Project Report

Image Restoration

By
Bhargava Bhatkurse(201105013) & Akash Gauns(201105004)

1. Introduction
The purpose of this project was to develop an image restoration software that utilizes edge inpainting to restore damaged or deteriorated images. The software allows the user to open an image, select a region of interest (ROI), and apply various image processing filters to restore the selected region.

2. Project Overview
The project involved the development of a graphical user interface (GUI) using the Tkinter library in Python. The GUI provides an interactive interface for the user to input the filename of the image to be restored. After opening the image, the user can select a region of interest within the image for restoration.

3. Code Explanation
The code begins by importing the necessary libraries, including OpenCV (cv2) for image processing, NumPy for array operations, Tkinter for GUI development, and PIL (Python Imaging Library) for image display.

The code defines various global variables and initializes them, including variables for storing the ROI coordinates, the image and its copy, and a flag for tracking the drawing state.

The select_roi function is defined to handle mouse events for selecting the ROI. When the left mouse button is pressed, the function records the starting coordinates. When the button is released, the function records the ending coordinates and draws a rectangle on the image copy to indicate the selected ROI.

The open_image function is responsible for opening the specified image file. It reads the filename from the input field, appends the appropriate folder path, and uses OpenCV to read the image. If the image is successfully loaded, a copy of the image is created for display, and a named window is created to show the image. The select_roi function is registered as the mouse callback for the window.

The apply_filters function is called when the user clicks the "Restore" button. It applies various image processing filters to the ROI and displays the restored image. The function first checks if an image is loaded and then reads the values from the sliders and radio buttons for filter parameters. Depending on the threshold mode selected, either local thresholding or thresholding over the full image is applied. Gaussian blur is applied to reduce noise, Canny edge detection is performed, and the edges are dilated. Finally, the inpainting algorithm is used to restore the image within the dilated edges.

The main part of the code creates the Tkinter GUI window and defines the layout. It includes labels for instructions and information, an entry field for the filename, sliders for adjusting filter parameters, radio buttons for selecting the threshold mode, and a button for applying the filters. The Enter key is bound to the open_image function for convenience.

Code:

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

drawing = False
x_start, y_start = 0, 0
x_end, y_end = 0, 0
image = None
image_copy = None

def select_roi(event, x, y, flags, param):  #region of interest
    global x_start, y_start, x_end, y_end, drawing, image_copy

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        x_start, y_start = x, y

    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        x_end, y_end = x, y
        cv2.rectangle(image_copy, (x_start, y_start), (x_end, y_end), (0, 255, 0), 2)
        cv2.imshow("Image", image_copy)

def open_image():
    # Read the filename from the entry field
    filename = filename_entry.get()
    filename = "images/" + filename   #folder name is 'images' where images are stored

    # Read the image
    global image, image_copy
    image = cv2.imread(filename)
    if image is None:
        print("Failed to load the image:", filename)
        return

    # Create a copy of the image for display
    image_copy = image.copy()

    cv2.namedWindow("Image")
    cv2.setMouseCallback("Image", select_roi)
    cv2.imshow("Image", image)

def apply_filters():

    # Apply the filters and process the ROI
    if image is None:
        print("No image loaded.")
        return

    # Read the values from the sliders
    blur_kernel_size = blur_slider.get()
    dilation_iterations = dilation_slider.get()
    threshold_value = threshold_slider.get()
    threshold_type = cv2.THRESH_BINARY if threshold_mode.get() == 0 else cv2.THRESH_BINARY_INV
    # Apply Gaussian blur to reduce noise
    blurred = cv2.GaussianBlur(image, (blur_kernel_size, blur_kernel_size), 0)

    if threshold_mode.get() == 0:  # Local Thresholding
        # Define the coordinates of the ROI
        roi = image[y_start:y_end, x_start:x_end]

        # ROI to grayscale
        gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

        # Apply thresholding to the grayscale ROI
        _, binary_roi = cv2.threshold(gray_roi, threshold_value, 255, threshold_type)

        # Create a mask with the same dimensions as the original image
        mask = np.zeros_like(image, dtype=np.uint8)

        # Insert the thresholded ROI into the mask
        mask[y_start:y_end, x_start:x_end] = cv2.cvtColor(binary_roi, cv2.COLOR_GRAY2BGR)
    else:  # Thresholding over the full image
        # Convert the image to grayscale
        gray_image = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)

        # Apply thresholding to the grayscale image
        _, mask = cv2.threshold(gray_image, threshold_value, 255, threshold_type)
        mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # Perform Canny edge detection
    edges = cv2.Canny(mask, 30, 100)

    # Dilate the edges to make them thicker
    kernel = np.ones((3, 3), np.uint8)
    dilated_edges = cv2.dilate(edges, kernel, iterations=dilation_iterations)

    inpainted = cv2.inpaint(image, dilated_edges, inpaintRadius=3, flags=cv2.INPAINT_NS)

    cv2.imshow("Restored image", inpainted)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Create the main window
root = tk.Tk()
root.title("Image Restoration")
root.geometry("600x600")  # Set the width and height of the window

label = tk.Label(root, text="\n\nThis software restores images using edge inpainting\n")
label.pack()

label = tk.Label(root, text="Made by\nBhargava Bhatkurse and Akash Gauns") #space
label.pack()

label = tk.Label(root, text="\n\nEnter the file name(with extension) and hit enter")
label.pack()

label = tk.Label(root, text="After image opens, select the region to restore")
label.pack()
```

```
# Create the filename entry field
filename_entry = tk.Entry(root)
filename_entry.pack()

# Bind the Enter key to open the image
root.bind('<Return>', lambda event: open_image())

label = tk.Label(root, text="\n") #space
label.pack()

# Create the sliders
blur_slider = tk.Scale(root, from_=1, to=31, orient=tk.HORIZONTAL, label="Blur Kernel Size")
blur_slider.set(13)
blur_slider.pack()

label = tk.Label(root, text="\n") #space
label.pack()

dilation_slider = tk.Scale(root, from_=1, to=10, orient=tk.HORIZONTAL, label="Dilation Iter")
dilation_slider.set(5)
dilation_slider.pack()

label = tk.Label(root, text="\n") #space
label.pack()

threshold_slider = tk.Scale(root, from_=0, to=255, orient=tk.HORIZONTAL, label="Threshold")
threshold_slider.set(250)
threshold_slider.pack()


# Create the threshold mode radio buttons
threshold_mode = tk.IntVar()
threshold_mode.set(0)

local_threshold_radio = tk.Radiobutton(root, text="Thresholding over a Region", variable=threshold_mode,
value=0)
local_threshold_radio.pack()

full_image_threshold_radio = tk.Radiobutton(root, text="Threshold over Full Image",
variable=threshold_mode, value=1)
full_image_threshold_radio.pack()

# Create the apply button
apply_button = tk.Button(root, text="Restore", command=apply_filters)
apply_button.pack()

# Start the Tkinter event loop
root.mainloop()
```

GUI:

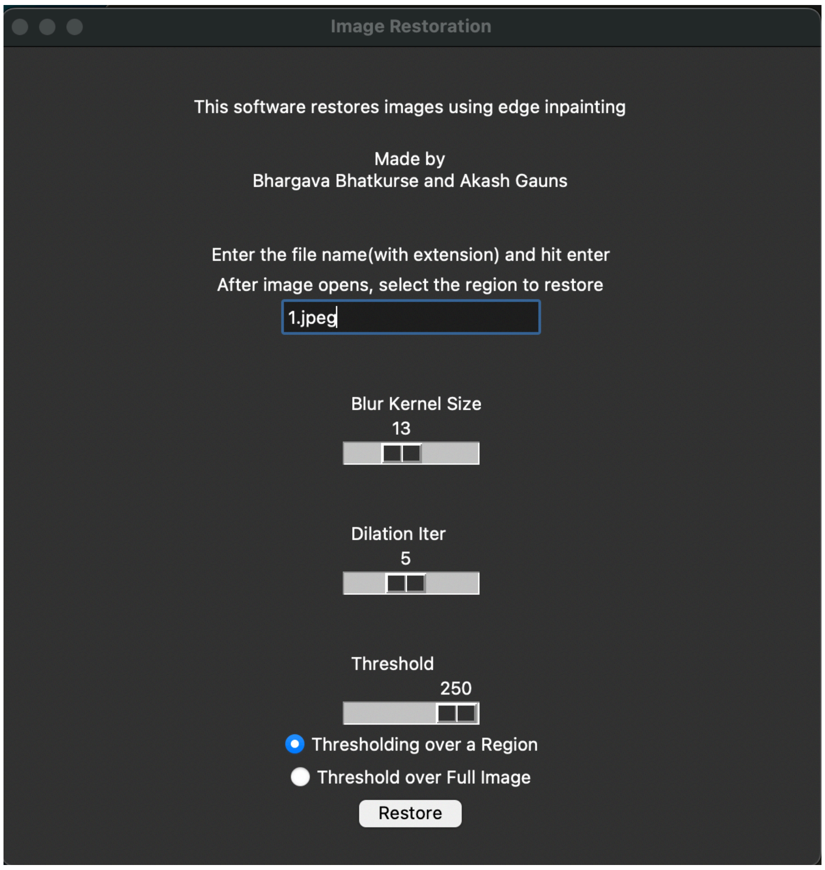


Image Restoration

This software restores images using edge inpainting

Made by
Bhargava Bhatkurse and Akash Gauns

Enter the file name(with extension) and hit enter
After image opens, select the region to restore

1.jpeg

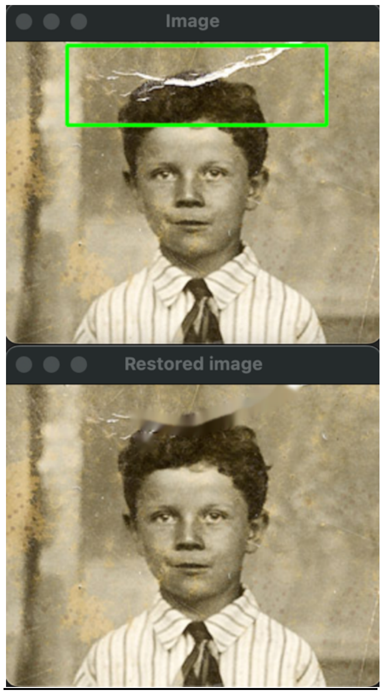
Blur Kernel Size
13

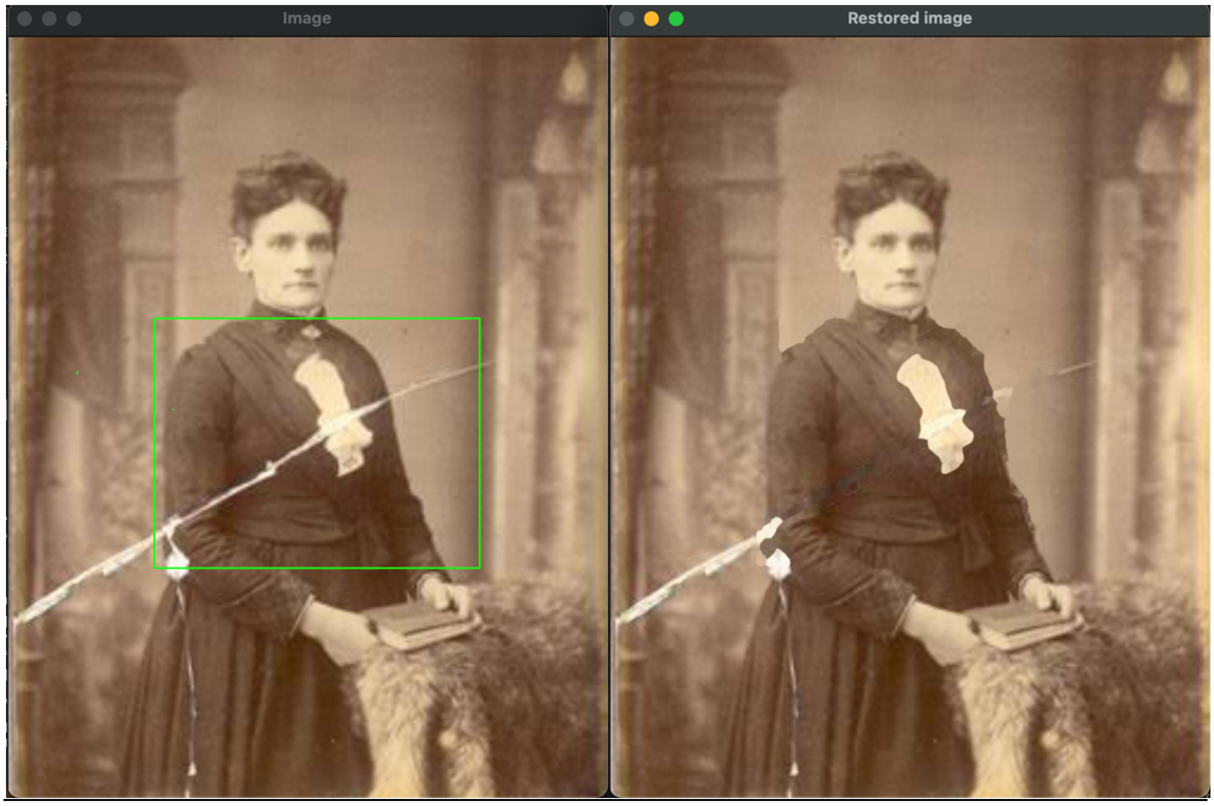
Dilation Iter
5

Threshold
250

[✓ Thresholding over a Region]
Threshold over Full Image

Sample Image restoration:





Conclusion
In conclusion, this project successfully implements an image restoration software using edge inpainting techniques. The software provides a user-friendly interface for opening and restoring images, allowing the user to select a specific region of interest. The implemented filters, such as Gaussian blur, thresholding, edge detection, and inpainting, contribute to the restoration process. The software can be further extended by incorporating additional image processing techniques and advanced inpainting algorithms to enhance the restoration results.