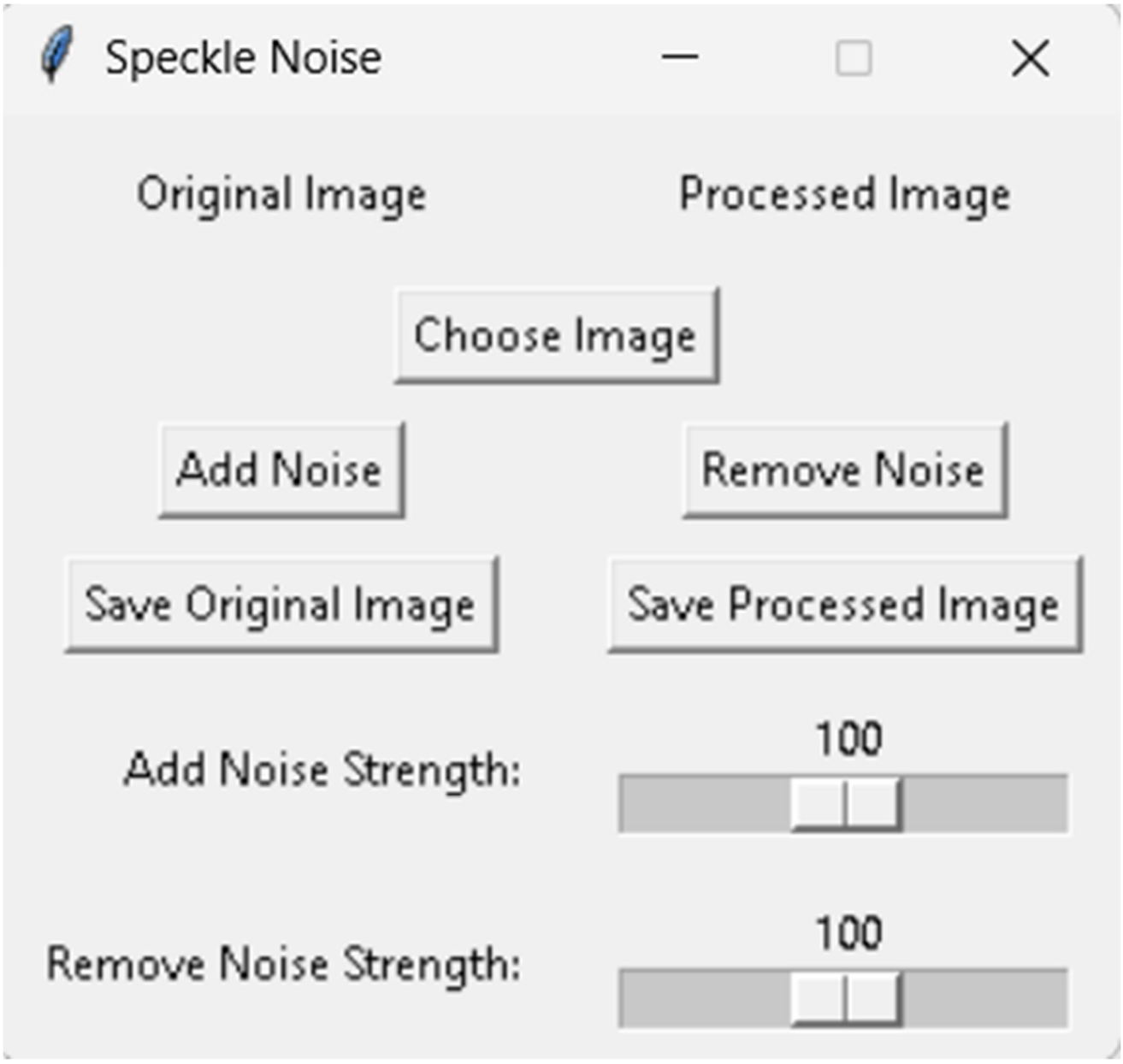
The libraries used

1. **OpenCV (cv2)**:
    - OpenCV is a popular library for computer vision and image processing tasks.
    - It is used for reading and processing images, adding speckle noise, and applying noise removal filters.
    - The application relies on OpenCV's functions for image loading (**cv2.imread**), noise addition, and bilateral filtering (**cv2.bilateralFilter**).
2. **Tkinter**:
    - Tkinter is the standard GUI (Graphical User Interface) toolkit for Python.
    - It is used to create the graphical user interface of the application, including buttons, sliders, labels, and file dialogs.
    - Tkinter provides widgets and methods for handling user interactions and displaying images within the GUI.
3. **NumPy**:
    - NumPy is a fundamental package for numerical computing with Python.
    - It is used for various numerical operations, including generating random noise for speckle noise addition and manipulating arrays representing images.
    - NumPy's arrays are compatible with OpenCV's image processing functions, facilitating seamless integration.
    -

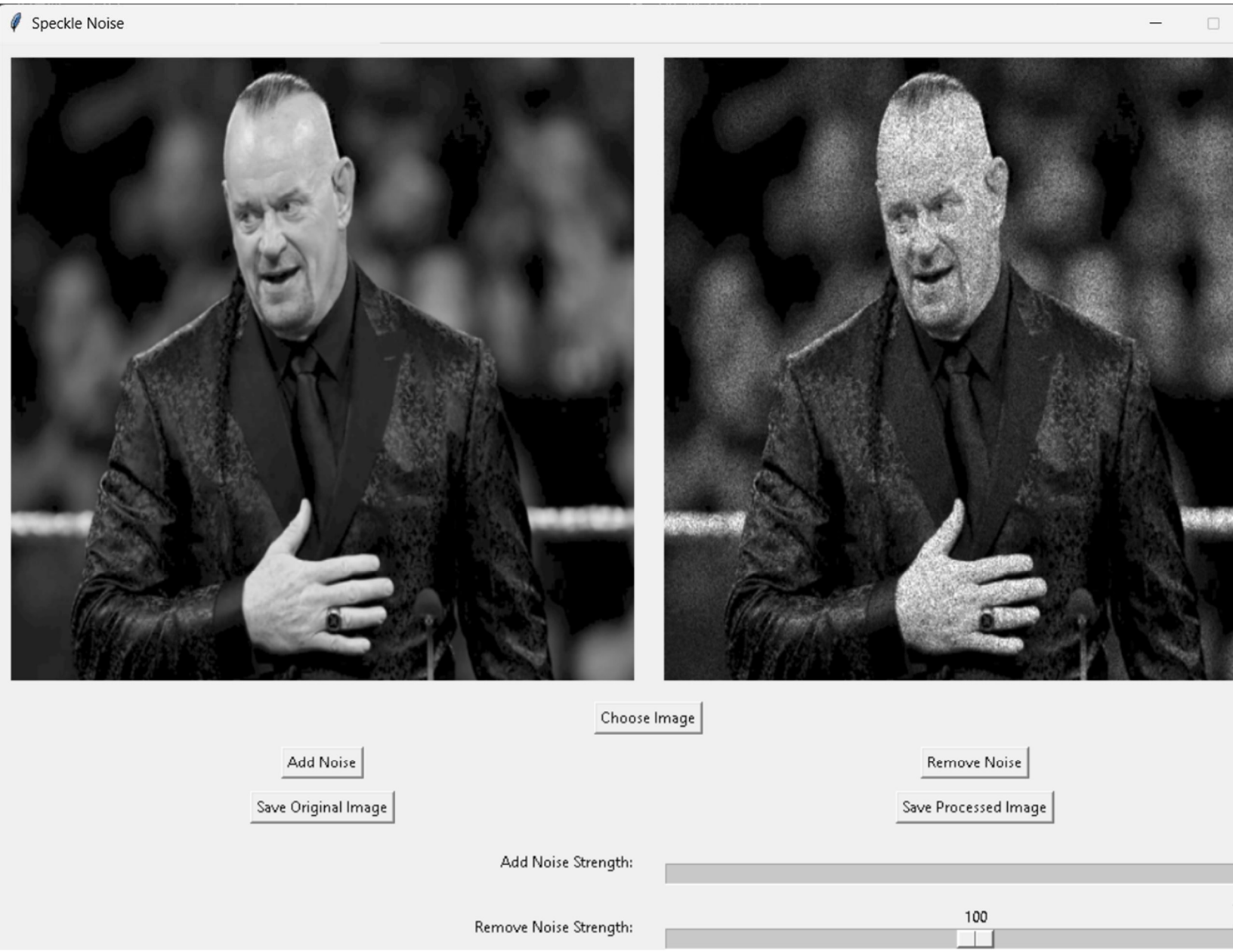
4. **PIL (Python Imaging Library) / Pillow**:
    - PIL (Python Imaging Library) is a library for opening, manipulating, and saving many different image file formats.
    - Pillow is a maintained fork of PIL and is used in this application.
    - PIL/Pillow is used to convert images between different formats and to resize images for display in the Tkinter GUI.

These libraries provide the necessary functionality for image processing, graphical user interface creation, numerical computations, and image file handling, enabling the application to add and remove speckle noise from images interactively.

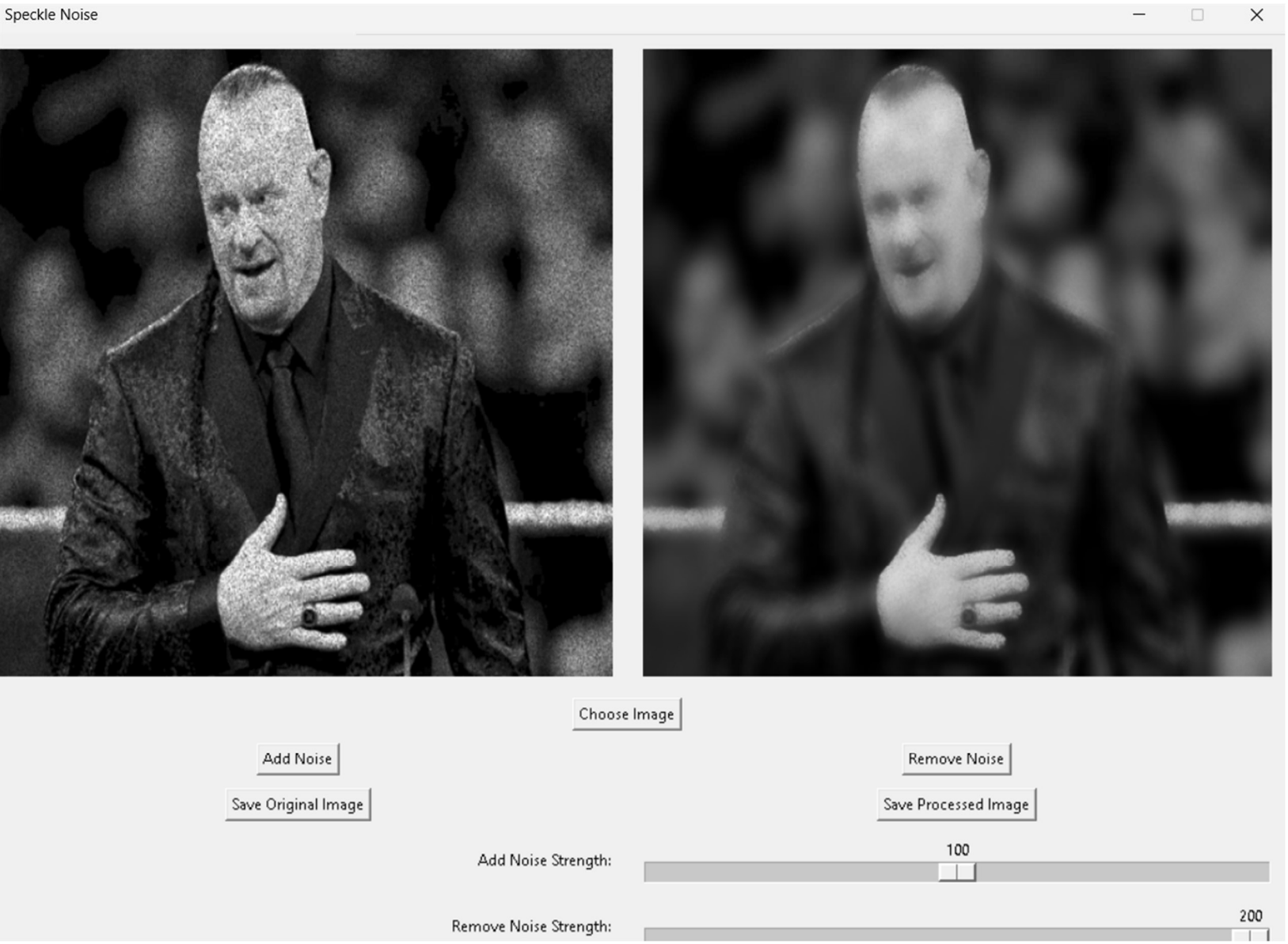
Shotscreens:-



User chooses an image from his local pc to apply and remove noises on it as simple as that for ex:-



This is a photo of the undertaker with added 200 noise percent



This is a photo of the undertaker with removed 200 noise percent

Code

```
    def choose_image(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            self.original_image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
            self.display_image(self.original_image, self.original_image_label)

    def display_image(self, image, label):
        image = Image.fromarray(image)
        # Resize the image to make it smaller
        image = image.resize((500, 500), Image.BILINEAR)
        image = ImageTk.PhotoImage(image)
        label.config(image=image)
        label.image = image

    def add_noise(self):
        if self.original_image is not None:
            add_noise_strength = self.add_noise_strength_scale.get() / 100
            self.noisy_image = self.add_speckle_noise(self.original_image, strength=add_noise_strength)
            self.display_image(self.noisy_image, self.processed_image_label)
        else:
            messagebox.showerror("Error", "No image selected.")

    def remove_noise(self):
        if self.original_image is not None:
            if self.noisy_image is not None:
                remove_noise_strength = self.remove_noise_strength_scale.get() / 100
                self.processed_image = self.remove_speckle_noise(self.noisy_image, strength=remove_noise_strength)
            else:
                remove_noise_strength = self.remove_noise_strength_scale.get() / 100
                self.processed_image = self.remove_speckle_noise(self.original_image, strength=remove_noise_strength)

            self.display_image(self.processed_image, self.processed_image_label)
        else:
            messagebox.showerror("Error", "No image selected.")

```

```
    def save_original_image(self):
        if self.original_image is not None:
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png"), ("All files", "*.*")])
            if file_path:
                cv2.imwrite(file_path, self.original_image)
                messagebox.showinfo("Save Original Image", "Original image saved successfully.")
        else:
            messagebox.showerror("Error", "No original image to save.")

    def save_processed_image(self):
        if self.processed_image is not None:
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png"), ("All files", "*.*")])
            if file_path:
                cv2.imwrite(file_path, self.processed_image)
                messagebox.showinfo("Save Processed Image", "Processed image saved successfully.")
        else:
            messagebox.showerror("Error", "No processed image to save.")

    def add_speckle_noise(self, image, mean=0, sigma=0.1, strength=0.5):
        noise = np.random.normal(mean, sigma, image.shape)
        noisy_image = image + image * noise * strength
        noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
        return noisy_image

    def remove_speckle_noise(self, image, d=9, sigma_color=50, sigma_space=50, strength=0.5):
        d = int(d * strength)
        sigma_color = int(sigma_color * strength)
        sigma_space = int(sigma_space * strength)
        return cv2.bilateralFilter(image, d, sigma_color, sigma_space)

def main():
    root = tk.Tk()
    root.title("Speckle Noise")
    root.resizable(False, False)  # Disable resizing
    app = SpeckleNoiseRemoverApp(root)
    root.mainloop()

```

```
import cv2
import numpy as np
import tkinter as tk
from tkinter import filedialog, messagebox, Scale
from PIL import Image, ImageTk


class SpeckleNoiseRemoverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Speckle Noise")
```

```
        # Image display
        self.original_image_label = tk.Label(self.root,
text="Original Image")
        self.original_image_label.grid(row=0, column=0, padx=10,
pady=10)

        self.processed_image_label = tk.Label(self.root,
text="Processed Image")
        self.processed_image_label.grid(row=0, column=1,
padx=10, pady=10)

        # Buttons
        self.choose_image_button = tk.Button(self.root,
text="Choose Image", command=self.choose_image)
        self.choose_image_button.grid(row=1, column=0,
columnspan=2, padx=10, pady=5)

        self.add_noise_button = tk.Button(self.root, text="Add
Noise", command=self.add_noise)
        self.add_noise_button.grid(row=2, column=0, padx=10,
pady=5)

        self.remove_noise_button = tk.Button(self.root,
text="Remove Noise", command=self.remove_noise)
```

```
        self.remove_noise_button.grid(row=2, column=1,
padx=10, pady=5)


        self.save_original_button = tk.Button(self.root, text="Save
Original Image", command=self.save_original_image)
        self.save_original_button.grid(row=3, column=0, padx=10,
pady=5)


        self.save_processed_button = tk.Button(self.root,
text="Save Processed Image",
command=self.save_processed_image)
        self.save_processed_button.grid(row=3, column=1,
padx=10, pady=5)


        # Noise filter strength scales
        self.add_noise_strength_label = tk.Label(self.root,
text="Add Noise Strength:")
        self.add_noise_strength_label.grid(row=4, column=0,
padx=10, pady=5, sticky=tk.E)


        self.add_noise_strength_scale = Scale(self.root, from_=0,
to=200, orient=tk.HORIZONTAL)
        self.add_noise_strength_scale.set(100)
```

```
        self.add_noise_strength_scale.grid(row=4, column=1,
columnspan=2, padx=10, pady=5, sticky=tk.W+tk.E)


        self.remove_noise_strength_label = tk.Label(self.root,
text="Remove Noise Strength:")
        self.remove_noise_strength_label.grid(row=5, column=0,
padx=10, pady=5, sticky=tk.E)


        self.remove_noise_strength_scale = Scale(self.root,
from_=0, to=200, orient=tk.HORIZONTAL)
        self.remove_noise_strength_scale.set(100)
        self.remove_noise_strength_scale.grid(row=5, column=1,
columnspan=2, padx=10, pady=5, sticky=tk.W+tk.E)


        # Image variables
        self.original_image = None
        self.noisy_image = None
        self.processed_image = None


    def choose_image(self):
        file_path = filedialog.askopenfilename()
        if file_path:
```

```
            self.original_image = cv2.imread(file_path,
cv2.IMREAD_GRAYSCALE)
            self.display_image(self.original_image,
self.original_image_label)


    def display_image(self, image, label):
        image = Image.fromarray(image)
        # Resize the image to make it smaller
        image = image.resize((500, 500), Image.BILINEAR)
        image = ImageTk.PhotoImage(image)
        label.config(image=image)
        label.image = image


    def add_noise(self):
        if self.original_image is not None:
            add_noise_strength =
self.add_noise_strength_scale.get() / 100
            self.noisy_image =
self.add_speckle_noise(self.original_image,
strength=add_noise_strength)
            self.display_image(self.noisy_image,
self.processed_image_label)
        else:
```

```
            messagebox.showerror("Error", "No image selected.")


    def remove_noise(self):
        if self.original_image is not None:
            if self.noisy_image is not None:
                remove_noise_strength =
self.remove_noise_strength_scale.get() / 100
                self.processed_image =
self.remove_speckle_noise(self.noisy_image,
strength=remove_noise_strength)
            else:
                remove_noise_strength =
self.remove_noise_strength_scale.get() / 100
                self.processed_image =
self.remove_speckle_noise(self.original_image,
strength=remove_noise_strength)


            self.display_image(self.processed_image,
self.processed_image_label)
        else:
            messagebox.showerror("Error", "No image selected.")


    def save_original_image(self):
```

```
        if self.original_image is not None:
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png"), ("All files", "*.*")])
            if file_path:
                cv2.imwrite(file_path, self.original_image)
                messagebox.showinfo("Save Original Image", "Original image saved successfully.")
        else:
            messagebox.showerror("Error", "No original image to save.")


    def save_processed_image(self):
        if self.processed_image is not None:
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png"), ("All files", "*.*")])
            if file_path:
                cv2.imwrite(file_path, self.processed_image)
                messagebox.showinfo("Save Processed Image", "Processed image saved successfully.")
        else:
```

```
            messagebox.showerror("Error", "No processed image
to save.")


    def add_speckle_noise(self, image, mean=0, sigma=0.1,
strength=0.5):
        noise = np.random.normal(mean, sigma, image.shape)
        noisy_image = image + image * noise * strength
        noisy_image = np.clip(noisy_image, 0,
255).astype(np.uint8)
        return noisy_image


    def remove_speckle_noise(self, image, d=9,
sigma_color=50, sigma_space=50, strength=0.5):
        d = int(d * strength)
        sigma_color = int(sigma_color * strength)
        sigma_space = int(sigma_space * strength)
        return cv2.bilateralFilter(image, d, sigma_color,
sigma_space)


def main():
    root = tk.Tk()
    root.title("Speckle Noise")
```

```
    root.resizable(False, False)  # Disable resizing
    app = SpeckleNoiseRemoverApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```