Assignment 1: Real-world Image Enhancement and Analysis

Name: Muhammad Sohaib

Roll No: 22F-3302

Section: BCS-7F

---

1. Introduction (The Problem)

X-rays are one of the most common medical imaging tools used by doctors to check bones, lungs, and other internal structures. However, sometimes these X-ray images have low contrast, which makes it difficult to clearly see important details like small fractures, ribs, or other internal features.

This problem can affect diagnosis because doctors rely on sharp, clear images. My goal in this assignment was to try some image enhancement techniques to improve the visibility of low-contrast X-rays. By doing so, we can make the bones and other important parts appear more clearly, which can help in better analysis.

2. Dataset

For this assignment, I used publicly available chest X-ray images. The dataset I referred to is from Kaggle's Chest X-ray dataset, which contains grayscale images in formats like PNG and JPEG. Most of the images are in the resolution range of 512×512 pixels. I worked with grayscale images only, since X-ray images don't usually contain color. The dataset is good for this task because chest X-rays often have areas that are too dark or too bright, making them perfect examples to test enhancement methods.

3. Methodology & Justification

Technique 1: Histogram Equalization

- **Justification:** Histogram equalization is a simple but powerful method to improve contrast. It works by spreading out the pixel intensity distribution so that dark and bright regions become more balanced. This is good for medical images where the contrast between tissues is important.
- **Transformation Function:** The function can be represented as:

$$s_k = \frac{(L - 1)}{MN} \sum_{j=0}^{k} n$$

$Where$:

- $s\_k = new\ intensity\ value$
- $L = total\ number\ of\ intensity\ levels\ (usually\ 256\ for\ grayscale)$
- $M \times N = total\ pixels\ in\ the\ image$
- $n\_j = number\ of\ pixels\ with\ intensity$ j

Technique 2: Power-Law (Gamma) Transformation

- **Justification:** Gamma correction is useful for adjusting the brightness of images. For X-rays that are too dark or too bright, choosing the right gamma value can reveal hidden details. If gamma < 1, the image becomes brighter, and if gamma > 1, the image becomes darker.
- **Transformation Function:**

$$\boldsymbol{s = c \cdot r^{\{\gamma\}}}$$

This brightened the darker parts of the image and made bones stand out.

4. Results & Analysis

Visual Comparison

Original Image: Looked dull with bones and ribs not very clear.

When applying Histogram Equalization**,** the image became much clearer overall, but sometimes the enhancement was too strong. While bones and tissues became more visible, small noise in the background was also enhanced.

When applying Gamma Transformation ($\gamma = 0.5$)**,** the darker parts of the X-ray were brightened in a smoother way. This made subtle details visible without making the image look artificial.

Histogram Analysis

- Original Image: The histogram was clustered in a narrow range, meaning low contrast.
- Histogram Equalization: The histogram spread out more evenly across all intensities, which shows better utilization of the dynamic range.
- Gamma Correction ($\gamma < 1$): The histogram shifted towards brighter intensities, which shows that dark pixels became lighter.

Critical Evaluation

Between the two, Gamma Transformation worked better for my X-ray images. Histogram equalization sometimes made the image look too harsh, while gamma correction gave smoother enhancement and revealed hidden details in darker regions.

Technique Combination

I also tried combining the two methods: first applying gamma correction ($\gamma = 0.6$), and then histogram equalization. The result was a much brighter image with strong contrast. However, in some cases it over-enhanced the image, which might not be suitable for medical diagnosis. So while the combination can work, it needs to be applied carefully.

5. Conclusion

In conclusion, both histogram equalization and gamma transformation improved the visibility of the X-ray images. For my dataset, gamma correction was more effective because it revealed details in low-intensity regions without adding too much noise. If I had to recommend one method, I would suggest gamma transformation with a carefully chosen value of gamma.

6. Appendix (Code)

The code for this project was written in Python using Tkinter for the GUI. It includes two main options:

1. **Histogram Equalization**
2. **Gamma Transformation**

The program allows the user to load an X-ray image, apply the enhancement, and compare the results with histograms.

Work of Each Library

- tkinter → for building the main GUI.
- Pillow → to display images inside Tkinter.
- OpenCV (cv2) → for image enhancement methods like histogram equalization.
- NumPy → to perform pixel-level array operations.
- Matplotlib → to plot and compare histograms.
- typing → just for cleaner function definitions (not required for execution).

Code:

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from PIL import Image, ImageTk
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from typing import Optional, Tuple
```

```
BG_COLOR = "#2e2e2e"
FRAME_COLOR = "#3c3c3c"
TEXT_COLOR = "#dcdcdc"
ACCENT_COLOR = "#007acc"
ACCENT_HOVER = "#009cff"
ERROR_COLOR = "#e74c3c"
CANVAS_BG = "#1c1c1c"

HISTOGRAM_PIXEL_LIMIT = 250000

class ImageEnhancerApp:
    def __init__(self, root: tk.Tk):
        self.root = root
        self.root.title("Medical Image Enhancement Studio")
        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.debounce_timer: Optional[str] = None
        self.lut_cache: dict = {}
        self.last_canvas_sizes: dict = {}

        self.setup_styles()
        self.setup_gui()

    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_frame = ttk.Frame(main_pane, style='TFrame')
        main_pane.add(display_frame, weight=5)

        self.setup_control_widgets(control_frame)
        self.setup_display_widgets(display_frame)

    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, 20))
                 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, ipady=5)
          self.save_button = ttk.Button(file_frame, text="Save Enhanced Image",
command=self.save_image, state=tk.DISABLED, style='TButton')
      self.save_button.pack(side=tk.LEFT, expand=True, fill=tk.X, padx=5, ipady=5)

         tech_frame = ttk.LabelFrame(parent_frame, text="Enhancement Techniques",
style='TLabelframe', padding=15)
        tech_frame.pack(fill=tk.X, pady=15)
        self.technique_var = tk.StringVar(value="None")

        techniques = [
            ("None (Show Original)", "None"),
            ("Histogram Equalization", "hist_eq"),
            ("Power-Law (Gamma)", "gamma"),
        ]
        for text, value in techniques:
```

```
            ttk.Radiobutton(tech_frame, text=text, variable=self.technique_var,
value=value, command=self.on_technique_change).pack(anchor=tk.W, pady=3)

            self.params_frame = ttk.LabelFrame(parent_frame, text="Parameters",
style='TLabelframe', padding=15)
        self.params_frame.pack(fill=tk.X, pady=15)

        self.gamma_label = ttk.Label(self.params_frame, text="Gamma (γ): 1.00")
        self.gamma_var = tk.DoubleVar(value=1.0)
            self.gamma_slider = ttk.Scale(self.params_frame, from_=0.1, to=5.0,
variable=self.gamma_var, orient=tk.HORIZONTAL, command=self.on_slider_change)

          self.reset_button = ttk.Button(parent_frame, text="Reset to Original",
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_technique_change()

    def setup_display_widgets(self, parent_frame: ttk.Frame):
            parent_frame.rowconfigure(0, weight=1); parent_frame.rowconfigure(1,
weight=1)
       parent_frame.columnconfigure(0, weight=3); parent_frame.columnconfigure(1,
weight=2)

              original_frame, self.original_canvas, self.original_info_label =
self._create_display_canvas(parent_frame, "Original Image")
        original_frame.grid(row=0, column=0, sticky="nsew", padx=(0, 10), pady=(0,
10))
            processed_frame, self.processed_canvas, self.processed_info_label =
self._create_display_canvas(parent_frame, "Enhanced Image")
      processed_frame.grid(row=1, column=0, sticky="nsew", padx=(0, 10), pady=(10,
0))

           self.hist_original_frame = self._create_histogram_frame(parent_frame,
"Original Histogram")
         self.hist_original_frame.grid(row=0, column=1, sticky="nsew", padx=(10,
0), pady=(0, 10))
          self.hist_processed_frame = self._create_histogram_frame(parent_frame,
"Enhanced Histogram")
        self.hist_processed_frame.grid(row=1, column=1, sticky="nsew", padx=(10,
0), pady=(10, 0))

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

    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 _create_histogram_frame(self, parent, title):
       return ttk.Labelframe(parent, text=title, style='TLabelframe', padding=10)

    def on_canvas_resize(self, event: tk.Event):
        canvas = event.widget
        canvas_w, canvas_h = canvas.winfo_width(), canvas.winfo_height()
        last_size = self.last_canvas_sizes.get(id(canvas))
        if last_size == (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 on_slider_change(self, _=None):
        if self.technique_var.get() == "gamma":
           self.gamma_label.config(text=f"Gamma (γ): {self.gamma_var.get():.2f}")

        if self.debounce_timer:
            self.root.after_cancel(self.debounce_timer)
        self.debounce_timer = self.root.after(100, self.apply_enhancement)

    def on_technique_change(self):
        self.gamma_label.pack_forget()
        self.gamma_slider.pack_forget()

        if self.technique_var.get() == "gamma":
            self.gamma_label.pack(anchor=tk.W)
            self.gamma_slider.pack(fill=tk.X, pady=(0, 10))

        self.apply_enhancement()

    def apply_enhancement(self):
```

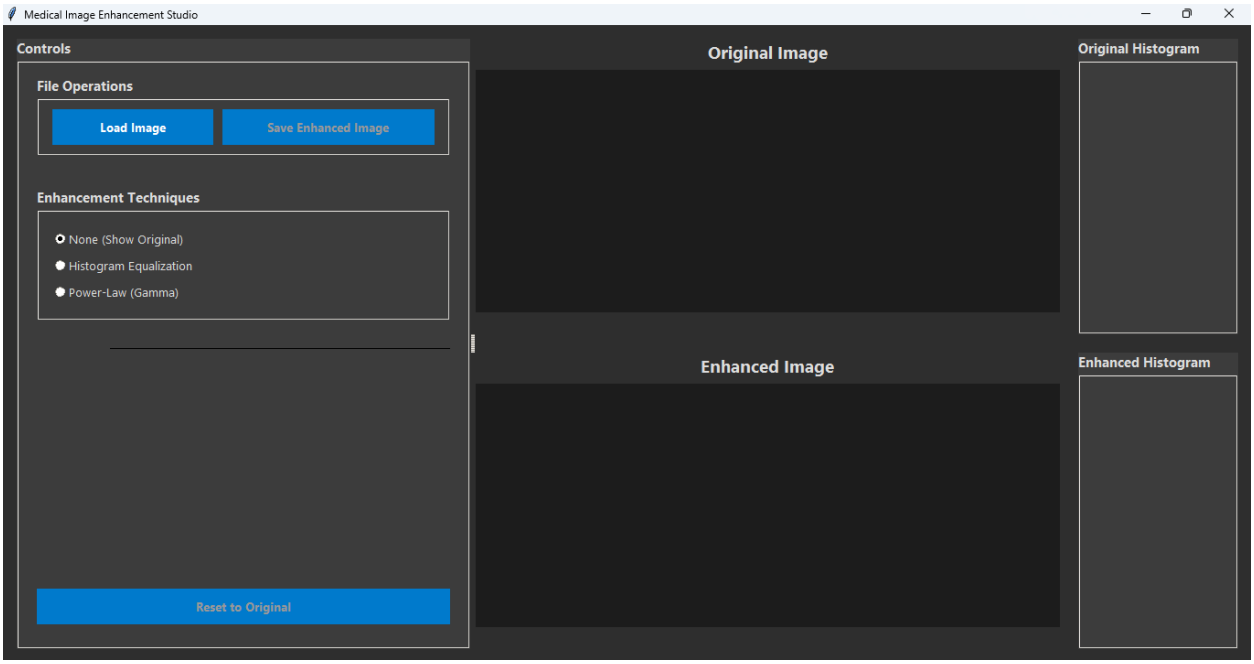
```
        if self.original_image is None: return

        technique = self.technique_var.get()

        if technique == "hist_eq":
            self.processed_image = cv2.equalizeHist(self.original_image)
        elif technique == "gamma":
            gamma = round(self.gamma_var.get(), 2)
            cache_key = f"gamma_{gamma}"
            if cache_key not in self.lut_cache:
                inv_gamma = 1.0 / gamma
                 self.lut_cache[cache_key] = np.array([((i / 255.0) ** inv_gamma)
* 255 for i in np.arange(256)]).astype("uint8")
            table = self.lut_cache[cache_key]
            self.processed_image = cv2.LUT(self.original_image, table)
        else:
            self.processed_image = self.original_image.copy()

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

    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 calculate_histogram_fast(self, image):
        if image.size > HISTOGRAM_PIXEL_LIMIT:
```

```
                        pixels, is_sampled = np.random.choice(image.ravel(),
HISTOGRAM_PIXEL_LIMIT, replace=False), True
        else:
            pixels, is_sampled = image.ravel(), False
        counts, bins = np.histogram(pixels, bins=256, range=[0, 256])
        return counts, bins, is_sampled

    def update_histograms(self):
        if self.original_image is not None:
                                                counts, bins, sampled =
self.calculate_histogram_fast(self.original_image)
                    self.plot_histogram(self.hist_original_frame, counts, bins,
ACCENT_COLOR, "Original Histogram", sampled)
        if self.processed_image is not None:
                                                counts, bins, sampled =
self.calculate_histogram_fast(self.processed_image)
                    self.plot_histogram(self.hist_processed_frame, counts, bins,
ERROR_COLOR, "Enhanced Histogram", sampled)

    def plot_histogram(self, parent_frame, counts, bins, color, title, sampled):
        for widget in parent_frame.winfo_children(): widget.destroy()
        parent_frame['text'] = title + (" (Sampled)" if sampled else "")
        fig, ax = plt.subplots(facecolor=FRAME_COLOR)
        ax.set_facecolor(BG_COLOR)
        ax.bar(bins[:-1], counts, width=1, color=color)
        ax.set_xlim([0, 255])
        ax.tick_params(colors=TEXT_COLOR, which='both')
        ax.set_xlabel("Pixel Intensity", color=TEXT_COLOR, fontsize=8)
        ax.set_ylabel("Frequency", color=TEXT_COLOR, fontsize=8)
        ax.tick_params(axis='both', which='major', labelsize=8)
        ax.grid(True, linestyle='--', alpha=0.2)
        fig.tight_layout(pad=0.5)
        canvas = FigureCanvasTkAgg(fig, master=parent_frame)
        canvas.draw()
        canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=5, pady=5)
        plt.close(fig)

    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.lut_cache.clear()
            self.last_canvas_sizes = {}
            self.reset_image()
            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_image(self):
          if self.processed_image is None: messagebox.showwarning("Warning", "No
enhanced image to save."); return
              file_path = filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG file", "*.png"), ("JPEG file", "*.jpg")])
        if file_path:
            try:
                cv2.imwrite(file_path, self.processed_image)
                messagebox.showinfo("Success", f"Image saved to:\n{file_path}")
            except Exception as e: messagebox.showerror("Error", f"Failed to save
image: {e}")

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


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

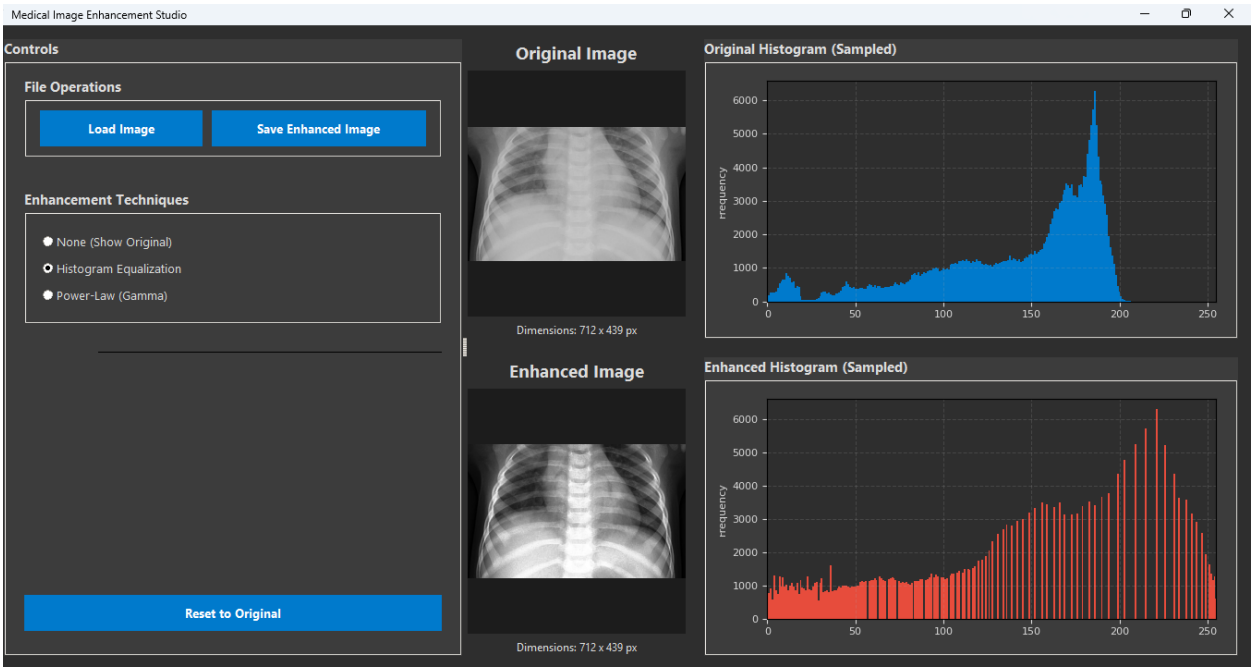
Outputs:

Before image upload

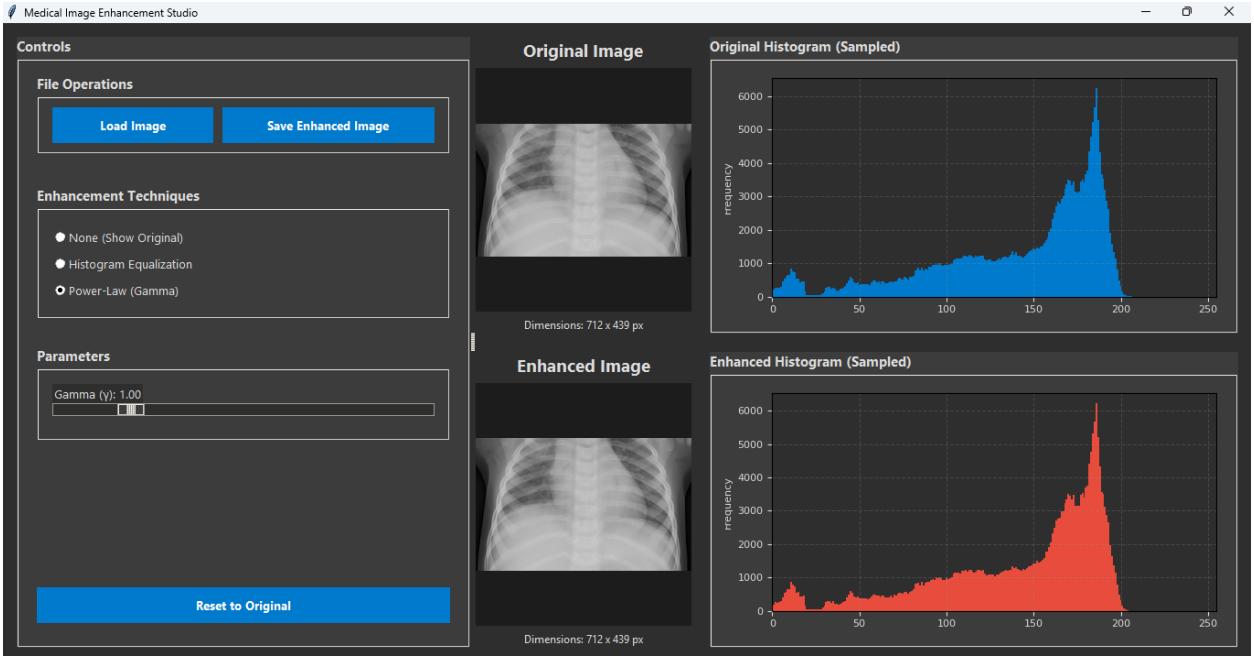


After image upload

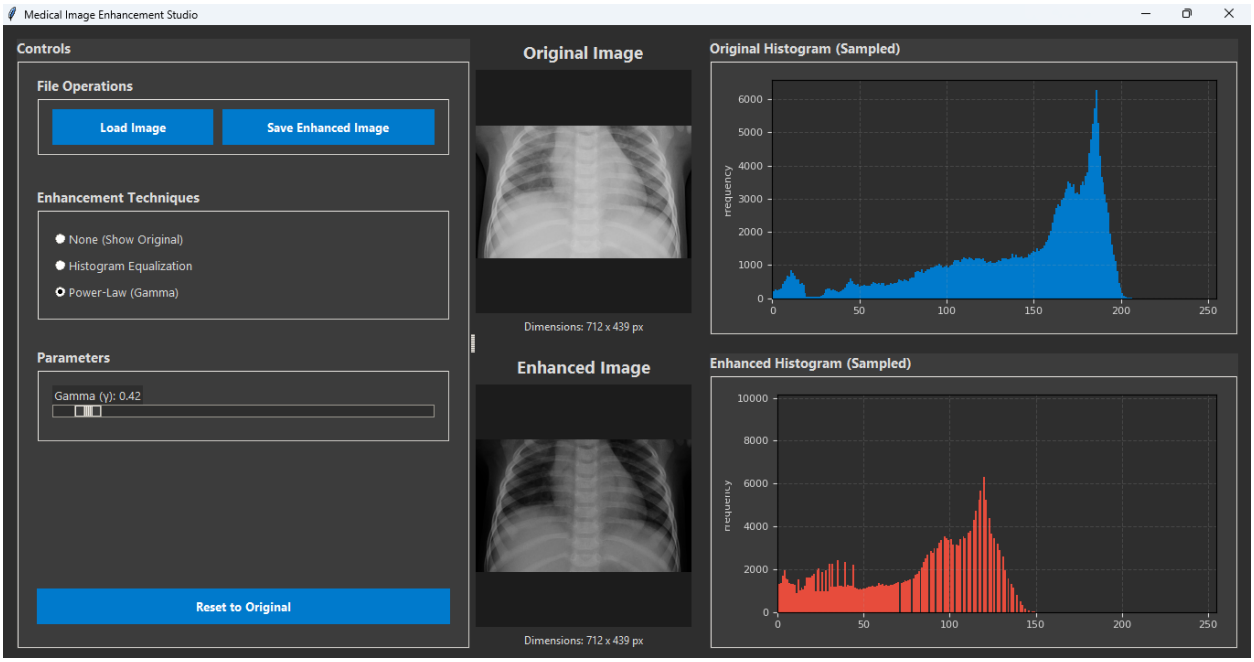
Apply histogram Equalization



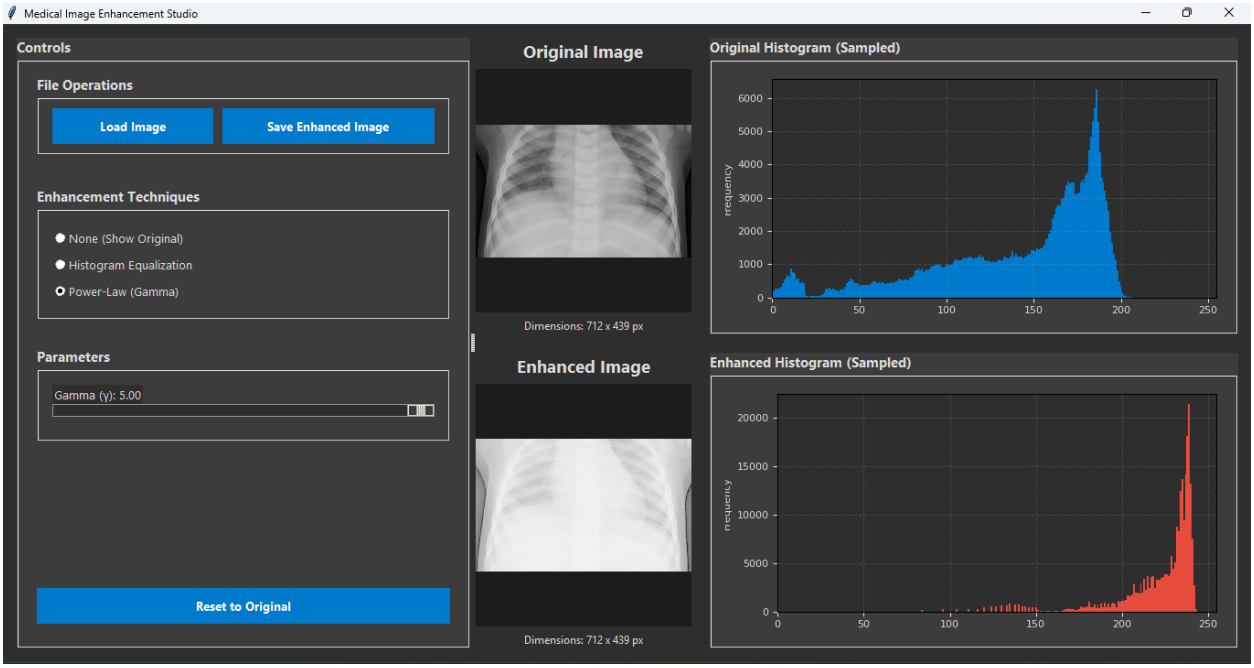
Apply Power-law(Gamma=1.00)



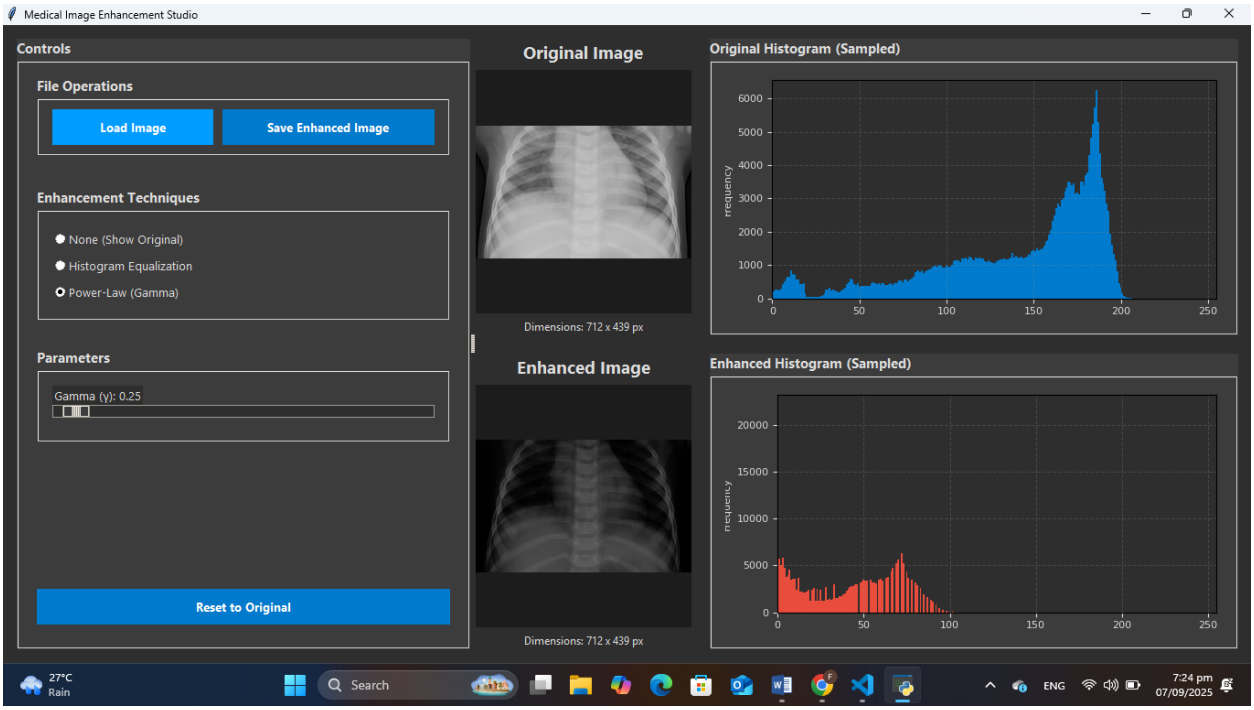
(Gamma=0.42)



(Gamma=5.00)



(Gamma=0.25)



Save Image