Assignment 3 – A Practical Application with a Graphical User Interface (GUI)

Ravulakola Akshitha

04:52 PM IST, Sunday, September 07, 2025

Abstract

This report presents a detailed analysis of an image processing toolkit developed using OpenCV, Pillow (PIL), and Streamlit, as implemented in the Python script `app.py` and demonstrated in the Jupyter notebook `ImageToolkit_Final.ipynb`. The toolkit provides a graphical user interface (GUI) for performing fundamental image processing and computer vision operations, including image information retrieval, color conversions, geometric transformations, filtering, morphology, enhancement, edge detection, compression, and file handling. Each module is explained with its underlying code, mathematical formulations, and integration into the Streamlit framework, supported by screenshots of the GUI in action. This document serves as a comprehensive guide to the system's design and functionality, tailored for Assignment 3 in a professional academic context.

Contents

1 Introduction **2**

2 System Overview **3**

3 Helper Functions **4**
- 3.1 PIL and OpenCV Conversions . . . 4
- 3.2 Image Normalization and Grayscale Conversion . . . 4
- 3.3 Image Information Retrieval . . . 5
- 3.4 Image Encoding and Saving . . . 5
- 3.5 Composite and Enhancement Functions . . . 5
- 3.6 Edge Detection Helpers . . . 6

4 Streamlit Application Structure **7**
- 4.1 File Upload and Session State . . . 7
- 4.2 Sidebar and Category Selection . . . 8

5 Detailed Operations by Category **9**
- 5.1 Color Conversions . . . 9
- 5.2 Filtering & Morphology . . . 9
- 5.3 Enhancement . . . 10
- 5.4 Edge Detection . . . 10

6 Save and Status Bar **12**

7 Jupyter Notebook Analysis **13**

8 Conclusion **14**

9 References **15**

Chapter 1

Introduction

The objective of Assignment 3 is to develop a practical application with a graphical user interface (GUI) for image processing and computer vision tasks. The implementation is provided in two files: `app.py`, a Streamlit-based web application offering an interactive GUI, and `ImageToolkit_Final.ipynb`, a Jupyter notebook demonstrating the backend logic using Matplotlib for visualization. The toolkit covers essential operations such as color conversions, filtering, enhancement, and edge detection, with a user-friendly interface demonstrated through screenshots (e.g., Figure **??**).

This report provides an in-depth explanation of the code, from helper functions to the Streamlit interface, including mathematical formulas and key code snippets, enhanced by visual examples of the GUI.

Chapter 2

System Overview

The system consists of two components:

1. `app.py`: A Streamlit application that enables users to upload images, select operations via a sidebar, and view original and processed images side-by-side. Screenshots illustrate these interactions (e.g., Figures **??**–**??**).

2. `ImageToolkit_Final.ipynb`: A Jupyter notebook that replicates the backend logic of `app.py`, using Matplotlib to display results. It serves as a non-interactive validation of the algorithms.

Both components share modular helper functions for image processing tasks, ensuring code reusability and maintainability.

Chapter 3

Helper Functions

The helper functions in `app.py` provide the core functionality for image manipulation and are reused across operations. These functions support the operations shown in the GUI screenshots.

3.1 PIL and OpenCV Conversions

```
def pil_open_bytes(file_bytes):
    return Image.open(io.BytesIO(file_bytes))

def cv2_from_pil(pil_img):
    return cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

def pil_from_cv2(bgr):
    return Image.fromarray(cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB))
```

Explanation: These functions handle conversions between PIL and OpenCV formats, enabling the display of processed images as seen in Figures ??–??.

3.2 Image Normalization and Grayscale Conversion

```
def ensure_uint8(img):
    if img.dtype == np.float32 or img.dtype == np.float64:
        img = np.clip(img, 0, 255).astype(np.uint8)
    elif img.dtype != np.uint8:
        img = img.astype(np.uint8)
    return img

def to_gray_if_needed(bgr):
    if len(bgr.shape) == 3 and bgr.shape[2] == 3:
        return cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
    return bgr
```

Explanation: These ensure proper image formatting, critical for grayscale conversion shown in Figure ??.

3.3 Image Information Retrieval

```
def get_image_info(pil_img, file_bytes=None, file_name=None):
    info = {}
    info["mode"] = pil_img.mode
    info["size_wh"] = pil_img.size  # (W, H)
    info["format"] = getattr(pil_img, "format", None) or (file_name.
        split(".")[-1].upper() if file_name else "N/A")
    info["dpi"] = pil_img.info.get("dpi", "N/A")
    info["file_size_bytes"] = len(file_bytes) if file_bytes is not
        None else "N/A"
    return info
```

Explanation: Extracts metadata used in the status bar of the GUI.

3.4 Image Encoding and Saving

```
def bytes_from_image(img: Image.Image, fmt="PNG", quality=95):
    buf = io.BytesIO()
    save_kwargs = {}
    if fmt.upper() in ["JPG", "JPEG"]:
        save_kwargs["quality"] = int(quality)
        save_kwargs["optimize"] = True
    img.save(buf, format=fmt, **save_kwargs)
    return buf.getvalue()
```

Explanation: Supports saving processed images, accessible via the download button in the GUI.

3.5 Composite and Enhancement Functions

```
def halfsplit_composite(orig_bgr, proc_bgr, vertical=True):
    h, w = orig_bgr.shape[:2]
    proc_bgr = cv2.resize(proc_bgr, (w, h), interpolation=cv2.INTER_
        LINEAR)
    if vertical:
        mid = w // 2
        left = orig_bgr[:, :mid]
        right = proc_bgr[:, mid:]
        return np.hstack([left, right])
    else:
```

```
        mid = h // 2
        top = orig_bgr[:mid, :]
        bottom = proc_bgr[mid:, :]
        return np.vstack([top, bottom])

def contrast_stretch(bgr, low_perc=2.0, high_perc=98.0):
    out = np.zeros_like(bgr)
    if len(bgr.shape) == 2:
        low = np.percentile(bgr, low_perc)
        high = np.percentile(bgr, high_perc)
        if high - low < 1e-5:
            return bgr.copy()
        out = np.clip((bgr - low) * 255.0 / (high - low), 0, 255).
           astype(np.uint8)
        return out
    for c in range(3):
        ch = bgr[:, :, c]
        low = np.percentile(ch, low_perc)
        high = np.percentile(ch, high_perc)
        if high - low < 1e-5:
            out[:, :, c] = ch
        else:
            out[:, :, c] = np.clip((ch - low) * 255.0 / (high - low),
                0, 255).astype(np.uint8)
    return out
```

Explanation and Formulas:

- `halfsplit_composite`: Creates split-view images, as shown in Figures ??–??. - `contrast_stretch`: Applies percentile-based stretching:

$$\text{out}_c = \left(\frac{\text{ch} - \text{low}}{\text{high} - \text{low}} \times 255, 0, 255 \right)$$

3.6 Edge Detection Helpers

```
def sobel_edges(gray, ksize=3):
    gx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=ksize)
    gy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=ksize)
    mag = cv2.magnitude(gx, gy)
    mag = np.uint8(np.clip(mag / (mag.max() + 1e-9) * 255, 0, 255))
    return mag
```

Explanation and Formulas: Computes gradient magnitude:

$$\text{mag} = \sqrt{g_x^2 + g_y^2}, \quad \text{normalized to } [0, 255]$$

Chapter 4

Streamlit Application Structure

The `app.py` script sets up the Streamlit GUI, as visualized in the screenshots.

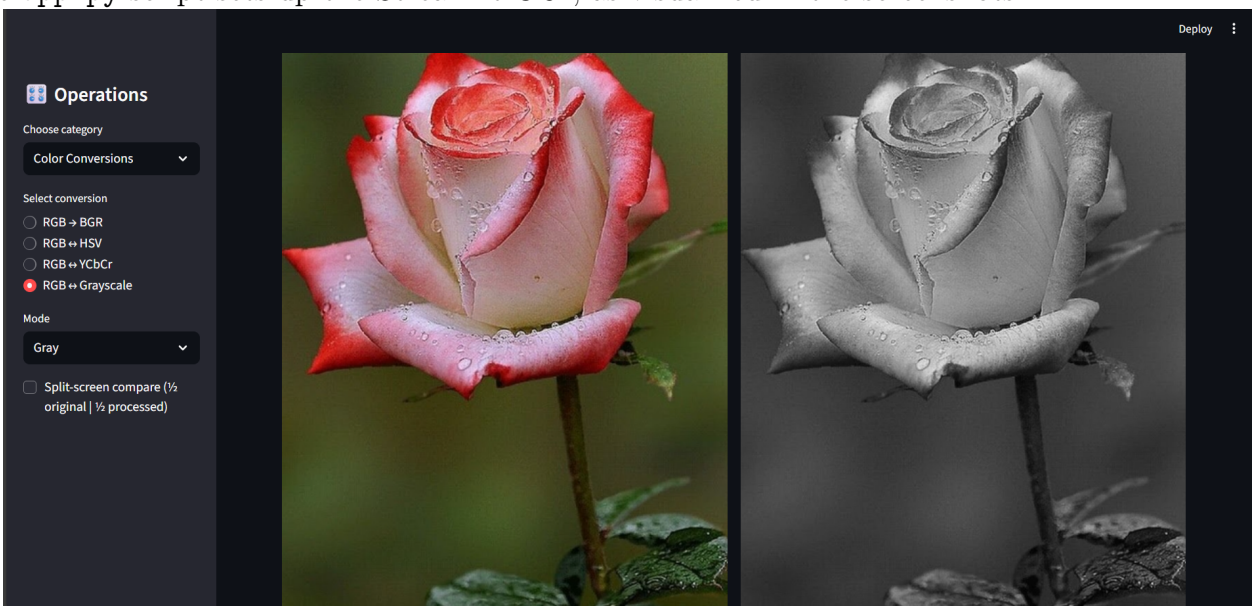

Fig 1
GUI showing Grayscale conversion (Color Conversions category).

```
st.set_page_config(page_title="Image Processing Toolkit (OpenCV + Streamlit)", layout="wide")
st.markdown(
    "<h1 style='margin-bottom:0'>       Image Processing Toolkit</h1>"
    "<div style='opacity:0.7;margin-bottom:1rem'>Module 1     Image Processing Fundamentals & Computer Vision</div>",
    unsafe_allow_html=True
)
```

4.1 File Upload and Session State

```
uploaded = st.file_uploader("Open     Upload an image", type=["png","jpg","jpeg","bmp","tiff"], accept_multiple_files=False)
if "orig_bytes" not in st.session_state:
    st.session_state.orig_bytes = None
if uploaded is not None:
    st.session_state.orig_bytes = uploaded.getvalue()
    pil_img = pil_open_bytes(st.session_state.orig_bytes).convert("RGB")
    st.session_state.proc_bgr = cv2_from_pil(pil_img)
```

4.2 Sidebar and Category Selection

```
st.sidebar.title("          Operations")
cat = st.sidebar.selectbox(
    "Choose category",
    [
        "Image Info",
        "Color Conversions",
        "Transformations",
        "Bitwise Ops",
        "Filtering & Morphology",
        "Enhancement",
        "Edge Detection",
        "Compression & File Handling",
        "Video Mode (Optional Demo)"
    ]
)
```

Chapter 5

Detailed Operations by Category

Each category is implemented as a conditional block, with GUI examples in Figures ??–??.

5.1 Color Conversions

```
elif cat == "Color Conversions":
    op = st.sidebar.radio(
        "Select conversion",
        ["RGB     BGR", "RGB     HSV", "RGB     YCbCr", "RGB
           Grayscale"]
    )
    if op == "RGB     Grayscale":
        gray = cv2.cvtColor(bgr_orig, cv2.COLOR_BGR2GRAY)
        proc = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
    split = st.sidebar.checkbox("Split-screen compare (   original |
          processed)", value=True)
    if split:
        show_bgr = halfsplit_composite(bgr_orig, proc, vertical=True)
    c1, c2 = st.columns(2)
    with c1:
        st.image(pil_from_cv2(bgr_orig), caption="Original", use_
           container_width=True)
    with c2:
        st.image(pil_from_cv2(show_bgr), caption="Processed", use_
           container_width=True)
```

5.2 Filtering & Morphology

GUI showing Gaussian filtering (Filtering Morphology category).

```
elif cat == "Filtering & Morphology":
    mode = st.sidebar.selectbox(
        "Select filter/morph operation",
```

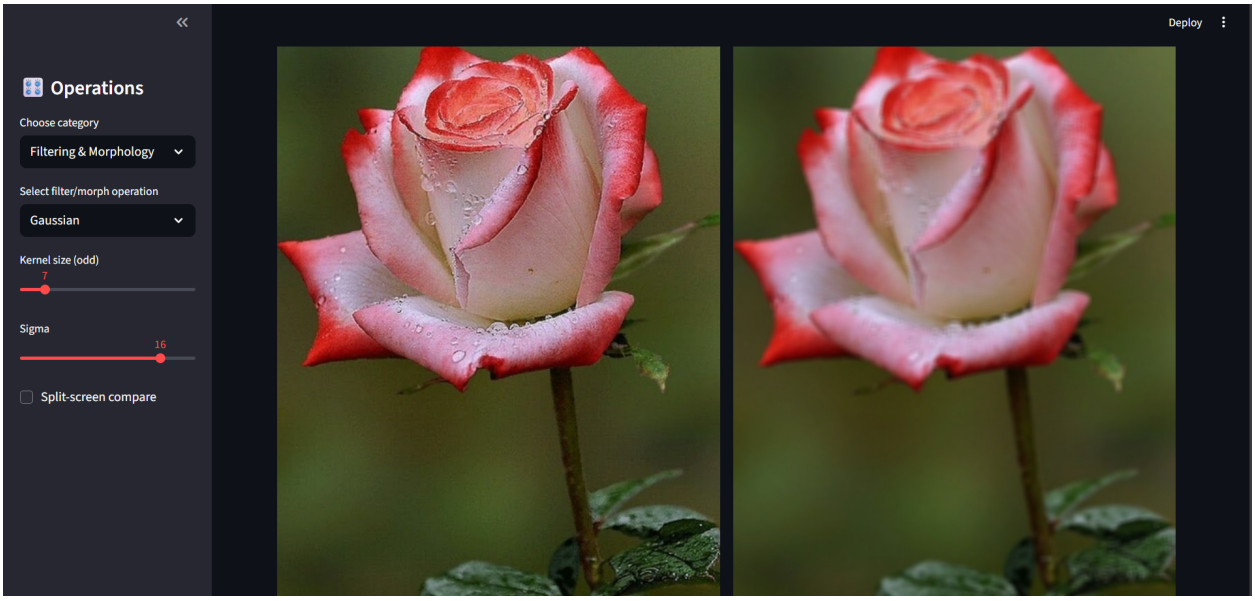

Figure 5.1: Filtering & Morphology

```
        ["Gaussian", "Median", "Mean (Box)", "Sobel", "Laplacian"]
    )
    if mode == "Gaussian":
        k = st.sidebar.slider("Kernel size (odd)", 3, 31, 7, step=2)
        sigma = st.sidebar.slider("Sigma", 0, 20, 2)
        proc = cv2.GaussianBlur(bgr_orig, (k, k), sigma)
```

5.3 Enhancement

```
elif cat == "Enhancement":
    op = st.sidebar.selectbox("Select enhancement", ["Contrast
       Stretching"])
    if op == "Contrast Stretching":
        low = st.sidebar.slider("Low percentile", 0, 20, 2)
        high = st.sidebar.slider("High percentile", 80, 100, 98)
        proc = contrast_stretch(bgr_orig, low_perc=float(low), high_
           perc=float(high))
```

5.4 Edge Detection

GUI showing Canny edge detection (Edge Detection category).

```
elif cat == "Edge Detection":
    mode = st.sidebar.radio("Select operator", ["Sobel", "Canny", "
       Laplacian"])
```

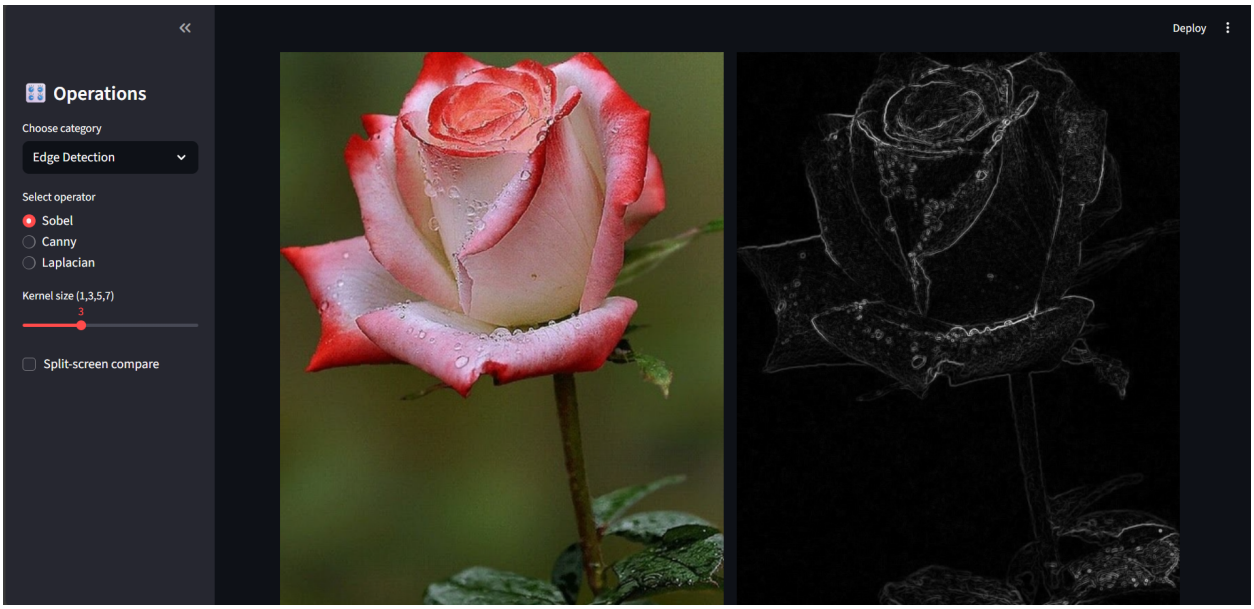

Figure 5.2: Edge Detection

```
    gray = cv2.cvtColor(bgr_orig, cv2.COLOR_BGR2GRAY)
    if mode == "Canny":
        th1 = st.sidebar.slider("Threshold 1", 0, 255, 50)
        th2 = st.sidebar.slider("Threshold 2", 0, 255, 150)
        proc_gray = cv2.Canny(gray, th1, th2)
        proc = cv2.cvtColor(proc_gray, cv2.COLOR_GRAY2BGR)
```

Chapter 6

Save and Status Bar

```
st.markdown("---")
st.subheader("        Save Processed Image")
fmt2 = st.selectbox("Format", ["PNG","JPG","BMP"], key="savefmt")
q2 = 95
if fmt2 == "JPG":
    q2 = st.slider("JPEG quality", 1, 100, 92, key="savequal")
save_bytes = cv2_encode_bytes(proc, ext=f".{fmt2.lower()}", quality=
    q2)
st.download_button(
    "Download processed image",
    data=save_bytes,
    file_name=f"processed.{fmt2.lower()}",
    mime=f"image/{'jpeg' if fmt2=='JPG' else fmt2.lower()}"
)
```

Chapter 7

Jupyter Notebook Analysis

The notebook `ImageToolkit_Final.ipynb` mirrors `app.py`'s logic.

```
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

def show_images(img1, img2, title1="Original", title2="Processed"):
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
    plt.title(title1)
    plt.axis('off')
    plt.subplot(1,2,2)
    if len(img2.shape) == 2:
        plt.imshow(img2, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
    plt.title(title2)
    plt.axis('off')
    plt.show()
```

Chapter 8

Conclusion

The image processing toolkit successfully integrates OpenCV, Pillow, and Streamlit to provide a user-friendly GUI, as demonstrated in Figures **??**–**??**. The Jupyter notebook validates the backend logic. Future enhancements could include advanced features like object detection.

Chapter 9

References

- OpenCV Documentation: `https://docs.opencv.org/`
- Streamlit Documentation: `https://docs.streamlit.io/`
- Pillow Documentation: `https://pillow.readthedocs.io/`