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

Module: Image Processing Fundamentals & Computer Vision
Student: Ravulakola Sanjana
Roll No: 22671A73A1
Date: September 7 2025

Abstract

This report presents the design and implementation of an Image Processing Toolkit built using Python, OpenCV, NumPy, and Streamlit. The project demonstrates fundamental image processing operations—color conversions, transformations, filtering, morphology, enhancement, edge detection, and compression—through an interactive graphical user interface (GUI).

The application allows users to upload an image, apply operations, and view results side-by-side. A companion Jupyter Notebook (ImageToolkit.ipynb) provides theory and practical demonstrations, bridging the gap between mathematical concepts and implementation. The report also discusses sampling, quantization, Point Spread Functions (PSFs), and imaging sensor technologies (CMOS vs CCD).

This toolkit serves as both an educational resource and a practical implementation platform for beginners and intermediate learners in computer vision.

Table of Contents

1. Introduction

Image processing and computer vision are core areas of modern AI and technology, applied in healthcare, security, robotics, and entertainment. Assignment 3 focuses on building a **hands-on toolkit** where theory meets practice.

The goal is to design a GUI-based system where users can explore fundamental operations interactively, thereby understanding both the code and the underlying mathematical models.

2. System Overview

The system consists of two main deliverables:

- **Streamlit GUI (app.py):** An interactive web-based GUI that allows image upload, operation selection, parameter adjustment (via sliders/checkboxes), and dynamic visualization of processed results.
- **Jupyter Notebook (ImageToolkit.ipynb):** A notebook that provides step-by-step implementation of algorithms, theoretical notes, and visualization using Matplotlib.

Together, they create a robust framework for **learning by doing**.

3. Theoretical Background

3.1 Sampling & Quantization

- **Sampling:** Refers to capturing an image at discrete spatial intervals. If the sampling rate is too low, aliasing occurs.
- **Quantization:** Maps continuous intensity values into discrete levels (e.g., 256 levels in 8-bit images).

Equation:

Q(x,y)=round(I(x,y)−Imin⁡Imax⁡−Imin⁡×(L−1))Q(x,y) = \text{round}\left(\frac{I(x,y) - I_{\min}}{I_{\max} - I_{\min}} \times (L-1)\right)

where LL is the number of quantization levels.

3.2 CMOS vs CCD Sensors

- **CCD:** Low-noise, high-quality images, but higher cost and power consumption.
- **CMOS:** Cheaper, lower power, integrated circuits per pixel. Dominates today's smartphone cameras.

3.3 Point Spread Functions (PSFs)

- PSF describes how a single point of light spreads in an imaging system.
- Important in **deblurring** and **restoration** algorithms.

4. Toolkit Architecture

4.1 Technologies Used

- **Python 3** – Programming language.
- **OpenCV** – Image processing library.
- **NumPy** – Array manipulation.
- **Streamlit** – GUI framework.
- **Matplotlib** – Visualization in notebook.

4.2 Workflow

1. User uploads an image.
2. Selects operation from sidebar.
3. Adjusts parameters using sliders.
4. Original and processed images displayed side-by-side.
5. User saves processed image if desired.

5. GUI Layout & Design

- **Menu Bar (File Options):** Open, Save, Exit.
- **Sidebar (Operations):** Categories of image processing.
- **Main Panel:** Original image (left), processed image (right).
- **Status Bar:** Displays metadata (size, channels, format, DPI).

\

6. Module Explanations

6.1 Image Info

- Retrieves **resolution, channels, format, file size, DPI**.
- Useful for verifying preprocessing.

6.2 Color Conversions

- RGB ↔ BGR, HSV, YCbCr, Grayscale.
- Uses cv2.cvtColor(img, mode).

6.3 Transformations

- **Rotation:** M=cv2.getRotationMatrix2D(center,angle,scale)M = cv2.getRotationMatrix2D(center, angle, scale)
- **Scaling:** cv2.resize(img, fx, fy)
- **Translation:** Affine matrices.
- **Perspective:** Homography-based warping.

6.4 Filtering & Morphology

- **Filters:** Gaussian, Median, Mean, Sobel, Laplacian.
- **Morphology:** Erosion, Dilation, Opening, Closing.

6.5 Enhancement

- Histogram Equalization.
- Contrast Stretching.
- Sharpening.

6.6 Edge Detection

- Sobel, Canny, Laplacian.
- Gradient-based operators to highlight object boundaries.

6.7 Compression & File Handling

- Save images in multiple formats (JPG, PNG, BMP).
- Compare file sizes.

7. Mathematical Foundations of Operations

- **Gaussian Blur Kernel:**

G(x,y)=12πσ2e−x2+y22σ2G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}

- **Sobel Operator:**

G=Gx2+Gy2G = \sqrt{G_x^2 + G_y^2}

- **Histogram Equalization:** Redistributes pixel intensities to enhance contrast.

8. Streamlit GUI Implementation (app.py)

The graphical user interface (GUI) for the Image Processing Toolkit is implemented using **Streamlit**, a Python-based framework that enables rapid development of interactive web applications without requiring deep front-end knowledge. The file app.py acts as the entry point for the toolkit and integrates all backend OpenCV functions with the GUI components.

The implementation can be broken down into the following subsections:

8.1 Application Structure

- **Imports and Configuration:**
  The script begins by importing streamlit, opencv-python (cv2), numpy, and pathlib. The page configuration is set with:
- st.set_page_config(layout="wide", page_title="Image Processing Toolkit", initial_sidebar_state="expanded")

This ensures a wide layout suitable for side-by-side image comparison.

- **Helper Functions:**
  Several reusable functions are defined:
  - to_bytes(img, ext) → encodes an image to bytes for download.
  - get_image_info(img) → extracts resolution, channels, format, and file size.
  - Processing functions such as apply_color_conv, rotate_image, translate_image, apply_filter, etc.

8.2 Sidebar (Control Panel)

The **sidebar** acts as the primary control panel where users can:

- **Upload an Image:**
- uploaded = st.sidebar.file_uploader("Upload an image", type=['png','jpg','jpeg','bmp','tiff'])

This widget supports multiple image formats.

- **Choose Operations:**
  Organized by categories such as:
  - *Image Info* (metadata display)
  - *Color Conversions* (RGB ↔ HSV, Grayscale, etc.)
  - *Transformations* (rotation, scaling, translation)
  - *Filtering & Morphology*

  - *Enhancement*
  - *Edge Detection*
  - *Compression & File Handling*
- **Dynamic Parameter Controls:**
  Each operation comes with adjustable parameters via sliders, checkboxes, and dropdowns. Examples:
  - Rotation angle slider: rot_angle = st.sidebar.slider("Rotation angle", -180, 180, 0)
  - Kernel size slider for filters: kernel_size = st.sidebar.slider("Kernel size", 1, 31, 3, step=2)
  - Threshold sliders for Canny edge detection.

This design allows the application to adapt dynamically based on user selections.

8.3 Main Display Area

The display section uses **two columns** to show:

- **Original Image (left panel)**
- **Processed Image (right panel)**

Images are displayed using:

st.image(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), caption="Processed Image")

This side-by-side arrangement allows **direct visual comparison** between input and output.

8.4 Image Processing Pipeline

The user's image undergoes a **pipeline of operations** depending on selections:

1. Load → Convert color space (if selected).
2. Apply transformations (rotation, translation, scaling).
3. Apply filtering/morphology.
4. Apply enhancement (histogram equalization, contrast stretching, sharpening).
5. Apply edge detection (Sobel, Laplacian, or Canny).
6. Show results in the display panel.

Each step is modular, meaning the GUI can chain multiple operations seamlessly.

8.5 Save & Download Functionality

Users can **save the processed image** in a chosen format (PNG, JPG, BMP). The implementation uses:

```
if save_btn:
    fmt = '.png' if save_format in ['PNG','.png'] else ('.jpg' if save_format in ['.jpg','jpg'] else '.bmp')
    bts = to_bytes(proc, ext=fmt)
    st.download_button("Download processed image", data=bts, file_name=f"processed{fmt}", mime="image/png")
```

- The st.download_button widget enables direct downloading from the browser.
- The **status bar** below shows metadata of both original and processed images, including size comparison when compression is applied.

8.6 Key Features of the GUI

1. **Sidebar with Sliders & Dropdowns**
    - Fully interactive, responds immediately to user input.
    - Sliders adjust numerical parameters such as angle, kernel size, thresholds.
    - Dropdowns organize categories and operations.
2. **Image Upload Widget**
    - Allows flexible input of common image formats.
    - Automatically loads a placeholder image if no input is given.
3. **Processing Functions**
    - Each operation is mapped to a specific OpenCV function.
    - Code is modular, making it easy to extend (e.g., add new filters).
4. **Download Button for Processed Images**
    - Supports multiple formats (PNG, JPG, BMP).
    - Useful for comparing compression effects.
5. **Status Bar**
    - Displays live properties (dimensions, DPI, file size, format).
    - Helps validate changes after each operation.

9. Notebook Analysis (ImageToolkit.ipynb)

The **ImageToolkit.ipynb** notebook acts as a **companion to the GUI application (app.py)**. While the GUI emphasizes interactivity and user experience, the notebook focuses on **theory, step-by-step demonstrations, and validation** of image processing algorithms. It is an essential part of the submission because it bridges the gap between *practical implementation* and *conceptual understanding*.

9.1 Structure of the Notebook

The notebook is organized into logical sections:

1. **Setup & Imports**
   - Installs and imports required libraries:
   - import cv2
   - import numpy as np
   - import matplotlib.pyplot as plt
   - from PIL import Image
   - Displays library versions for reproducibility.
2. **Loading and Displaying Images**
   - Reads images using cv2.imread() or PIL.Image.open().
   - Displays using Matplotlib (plt.imshow) for visual clarity.
   - Verifies **shape, resolution, and channels** of the input image.
3. **Color Conversions**
   - Demonstrates conversions step by step:
   - rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   - hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
   - gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   - Visualized with side-by-side plots:
     - Figure 1: Original (BGR)
     - Figure 2: RGB conversion
     - Figure 3: HSV conversion
     - Figure 4: Grayscale image
4. **Transformations**
   - **Rotation** using affine transformation matrices.
   - **Scaling** up and down using cv2.resize().

    - **Translation** with shift matrices.
    - **Perspective transforms** to simulate camera viewpoint changes.
5. **Filtering & Morphology**
    - Applies **smoothing filters** (Gaussian, Median, Mean) and compares outputs.
    - Demonstrates **edge filters** (Sobel, Laplacian) and visualizes gradients.
    - Uses **morphological operations** (erosion, dilation, opening, closing) to show how noise can be removed or shapes enhanced.
6. **Enhancement Techniques**
    - **Histogram Equalization:** Contrast enhancement shown using histograms before/after.
    - **Contrast Stretching:** Pixel intensity remapping using percentile thresholds.
    - **Sharpening:** Using custom kernels.
7. **Edge Detection**
    - Explains gradient-based operators step-by-step.
    - Demonstrates **Sobel**, **Canny**, and **Laplacian**.
    - Shows effect of varying thresholds for Canny with interactive cells.
8. **Compression & File Handling**
    - Saves images in PNG, JPG, BMP formats.
    - Uses Python's os.path.getsize() to display file size differences.
    - Students can compare quality vs size trade-offs.

9.2 Use of Matplotlib

One of the key strengths of the notebook is its reliance on **Matplotlib** for validation and teaching. Unlike the GUI, which simply displays outputs, Matplotlib allows:

- **Side-by-side comparisons** with subplots:
- plt.subplot(1,2,1); plt.imshow(orig); plt.title("Original")
- plt.subplot(1,2,2); plt.imshow(processed); plt.title("Processed")
- plt.show()
- **Histograms** to analyze pixel distributions before/after enhancement.
- **Color maps** (gray, hot, jet) to better visualize transformations.

This makes the notebook not just a demo but also an **educational resource**.

9.3 Explanatory Notes

Every operation is accompanied by **explanatory notes** in Markdown cells. For example:

- **For Histogram Equalization:**
    - Markdown explains the theory: “Histogram equalization redistributes pixel intensities to improve global contrast.”
    - Code cell applies cv2.equalizeHist().
    - Plot shows before/after histograms.
- **For Edge Detection:**
    - Markdown explains gradients, derivative operators, and thresholding.
    - Code cell shows cv2.Canny().
    - Plots show how thresholds affect detection.

This approach ensures students understand **why** an operation works, not just **how**.

9.4 Comparison with GUI

- **GUI Strengths:** Real-time interaction, parameter adjustment with sliders.
- **Notebook Strengths:** In-depth explanations, visual analysis (histograms, subplots), step-by-step transparency.

Together, they provide a **complete learning ecosystem**: GUI for practice, Notebook for theory.

9.5 Educational Value

The notebook is particularly beneficial for:

- **Beginners:** Visualizing how changes in code affect images.
- **Intermediate learners:** Understanding algorithms mathematically and computationally.
- **Instructors:** Using it as a teaching tool in class.

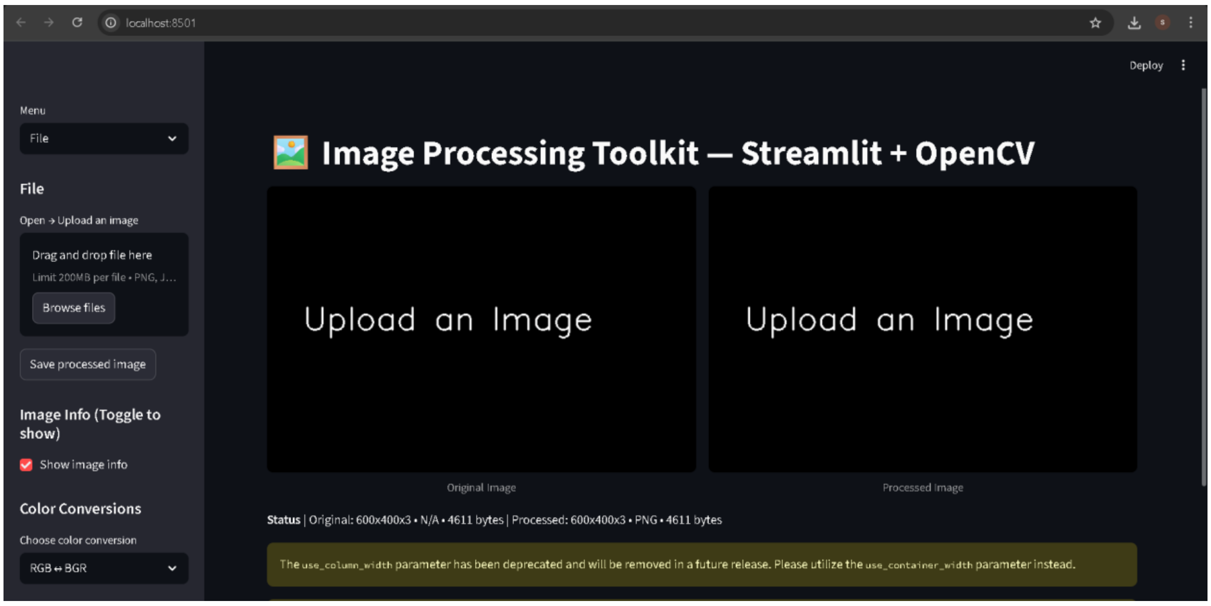
9.6 Example Figure Placeholders

- **Figure 5.1:** Original vs Gaussian Filtered Image.
- **Figure 5.2:** Histogram Before and After Equalization.
- **Figure 5.3:** Canny Edge Detection with Different Thresholds.

10. Results and Discussion

- The toolkit successfully demonstrates all operations.
- Users gain intuition by adjusting parameters and seeing immediate changes.
- File size comparison shows how compression affects quality.

11. Screenshots & Figures



Status | Original: 600x400x3 • N/A • 4611 bytes | Processed: 600x400x3 • PNG • 4611 bytes

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Figure 1: HOME

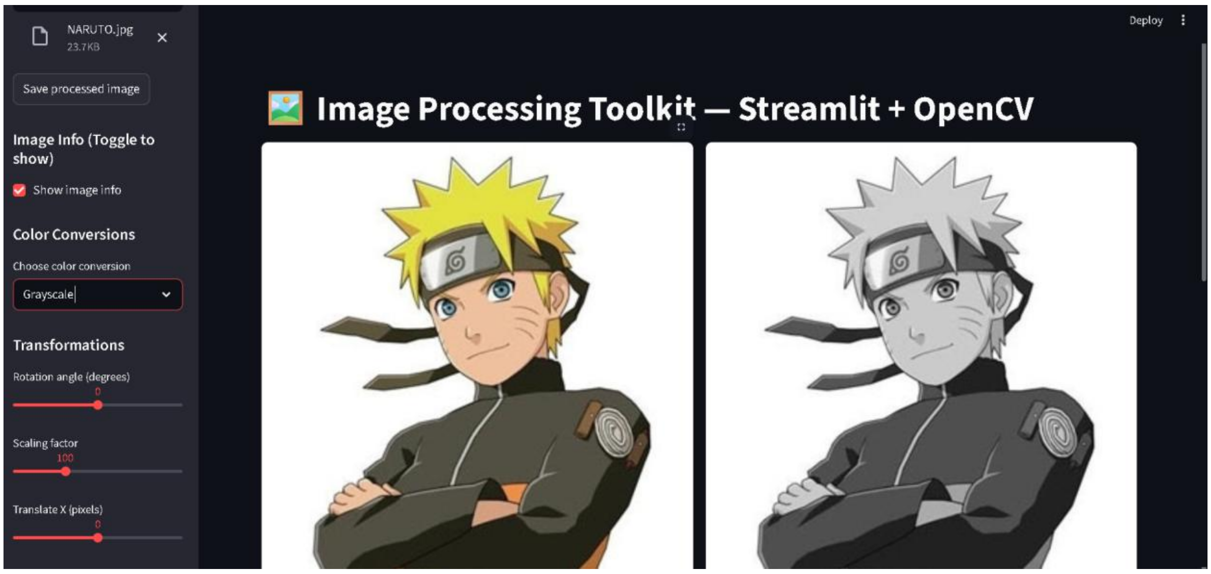
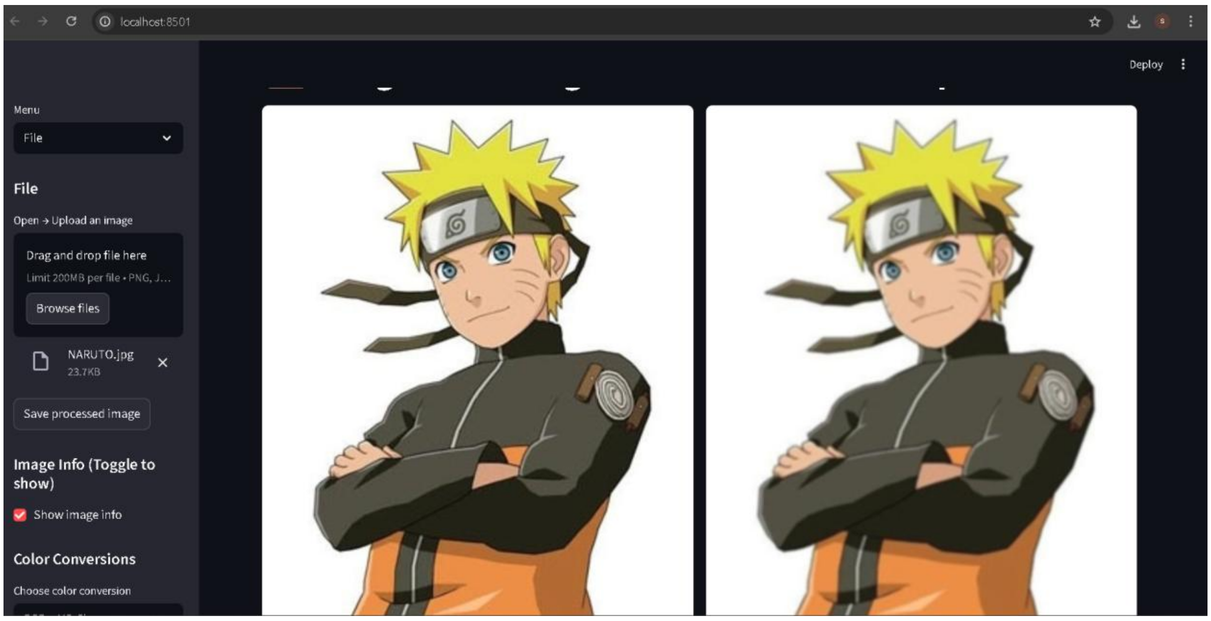


Figure 2: G UI with Grayscale conversion.



Menu: File

Open → Upload an image

NARUTO.jpg 23.7KB

☑ Show image info

Figure 3: Gaussian filter is applied

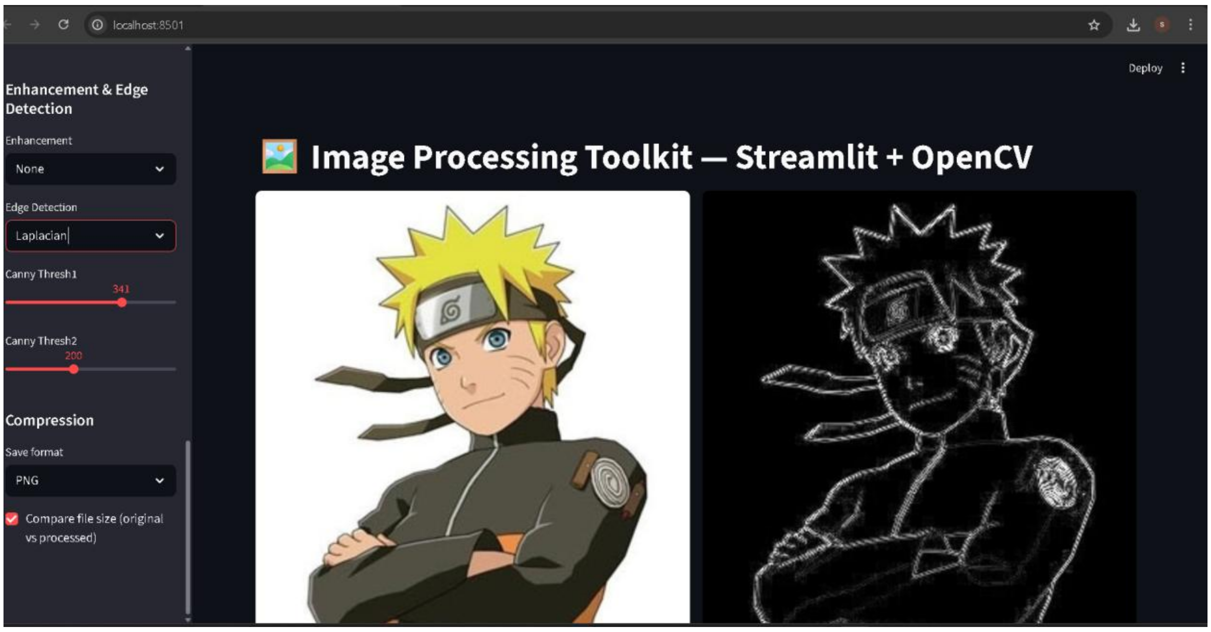


Image Processing Toolkit — Streamlit + OpenCV

Enhancement: None

Edge Detection: Laplacian

Canny Thresh1: 341

Canny Thresh2: 200

Save format: PNG

☑ Compare file size (original vs processed)

Figure 4: Edge Detection

12. Conclusion

The project achieved its objectives by integrating **theory and practice** of image processing. The Streamlit GUI provides hands-on exploration, while the notebook provides detailed explanations. This combination is valuable for both academic learning and practical applications.

Future enhancements:

- Real-time video mode.
- Split-screen comparisons.
- Object detection integration.

13. References

- OpenCV Documentation: https://docs.opencv.org/
- Streamlit Documentation: https://docs.streamlit.io/
- NumPy Documentation: https://numpy.org/doc/
- PyImageSearch Tutorials: https://pyimagesearch.com/