Spatial Filtering in Computer Vision

1. Introduction

In medical imaging, especially when dealing with chest X-rays, one of the main challenges faced by doctors is the lack of clarity in images. X-rays are often blurry, contain noise, and have edges that are not clearly visible. This creates difficulty in identifying small details like fractures, bone structures, and the boundaries of lungs. To deal with this issue, spatial filtering is used as an important image processing technique. Spatial filtering helps in enhancing images by sharpening them, detecting edges, and highlighting sudden intensity changes. Sharpening filters make the image clearer, while derivative operators like Sobel and Laplacian bring out edges and fine details. In this assignment, the goal was to apply different spatial filtering methods on X-ray images to improve their quality and make them easier to analyze.

2. Dataset

For this assignment, I selected a chest X-ray dataset that is publicly available on Kaggle under the title "Chest X-ray Pneumonia Dataset." The dataset contains grayscale images in PNG and JPEG formats, with an approximate resolution of 512 by 512 pixels. These images are highly suitable for spatial filtering experiments because X-rays naturally contain both sharp edges, such as ribs and bones, and smooth intensity regions, like soft tissues. The reason for selecting this dataset is that it allows a good balance between smooth areas and sharp boundaries, which makes it ideal for testing filters like Laplacian, Sobel, and unsharp masking. Since spatial filtering generally works on pixel intensity values, only grayscale images from this dataset were used.

3. Methodology & Justification

I implemented and tested three main spatial filtering techniques**:**

Technique 1: Laplacian Filter (Second Derivative))

- **Justification** The Laplacian filter is a second derivative operator that highlights regions of rapid intensity change. It is useful for detecting edges and fine details in chest X-rays, such as bone outlines and possible fractures. However, one drawback is that it may also enhance noise in the image.
- **Mathematical Kernel:**

$$\nabla^2 f(x, y) = \left(\partial^2 f — \partial x^2\right) + \left(\partial^2 f — \partial y^2\right)$$

- **Discrete Kernel (4-neighborhood):**

The discrete kernel of the Laplacian filter in 4-neighborhood form is:

[0 -1 0]
[-1 4 -1]
[0 -1 0]

This kernel enhances the sharp changes in the image, which is why it is suitable for detecting fine fractures and details in X-rays.

Technique 2: Sobel Operator (First Derivative)

Justification:
The Sobel operator detects edges in horizontal and vertical directions by calculating intensity gradients. It is very effective for finding the boundaries of ribs and lungs in chest X-rays. However, it mainly produces edge maps, which means smooth details are not preserved.

Transformation Function:

$$G = \sqrt{}(Gx^2 + Gy^2)$$

Sobel Kernels:

Gx =
[-1 0 +1]
[-2 0 +2]
[-1 0 +1]

Gy =
[-1 -2 -1]
[0 0 0]
[+1 +2 +1]

Technique 3: Unsharp Masking (Sharpening Filter)

Justification:
Unsharp masking is used to sharpen blurry images. It works by subtracting a blurred version of the image from the original and then adding the difference back. This method is suitable for X-rays because it sharpens the important details without making the image look unnatural.

Transformation Function:

$$g(x,y) = f(x,y) + k * (f(x,y) - fblur(x,y))$$

Where:

- f(x,y) = original image
- fblur(x,y) = blurred image
- k = sharpening constant (0.5 to 2.0)

4. Results and Analysis

Visual Comparison of Filters

The original X-ray image looked a little blurred, and the ribs and lungs were not very clear. With the Laplacian filter, the image became sharper and small fractures were easier to see, but it also added too much background noise, which reduced the overall quality. The Sobel operator highlighted the edges of ribs and lungs clearly, but the rest of the image lost its smoothness and looked more like an edge map, making it less useful in medical viewing. Unsharp masking gave the best results as it made the image sharper and clearer without adding noise, keeping the image natural and easy to understand.

Critical Evaluation

From the results, it is clear that each filter has both advantages and disadvantages. The Laplacian is good for highlighting very fine details but creates too much noise. The Sobel operator is strong for edge detection but does not give a smooth image for practical use. Unsharp masking balances sharpness and natural appearance, which makes it more reliable for medical images.

Technique Combination

A combination of these techniques can also be useful. For example, Sobel can be applied first to detect edges, and then unsharp masking can be used to improve clarity while keeping the image natural. Such combinations can give better results where both edge details and overall smoothness are required.

Overall Findings

Overall, unsharp masking proved to be the most effective single method for enhancing chest X-rays. However, using a smart combination of filters can sometimes improve results further, depending on the need.

5. Conclusion

In conclusion, I applied three different spatial filtering techniques on chest X-ray images. These techniques were the Laplacian filter, the Sobel operator, and the Unsharp Masking method. The

Laplacian filter made the image very sharp but increased the noise. The Sobel operator was very effective in edge detection but not in creating a natural-looking image. Unsharp masking provided the best overall result because it enhanced the sharpness without making the image unnatural.

From these results, I concluded that unsharp masking is the most effective method for X-ray enhancement in medical imaging. It provides a good balance between sharpness and clarity, which is very important for diagnosis.

6. Appendix (Code)

The code for this assignment was written in Python using Tkinter for GUI, OpenCV for image processing, and other supporting libraries. Tkinter was used to create the interface, Pillow was used for displaying images inside Tkinter, OpenCV was used for applying Laplacian, Sobel, and Unsharp filters, NumPy helped in handling pixel values, and Matplotlib was used for displaying histograms.

Code:

```python
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from PIL import Image, ImageTk
import cv2
import numpy as np
import os
from typing import Optional

BG_COLOR = "#2e2e2e"
FRAME_COLOR = "#3c3c3c"
TEXT_COLOR = "#dcdcdc"
ACCENT_COLOR = "#007acc"
ACCENT_HOVER = "#009cff"
CANVAS_BG = "#1c1c1c"
OUTPUT_DIR = "output_filtered"
DEBOUNCE_DELAY_MS = 200

class SpatialFilteringApp:
    def __init__(self, root: tk.Tk):
        self.root = root
        self.root.title("Spatial Filtering Lab (Stable Version)")
        self.root.geometry("1600x900")
        self.root.configure(bg=BG_COLOR)

        self.original_image: Optional[np.ndarray] = None
        self.processed_image: Optional[np.ndarray] = None
        self.tk_original_image: Optional[ImageTk.PhotoImage] = None
        self.tk_processed_image: Optional[ImageTk.PhotoImage] = None
        self.original_filename: Optional[str] = None
        self.last_canvas_sizes: dict = {}
        self.debounce_timer: Optional[str] = None

        self.setup_styles()
        self.setup_gui()

        if not os.path.exists(OUTPUT_DIR):
            os.makedirs(OUTPUT_DIR)

    def setup_styles(self):
        style = ttk.Style()
        style.theme_use('clam')
        style.configure('.', background=BG_COLOR, foreground=TEXT_COLOR, 
fieldbackground=FRAME_COLOR, borderwidth=1)
        style.configure('TFrame', background=BG_COLOR)
        style.configure('TLabel', background=BG_COLOR, foreground=TEXT_COLOR, 
font=('Segoe UI', 10))
        style.configure('TRadiobutton', background=FRAME_COLOR, 
foreground=TEXT_COLOR, font=('Segoe UI', 10))
        style.map('TRadiobutton', background=[('active', BG_COLOR)])
        style.configure('TButton', background=ACCENT_COLOR, foreground='white', 
font=('Segoe UI', 10, 'bold'), borderwidth=0)
        style.map('TButton', background=[('active', ACCENT_HOVER)])
        style.configure('TLabelframe', background=FRAME_COLOR, 
bordercolor=FRAME_COLOR)
        style.configure('TLabelframe.Label', background=FRAME_COLOR, 
foreground=TEXT_COLOR, font=('Segoe UI', 11, 'bold'))
        style.configure('Horizontal.TScale', background=FRAME_COLOR, 
troughcolor=BG_COLOR)

    def setup_gui(self):
        main_pane = ttk.PanedWindow(self.root, orient=tk.HORIZONTAL)
        main_pane.pack(fill=tk.BOTH, expand=True, padx=15, pady=15)

        control_frame = ttk.Labelframe(main_pane, text="Controls", 
style='TLabelframe')
        main_pane.add(control_frame, weight=2)

        display_pane = ttk.PanedWindow(main_pane, orient=tk.VERTICAL)
        main_pane.add(display_pane, weight=5)

        original_frame, self.original_canvas, self.original_info_label = 
self._create_display_canvas(display_pane, "Original Image")
```

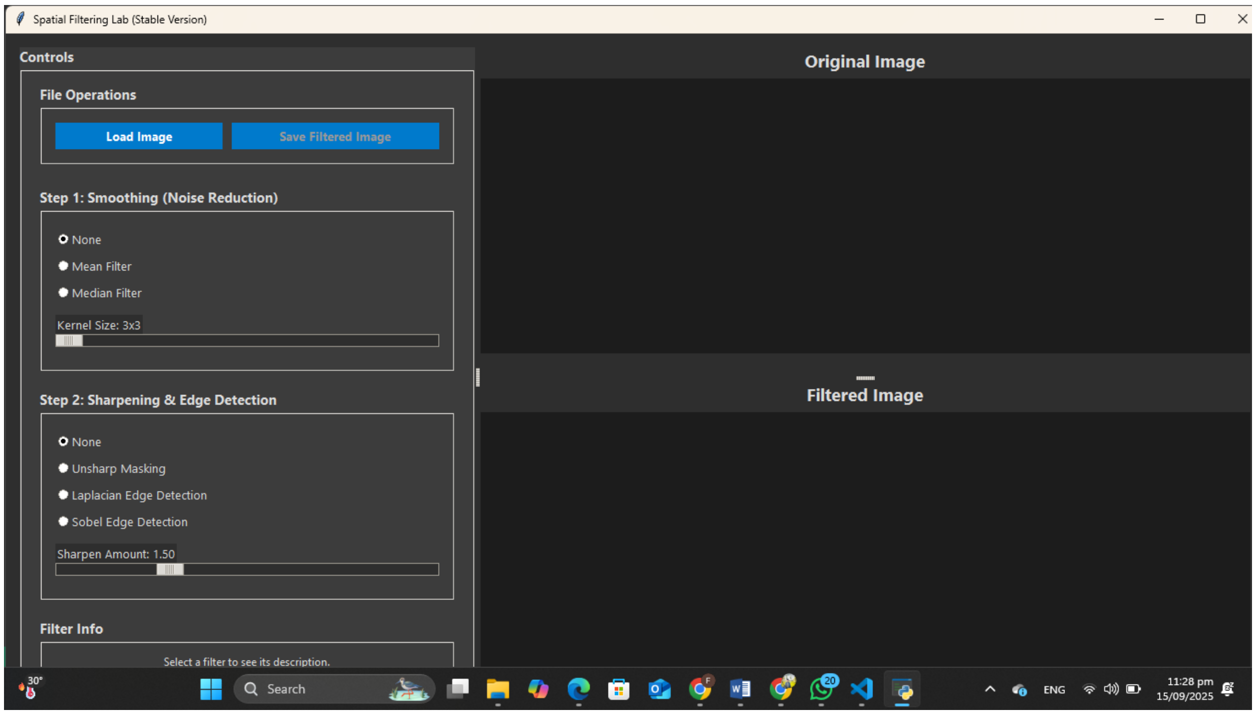
```
        display_pane.add(original_frame, weight=1)

        processed_frame, self.processed_canvas, self.processed_info_label = 
self._create_display_canvas(display_pane, "Filtered Image")
        display_pane.add(processed_frame, weight=1)

        self.setup_control_widgets(control_frame)

        self.original_canvas.bind('<Configure>', self.on_canvas_resize)
        self.processed_canvas.bind('<Configure>', self.on_canvas_resize)

    def setup_control_widgets(self, parent_frame: ttk.Labelframe):
        parent_frame['padding'] = (20, 15)

        file_frame = ttk.LabelFrame(parent_frame, text="File Operations", 
style='TLabelframe', padding=10)
        file_frame.pack(fill=tk.X, pady=(0, 15), ipady=5)
        self.load_button = ttk.Button(file_frame, text="Load Image", 
command=self.load_image, style='TButton')
        self.load_button.pack(side=tk.LEFT, expand=True, fill=tk.X, padx=5)
        self.save_button = ttk.Button(file_frame, text="Save Filtered Image", 
command=self.save_output, state=tk.DISABLED, style='TButton')
        self.save_button.pack(side=tk.LEFT, expand=True, fill=tk.X, padx=5)

        self.sharpen_var = tk.StringVar(value="None")
        self.smooth_var = tk.StringVar(value="None")

        smooth_frame = ttk.LabelFrame(parent_frame, text="Step 1: Smoothing 
(Noise Reduction)", style='TLabelframe', padding=15)
        smooth_frame.pack(fill=tk.X, pady=10)
        smooth_techniques = [("None", "None"), ("Mean Filter", "mean"), ("Median 
Filter", "median")]
        for text, value in smooth_techniques:
            ttk.Radiobutton(smooth_frame, text=text, variable=self.smooth_var, 
value=value, command=self.on_filter_change).pack(anchor=tk.W, pady=3)

        self.kernel_label = ttk.Label(smooth_frame, text="Kernel Size: 3x3")
        self.kernel_label.pack(anchor=tk.W, pady=(10,0))
        self.kernel_var = tk.IntVar(value=3)
        self.kernel_slider = ttk.Scale(smooth_frame, from_=3, to=31, 
variable=self.kernel_var, orient=tk.HORIZONTAL, command=self.on_slider_move)
        self.kernel_slider.pack(fill=tk.X, pady=(0, 10))

        sharpen_frame = ttk.LabelFrame(parent_frame, text="Step 2: Sharpening & 
Edge Detection", style='TLabelframe', padding=15)
```

```
        sharpen_frame.pack(fill=tk.X, pady=10)
        sharpen_techniques = [("None", "None"), ("Unsharp Masking", 
"unsharp_mask"), ("Laplacian Edge Detection", "laplacian"), ("Sobel Edge 
Detection", "sobel")]
        for text, value in sharpen_techniques:
            ttk.Radiobutton(sharpen_frame, text=text, variable=self.sharpen_var, 
value=value, command=self.on_filter_change).pack(anchor=tk.W, pady=3)

        self.amount_label = ttk.Label(sharpen_frame, text="Sharpen Amount: 1.50")
        self.amount_label.pack(anchor=tk.W, pady=(10,0))
        self.amount_var = tk.DoubleVar(value=1.5)
        self.amount_slider = ttk.Scale(sharpen_frame, from_=0.1, to=5.0, 
variable=self.amount_var, orient=tk.HORIZONTAL, command=self.on_slider_move)
        self.amount_slider.pack(fill=tk.X, pady=(0, 10))

        info_frame = ttk.LabelFrame(parent_frame, text="Filter Info", 
style='TLabelframe', padding=10)
        info_frame.pack(fill=tk.BOTH, expand=True, pady=10)
        self.info_text = tk.Label(info_frame, text="Select a filter to see its 
description.", wraplength=350, justify=tk.LEFT, background=FRAME_COLOR, 
foreground=TEXT_COLOR)
        self.info_text.pack(fill=tk.BOTH, expand=True)

        self.reset_button = ttk.Button(parent_frame, text="Reset All Filters", 
command=self.reset_image, state=tk.DISABLED, style='TButton')
        self.reset_button.pack(fill=tk.X, side=tk.BOTTOM, pady=10, ipady=5)

        self.on_filter_change()

    def on_filter_change(self, *args):
        self.update_ui_visibility()
        self.apply_filters()
        self.update_filter_info()

    def update_ui_visibility(self):
        self.kernel_slider.state(['!disabled' if self.smooth_var.get() != 'None' 
else 'disabled'])
        self.amount_slider.state(['!disabled' if self.sharpen_var.get() == 
'unsharp_mask' else 'disabled'])

    def _create_display_canvas(self, parent, title):
        frame = ttk.Frame(parent, style='TFrame')
        label = ttk.Label(frame, text=title, font=('Segoe UI', 14, 'bold'))
        label.pack(pady=(0, 5))
        canvas = tk.Canvas(frame, bg=CANVAS_BG, relief=tk.FLAT, bd=0, 
highlightthickness=0)
        canvas.pack(fill=tk.BOTH, expand=True)
        info_label = ttk.Label(frame, text="", font=('Segoe UI', 9))
        info_label.pack(pady=(5, 0))
        return frame, canvas, info_label

    def on_slider_move(self, value):
        self.amount_label.config(text=f"Sharpen Amount: 
{self.amount_var.get():.2f}")
        kernel_size = round(float(self.kernel_var.get()))
        if kernel_size % 2 == 0: kernel_size += 1
        self.kernel_var.set(kernel_size)
        self.kernel_label.config(text=f"Kernel Size: 
{kernel_size}x{kernel_size}")

        if self.debounce_timer: self.root.after_cancel(self.debounce_timer)
        self.debounce_timer = self.root.after(DEBOUNCE_DELAY_MS, 
self.apply_filters)

    def apply_filters(self):
        if self.original_image is None: return
        if self.debounce_timer:
            self.root.after_cancel(self.debounce_timer)
            self.debounce_timer = None

        processed = self._process_image(self.original_image, 
self.smooth_var.get(), self.sharpen_var.get())

        self.processed_image = processed
        self.display_image(self.processed_image, self.processed_canvas, 
'processed', self.processed_info_label)

    def _process_image(self, image: np.ndarray, smooth_filter: str, 
sharpen_filter: str) -> np.ndarray:
        img_float = image.astype(np.float32)

        if smooth_filter != "None":
            ksize = self.kernel_var.get()
            if smooth_filter == "mean":
                img_float = cv2.blur(img_float, (ksize, ksize))
            elif smooth_filter == "median":
                median_blurred_8bit = cv2.medianBlur(image, ksize)
                img_float = median_blurred_8bit.astype(np.float32)
        if sharpen_filter != "None":
            if sharpen_filter == "unsharp_mask":
                blur_float = cv2.GaussianBlur(img_float, (0, 0), 2.0)
                amount = self.amount_var.get()
                img_float = cv2.addWeighted(img_float, 1 + amount, blur_float, -
amount, 0)
            elif sharpen_filter == "laplacian":
                lap = cv2.Laplacian(img_float, cv2.CV_32F, ksize=3)
                img_float = cv2.convertScaleAbs(lap)
            elif sharpen_filter == "sobel":
                sobelx = cv2.Sobel(img_float, cv2.CV_32F, 1, 0, ksize=5)
                sobely = cv2.Sobel(img_float, cv2.CV_32F, 0, 1, ksize=5)
                magnitude = cv2.magnitude(sobelx, sobely)
                img_float = cv2.convertScaleAbs(magnitude)

        processed_final = np.clip(img_float, 0, 255)
        return processed_final.astype(np.uint8)

    def update_filter_info(self):
        sharp_filter = self.sharpen_var.get()
        smooth_filter = self.smooth_var.get()
        info = "Select a filter to see its description."

        if sharp_filter != "None":
            if sharp_filter == "unsharp_mask": info = "Unsharp Masking:\nThe
standard method for sharpening. Enhances edges by adding a scaled version of the
image's details back into the original.\n\nFormula: S = O + A*(O -
B)\n(S=Sharpened, O=Original, A=Amount, B=Blurred)"
            elif sharp_filter == "laplacian": info = "Laplacian (2nd
Derivative):\nA common kernel is:\n[[0, 1, 0],\n [1,-4, 1],\n [0, 1, 0]]\nIt's
very sensitive to noise and produces a detailed edge map."
            elif sharp_filter == "sobel": info = "Sobel (1st
Derivative):\nCalculates the gradient using two kernels (for x and y axes) to
find edges. Provides a cleaner edge map than the Laplacian."
        elif smooth_filter != "None":
            if smooth_filter == "mean": info = "Mean (Averaging)
Filter:\nReplaces each pixel with the average of its neighbors. A 3x3 kernel
is:\n1/9 * [[1, 1, 1],\n       [1, 1, 1],\n       [1, 1, 1]]"
            elif smooth_filter == "median": info = "Median Filter:\nReplaces each
pixel with the median value of its neighbors. Not a kernel-based convolution, but
very effective at removing salt-and-pepper noise while preserving edges."

        self.info_text.config(text=info)

    def display_image(self, image_data, canvas, image_type, info_label):
        canvas.delete("all")
        canvas_w, canvas_h = canvas.winfo_width(), canvas.winfo_height()
        if image_data is None or canvas_w < 2 or canvas_h < 2: return

        img_h, img_w = image_data.shape[:2]
        aspect = img_w / img_h
        new_w, new_h = (canvas_w, int(canvas_w / aspect)) if (canvas_w / aspect)
<= canvas_h else (int(canvas_h * aspect), canvas_h)
        if new_w < 1 or new_h < 1: return

        resized_img = cv2.resize(image_data, (int(new_w), int(new_h)),
interpolation=cv2.INTER_AREA)
        photo_img = ImageTk.PhotoImage(image=Image.fromarray(resized_img))

        if image_type == 'original': self.tk_original_image = photo_img
        else: self.tk_processed_image = photo_img

        x, y = (canvas_w - new_w) / 2, (canvas_h - new_h) / 2
        canvas.create_image(x, y, anchor=tk.NW, image=photo_img)
        info_label.config(text=f"Dimensions: {img_w} x {img_h} px")

    def on_canvas_resize(self, event: tk.Event):
        canvas = event.widget
        canvas_w, canvas_h = canvas.winfo_width(), canvas.winfo_height()
        if self.last_canvas_sizes.get(id(canvas)) == (canvas_w, canvas_h): return

        self.last_canvas_sizes[id(canvas)] = (canvas_w, canvas_h)
        if canvas == self.original_canvas and self.original_image is not None:
            self.display_image(self.original_image, self.original_canvas,
'original', self.original_info_label)
        elif canvas == self.processed_canvas and self.processed_image is not
None:
            self.display_image(self.processed_image, self.processed_canvas,
'processed', self.processed_info_label)

    def load_image(self):
        file_path = filedialog.askopenfilename(filetypes=[("Image Files",
"*.png;*.jpg;*.jpeg;*.bmp;*.tif;*.dcm")])
        if not file_path: return
        try:
            img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
            if img is None: raise ValueError("File is not a valid image.")
            self.original_image = img
            self.original_filename = os.path.basename(file_path)
            self.reset_image()
```

```python
                self.save_button['state'] = tk.NORMAL
                self.reset_button['state'] = tk.NORMAL
            except Exception as e:
                messagebox.showerror("Error", f"Failed to load image: {e}")

        def save_output(self):
            if self.processed_image is None: return
            smooth_f = self.smooth_var.get()
            sharp_f = self.sharpen_var.get()
            if smooth_f == "None" and sharp_f == "None":
                messagebox.showinfo("Info", "No filter applied. Cannot save.")
                return

            base_name, _ = os.path.splitext(self.original_filename)
            suffix = ""
            if smooth_f != 'None': suffix += f"_{smooth_f}_k{self.kernel_var.get()}"
            if sharp_f != 'None':
                if sharp_f == 'unsharp_mask': suffix +=
f"_unsharp_a{self.amount_var.get():.2f}"
                else: suffix += f"_{sharp_f}"

            new_filename = f"{base_name}{suffix}.png"
            save_path = os.path.join(OUTPUT_DIR, new_filename)
            try:
                cv2.imwrite(save_path, self.processed_image)
                messagebox.showinfo("Success", f"Image saved to '{OUTPUT_DIR}'
as:\n{new_filename}")
            except Exception as e: messagebox.showerror("Error", f"Failed to save
file: {e}")

        def reset_image(self):
            if self.original_image is not None:
                self.sharpen_var.set("None")
                self.smooth_var.set("None")
                self.last_canvas_sizes = {}
                self.on_filter_change()
                self.display_image(self.original_image, self.original_canvas,
'original', self.original_info_label)

if __name__ == "__main__":
    root = tk.Tk()
    app = SpatialFilteringApp(root)
    root.mainloop()
```

Outputs:

Before uploading image



Spatial Filtering Lab (Stable Version)

Controls

File Operations

Load Image | Save Filtered Image

Step 1: Smoothing (Noise Reduction)

[✓ None]
Mean Filter
Median Filter

Kernel Size: 3x3

Step 2: Sharpening & Edge Detection

[✓ None]
Unsharp Masking
Laplacian Edge Detection
Sobel Edge Detection

Sharpen Amount: 1.50

Filter Info

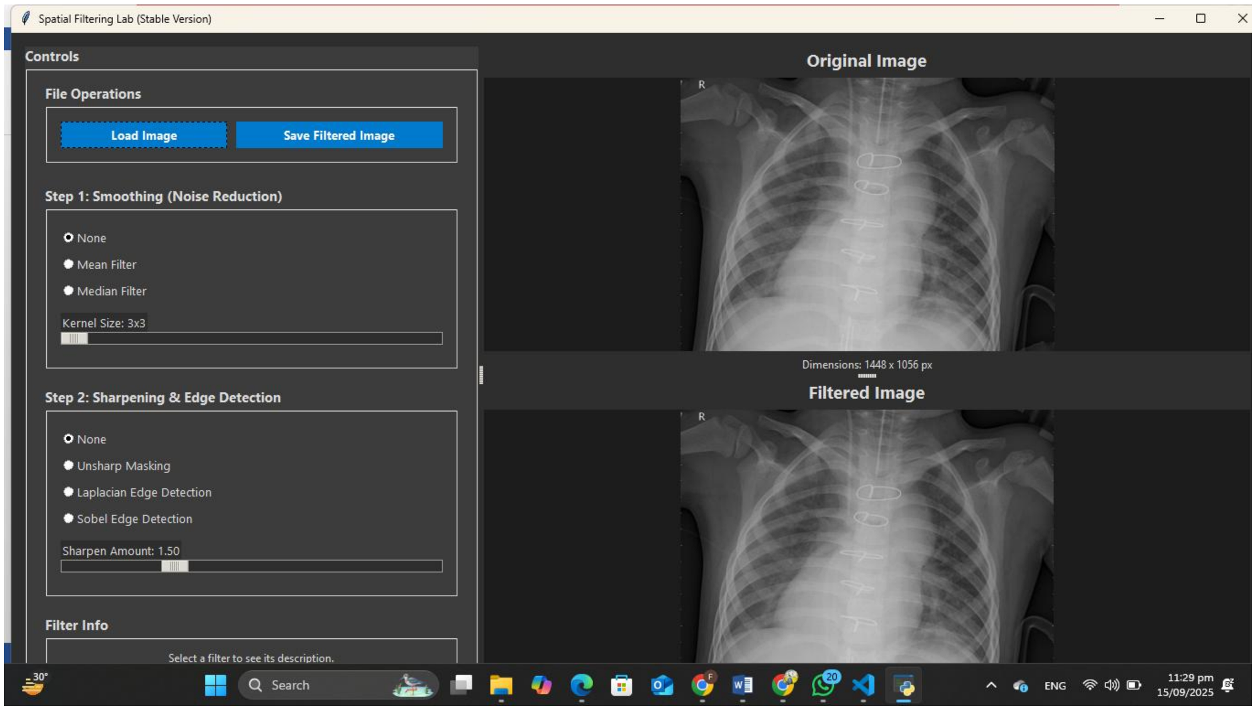
Select a filter to see its description.

Original Image

Filtered Image

30° Search ENG 11:28 pm 15/09/2025

After uploading image



Spatial Filtering Lab (Stable Version)

Controls

File Operations

Load Image | Save Filtered Image

Step 1: Smoothing (Noise Reduction)

[✓ None]
Mean Filter
Median Filter

Kernel Size: 3x3

Step 2: Sharpening & Edge Detection

[✓ None]
Unsharp Masking
Laplacian Edge Detection
Sobel Edge Detection

Sharpen Amount: 1.50

Filter Info

Select a filter to see its description.

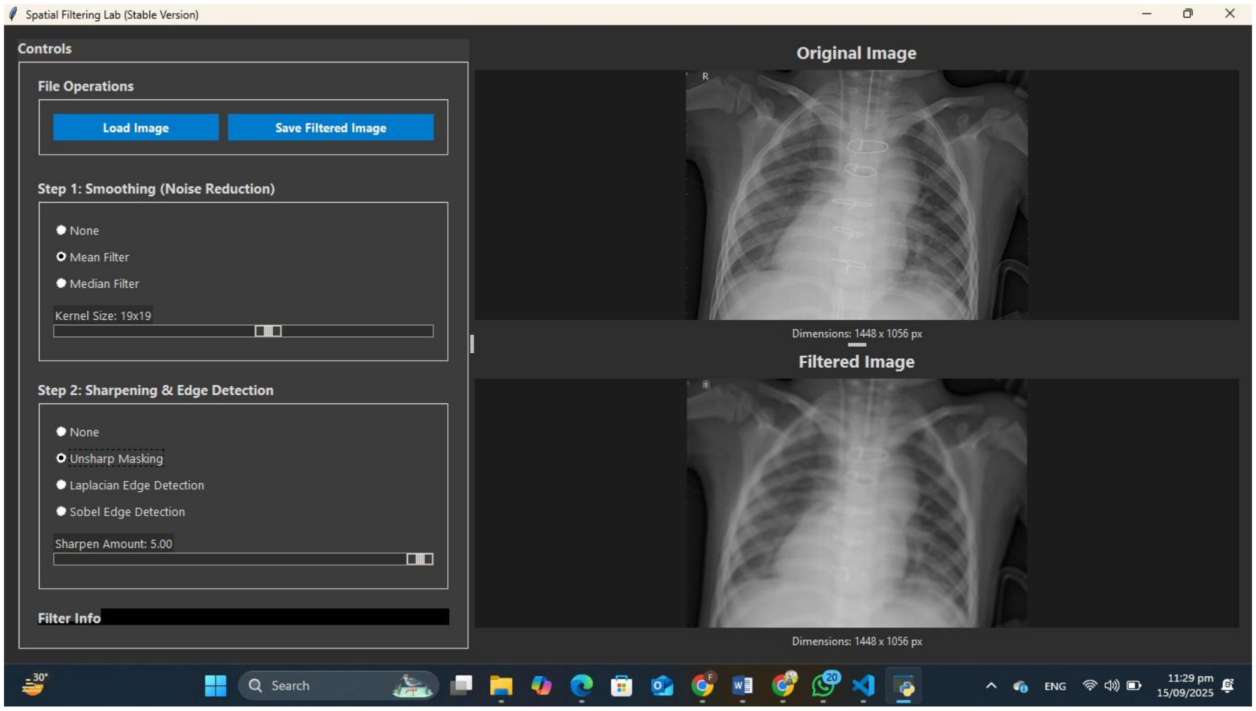
Original Image

Dimensions: 1448 x 1056 px

Filtered Image

30° Search ENG 11:29 pm 15/09/2025

Apply mean


Spatial Filtering Lab (Stable Version)

Controls

File Operations

Load Image | Save Filtered Image

Step 1: Smoothing (Noise Reduction)

- None
- Mean Filter [✓]
- Median Filter

Kernel Size: 19x19

Step 2: Sharpening & Edge Detection

- None
- Unsharp Masking [✓]
- Laplacian Edge Detection
- Sobel Edge Detection

Sharpen Amount: 5.00

Filter Info

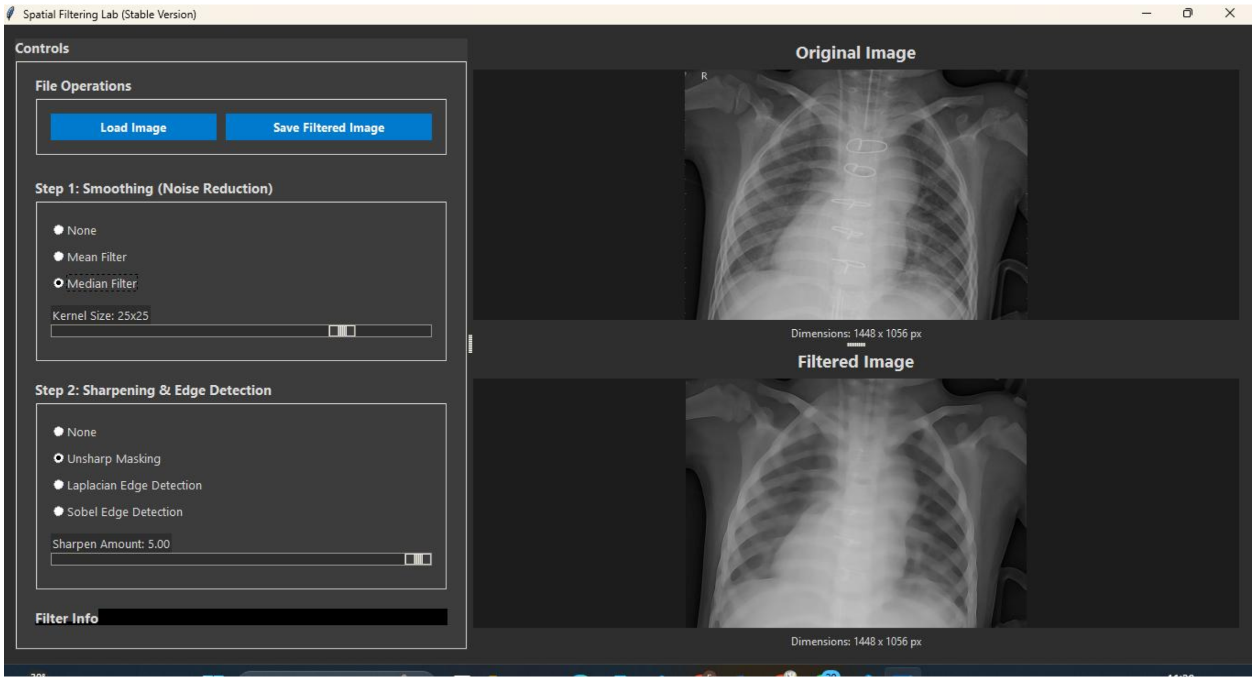
Original Image

Dimensions: 1448 x 1056 px

Filtered Image

Dimensions: 1448 x 1056 px

Apply median & unsharping masking


Spatial Filtering Lab (Stable Version)

Controls

File Operations

Load Image | Save Filtered Image

Step 1: Smoothing (Noise Reduction)

- None
- Mean Filter
- Median Filter [✓]

Kernel Size: 25x25

Step 2: Sharpening & Edge Detection

- None
- Unsharp Masking [✓]
- Laplacian Edge Detection
- Sobel Edge Detection

Sharpen Amount: 5.00

Filter Info

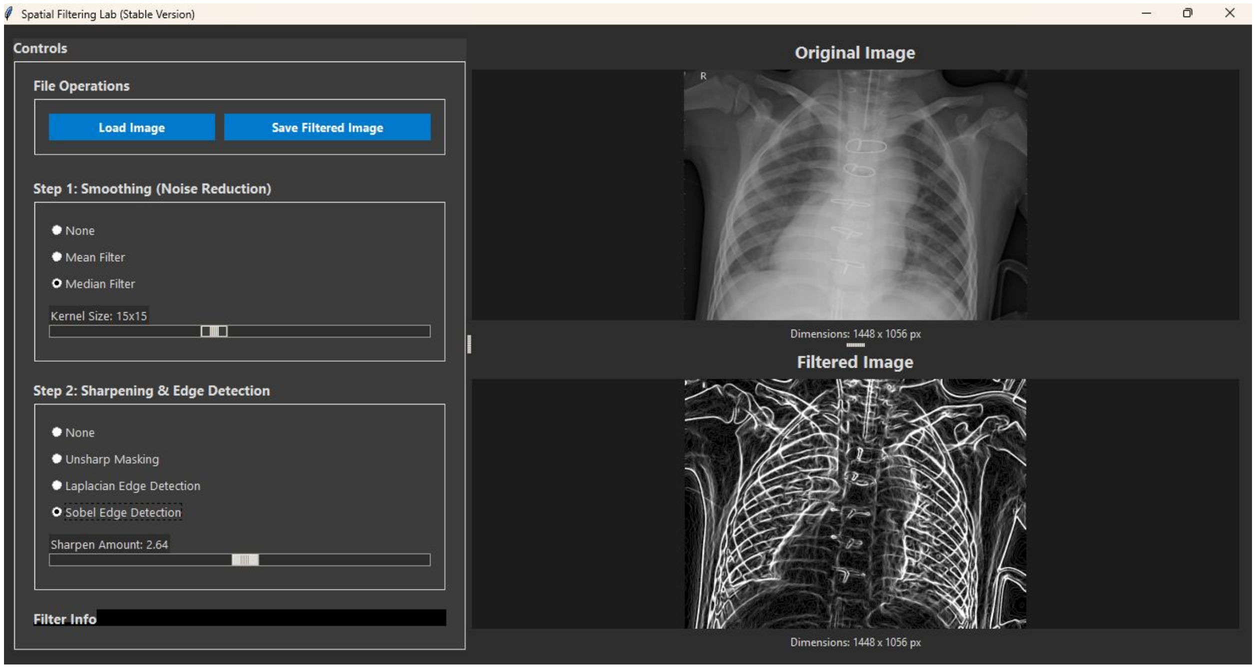
Original Image

Dimensions: 1448 x 1056 px

Filtered Image

Dimensions: 1448 x 1056 px

Apply median & sobal edge



Spatial Filtering Lab (Stable Version)

Controls

File Operations

Load Image | Save Filtered Image

Step 1: Smoothing (Noise Reduction)

- None
- Mean Filter
- [✓ Median Filter]

Kernel Size: 15x15

Step 2: Sharpening & Edge Detection

- None
- Unsharp Masking
- Laplacian Edge Detection
- [✓ Sobel Edge Detection]

Sharpen Amount: 2.64

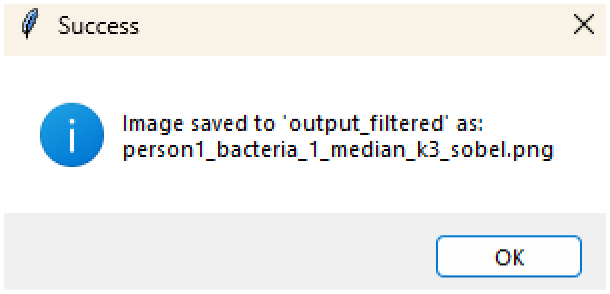
Filter Info

Original Image

Dimensions: 1448 x 1056 px

Filtered Image

Dimensions: 1448 x 1056 px



Success

Image saved to 'output_filtered' as:
person1_bacteria_1_median_k3_sobel.png

OK

Final Saved Output: