IQRA UNIVERSITY IU

REPORT

[DIGITAL IMAGE PROCESSING LAB]

Course Instructor: Muhammad Anas

GROUP MEMBERS

- **AREEBA ABDUL RAZZAQ 23163**
- **SHARMEEN ZUBAIR 22509**
- **LAREB RAZZAQ 22635**

Project Name: Image Processing Tool Using Stream-lit and Open-CV

1. Introduction

The advent of digital technology has enhanced the capabilities of image processing, making it an essential field in computer vision, graphic design, and various industrial applications. This report discusses an Image Processing Tool, developed using Python libraries such as Open-CV, NumPy, Stream-lit, and PIL. The tool allows users to perform a wide range of operations on images, such as background removal, edge detection, blurring, re-sizing, rotation, and image addition.

2. Problem Statement

In real life, we encounter several challenges related to image manipulation:

- **Background Distractions**: Unwanted backgrounds in images reduce their clarity and utility.
- **Edge Identification**: Detecting the boundaries of objects in an image is difficult without automation.
- **Image Quality Issues**: Images may need to be blurred for artistic purposes or to reduce noise.
- **Re-sizing Constraints**: Images must often be re-sized for compatibility with specific devices or platforms.
- **Alignment Challenges**: Rotating images to desired orientations can be complex without precise tools.
- **Composite Image Creation**: Merging multiple images seamlessly requires specialized software.

3. Project Objective

The objectives of the project are:

- To provide an easy-to-use tool for performing common image processing tasks.
- To enable users to:
- Remove unwanted backgrounds from images.
- Detect edges for object identification.
- Apply blurring for aesthetic and technical purposes.
- Resize images to desired dimensions.

- Rotate images to correct alignment or for artistic effects.
- Add or blend multiple images to create composites.

4. Scope of the Project

This tool is designed for:

- Individuals working in photography, graphic design, or content creation.
- Developers needing an efficient solution for preprocessing images in machine learning or computer vision applications.
- Educators and students in digital image processing courses.
- Casual users interested in enhancing their images for personal use.

5. Methodology

The tool is built using the following technologies and methods:

1. Python Libraries:

- **Open-CV**: For core image processing operations.
- **NumPy**: For handling multidimensional arrays and matrices.
- **PIL (Pillow)**: For advanced image manipulation.
- **Stream-lit**: For creating an intuitive web-based interface.

2. User Interface:

- A sidebar allows users to select desired operations.
- Sliders and buttons facilitate customization (e.g., rotation angles, blur scales).

3. Operations:

- **Background Removal**: Uses thresholding and morphological operations to isolate the subject.

- **Edge Detection**: Applies the Canny edge detection algorithm.
- **Blurring**: Implements Gaussian blurring with user-defined parameters.
- **Re-sizing**: Re-sizes images to specified dimensions.
- **Rotation**: Rotates images using affine transformation matrices.
- **Image Addition**: Blends two images using weighted sums.

4. Deployment:

- Hosted as a web application using the Stream-lit framework.
- Allows image uploads, previews, and downloads.

5. Code

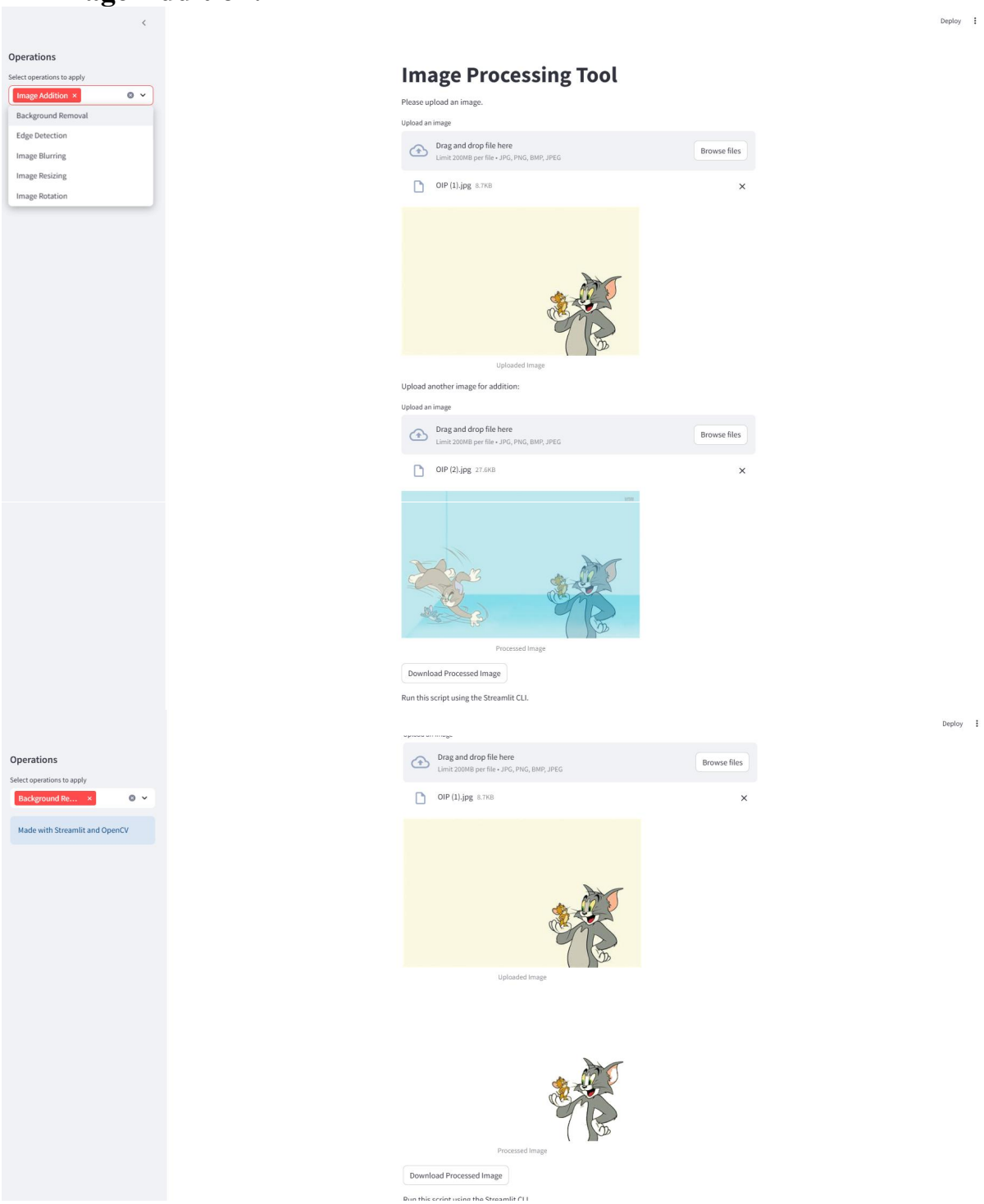
```
import cv2  # OpenCV library for image processing
import numpy as np  # for arrays and matrices
import streamlit as st  # Streamlit library for building web apps
from PIL import Image  # Pfor imagemanipulation
import io  # Input/Output library for handling binary streams

# Streamlit app setup
st.title("Image Processing Tool")  # Set the title of the Streamlit app

# Sidebar for user navigation
st.sidebar.header("Operations")  # Add a header in the sidebar for clarity
# Create a multiselect widget in the sidebar for selecting operations to apply
operations = st.sidebar.multiselect("Select operations to apply", [
    "Background Removal",
    "Edge Detection",
    "Image Blurring",
    "Image Resizing",
    "Image Rotation",
    "Image Addition"
])

# Function to upload an image
def upload_image(key=None):
    # Allow user to upload an image with specified file types
    uploaded_file = st.file_uploader("Upload an image", type=["jpg", "png", "bmp", "jpeg"], key=key)
    if uploaded_file is not None:  # Check if an image was uploaded
        # Open the uploaded image and convert it to RGBA format
        image = Image.open(uploaded_file).convert("RGBA")
        return np.array(image), image  # Return both NumPy array and PIL image
    return None, None  # Return None if no image is uploaded

# Function to enable image download
def download_image(image):
    buffer = io.BytesIO()  # Create an in-memory binary stream
    image.save(buffer, format="PNG")  # Save the image to the stream in PNG format
    st.download_button(
        label="Download Processed Image",  # Button label
```

```
            data=buffer.getvalue(),  # Binary data of the image
            file_name="processed_image.png",  # Default file name for download
            mime="image/png"  # MIME type for the download
        )

    # Function to remove the background from an image
    def remove_background(image):
        if image.shape[2] == 4:  # Check if the image has an alpha channel (RGBA)
            b, g, r, a = cv2.split(image)  # Split into individual channels
            rgb_image = cv2.merge((b, g, r))  # Merge RGB channels (discard alpha temporarily)
        else:  # If the image has no alpha channel (RGB)
            rgb_image = image

        gray = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        # Apply binary thresholding with Otsu's method to create a mask
        _, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
        kernel = np.ones((5, 5), np.uint8)  # Create a morphological kernel
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)  # Close small holes in the mask

        b, g, r = cv2.split(rgb_image)  # Split RGB channels again
        result = cv2.merge((b, g, r, mask))  # Add the mask as an alpha channel
        return result  # Return the image with the background removed

    # Function to detect edges in an image
    def detect_edges(image):
        return cv2.Canny(image, 100, 200)  # Apply Canny edge detection with thresholds

    # Function to blur an image
    def blur_image(image, blur_scale):
        # Ensure the blur scale is greater than 1 and an odd number
        if blur_scale <= 1:
            blur_scale = 3  # Set to minimum acceptable value
        if blur_scale % 2 == 0:
            blur_scale += 1  # Make odd if it's even
        kernel_size = (blur_scale, blur_scale)  # Define kernel size for blurring
        return cv2.GaussianBlur(image, kernel_size, 0)  # Apply Gaussian blur

    # Function to resize an image
    def resize_image(image, width, height):
        return cv2.resize(image, (width, height))  # Resize to the specified dimensions

    # Function to rotate an image
    def rotate_image(image, angle):
        (h, w) = image.shape[:2]  # Get height and width of the image
        center = (w // 2, h // 2)  # Define the center of rotation
        matrix = cv2.getRotationMatrix2D(center, angle, 1.0)  # Compute rotation matrix
        return cv2.warpAffine(image, matrix, (w, h))  # Apply rotation using the matrix

    # Function to add two images
    def add_images(image1, image2):
        # Convert grayscale images to RGBA if needed
        if len(image1.shape) == 2:
            image1 = cv2.cvtColor(image1, cv2.COLOR_GRAY2BGRA)
        if len(image2.shape) == 2:
            image2 = cv2.cvtColor(image2, cv2.COLOR_GRAY2BGRA)

        # Resize second image to match dimensions of the first
        if image1.shape[:2] != image2.shape[:2]:
            image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

        # Ensure both images have the same number of channels
```

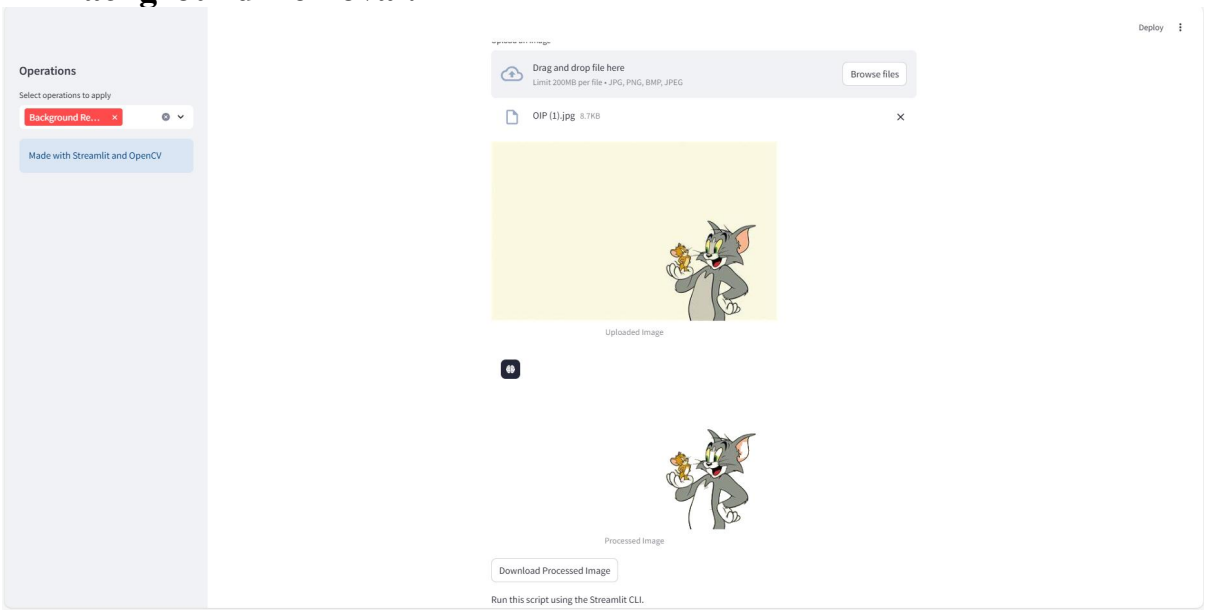
```
    if image1.shape[2] != image2.shape[2]:
        if image1.shape[2] == 4:
            image2 = cv2.cvtColor(image2, cv2.COLOR_RGB2BGRA)
        else:
            image1 = cv2.cvtColor(image1, cv2.COLOR_BGRA2BGR)

    # Blend the two images with equal weights
    return cv2.addWeighted(image1, 0.5, image2, 0.5, 0)

# Main application logic
st.write("Please upload an image.")  # Prompt the user to upload an image
image, pil_image = upload_image(key="main_image")  # Upload image with a unique key
if image is not None:
    st.image(pil_image, caption="Uploaded Image")  # Display the uploaded image

    # Convert RGBA to BGRA for OpenCV processing
    result = cv2.cvtColor(image, cv2.COLOR_RGBA2BGRA)
    for operation in operations:  # Iterate over selected operations
        if operation == "Background Removal":
            result = remove_background(result)
        elif operation == "Edge Detection":
            result = detect_edges(result)
        elif operation == "Image Blurring":
            # Allow user to adjust blur scale via sidebar slider
            blur_scale = st.sidebar.slider("Blur Scale", 1, 50, 15, step=2)
            result = blur_image(result, blur_scale)
        elif operation == "Image Resizing":
            # Allow user to adjust width and height via sliders
            width = st.sidebar.slider("Width", 10, 1000, result.shape[1])
            height = st.sidebar.slider("Height", 10, 1000, result.shape[0])
            result = resize_image(result, width, height)
        elif operation == "Image Rotation":
            # Allow user to set rotation angle via slider
            angle = st.sidebar.slider("Angle", -180, 180, 0)
            result = rotate_image(result, angle)
        elif operation == "Image Addition":
            st.write("Upload another image for addition:")  # Prompt for another image
            add_image, add_pil_image = upload_image(key="add_image")
            if add_image is not None:
                result = add_images(result, add_image)
            else:
                st.warning("Please upload another image to perform addition.")

    # Convert BGRA to RGBA for display
    result_rgb = cv2.cvtColor(result, cv2.COLOR_BGRA2RGBA)
    st.image(result_rgb, caption="Processed Image", channels="RGBA")  # Display the result

    # Convert back to PIL format for downloading
    result_pil = Image.fromarray(result_rgb)
    download_image(result_pil)  # Provide download option

st.sidebar.info("Made with Streamlit and OpenCV")  # Add footer in sidebar

if __name__ == "__main__":
    st.write("Run this script using the Streamlit CLI.")  # Instruction for running the script
```

6. Code snapshots

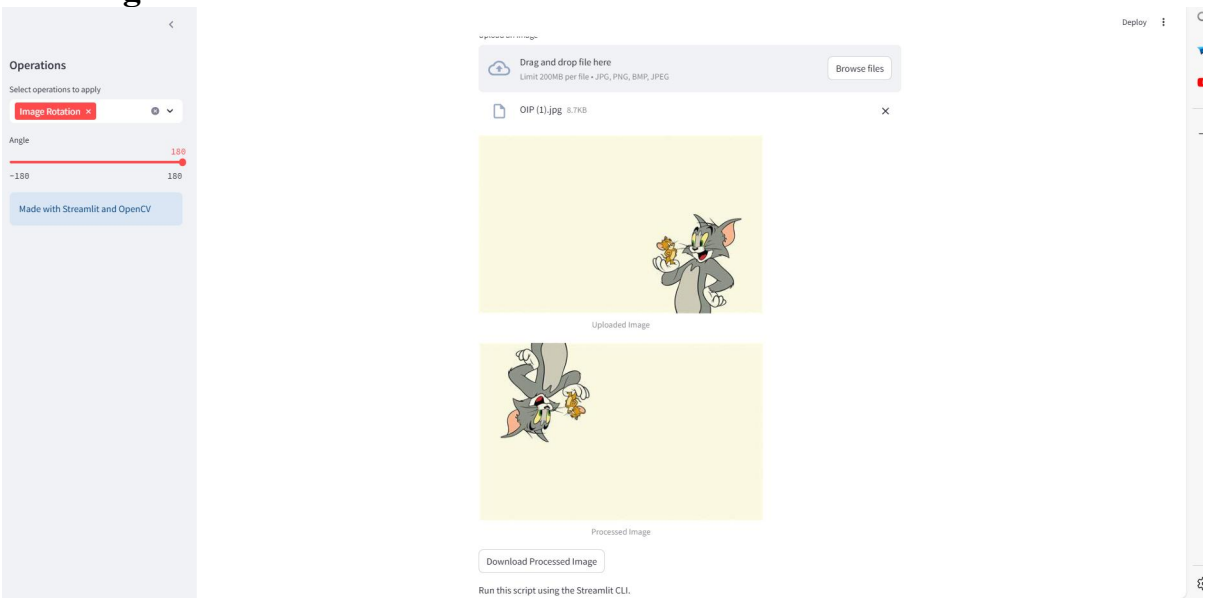
- **Image Addition:**



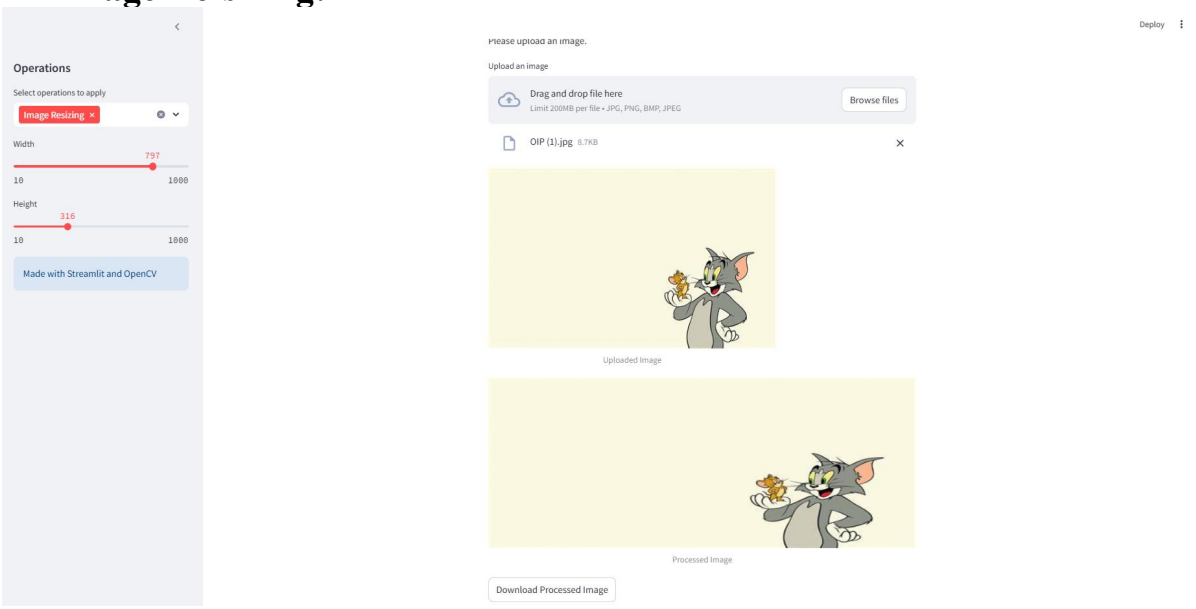
- **Background Removal:**



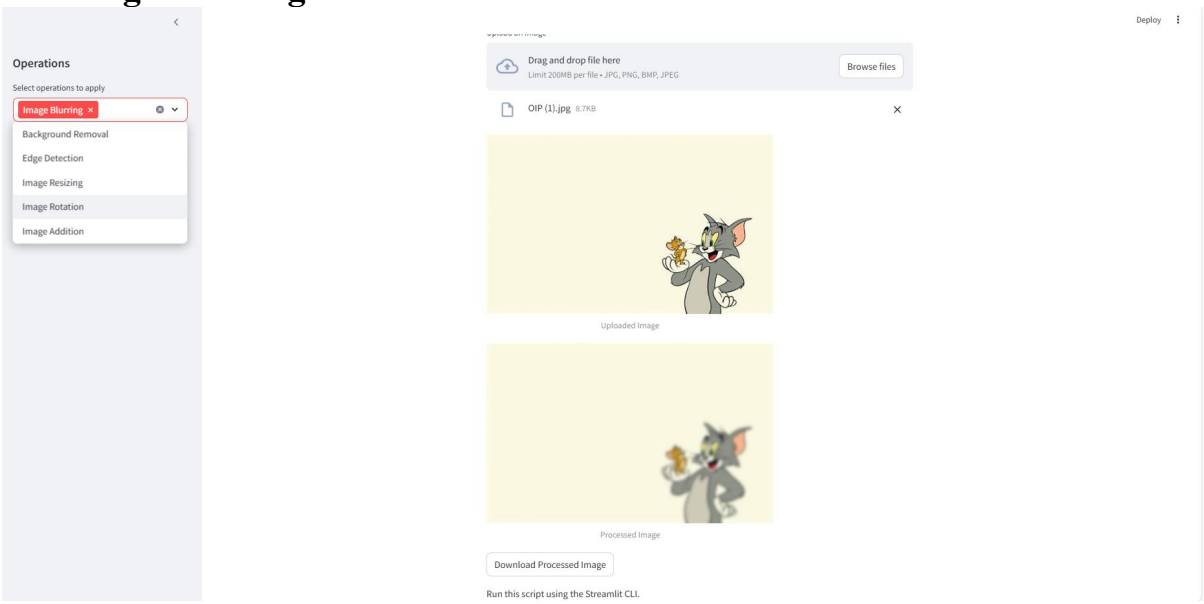
- **Image Rotation:**



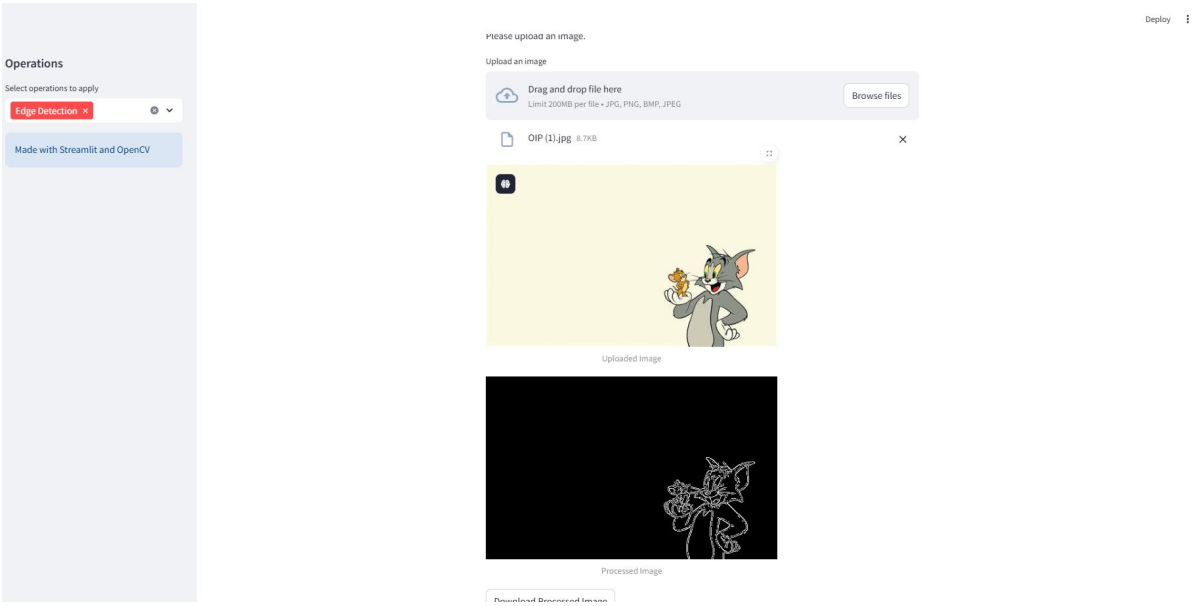
- **Image Re-sizing:**



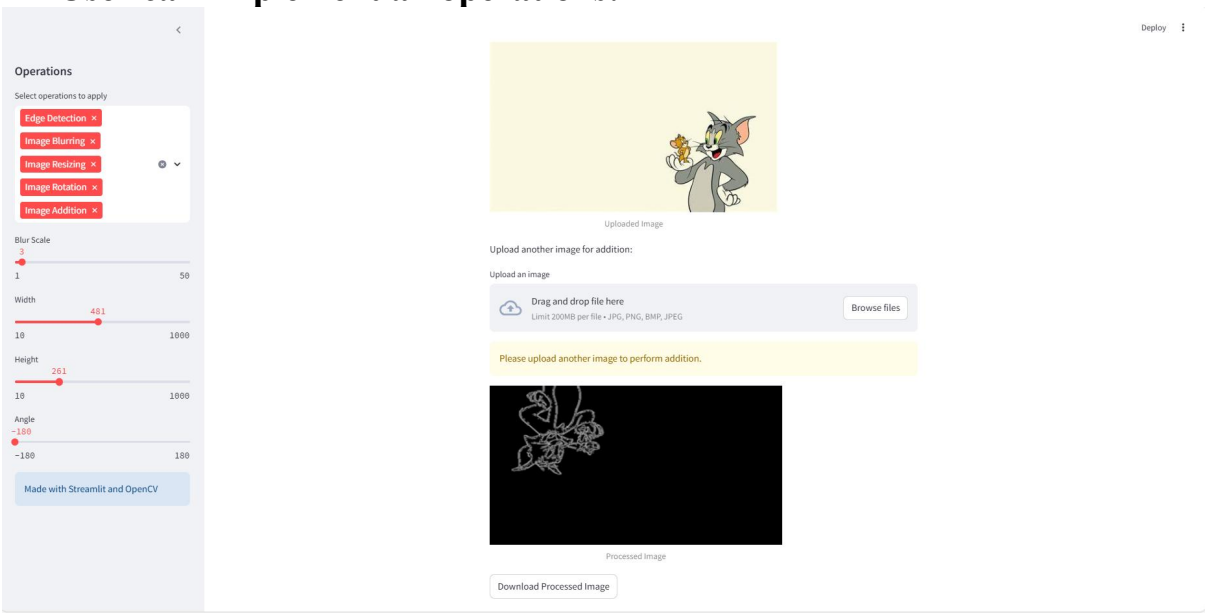
- **Image Blurring:**



- **Edge Detection:**



- **User can Implement all operations:**



References

Bradski, G., & Kaehler, A. (2008). **Learning OpenCV: Computer Vision with the OpenCV Library**. O'Reilly Media.

NumPy Documentation. **NumPy: The fundamental package for array computing with Python**. [Online]. Available: https://numpy.org

Streamlit Documentation. **Streamlit: The fastest way to build data apps**. [Online]. Available: https://streamlit.io

Pillow (PIL) Documentation. **Pillow: Python Imaging Library**. [Online]. Available: https://pillow.readthedocs.io

OpenCV Documentation. **Open Source Computer Vision Library**. [Online]. Available: https://opencv.org