PES UNIVERSITY

PROJECT TITLE

Image processing

TEAM MEMBERS DETAILS

Section P2

SRN	Names
PES1UG25AM261	Patel Shlok
PES1UG25CS352	Nishanth U
PES1UG25AM250	P Kelvin OM
PESU1UG25CS344	Niranjana M

PROBLEM STATEMENT

This project aims to develop a GUI-based application that allows users to load an image, detect red pixels, and display the results clearly.

Approach

1. A graphical user interface is created using Tkinter.
2. The user selects an image using a file dialog.
3. The image is converted to RGB format.
4. Each pixel is examined to determine whether it qualifies as a red pixel based on predefined threshold values.
5. The total number of red pixels is counted.
6. The percentage of red pixels is calculated and displayed.
7. The image can optionally be converted to grayscale.

Methodology

Pixel-by-pixel traversal of the image

Comparison of RGB values using threshold logic

Displaying results using GUI labels

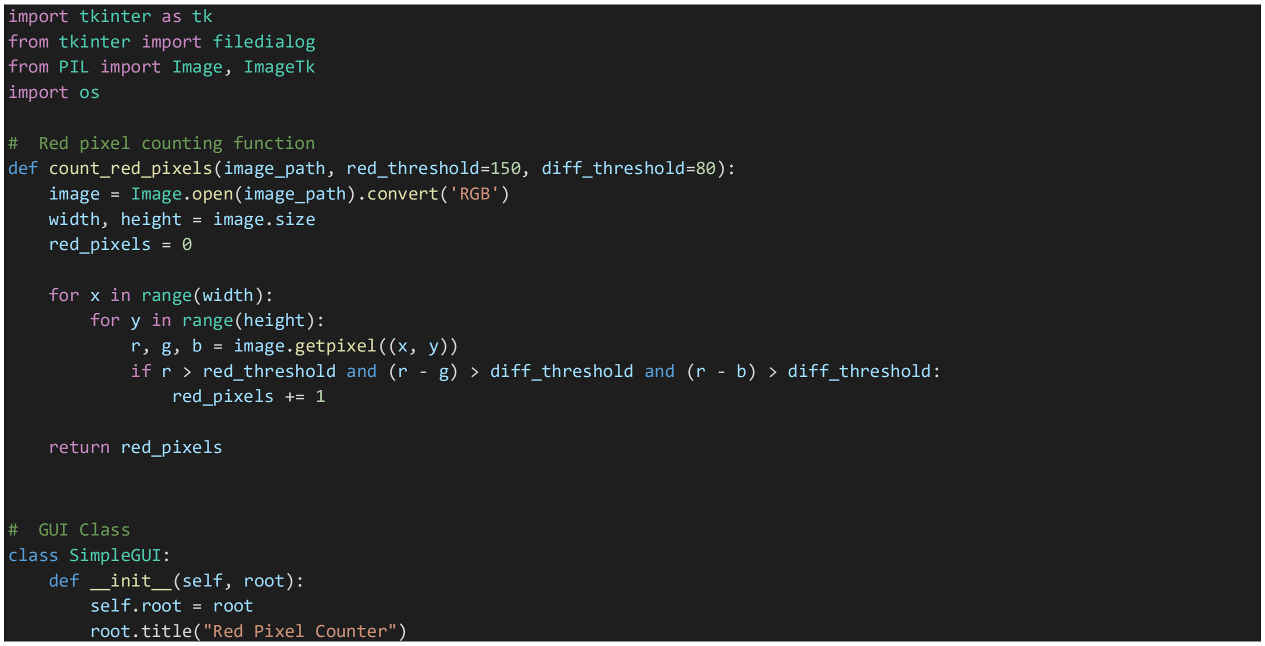
Data Structures Used

Variables (for counting pixels and storing thresholds)

Tuples (to store RGB pixel values)

Objects (Image objects from PIL library)

Solution



```python
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import os

#  Red pixel counting function
def count_red_pixels(image_path, red_threshold=150, diff_threshold=80):
    image = Image.open(image_path).convert('RGB')
    width, height = image.size
    red_pixels = 0

    for x in range(width):
        for y in range(height):
            r, g, b = image.getpixel((x, y))
            if r > red_threshold and (r - g) > diff_threshold and (r - b) > diff_threshold:
                red_pixels += 1

    return red_pixels


#  GUI Class
class SimpleGUI:
    def __init__(self, root):
        self.root = root
        root.title("Red Pixel Counter")
        root.geometry("700x600")

        self.image_path = None
        self.current_image = None

        # Title
        tk.Label(root, text="Red Pixel Counter",
                 font=("Arial", 18, "bold")).pack(pady=10)

        # Buttons
        tk.Button(
            root, text="Load Image", command=self.load_image,
            font=("Arial", 12), bg="#4CAF50", fg="white",
            padx=20, pady=8
        ).pack(pady=5)

        self.count_btn = tk.Button(
            root, text="Count Red Pixels", command=self.count,
            font=("Arial", 12), bg="#2196F3", fg="white",
            padx=20, pady=8, state=tk.DISABLED
        )
        self.count_btn.pack(pady=5)

        self.gray_btn = tk.Button(
            root, text="Convert to Grayscale",
            command=self.convert_grayscale,
            font=("Arial", 12), bg="#FF9800", fg="white",
            padx=20, pady=8, state=tk.DISABLED
        )
        self.gray_btn.pack(pady=5)

        # Threshold info
        tk.Label(
            root,
            text="Threshold Settings: Red = 150, Difference = 80",
            font=("Arial", 10, "italic"), fg="gray"
        ).pack(pady=5)

        # Image frame
        img_frame = tk.Frame(root, bg="lightgray", width=500, height=350)
        img_frame.pack(pady=10)
        img_frame.pack_propagate(False)

        self.img_label = tk.Label(img_frame, text="No image loaded", bg="lightgray")
        self.img_label.pack(fill=tk.BOTH, expand=True)

        # Result label
        self.result_label = tk.Label(root, text="", font=("Arial", 12, "bold"), fg="green")
        self.result_label.pack(pady=10)

    # Load Image
    def load_image(self):
        path = filedialog.askopenfilename(
            filetypes=[("Images", "*.png *.jpg *.jpeg *.bmp")]
        )
        if path:
            self.image_path = path
            self.current_image = Image.open(path).convert('RGB')

            img = self.current_image.copy()
            img.thumbnail((480, 330), Image.Resampling.LANCZOS)
            photo = ImageTk.PhotoImage(img)

            self.img_label.config(image=photo, text="", bg="white")
            self.img_label.image = photo

            self.count_btn.config(state=tk.NORMAL)
            self.gray_btn.config(state=tk.DISABLED)
```

```python
            self.result_label.config(text="")

    #  Count Red Pixels
    def count(self):
        if self.current_image:
            temp_path = "temp_count.png"
            self.current_image.save(temp_path)

            red_count = count_red_pixels(temp_path, 150, 80)
            total = self.current_image.size[0] * self.current_image.size[1]
            percentage = (red_count / total) * 100

            self.result_label.config(
                text=f"Red Pixels: {red_count:,} | Total: {total:,} | {percentage:.2f}%"
            )

            self.gray_btn.config(state=tk.NORMAL)

            if os.path.exists(temp_path):
                os.remove(temp_path)

    # Convert to Grayscale
    def convert_grayscale(self):
        if self.current_image:
            self.current_image = self.current_image.convert('L').convert('RGB')

            img = self.current_image.copy()
            img.thumbnail((480, 330), Image.Resampling.LANCZOS)
            photo = ImageTk.PhotoImage(img)

            self.img_label.config(image=photo)
            self.img_label.image = photo

            self.result_label.config(
                text="Converted to Grayscale! Count again to see red pixels."
            )
            self.gray_btn.config(state=tk.DISABLED)


#  Run App
root = tk.Tk()
app = SimpleGUI(root)
root.mainloop()
```

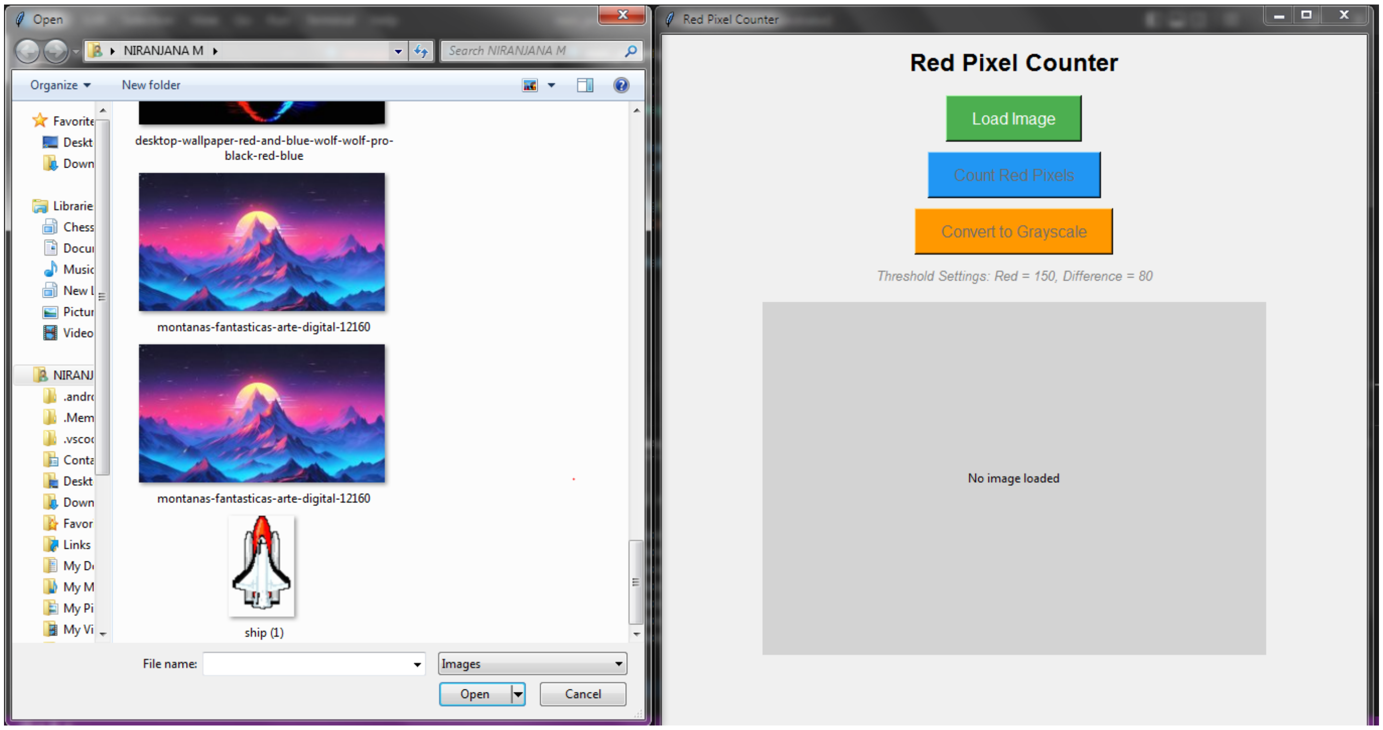
SAMPLE INPUT AND OUTPUT

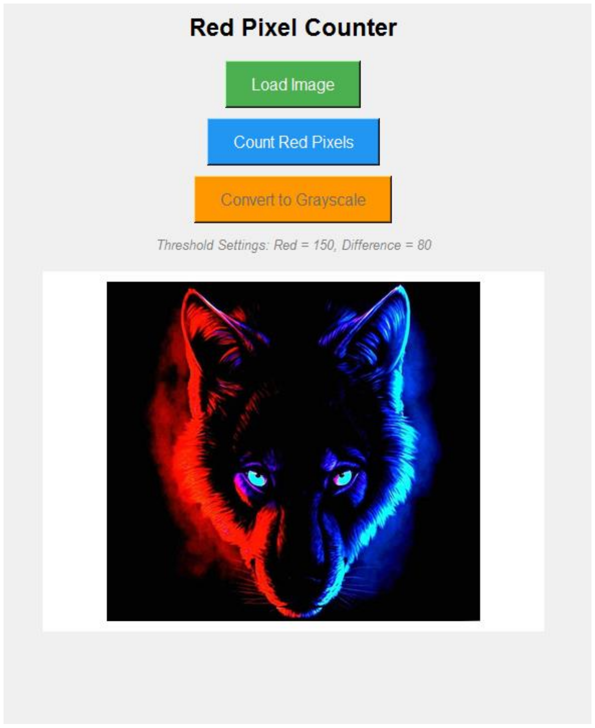
Sample Input

Input Type-Image file (.jpg /.png)

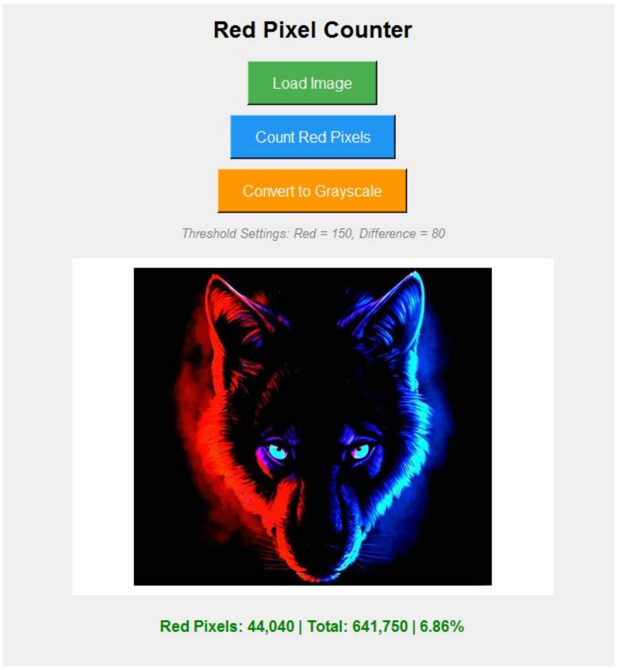
Input Method:-File selection using GUI

Sample Input



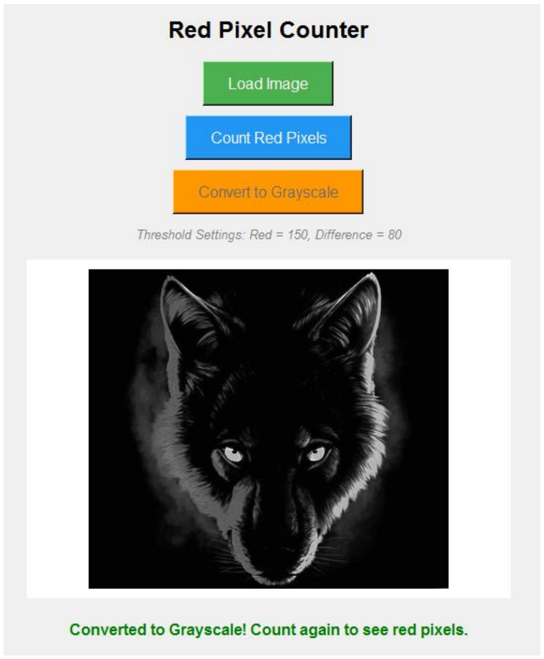


Sample Output



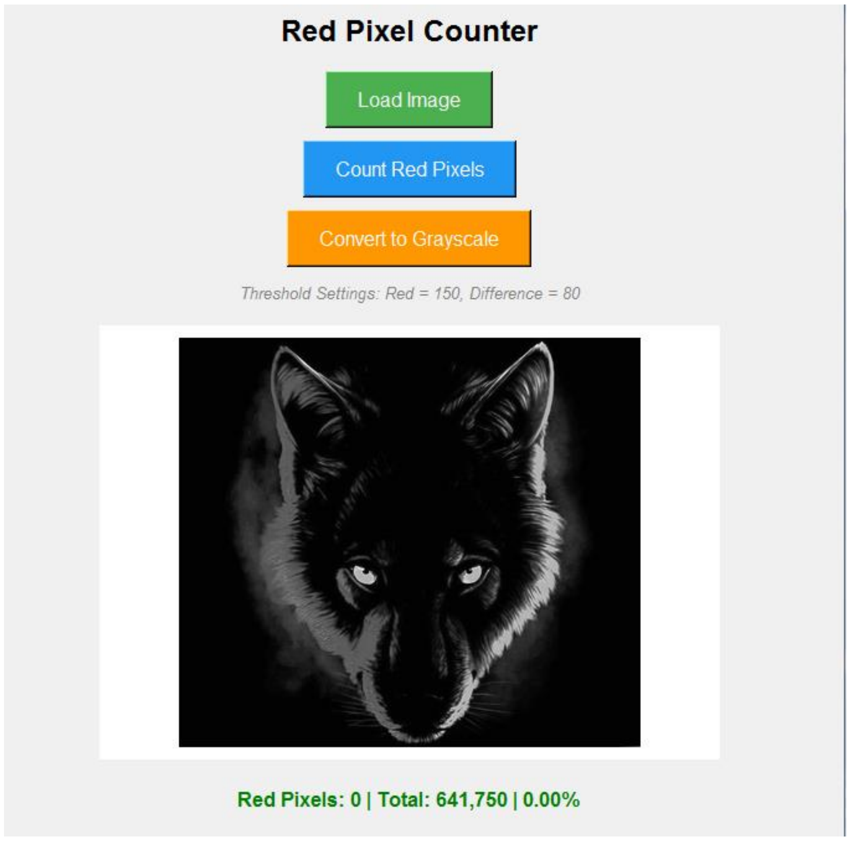
Threshold Settings: Red = 150, Difference = 80

Red Pixels: 44,040 | Total: 641,750 | 6.86%



Threshold Settings: Red = 150, Difference = 80

Converted to Grayscale! Count again to see red pixels.



Threshold Settings: Red = 150, Difference = 80

Red Pixels: 0 | Total: 641,750 | 0.00%

Number of red pixels |Total number of pixels | Percentage of red pixels

CHALLENGES FACED

Selecting appropriate threshold values for accurate red detection

Handling large image files efficiently

Maintaining image quality while resizing for GUI display

Avoiding misclassification of similar colors

SCOPE FOR IMPROVEMENT

Extend the application to detect other colors

Add support for real-time camera input

Export analysis results to a text or CSV file

Improve accuracy using advanced image processing techniques