Web-Based Image Processing Application

Computer Vision Project Report

Amira Yasser Mohamed
Alhussien Ayman Hanafy
Ahmed Ahmed Mokhtar
Osama Magdy Ali

February 26, 2026

Contents

1 Project Overview

This project implements a web-based image processing application using:

- **Backend:** FastAPI (Python + OpenCV)
- **Frontend:** Vanilla JavaScript (HTML, CSS)

The system allows users to:

- Upload images
- Apply spatial domain filters
- Perform edge detection
- Apply histogram operations
- Perform frequency domain filtering
- Generate hybrid images

The application provides an interactive UI to test different parameters and observe their effect in real-time.

2 System Architecture

2.1 Backend (FastAPI)

- Image upload handling
- OpenCV-based image processing functions
- REST API endpoints for each operation
- Parameter validation

2.2 Frontend (JavaScript)

- Image preview
- Parameter input forms
- Asynchronous API calls
- Result visualization

3 Application Interface

3.1 Home Page

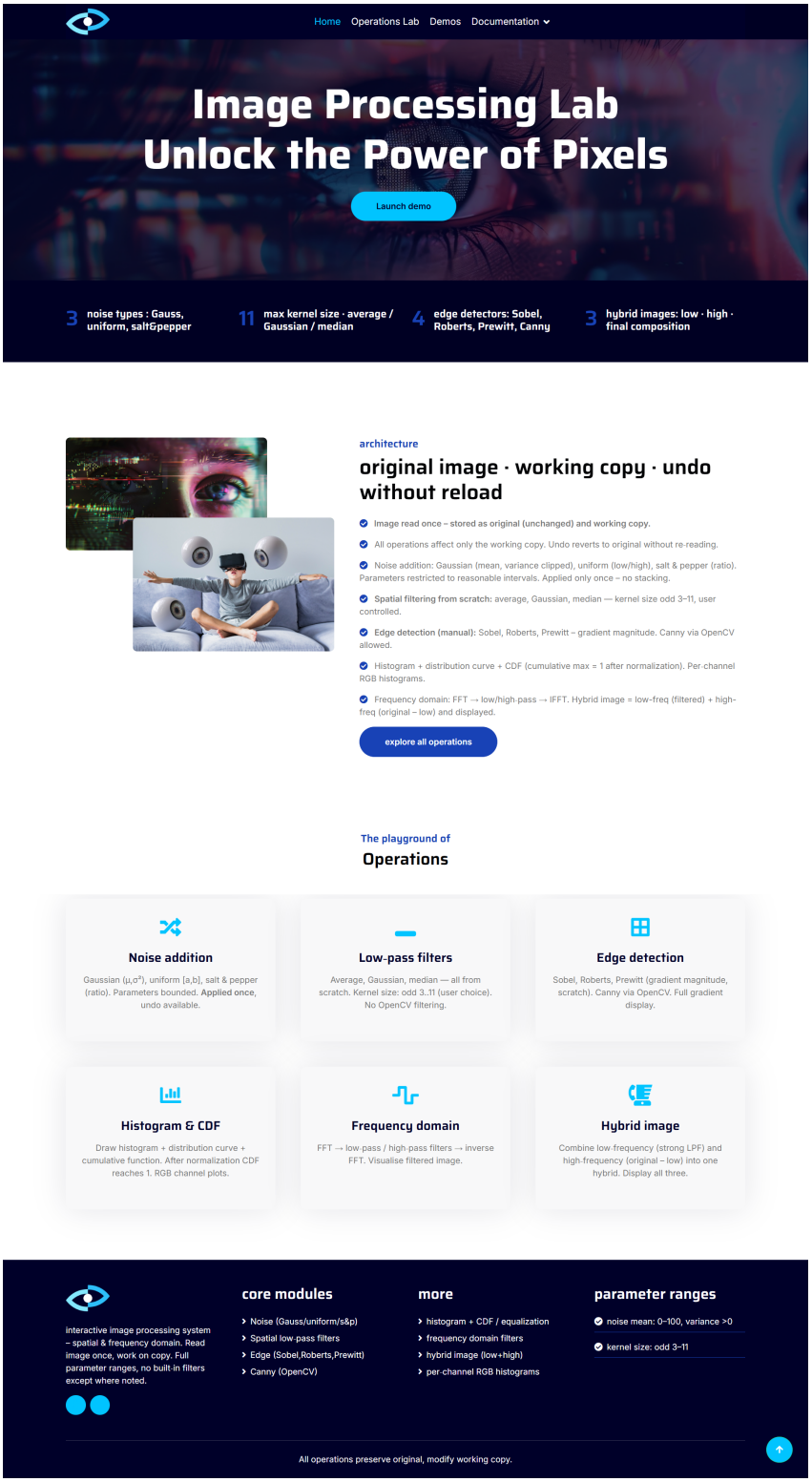

Figure 1: Application Home Page

3.2 Operations UI

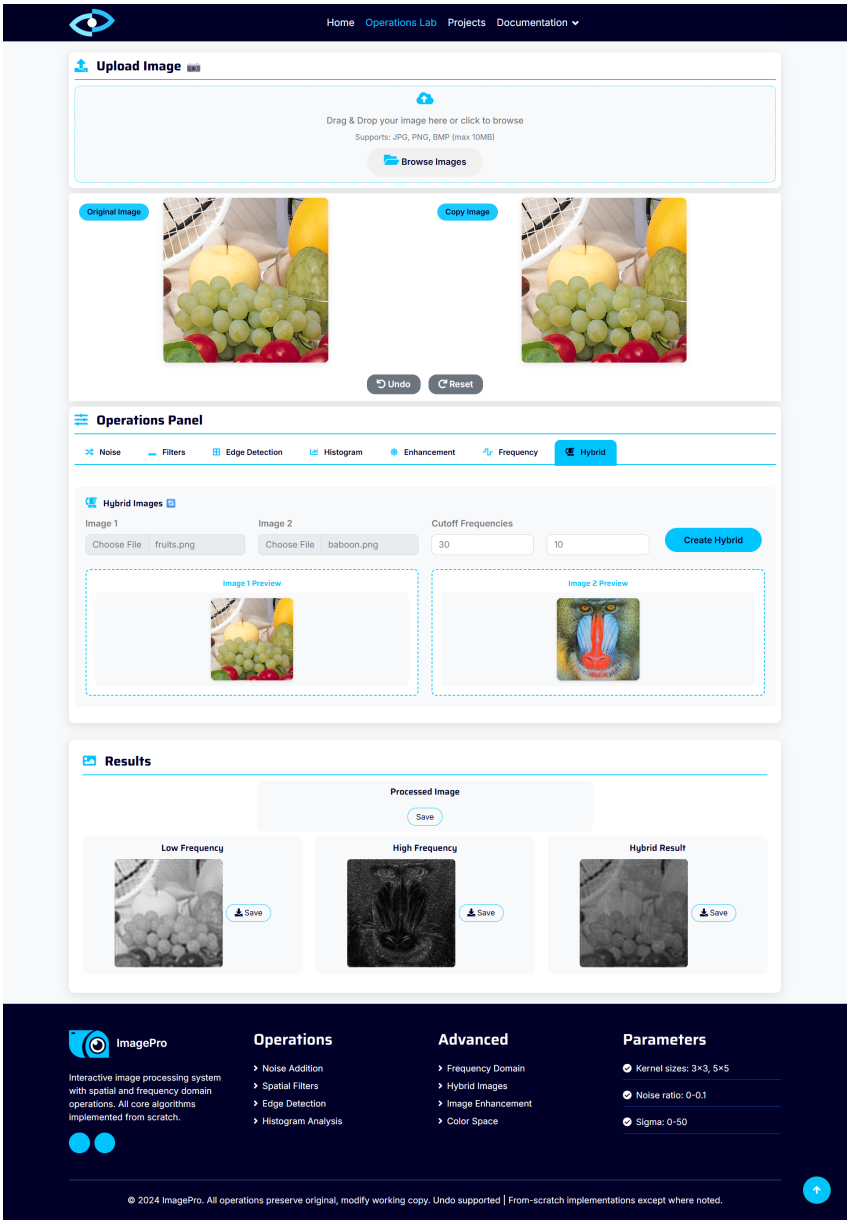

Figure 2: Operations User Interface

4 Task 1: Noise Addition

We implemented additive noise using OpenCV:

- Uniform Noise
- Gaussian Noise
- Salt & Pepper Noise

4.1 Observations

- Gaussian noise produces smooth intensity variation.

- Salt & Pepper noise creates sharp black and white pixels.
- Increasing variance increases distortion level.

Noise Type	Parameter	Effect
Gaussian	Variance ↑	Stronger distortion
Salt & Pepper	Density ↑	More impulse pixels
Uniform	Range ↑	Wider intensity spread

Table 1: Noise Parameter Effects

5 Task 2: Low Pass Filtering

Implemented filters:

- Average Filter (3x3, 5x5)
- Gaussian Filter (3x3, 5x5)
- Median Filter (3x3, 5x5)

5.1 Observations

- Median filter performs best for Salt & Pepper noise.
- Gaussian filter preserves edges better than average filter.
- Larger kernel size increases smoothing but blurs edges.

6 Task 3: Edge Detection

Implemented:

- Sobel (X and Y)
- Roberts
- Prewitt
- Canny (OpenCV built-in)

6.1 Observations

- Sobel provides directional edge information.
- Roberts is sensitive to noise.
- Prewitt is similar to Sobel but slightly less accurate.
- Canny produces the cleanest and thinnest edges.

Method	Noise Sensitivity	Edge Quality
Sobel	Medium	Good
Roberts	High	Weak
Prewitt	Medium	Moderate
Canny	Low	Excellent

Table 2: Edge Detection Comparison

7 Histogram and Distribution Curve

We plotted:

- Intensity histogram
- Cumulative distribution function (CDF)

7.1 Observations

- Low contrast images show narrow histogram distribution.
- Equalization spreads histogram across full intensity range.

8 Histogram Equalization

8.1 Observations

- Improves contrast in low contrast images.
- May over-enhance already high contrast images.

9 Normalization

Image intensities were scaled to [0, 255].

9.1 Observations

- Useful before thresholding.
- Improves dynamic range usage.

10 Frequency Domain Filtering

Implemented:

- Low Pass Filter (Gaussian)
- High Pass Filter

10.1 Observations

- Low pass removes noise but blurs image.
- High pass enhances edges.
- Cutoff frequency controls level of detail.

11 Hybrid Images

Hybrid images were created by:

- Applying low-pass filter to one image
- Applying high-pass filter to another image
- Adding both results

11.1 Observations

- Seen up close → high frequency image dominates.
- Seen from far → low frequency image dominates.

12 Conclusion

This project successfully implemented a complete image processing pipeline in a web-based interactive application.

Key achievements:

- Modular FastAPI backend
- Interactive frontend UI
- Parameter-based experimentation
- Spatial and frequency domain processing

The system allows testing different algorithms and parameters dynamically, which helps understand the behavior of image processing techniques practically.