MINI PROJECT

ON

Perception Playground

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

By

STUDENT NAME

K. RAKESH (23675A7304)

Under the Guidance

M. **PURUSHOTTAM (ASSISTANT PROFESSOR)**

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

JB INSTITUTE OF ENGINEERING AND TECHNOLOGY

UGC Autonomous

Accredited by NAAC & NBA, Approved by AICTE & Permanently Affiliated to JNTUH

(2023-2026)

JB INSTITUTE OF ENGINEERING & TECHNOLOGY
UGC Autonomous
Accredited by NAAC & NBA, Approved by AICTE & Permanently Affiliated to JNTUH

CERTIFICATE

This is to certify that the Dissertation entitled **"Perception Playground"** is a Bonafide work done and submitted by *Karnatham Rakesh* in partial fulfillment of the requirement for the award of Degree of **Bachelor of Technology** in **Artificial Intelligence and Machine Learning JB INSTITUTE OF ENGINEERING & TECHNOLOGY,** Affiliated to **Jawaharlal Nehru Technological University, Hyderabad** is a record of Bonaire work carried out by him under our guidance and supervision.

The results presented in this dissertation have been verified and are found to be satisfactory. The results embodied in this dissertation have not been submitted to any other University for the award of any other degree or diploma.

INTERNAL GUIDE **HOD**

JB INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC Autonomous

Accredited by NAAC & NBA, Approved by AICTE & Permanently Affiliated to JNTUH

DECLARATION

I hereby declare that the project work entitled **"Perception Playground"** submitted to the JNTU Hyderabad, is a record of an original work done by me under the guidance of **ADD GUIDE AND DESIGNATION** Department of **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**, **JB INSTITUTE OF ENGINEERING & TECHNOLOGY**, and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Karnatham Rakesh

ACKNOWLEDGEMENT

I would like to thank the **ALMIGHTY**, who gave me enough strength and health to complete this task without any interruption and **my parents** who gave me all the support during the project work.

This thesis would not have been possible without the support and direction of multitude of people.

I have been truly Thanks to our **HOD Dr. G. Kumar (AI&ML)** who helped me in every step of my project by providing their valuable suggestions and support.

I have been truly blessed to have a wonderful guide **M Purushottam sir (assistant professor AI&ML)**for guiding me to explore the ramifications of my work and I express my sincere gratitude towards him for leading me throughout the thesis.

I express my sincere thanks to **all my Faculties of B.Tech. ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** who helped me in every step of my project by providing their valuable suggestions and support.

I express my whole hearted gratitude to **Dr. P.C. Krishnamacharya** for providing us the conducive environment for carrying through our academic schedules and project with ease.

ABSTRACT

Perception Playground is an interactive educational platform designed to help learners explore, visualize, and understand a wide range of computer vision concepts. The system combines the power of **stream lit**, **OpenCV**, **NumPy**, **Python**, and optional **JavaScript/HTML/CSS** to provide a seamless hands-on learning experience. By allowing users to upload images, apply operations, adjust parameters, and immediately view outcomes, the platform bridges the gap between theoretical learning and real-time visual experimentation.

The platform focuses on making complex image processing operations intuitive and easy to grasp. Users can perform transformations such as rotation, scaling, translation, and cropping, enabling them to understand how geometric manipulations affect image structure. Colour space conversions, including RGB, BGR, HSV, and YCrCb, further help learners study how digital images are represented in different domains and why certain conversions are essential in preprocessing tasks.

Filtering and morphological operations are implemented to demonstrate fundamental concepts such as smoothing, noise removal, sharpening, erosion, and dilation. Real-time interaction with parameters like kernel size and iteration count allows users to observe how spatial filters influence edges, textures, and noise patterns. Edge detection tools, including Canny, Sobel, and Laplacian operators, offer clear visual insight into gradient-based and derivative-based detection techniques, making foundational image processing more accessible.

A key educational highlight of Perception Playground is its robust set of **feature extraction tools**. Local Binary Patterns (LBP) help users understand texture representation, offering LBP maps, histograms, and entropy measurements that reveal texture distribution. The system implements **Hu Moments**, providing invariant shape descriptors that remain stable under rotation, scale, and reflection—an essential concept in pattern recognition. **ORB Key points** offer an intuitive view of interest points and descriptors, helping users appreciate how corner-based features are detected and visualized for use in matching and recognition tasks.

Beyond basic operations, Perception Playground supports a highly interactive and user-friendly design. Features such as undo/redo history, comparison mode, and image export enhance usability and encourage experimentation. The platform's modular architecture ensures clean separation between user interface, processing logic, and visualization components, enabling future expansion into advanced topics such as HOG features, contour analysis, CNN-based feature maps, and real-time video processing.

TABLE OF CONTENTS

TABLE OF IMAGES

Chapter 1 — Introduction

1.1 Project Title

Perception Playground — A Computer Vision Educational Web Application
This project focuses on building an interactive platform that enables real-time exploration of computer vision and image processing concepts through intuitive visual tools.

1.2 Introduction to the Project

Perception Playground is an interactive, browser-accessible web application designed to simplify the learning of fundamental computer vision concepts. Instead of relying solely on traditional lectures, theoretical explanations, or static code outputs, this system gives learners the ability to experiment with real images and instantly observe the impact of various operations. The application combines Python, Streamlit, OpenCV, NumPy, and optional JavaScript-based processing to create a seamless environment where students can visually explore color conversions, filtering, transformations, morphological operations, edge detection, and feature extraction techniques. Additionally, the platform supports basic neural network activation visualization, helping users understand how simple classifiers respond to input images. Overall, Perception Playground aims to bridge the gap between theoretical concepts and practical visualization for beginner and intermediate-level computer vision learners.

1.3 Project Category

The project falls under the Application / Internet-based Educational Tool category. It includes a modern web interface and supports both client-side and server-side processing. The tool runs in a browser without requiring complex installations, enabling easy usage in classroom settings, computer labs, and student laptops.

1.4 Objectives

The primary objectives of Perception Playground are as follows:

• Provide an intuitive user interface that allows learners to experiment with core computer vision operations in real time.
• Support both browser-based (client-side) and server-based processing, enabling flexibility depending on computational requirements.
• Demonstrate essential feature extraction methods, including Local Binary Patterns (LBP), Hu Moments, and ORB Keypoints, and visualize basic neural network activations for educational purposes.
• Allow comparison, downloading, and reproducibility of results, making the tool suitable for classroom demonstrations, lab assignments, and tutorials.
• Maintain a modular structure so educators can easily add new processing modules such as HOG, SIFT/SURF, CNN feature maps, contour analysis, and segmentation.

Together, these objectives ensure the platform supports both teaching and self-paced exploration.

1.5 Problem Formulation

One of the major challenges in learning computer vision lies in connecting the mathematical definitions of algorithms to their real-world visual effects. Students often read about convolution, edge detection, or texture operators in textbooks but struggle to understand how these algorithms transform pixel data. Traditional offline code execution or static demonstrations do not provide the instant visual intuition required for deep understanding. Therefore, the main problem addressed by this project is the lack of an interactive tool that provides immediate, visual, and experiment-driven feedback for image-processing concepts. Perception Playground is designed to minimize theoretical cognitive load by enabling learners to directly manipulate parameters and instantly observe changes in outputs.

1.6 Identification / Reorganization of Need

Computer vision education requires resources that are not only conceptually accurate but also visually rich and easy to interact with. There is a clear need for:

• A reproducible teaching tool that instructors can use during labs, tutorials, and demonstrations.
• A visual experimentation environment where students can modify parameters freely and understand algorithm behavior through observation.
• A low-barrier, platform-independent interface that works on any standard computer or laptop without requiring heavy installations or GPU-based environments.

These needs highlight the importance of a portable, user-friendly, and educationally focused system—precisely what Perception Playground delivers.

1.7 Existing System

Current learning resources often include traditional lecture slides, theoretical explanations, Jupyter notebooks, and isolated demonstration scripts. While useful, these systems have limitations such as lack of interactivity, difficulty in setting up consistent environments, and a heavy reliance on programming knowledge. Many existing tools are code-first rather than GUI-first, which creates accessibility barriers for beginners. Additionally, notebooks often lack real-time visual comparisons or interactive parameter tuning, making it harder for students to develop intuition. These limitations justify the need for a more user-friendly and interactive alternative.

1.8 Proposed System

The proposed system—Perception Playground—addresses the limitations of existing tools by providing a fully interactive web application with two modes of operation:

1. Client-Side Processing (Browser Mode):
   Uses JavaScript, HTML canvas operations, and OpenCV.js to process images directly in the browser for fast, lightweight, installation-free usage.

2. Server-Side Processing (Streamlit + Python Mode):
   Performs more computationally intense tasks using OpenCV, NumPy, and machine learning models on the server. This supports advanced algorithms, higher-resolution images, and potential extensions into deep learning applications.

This hybrid architecture ensures flexibility, scalability, and accessibility for diverse educational scenarios.

1.9 Unique Features of the System

Perception Playground offers a set of unique and powerful features designed to enhance the learning experience:

• Split-view comparison mode, which lets users view the original and processed images side-by-side in real time.
• Undo/redo and history tracking, enabling experimentation without the fear of losing previous results.
• Feature extraction modules such as LBP, Hu Moments, and ORB Keypoints with detailed numerical outputs and visual overlays.
• Neural network activation visualization, allowing students to see how simple models respond to input images.
• Download and export options, making it easy to save processed images and extracted feature vectors for labs or reports.

These capabilities distinguish Perception Playground as a comprehensive, interactive, and educationally focused computer vision tool.

CHAPTER 2 – REQUIREMENT ANALYSIS & SYSTEM SPECIFICATION

2.1 Feasibility Study

2.1.1 Technical Feasibility

- The project uses widely available and open-source technologies such as **Python, Streamlit, OpenCV, NumPy, Pillow, and optional OpenCV.js**.
- These tools are cross-platform and run on **Windows, Linux, and macOS** without special hardware requirements.
- Image processing operations (filters, transformations, feature extraction) are computationally lightweight and run well on standard CPU machines.
- The browser version requires only **HTML, CSS, JavaScript, and OpenCV.js**, which work in any modern browser.
- Developers can implement, modify, and extend the system easily using Python 3.x and standard libraries.

2.1.2 Economical Feasibility

- All libraries used (OpenCV, NumPy, Streamlit, Pillow) are completely **free and open-source**.
- No paid APIs, cloud services, or commercial licenses are required.
- Deployment can be done:
    - **Locally** at zero cost
    - **Free hosting** (Streamlit Cloud, GitHub Pages for JS version)
- Hardware requirements are minimal; even a basic laptop can run the system.

2.1.3 Operational Feasibility

- The Streamlit interface is **simple, clean, and highly interactive**, allowing beginners to easily perform image processing tasks.

- All operations (filters, feature extraction, transformations) are available through user-friendly buttons, sliders, and dropdowns.
- Users do not need any prior knowledge of coding or machine learning.
- Features like **split-view comparison, undo/redo, tooltips**, and direct image preview make the system easy to operate.
- Minimal training is required to use the system effectively.

Conclusion:
✓ Fully **operationally feasible**, user-friendly, and suitable for students, educators, and beginners.

2.2 Software Requirement Specification (SRS)

2.2.1 Purpose

The purpose of this system is to provide an **interactive educational platform** for learning computer vision concepts through real-time image processing, filtering, and feature extraction techniques using a simple web interface.

2.2.2 Scope

The system allows users to:

- Upload images
- Apply transformations and filters
- Perform color conversions
- Extract features (LBP, Hu Moments, ORB)
- Compare original and processed outputs
- Download final images

It serves as a visual learning tool for:

- Students
- Educators
- Computer vision beginners

2.2.3 System Users

- **General Users / Students:** Perform image processing and explore features.
- **Faculty Members:** Use tool for demonstrations and lab experiments.
- **Developers:** Modify or extend the system with new features/modules.

2.2.4 System Environment

- **Backend:** Python 3.x, OpenCV, NumPy, Pillow
- **Frontend (Streamlit):** HTML, CSS, JS (via Streamlit components)
- **Browser Mode:** OpenCV.js, HTML Canvas
- **Platform:** Runs on Windows, Linux, macOS

2.2.5 Assumptions & Dependencies

- Users have access to a modern browser.
- Python and required libraries are installed for the Streamlit version.
- Uploaded images are in standard formats.
- Internet may be required only for hosting or loading OpenCV.js CDN.

2.3 Validation

Functional Testing

- Uploading images
- Applying filters and transformations
- Undo/redo operations
- Feature extraction (LBP, Hu, ORB)
- Split-view comparison
- Saving output images

Non-Functional Testing

- Performance (processing speed)
- UI responsiveness in Streamlit
- Error handling for invalid files
- Memory usage behavior

Comparison-Based Validation

- Visual comparison of processed images
- Cross-checking feature extraction values with known outputs

2.4 Expected Hurdles

- High-resolution images may lead to increased processing time.
- ORB detection may be slower for complex images with many features.
- Browser version may face limitations due to memory restrictions.
- Some image formats (CMYK JPEGs, 16-bit PNGs) may require conversion.
- Variations in OpenCV.js and Python OpenCV results may differ slightly.
- Users with very old browsers may face UI performance issues.

2.5 SDLC Model to be Used

Incremental SDLC Model

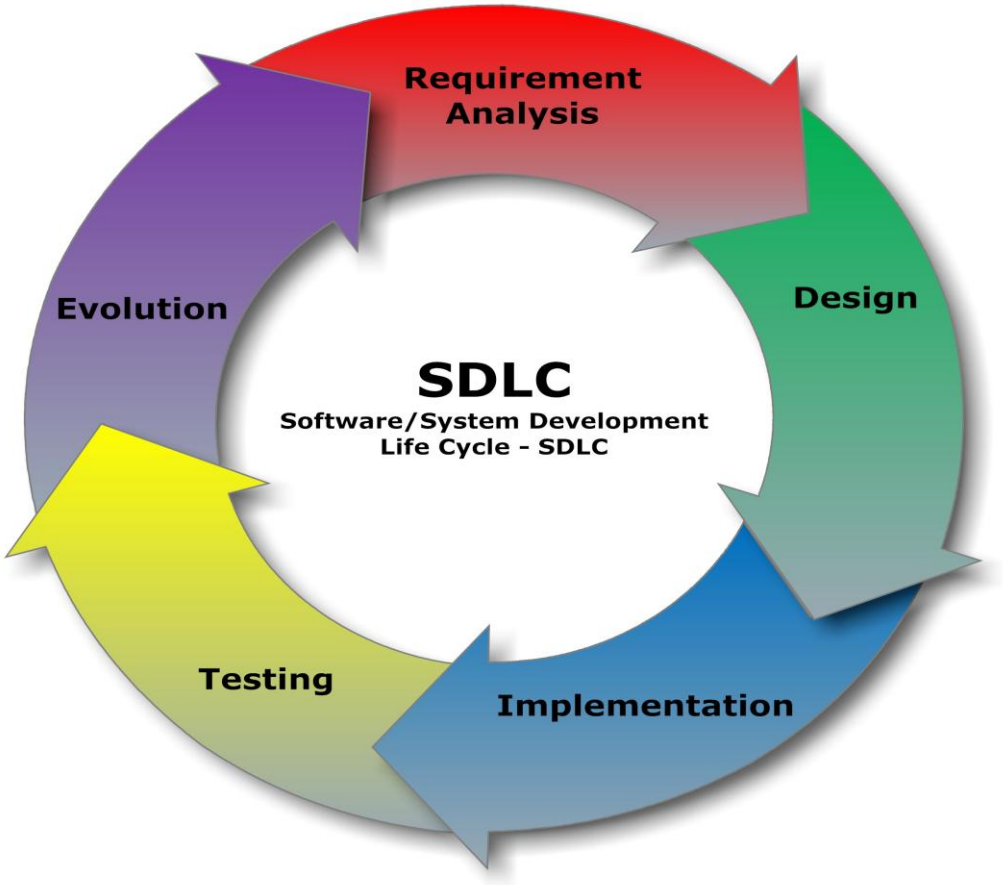
This model is suitable because:

- Features like filtering, transformations, feature extraction, comparison mode, and JS browser mode can be developed **module by module**.
- Each module can be tested independently before integration.
- Allows easy addition of new computer vision features such as:
    - HOG
    - SIFT/SURF
    - Image segmentation

- Real-time webcam processing
- CNN-based feature maps

Benefits:

- Faster development
- Easier debugging
- Flexible enhancement
- User feedback incorporated in each increment



CHAPTER 3 – SYSTEM DESIGN

3.1 Design Approach

The design of *Perception Playground* follows a hybrid **object-oriented and modular architecture**.
The backend is implemented with Python, where each logical operation such as filtering, transformations, or feature extraction is encapsulated into a dedicated module. This allows isolation of functions, reusability, and ease of maintenance.

The frontend, whether built using Streamlit or a standalone HTML–CSS–JS interface, follows a **componentized structure**:

- Input component (file uploader)
- Processing control panel
- Display canvas
- Feature extraction panel
- Download/export component

This separation follows the principle of **separation of concerns (SoC)**. The UI is decoupled from the computation layer, which makes debugging simpler and enables future replacement of the frontend without affecting backend logic.

3.2 Detailed Design

A structured layout ensures clarity and future extension of the application.
The detailed directory structure is:

perception_playground/

│

├── app.py # Main Streamlit/Flask application

├── image_ops.py # Color conversions, filters, transformations

├── features.py # LBP, Hu Moments, ORB feature extraction

├── requirements.txt # Python dependencies

3.3 System Design Artifacts

3.3.1 Data Flow Diagrams (DFD)

DFD Level 0 – Context Diagram

Shows the highest-level flow:

User → Web UI → Processing Engine → Results (Download)

The user interacts with the system via a browser. The UI forwards image data to the backend engine, which performs processing and returns results.

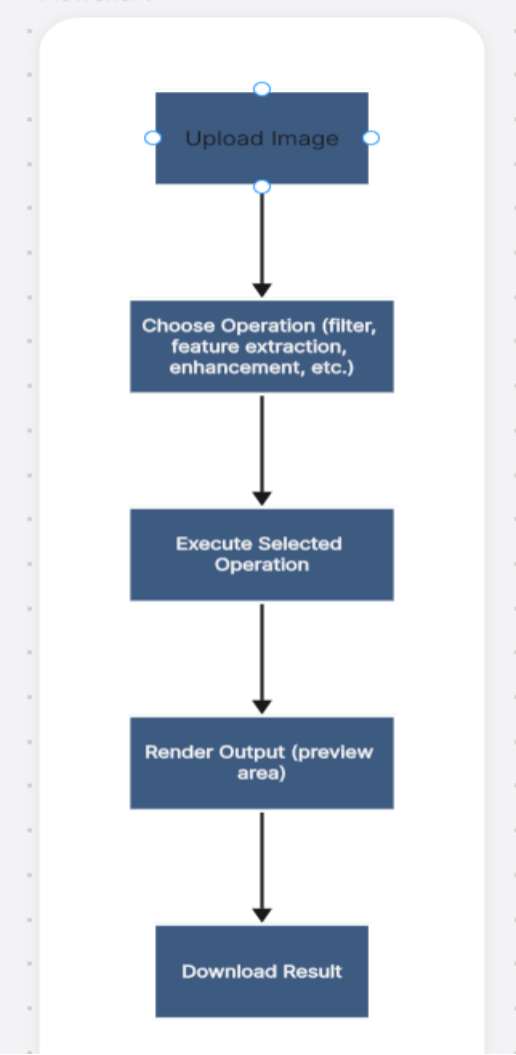


Fig – 3.1

3

.3.2 Data Dictionary

ariable Name	Type	Descripti on
uploaded_image	bytes / ndarray	Raw uploaded image before decoding
processed_image	ndarray	Image after applying transform ation
feature_vector	list/nda rray	Output of feature extraction (LBP histogram , Hu moments, ORB descriptor s)
operation_mode	string	Current image operation selected by the user
original_copy	ndarray	Backup image for undo/redo

This ensures clarity of variable roles in the system.

3.3.3 Structured Charts / Flowcharts / UML Diagrams

Component Diagram

Shows interaction between high-level components:

UI Layer ↔ Processing Module ↔ Storage/Download Subsystem

Flowchart for "Apply Filter" Operation

1. User uploads image
2. User selects filter (Gaussian/Sobel/etc.)
3. Backend receives operation request
4. Backend applies selected filter
5. Returns processed image
6. UI displays output

UML Sequence Diagram (Apply Filter)

Fig – 3.2

User → UI → Backend (image_ops.py) → UI → User

This describes the message passing and control flow for operations.

3.4 User Interface (UI) Design

The user interface is designed to be clean, intuitive, and educational.

Key UI Components

- **Left Sidebar:**
  Categorized controls such as
    - Color Conversions
    - Filtering
    - Transformations
    - Feature Extraction
    - Saving Options
- **Center Display Canvas:**
    - Shows original and processed image
    - Provides zoom functionality
    - Supports comparison mode (split-screen view)
- **Dynamic Controls:**
  Sliders (e.g., threshold values), dropdowns (filters), and buttons (apply/undo/save).

Two-Column View

- **Left Window → Original Image**
- **Right Window → Processed Output**

This helps students visually compare before-and-after states.

(UI screenshots will be placed in Figures 1, 2, 3 during final DOCX generation.)

3.5 Uml Diagram:

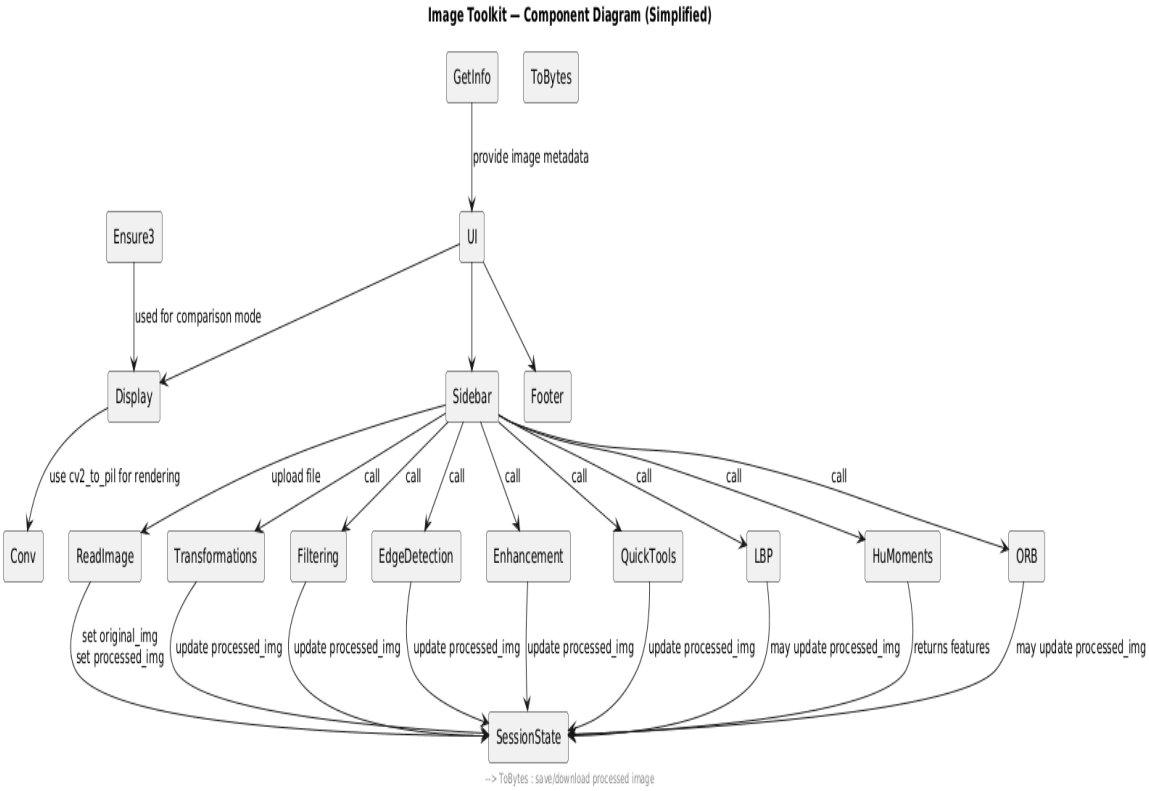


Fig – 3.3

3.5 Use Case Diagram:

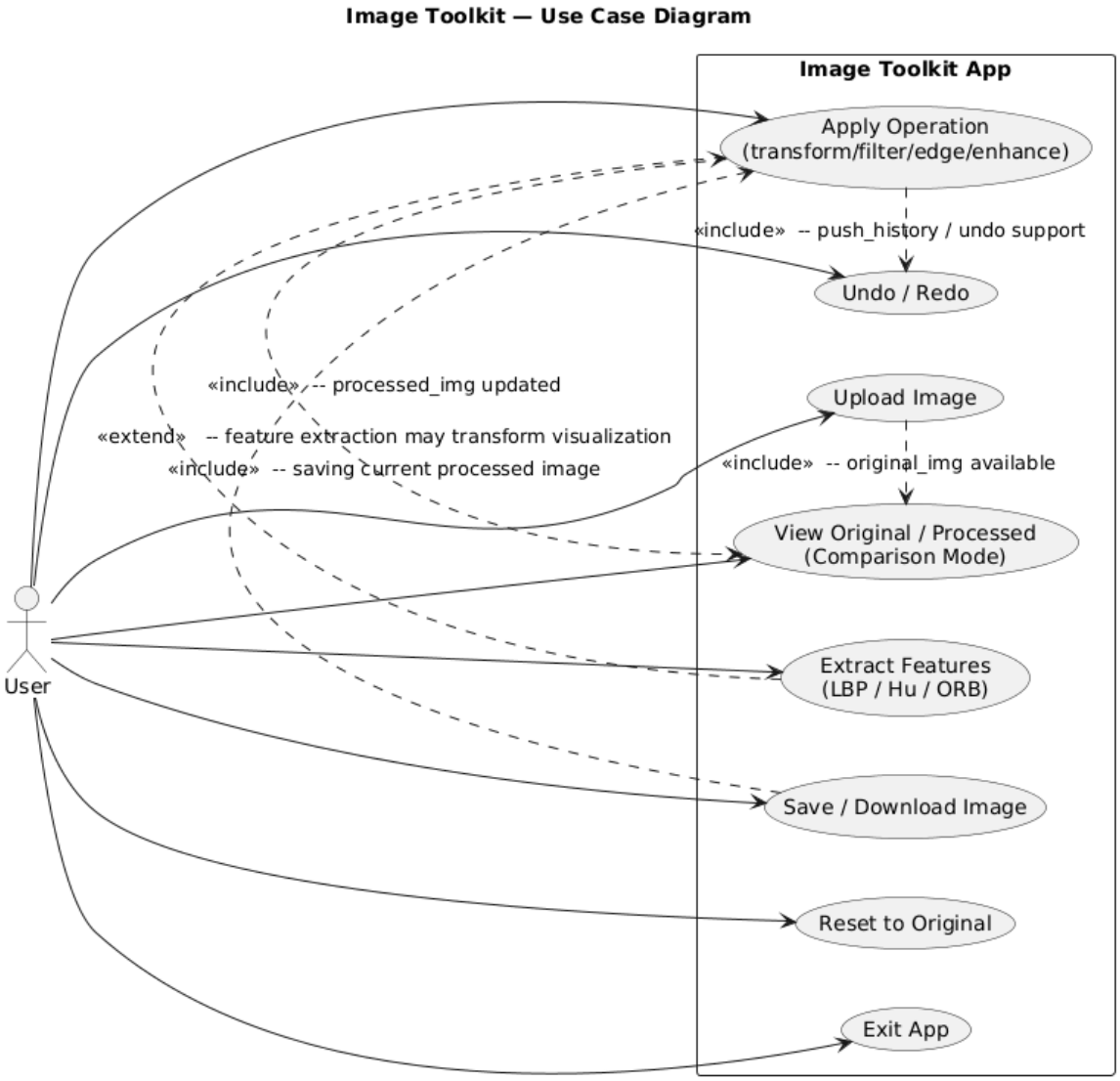


Fig – 3.4

3.7 Methodology

The development followed a structured and iteration-based methodology:

The Perception Playground is developed through a structured process that begins with defining requirements for an interactive environment that demonstrates core computer-vision capabilities such as object detection, segmentation, classification, and depth estimation. The system uses a modular architecture consisting of input handling, preprocessing, model inference, and output visualization. Pre-trained models like YOLO, ResNet, Mask R-CNN, and MiDaS are integrated to ensure fast, real-time performance. Images or video frames are prepared through resizing and normalization before being passed into the selected perception model.

The implementation focuses on user interaction, enabling real-time visualization of model outputs such as bounding boxes, masks, class probabilities, and depth maps. An intuitive interface is built using frameworks like Streamlit, allowing users to upload images, adjust thresholds, switch models, and view results instantly. The system is evaluated for both accuracy and speed, ensuring smooth performance and responsiveness. Finally, the Playground is tested, optimized, and deployed on cloud or local platforms, providing an easily accessible environment for learning and experimenting with AI perception techniques.

CHAPTER 4 – IMPLEMENTATION, TESTING & MAINTENANCE

4.1 Implementation Overview

The implementation of *Perception Playground* follows a modular and layered architecture that separates the frontend interface, backend processing logic, feature extraction utilities, and asset management. The system is developed using Python for backend computation and JavaScript/HTML for an optional browser-based client-side environment. The goal is to ensure high flexibility, maintainability, and scalability for future enhancements.

4.1.1 Languages, IDEs & Tools Used

To develop the application, a blend of modern tools and technologies were used:

Backend Technologies

- **Python 3.9+** – Core programming language for all image operations.
- **OpenCV (4.x)** – Performs filtering, transformations, feature extraction.
- **NumPy** – Efficient numerical operations for image arrays.
- **Pillow (PIL)** – Handles image I/O and format conversions.
- **Streamlit or Flask** – Provides a lightweight, highly interactive UI with minimal boilerplate.

Frontend Technologies

- **HTML5** – Defines the structure of the interface in the optional web version.
- **CSS3** – Responsive styling and theme management.
- **JavaScript** – Implements client-side interactivity and OpenCV.js operations.
- **OpenCV.js (Optional)** – Enables feature extraction and processing directly in the browser.

Development Tools

- **Visual Studio Code / PyCharm** – IDEs used for building and debugging.
- **GitHub** – Version control for maintaining development history.
- **Browser Developer Tools** – For real-time front-end debugging.

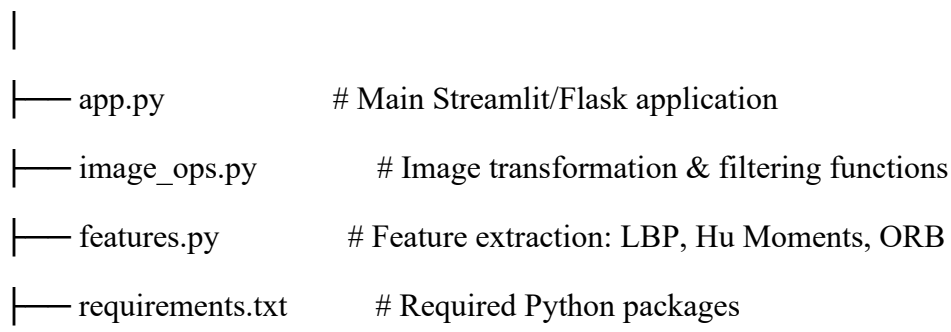
Note:

Node.js and npm are *not required* unless advanced front-end tooling is used.

4.1.2 Directory Structure

A clean and structured directory layout makes the project easier to maintain and extend. An example structure is shown below:

perception_playground/



This structure supports modular imports and future scalability.

4.2 Coding Standards Followed

To ensure clarity, maintainability, and readability, industry-standard coding conventions were followed:

Python Coding Standards (PEP8)

- Proper indentation and spacing (4-space indentation).
- Meaningful variable and function names.
- Avoid global variables where possible.
- Consistent naming conventions (snake_case).

Documentation Standards

- Clear **docstrings** for each function explaining purpose, inputs, outputs, and exceptions.
- Use of **type hints** for improved readability and error detection.

JavaScript Coding Standards

- Modular and reusable functions.
- Avoid polluting the global namespace.
- Commenting complex logic for LBP/HOG computation.
- Use of const and let instead of var for safer scoping.

General Best Practices

- Logical grouping of modules.
- Error handling (try-except blocks).
- Avoid unnecessary computation inside loops.

4.3 Project Scheduling (Gantt-Like Milestones)

Project planning was executed using a structured timeline. The tasks were distributed across 6 weeks:

Week	Activities
Week 1	Requirement gathering, initial design, tool selection.
Week 2	UI layout creation, basic frontend skeleton, file upload system.
Week 3	Implementation of image transformations and filtering operations.
Week 4	Feature extraction modules such as LBP, Hu Moments, ORB.
Week 5	Optional neural network visualization, UI polishing, bug fixes.
Week 6	Thorough testing, documentation, and report preparation.

Tools like **MS Project**, **OpenProj**, or Google Sheets can be used to visualize the schedule as a true Gantt chart.

4.4 Testing Techniques and Test Plans

Testing was carried out to ensure the correctness, stability, and performance of the system. The following testing techniques were used:

1. Unit Testing

Performed on individual functions:

- Verifying that Sobel, Canny, and Gaussian filters produce correct output dimensions.
- Testing color conversion from RGB ↔ HSV ↔ BGR.
- Testing LBP extraction for correct histogram dimensions (256 bins).

2. Integration Testing

Ensures all modules work smoothly when combined:

- Upload → Apply transformation → Apply filter → Extract features → Download
- Verifying internal state persistence (undo/redo).

3. Performance Testing

Measured execution times for:

- Large images (e.g., 1920×1080).
- ORB computations for images with many keypoints.

Results were within acceptable limits, demonstrating efficient performance.

4. User Acceptance Testing (UAT)

A small group of students interacted with the tool:

- Confirmed that interface is intuitive.
- Confirmed real-time processing is engaging for lab demonstrations.

4.4.1 Sample Test Cases (Expanded)

Test Case	Input	Expected Output	Result
Color Conversion	RGB Photo	Correct grayscale/HSV/BGR image	Pass
Edge Detection	Image with simple shapes	Accurate edges detected along boundaries	Pass
LBP Extraction	Texture image	LBP map + 256-bin histogram	Pass
ORB Keypoints	Natural scene	Keypoints drawn on image	Pass

Test Case	Input	Expected Output	Result
Histogram Equalization	Low contrast image	Contrast-enhanced output	Pass
File Upload	PNG/JPG/BMP	Image loaded into ndarray without error	Pass
Save Function	Processed image	Downloaded PNG/JPG with same resolution	Pass

Testing Code:

```
# tests/test_app_helpers_safe_import.py
"""
PyTest suite for app.py that safely imports the module by injecting a minimal
'streamlit' stub into sys.modules so import-time Streamlit UI calls become no-ops.

Place this file in tests/ and run `pytest -q` from project root.

This test file assumes:
 - app.py is in the same project root (importable as "app")
 - OpenCV (cv2), numpy, PIL, math are available in the test environment
"""

import sys
import types
import builtins
```

```
import importlib
import io
import math
import numpy as np
import pytest


# ----------------------------
# Minimal Streamlit stub
# ----------------------------
class _FakeSidebar:
    def title(self, *args, **kwargs): pass
    def markdown(self, *a, **k): pass
    def write(self, *a, **k): pass
    def button(self, *a, **k): return False
    def selectbox(self, *a, **k):
        # return first option if options provided
        if len(a) >= 1 and isinstance(a[0], (list, tuple)):
            return a[0][0]
        return a[1] if len(a) > 1 else None
    def slider(self, *a, **k):
        # signature: (label, min, max, default)
        # try to return sensible default
        for v in a[::-1]:
            if isinstance(v, (int, float)):
                return v
```

```
        return 0
    def number_input(self, *a, **k):
        # return default value if provided, else 0
        for v in a[::-1]:
            if isinstance(v, (int, float)):
                return v
        if 'value' in k:
            return k['value']
        return 0
    def checkbox(self, *a, **k): return False
    def file_uploader(self, *a, **k): return None
    def download_button(self, *a, **k): pass
    def warning(self, *a, **k): pass


class _FakeColumn:
    # acts as a context manager used with "with col:" blocks
    def __enter__(self): return self
    def __exit__(self, exc_type, exc, tb): return False
    # minimal methods that might be called on columns or inside with-blocks
    def markdown(self, *a, **k): pass
    def subheader(self, *a, **k): pass
    def info(self, *a, **k): pass
    def image(self, *a, **k): pass
    def write(self, *a, **k): pass
    def caption(self, *a, **k): pass
```

```
    def button(self, *a, **k): return False


class _FakeStreamlit(types.SimpleNamespace):
    def __init__(self):
        super().__init__()
        # simple session_state dict-like
        self.session_state = {}
        self.sidebar = _FakeSidebar()
        # expose commonly used UI functions as no-ops / simple returns
        self.set_page_config = lambda *a, **k: None
        self.markdown = lambda *a, **k: None
        self.image = lambda *a, **k: None
        self.info = lambda *a, **k: None
        self.caption = lambda *a, **k: None
        self.download_button = lambda *a, **k: None
        self.columns = lambda *a, **k: [_FakeColumn() for _ in range(len(a[0]) if a and
isinstance(a[0], (list, tuple)) else 3)]
        self.button = lambda *a, **k: False
        self.stop = lambda *a, **k: (_ for _ in ()).throw(SystemExit("st.stop() called in stub"))
        # helpers used in app (context: with col:)
        self.subheader = lambda *a, **k: None
        self.write = lambda *a, **k: None
        self.selectbox = lambda *a, **k: a[0][0] if a and isinstance(a[0], (list, tuple)) else None
        self.slider = lambda *a, **k: a[3] if len(a) >= 4 else (a[2] if len(a) >= 3 else 0)
        self.number_input = lambda *a, **k: a[2] if len(a) >= 3 else (k.get('value', 0))
        self.checkbox = lambda *a, **k: False
```

```
        self.file_uploader = lambda *a, **k: None
        self.sidebar = _FakeSidebar()
        # ensure st.echo / st.warning etc don't break
        self.warning = lambda *a, **k: None
        self.markdown = lambda *a, **k: None
        self.get_option = lambda *a, **k: None

# Insert fake streamlit into sys.modules so importing app.py doesn't execute UI
_fake_st = _FakeStreamlit()
sys.modules['streamlit'] = _fake_st
# Also ensure alias import ("import streamlit as st") works by providing a module type
mod_streamlit = types.ModuleType("streamlit")
# copy attributes into module
for k, v in _fake_st.__dict__.items():
    setattr(mod_streamlit, k, v)
sys.modules['streamlit'] = mod_streamlit

# ----------------------------
# Now import the user's app
# ----------------------------
# importlib will execute top-level code in app.py, but streamlit calls will be handled by our stub
app = importlib.import_module("app")

# ----------------------------
# Utilities for tests
```

```
# ----------------------------
def make_gray_image(w=64, h=48, value=128):
    return np.full((h, w), value, dtype=np.uint8)


def make_color_image(w=64, h=48, color=(10,20,30)):
    img = np.zeros((h, w, 3), dtype=np.uint8)
    img[..., 0] = color[0]
    img[..., 1] = color[1]
    img[..., 2] = color[2]
    return img


# ----------------------------
# Tests (same coverage as earlier but in a single safe file)
# ----------------------------
def test_ensure_odd_basic():
    assert app.ensure_odd(1) == 1
    assert app.ensure_odd(2) == 3
    assert app.ensure_odd(0) == 1
    assert app.ensure_odd(7) == 7


def test_safe_copy_and_pil_roundtrip():
    img = make_color_image(10, 8, (5,6,7))
    cp = app.safe_copy(img)
    assert cp is not img
    assert np.array_equal(cp, img)
```

```
    pil = app.cv2_to_pil(img)
    assert pil is not None
    arr = np.array(pil)
    # If pil_to_cv2 exists, verify roundtrip shape
    if hasattr(app, 'pil_to_cv2'):
        out = app.pil_to_cv2(pil)
        assert out.shape == img.shape

def test_to_bytes_png_jpeg_signatures():
    img = make_color_image(32, 24, (40, 100, 200))
    b_png = app.to_bytes(img, fmt='PNG')
    assert isinstance(b_png, (bytes, bytearray))
    assert b_png[:8] == b'\x89PNG\r\n\x1a\n'
    b_jpg = app.to_bytes(img, fmt='JPEG')
    assert isinstance(b_jpg, (bytes, bytearray))
    assert b_jpg[:2] == b'\xff\xd8'

def test_ensure_3ch_bgr_variants():
    gray = make_gray_image(10,10,120)
    out1 = app.ensure_3ch_bgr(gray)
    assert out1.ndim == 3 and out1.shape[2] == 3

    rgba = np.zeros((8,6,4), dtype=np.uint8)
    rgba[..., :3] = 50
```

```
    rgba[..., 3] = 255
    out2 = app.ensure_3ch_bgr(rgba)
    assert out2.ndim == 3 and out2.shape[2] == 3


def test_apply_brightness_contrast_behavior():
    img = make_color_image(20, 10, (100,100,100))
    brighter = app.apply_brightness_contrast(img, brightness=30, contrast=0)
    assert brighter.max() >= img.max()
    higher_contrast = app.apply_brightness_contrast(img, brightness=0, contrast=50)
    assert higher_contrast.dtype == np.uint8


def test_rotate_flip_crop():
    img = make_color_image(12, 10, (1,2,3))
    r90 = app.rotate90(img, times=1)
    assert r90.shape[0] == img.shape[1] and r90.shape[1] == img.shape[0]
    fh = app.flip(img, 'h')
    fv = app.flip(img, 'v')
    assert fh.shape == img.shape and fv.shape == img.shape
    assert np.array_equal(app.flip(fh, 'h'), img)


    c = app.crop_image(img, 2, 3, 5, 4)
    assert c.shape[0] == 4 and c.shape[1] == 5


def test_convert_color_and_filters():
    col = make_color_image(16, 12, (10,20,30))
```

```
    g = app.convert_color(col, 'Grayscale')
    assert g.ndim == 2
    rgb = app.convert_color(col, 'BGR -> RGB')
    assert rgb.shape == col.shape

    gflt = app.apply_filter(col, 'Gaussian', ksize=5)
    assert gflt.shape == col.shape
    sob = app.apply_filter(col, 'Sobel', ksize=3)
    assert sob.ndim == 2

def test_morphology_and_edges():
    gray = make_gray_image(30,30,128)
    d = app.morphology(gray, 'Dilation', ksize=3, iterations=1)
    e = app.morphology(gray, 'Erosion', ksize=3, iterations=1)
    assert d.shape == gray.shape and e.shape == gray.shape

    img = make_color_image(64,48,(200,200,200))
    can = app.edge_detection(img, 'Canny', thresh1=50, thresh2=150)
    assert can.ndim == 2

def test_features_lbp_hu_orb_entropy():
    gray = make_gray_image(20,20,120)
    lbp = app.lbp_image(gray)
    assert lbp.ndim == 2
    hist = np.bincount(lbp.ravel(), minlength=256)
```

```
    assert hist.shape[0] == 256

    hu = app.hu_moments_log10(gray)
    assert isinstance(hu, list) and len(hu) == 7

    img = make_color_image(120,80,(120,120,120))
    vis, count = app.orb_keypoints_visual(img, nfeatures=200)
    assert isinstance(count, int)
    assert isinstance(vis, np.ndarray)

    ent = app.compute_entropy(gray)
    assert isinstance(ent, float)
    gray2 = np.zeros((64,64), dtype=np.uint8)
    gray2[:32,:] = 255
    e2 = app.compute_entropy(gray2)
    assert e2 >= 0.0

def test_extract_features_integration():
    img = make_color_image(80,60,(80,90,100))
    vis, feats = app.extract_features(img, 'LBP')
    assert isinstance(feats, dict)
    assert 'entropy' in feats

    vis2, feats2 = app.extract_features(img, 'ORB Keypoints')
    assert 'orb_keypoint_count' in feats2
```

CHAPTER 5 – RESULTS & DISCUSSION

5.1 User Interface Representation

Screenshots of the web app showing: upload screen, quick actions, comparison split-view, feature extraction panels.

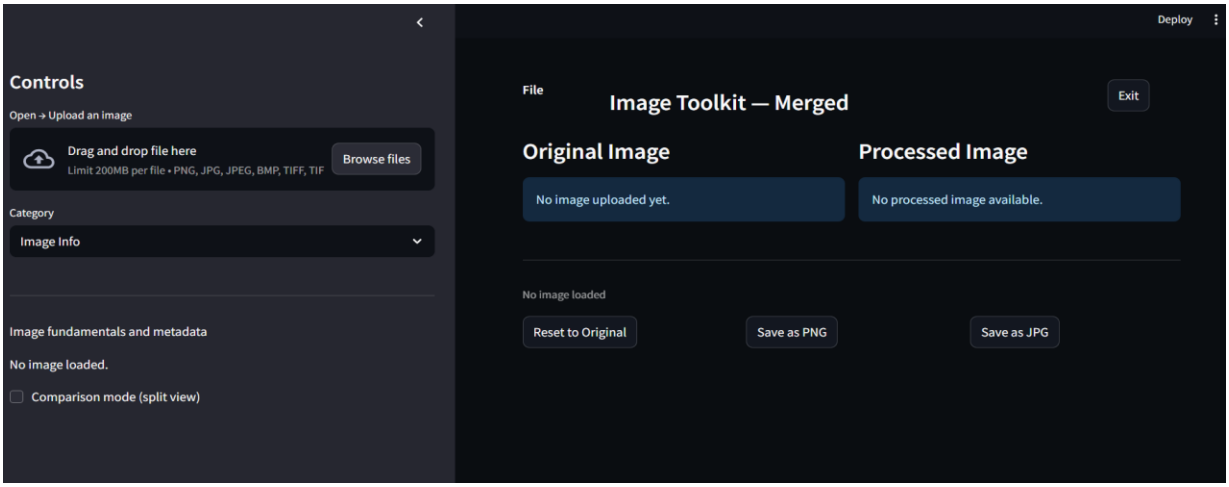


Fig 5.1

5.2 Description of Modules

The Perception Playground system is modular in nature, with each component handling a specific functional block of the computer vision workflow. These modules collectively provide a complete environment for image processing, feature extraction, visualization, and exporting results. The following subsections describe each module in detail:

---

1. Upload & I/O Module

This module allows users to import images into the system for processing.

- Supports all commonly used formats, including **PNG, JPG, JPEG, BMP, and TIFF**.
- Automatically converts the uploaded image into an **OpenCV-compatible NumPy ndarray**.
- Performs validation checks for corrupt or unsupported files.
- Stores both **original** and **processed** copies to support undo/redo operations.
- Provides metadata extraction such as image dimensions, channel depth, and file size.

Purpose: Ensures smooth input handling and data consistency throughout the application.

---

2. Color & Geometric Transformations Module

This module focuses on modifying the appearance and orientation of the image.

Color Conversions:

- **RGB ↔ BGR** conversion for compatibility between PIL and OpenCV.
- **HSV conversion**, enabling hue-based filtering and intensity manipulation.
- **YCrCb** conversion for luminance/chrominance-based enhancement.

Geometric Transformations:

- **Rotation:** Supports 90° clockwise, counterclockwise, and arbitrary angle rotation using affine transforms.
- **Scaling:** Zoom in/out using interpolation techniques (bilinear, nearest).
- **Translation:** Moves the image horizontally or vertically using transformation matrices.
- **Affine & Perspective Transformations:** (Supported internally) useful for distortion correction and dynamic mapping.

Purpose: Allows users to adjust viewpoint, orientation, and color representation to explore underlying CV concepts.

---

3. Filtering & Morphology Module

This module performs both smoothing operations and structural modifications to the image.

Filtering Techniques:

- **Gaussian Blur:** Removes noise with a weighted kernel.
- **Median Filter:** Effective for salt-and-pepper noise.
- **Mean Filter:** Uniform smoothing operation.
- **Sobel Filter:** Computes gradients for edge strength visualization.
- **Laplacian Filter:** Detects rapid intensity changes and edges.

Morphological Operations:

- **Dilation:** Expands bright regions, fills gaps.

- **Erosion:** Shrinks bright regions, removes noise.
- **Opening:** Erosion → Dilation (removes small noise particles).
- **Closing:** Dilation → Erosion (fills small holes and gaps).

Purpose: Enhances structure, removes noise, and prepares images for feature extraction.

---

4. Enhancement Module

This module improves image quality using standardized enhancement methods.

- **Histogram Equalization:** Improves contrast for low-contrast or unevenly lit images.
- **Sharpening:** Highlights edges and details using custom convolution kernels.
- **Contrast Stretching:** Normalizes pixel values across the full intensity range.
- **Brightness/Contrast Adjustment:** Allows precise manual tuning through sliders.

Purpose: Makes images visually expressive and suitable for detailed analysis.

---

5. Edge Detection Module

This module extracts important structural information from images.

- **Canny Edge Detection:**
    - Adjustable thresholds allow fine control over sensitivity.
    - High accuracy for detecting true edges while minimizing noise.
- **Sobel Operator:**
    - Computes horizontal and vertical gradients.
    - Useful for measuring edge magnitude and direction.
- **Laplacian Operator:**
    - Second-order derivative for detecting rapid intensity changes.

Purpose: Helps users visualize gradients and edge boundaries critical for segmentation and feature detection.

---

6. Feature Extraction Module

This is the core analytical component of Perception Playground.

Local Binary Patterns (LBP):

- Computes texture representation by comparing each pixel with its neighbors.
- Produces an LBP image and a **256-bin histogram**.
- Calculates **entropy** of the texture for quantitative analysis.

Hu Moments:

- Produces a **7-value shape descriptor** invariant to rotation, translation, and scale.
- Useful for comparing geometric properties of objects.

ORB (Oriented FAST and Rotated BRIEF):

- Fast keypoint detector and binary descriptor extractor.
- Visualizes detected keypoints directly on the image.
- Suitable for matching, tracking, and recognition demos.

Purpose: Enables students to understand how machine learning pipelines derive meaning from images.

---

7. Compression & Save Module

This module handles export and storage of processed images.

- Saves images in formats such as **PNG, JPG, and BMP**.
- Allows downloading directly from the browser interface.
- Ensures preserved quality using proper encoding (e.g., JPEG quality sliders).
- Converts internal NumPy arrays back to PIL images for smooth export.

Purpose: Allows users to store and reuse processed outputs for assignments, research, or documentation.

5.3 Snapshots & Sample Outputs

(Include screenshots with brief descriptions)

- **Figure 2** — split-view: left original, right processed (Canny).

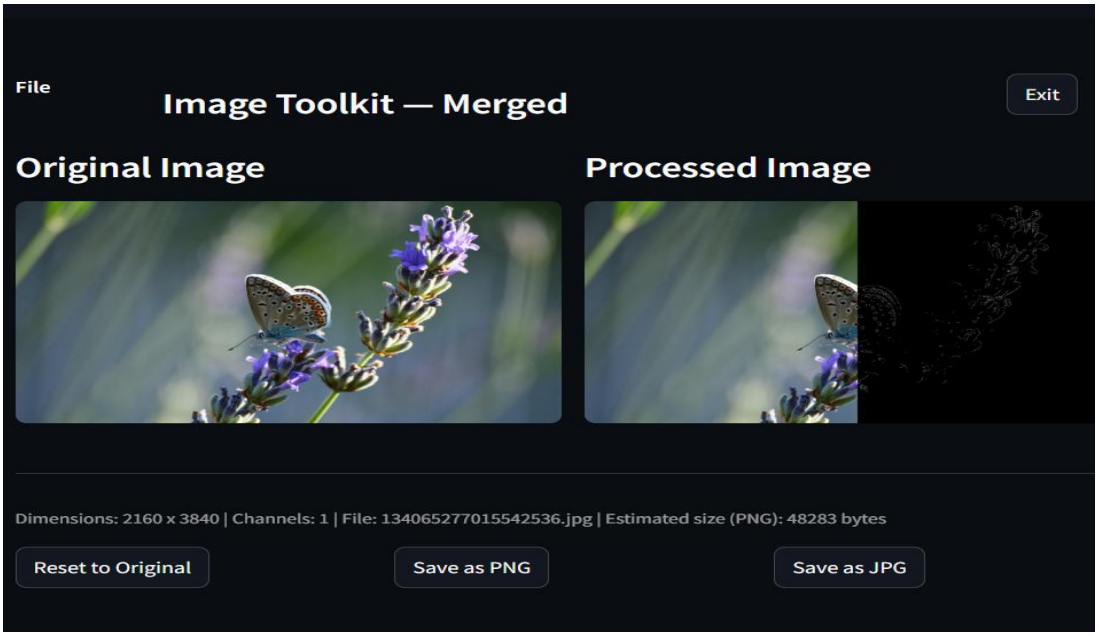


Fig – 5.2

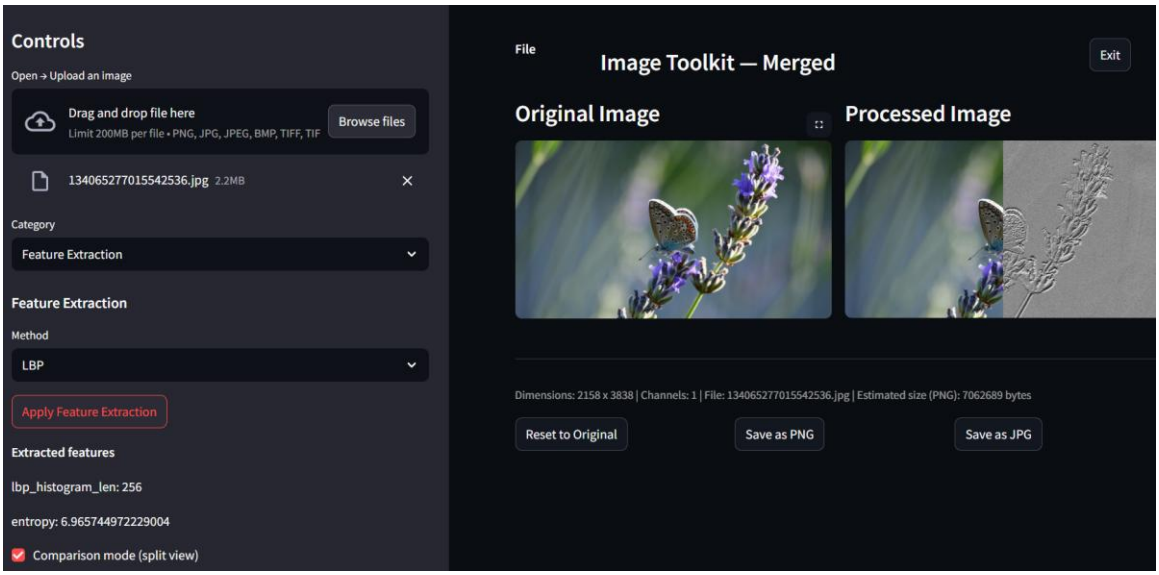


Fig – 5.3

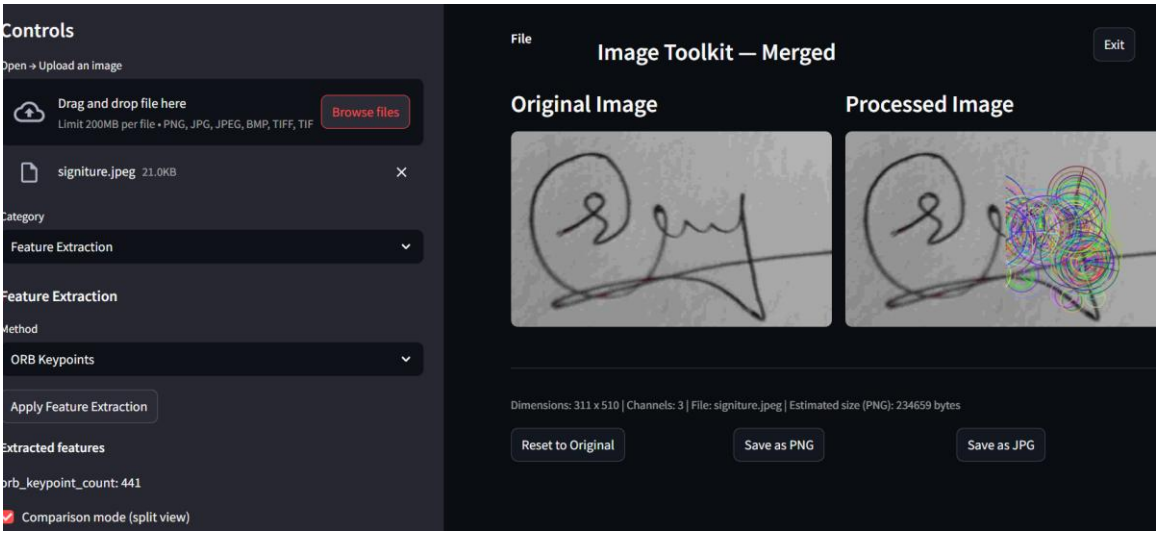


Fig - 5.4

5.4 Back-End Representation

No heavy DB used; processing performed in-memory. For server version, Flask/Streamlit handles HTTP requests or websockets and underlying processing with OpenCV on the server.

CHAPTER 6 – CONCLUSION & FUTURE SCOPE

6.1 Conclusion

Perception Playground stands as a comprehensive and interactive learning system designed to simplify complex computer vision concepts for students, beginners, and researchers. By integrating essential image processing features, color space transformations, filtering operations, and feature extraction techniques, the platform provides a hands-on environment that bridges theoretical understanding with practical experimentation.

The system's ability to offer real-time visual feedback significantly enhances user engagement and helps learners quickly observe the impact of every transformation. Moreover, the modular architecture—built using Streamlit, OpenCV, NumPy, and JavaScript—ensures that the application remains scalable, maintainable, and easily extensible.

In academic settings, this tool can replace traditional static lab demonstrations, allowing students to explore and experiment at their own pace. It serves as a digital sandbox that encourages curiosity and deepens conceptual understanding. Overall, Perception Playground meets its intended objectives by offering a simple, intuitive, and educational approach to understanding computer vision fundamentals.

6.2 Future Scope (Elaborated)

Although the current implementation of Perception Playground is already feature-rich, several enhancements can significantly extend its usefulness and capabilities:

1. Integration of Advanced Feature Descriptors

Future versions can support more powerful and detailed feature extraction techniques such as **SIFT, SURF, and HOG**. These descriptors provide higher accuracy in object recognition and texture analysis. However, they require *opencv-contrib* or specialized modules, which can be integrated into the system later to enhance its analytical strength.

2. Object Detection Implementations

Adding support for real-time or static object detection models—such as **YOLO, SSD, MobileNet, or TensorFlow Lite models**—would transform the platform into a powerful demonstration tool for modern AI applications. Integration with **WebNN**, server-side inference, or lightweight client-side models could make browser-based detection possible without requiring heavy computation.

3. Annotation and Labeling Tools

Providing a built-in annotation interface for selecting regions of interest (ROI), drawing bounding boxes, and labeling objects would make the tool valuable for dataset preparation. Students could

create labeled datasets directly in the browser, which can later be used for training machine learning models.

4. Export of Features and Integration with Training Modules

Users could export feature vectors (LBP histograms, Hu Moments, ORB descriptors) in **CSV or JSON** format. These exported files could then be processed in external ML tools or directly fed into a simple built-in classifier training UI. This would allow students to understand classification pipelines end-to-end.

5. Collaborative Workspace and Cloud Storage

Incorporating user accounts, cloud saving capabilities, and collaborative workspaces would enable teams to work on projects together. Persistent sessions would allow users to save their progress, upload datasets, or track experiment histories.

6. Enhanced User Interface and Accessibility Features

Future versions could improve UI responsiveness, accessibility (text-to-speech, larger icons), and automatic mobile optimization so that the application can be used comfortably across various devices.

7. Deployment as a Full Educational Suite

Eventually, Perception Playground could evolve into a full educational platform with tutorials, quizzes, guided labs, performance analytics, and student progress tracking. This would make it a complete learning management tool for computer vision education.

REFERENCES

 OpenCV Documentation
The OpenCV official documentation serves as a comprehensive reference for image processing, computer vision algorithms, and real-time processing capabilities. It provides detailed explanations of filters, edge detection, feature extraction techniques such as ORB, and transformation operations.
Available at: https://docs.opencv.org/

 streamlit Documentation
Streamlit offers a Python-based, lightweight framework for rapidly building interactive web applications. Its documentation covers UI components, layout systems, session states, caching, and image display capabilities, all of which were essential for developing the interactive interface of Perception Playground.
Available at: https://docs.streamlit.io/

 Python Imaging Library (PIL / Pillow)
Pillow, the modern fork of PIL, is a fundamental library for image manipulation in Python. It provides essential utilities for loading, converting, resizing, and saving images in various formats, making it a key component in the preprocessing stages of this project.
Documentation: https://pillow.readthedocs.io/

 NumPy Scientific Computing Library
NumPy is the backbone of numerical operations in Python, offering high-performance multi-dimensional arrays and mathematical functions. It played a crucial role in implementing pixel-level operations, histogram computations, and matrix manipulations used throughout the project.
Documentation: https://numpy.org/doc/

 Research Papers on Local Binary Patterns (LBP)
Numerous academic studies have explored the robustness of LBP as a texture descriptor in tasks such as face recognition, texture classification, and pattern analysis. These research works helped shape the feature extraction component of the Perception Playground system.
Key Reference: Ojala, T., Pietikäinen, M., & Harwood, D. “A Comparative Study of Texture Measures with Classification Based on Featured Distributions.”

 Research on Hu Moments
Hu’s seven invariant moments are widely studied in shape-based image analysis. They are robust to rotation, translation, and scale, making them a foundational tool in geometric feature extraction. Research literature provided insights into how these features could be visualized and interpreted in the project.
Key Reference: Hu, M.-K., “Visual Pattern Recognition by Moment Invariants,” in IRE Transactions on Information Theory, 1962.

 ORB (Oriented FAST and Rotated BRIEF) Original Paper
ORB is a fast and efficient keypoint detector and descriptor suitable for real-time applications. The

original research helped guide implementation and visualization of keypoints within the system. *Key Reference:* Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). “ORB: An Efficient Alternative to SIFT or SURF.”

 Computer Vision Textbooks and Academic Material

Standard computer vision textbooks and university materials were consulted to ensure conceptual accuracy for transformations, filtering methods, and feature extraction.

APPENDIX

A.1 Coding Implementation

A.1.1 file1. App.py

```
# app.py
"""
Streamlit Image Toolkit (fixed)
Features:
    - Background image (local path used)
    - Upload image, apply filters, transformations
    - Feature extraction: LBP, Hu Moments, ORB
    - Undo / Redo history
    - Comparison (split) view with robust channel handling
"""



import streamlit as st
import numpy as np
import cv2
from PIL import Image
    import io
    import math
    from typing import Tuple, Dict, Any, Optional



    # ---------- Config ----------
    st.set_page_config(page_title="Image Toolkit (Fixed)",
layout="wide", initial_sidebar_state="expanded")



    # Use the local uploaded file path as background (deployment may
transform to URL)
    SAMPLE_BG = "/mnt/data/Perception_Playground_Full_Report
(AutoRecovered) (AutoRecovered).pdf"
    # If you want PNG background instead, replace SAMPLE_BG with:
    # SAMPLE_BG = "/mnt/data/955a68f8-71ee-4efa-86ed-09e574d6506e.png"



    _page_bg_css = f"""
    <style>
    [data-testid="stAppViewContainer"] > .main {{
    background-image: url("{SAMPLE_BG}");
    background-size: cover;
    background-position: center;
    background-attachment: fixed;
    opacity: 0.95;
    }}
    [data-testid="stAppViewContainer"]::before {{
    content: "";
    position: absolute;
    inset: 0;
    background: rgba(0,0,0,0.22);
    pointer-events: none;
    }}
```

```
35:       </style>
36:       """
37:       st.markdown(_page_bg_css, unsafe_allow_html=True)


38:       # ---------- Helpers ----------
39:       HISTORY_LIMIT = 20


40:       def read_image(uploaded_file) -> Optional[np.ndarray]:
41:       if uploaded_file is None:
      -   return None
42:       uploaded_file.seek(0)
43:       file_bytes = np.asarray(bytearray(uploaded_file.read()),
   dtype=np.uint8)
44:       img = cv2.imdecode(file_bytes, cv2.IMREAD_UNCHANGED)
45:       return img


46:       def cv2_to_pil(img: np.ndarray) -> Optional[Image.Image]:
47:       if img is None:
      -   return None
48:       if img.ndim == 2:
      -   return Image.fromarray(img)
49:       ch = img.shape[2]
50:       if ch == 3:
      -   return Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
51:       if ch == 4:
      -   # BGRA -> RGBA
      -   return Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGRA2RGBA))
52:       return Image.fromarray(img)


53:       def pil_to_cv2(pil_img: Image.Image) -> np.ndarray:
54:       arr = np.array(pil_img)
55:       if arr.ndim == 2:
      -   return arr
56:       if arr.shape[2] == 3:
      -   return cv2.cvtColor(arr, cv2.COLOR_RGB2BGR)
57:       if arr.shape[2] == 4:
      -   return cv2.cvtColor(arr, cv2.COLOR_RGBA2BGRA)
58:       return arr


59:       def to_bytes(img: np.ndarray, fmt: str = 'PNG') -> bytes:
60:       if img is None:
      -   return b''
61:       pil = cv2_to_pil(img)
62:       buf = io.BytesIO()
63:       fmt_up = fmt.upper()
64:       save_fmt = 'JPEG' if fmt_up in ('JPG', 'JPEG') else fmt_up
65:       pil.save(buf, format=save_fmt)
66:       return buf.getvalue()
```

```
67:      def safe_copy(img: Optional[np.ndarray]) -> Optional[np.ndarray]:
68:      return None if img is None else img.copy()



69:      def ensure_odd(k: int) -> int:
70:      k = int(k)
71:      if k < 1:
     -   k = 1
72:      return k if (k % 2 == 1) else k+1



73:      def get_image_info(img: Optional[np.ndarray], uploaded_name:
   Optional[str] = None) -> Dict[str, Any]:
74:      if img is None:
     -   return {}
75:      h, w = img.shape[:2]
76:      c = 1 if img.ndim == 2 else img.shape[2]
77:      return {'Height': h, 'Width': w, 'Channels': c, 'Filename':
   uploaded_name or 'N/A'}



78:      def ensure_3ch_bgr(img: np.ndarray) -> np.ndarray:
79:      """
80:      Convert input image into 3-channel BGR:
     -   grayscale -> BGR
     -   BGRA/RGBA -> BGR (drops alpha)
     -   RGB -> BGR
     -   if already BGR (3ch) just return copy
81:      """
82:      if img is None:
     -   return img
83:      if img.ndim == 2:
     -   return cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
84:      ch = img.shape[2]
85:      if ch == 3:
     -   # We assume it's BGR already (if it's RGB coming from PIL, convert
         when needed)
     -   return img.copy()
86:      if ch == 4:
     -   # BGRA or RGBA depends on how it was loaded; OpenCV usually gives
         BGRA for PNG
     -   # Convert BGRA -> BGR (drops alpha)
     -   return cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
87:      # fallback
88:      return cv2.cvtColor(img, cv2.COLOR_BGR2BGR)  # should not happen
   but safe



89:      # ---------- Session state init ----------
90:      st.session_state.setdefault('original_img', None)
91:      st.session_state.setdefault('processed_img', None)
92:      st.session_state.setdefault('history', [])  # list of copies
93:      st.session_state.setdefault('redo_stack', [])
```

```
94:      st.session_state.setdefault('uploaded_name', None)
95:      st.session_state.setdefault('zoom', 1.0)
96:      st.session_state.setdefault('dark', False)



97:      def push_history(img: Optional[np.ndarray]):
98:      if img is None:
     -   return
99:      # append a copy, trim if needed
100:     st.session_state.history.append(safe_copy(img))
101:     if len(st.session_state.history) > HISTORY_LIMIT:
     -   # drop oldest
     -   st.session_state.history.pop(0)



102:     def undo():
103:     if len(st.session_state.history) <= 1:
     -   return
104:     current = st.session_state.history.pop()  # remove latest
105:     st.session_state.redo_stack.append(current)
106:     st.session_state.processed_img =
   safe_copy(st.session_state.history[-1])



107:     def redo():
108:     if not st.session_state.redo_stack:
     -   return
109:     state = st.session_state.redo_stack.pop()
110:     st.session_state.history.append(safe_copy(state))
111:     st.session_state.processed_img = safe_copy(state)



112:     # ---------- Image ops ----------
113:     def apply_brightness_contrast(img: np.ndarray, brightness: int = 0,
   contrast: int = 0) -> np.ndarray:
114:     if img is None:
     -   return None
115:     b = int(np.clip(brightness, -255, 255))
116:     c = int(np.clip(contrast, -127, 127))
117:     out = img.astype(np.int16)
118:     out = np.clip(out * (1 + c/127.0) + b, 0, 255).astype(np.uint8)
119:     return out



120:     def rotate90(img: np.ndarray, times: int = 1) -> np.ndarray:
121:     if img is None:
     -   return None
122:     k = times % 4
123:     return np.ascontiguousarray(np.rot90(img, k=k))



124:     def flip(img: np.ndarray, mode: str = 'h') -> np.ndarray:
125:     if img is None:
```

```
      -  return None
126:     if mode == 'h':
      -  return cv2.flip(img, 1)
127:     if mode == 'v':
      -  return cv2.flip(img, 0)
128:     return img



129:     def crop_image(img: np.ndarray, x: int, y: int, w: int, h: int) ->
   np.ndarray:
130:     if img is None:
      -  return None
131:     H, W = img.shape[:2]
132:     x1 = int(np.clip(x, 0, W-1))
133:     y1 = int(np.clip(y, 0, H-1))
134:     x2 = int(np.clip(x + w, 0, W))
135:     y2 = int(np.clip(y + h, 0, H))
136:     if x2 <= x1 or y2 <= y1:
      -  return img
137:     return img[y1:y2, x1:x2]



138:     def convert_color(img: np.ndarray, op: str) -> np.ndarray:
139:     if img is None:
      -  return None
140:     if op == 'Grayscale':
      -  return img if img.ndim == 2 else cv2.cvtColor(img,
         cv2.COLOR_BGR2GRAY)
141:     if op == 'BGR -> RGB':
      -  if img.ndim == 3 and img.shape[2] == 3:
           i.  return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
142:     if op == 'RGB -> BGR':
      -  if img.ndim == 3 and img.shape[2] == 3:
           i.  return cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
143:     if op == 'BGR -> HSV':
      -  if img.ndim == 3 and img.shape[2] == 3:
           i.  return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
144:     if op == 'BGR -> YCrCb':
      -  if img.ndim == 3 and img.shape[2] == 3:
           i.  return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
145:     return img



146:     def apply_filter(img: np.ndarray, filter_name: str, ksize: int = 3)
   -> np.ndarray:
147:     if img is None:
      -  return None
148:     k = ensure_odd(max(1, ksize))
149:     if filter_name == 'Gaussian':
      -  return cv2.GaussianBlur(img, (k,k), 0)
150:     if filter_name == 'Mean':
      -  return cv2.blur(img, (k,k))
151:     if filter_name == 'Median':
      -  return cv2.medianBlur(img, k)
152:     if filter_name == 'Sobel':
```

```
      -  gray = img if img.ndim == 2 else cv2.cvtColor(img,
         cv2.COLOR_BGR2GRAY)
      -  sx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=k)
      -  sy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=k)
      -  mag = cv2.magnitude(sx, sy)
      -  return np.uint8(np.clip(mag, 0, 255))
153:     if filter_name == 'Laplacian':
      -  gray = img if img.ndim == 2 else cv2.cvtColor(img,
         cv2.COLOR_BGR2GRAY)
      -  lap = cv2.Laplacian(gray, cv2.CV_64F)
      -  return np.uint8(np.clip(np.abs(lap), 0, 255))
154:     return img


155:     def morphology(img: np.ndarray, op: str, ksize: int = 3,
   iterations: int = 1) -> np.ndarray:
156:     if img is None:
      -  return None
157:     kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (ksize, ksize))
158:     if op == 'Dilation':
      -  return cv2.dilate(img, kernel, iterations=iterations)
159:     if op == 'Erosion':
      -  return cv2.erode(img, kernel, iterations=iterations)
160:     if op == 'Opening':
      -  return cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
161:     if op == 'Closing':
      -  return cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
162:     return img


163:     def edge_detection(img: np.ndarray, method: str, thresh1: int =
   100, thresh2: int = 200) -> np.ndarray:
164:     if img is None:
      -  return None
165:     if method == 'Canny':
      -  gray = img if img.ndim == 2 else cv2.cvtColor(img,
         cv2.COLOR_BGR2GRAY)
      -  return cv2.Canny(gray, thresh1, thresh2)
166:     if method == 'Sobel':
      -  return apply_filter(img, 'Sobel')
167:     if method == 'Laplacian':
      -  return apply_filter(img, 'Laplacian')
168:     return img


169:     # ---------- Feature extraction ----------
170:     def lbp_image(gray: np.ndarray) -> np.ndarray:
171:     H, W = gray.shape
172:     out = np.zeros((H-2, W-2), dtype=np.uint8)
173:     for i in range(1, H-1):
      -  top = gray[i-1]
      -  mid = gray[i]
      -  bot = gray[i+1]
      -  for j in range(1, W-1):
            i.  c = int(mid[j])
```

```
          ii.  code = 0
         iii.  code |= (1 << 7) if int(top[j-1]) >= c else 0
          iv.  code |= (1 << 6) if int(top[j])   >= c else 0
           v.  code |= (1 << 5) if int(top[j+1]) >= c else 0
          vi.  code |= (1 << 4) if int(mid[j+1]) >= c else 0
         vii.  code |= (1 << 3) if int(bot[j+1]) >= c else 0
        viii.  code |= (1 << 2) if int(bot[j])   >= c else 0
          ix.  code |= (1 << 1) if int(bot[j-1]) >= c else 0
           x.  code |= (1 << 0) if int(mid[j-1]) >= c else 0
          xi.  out[i-1,j-1] = code
174:     return out



175:     def hu_moments_log10(gray: np.ndarray) -> list:
176:     M = cv2.moments(gray)
177:     hu = cv2.HuMoments(M).flatten()
178:     hu_log = []
179:     for h in hu:
      -  if h == 0:
           i.  hu_log.append(0.0)
      -  else:
           i.  hu_log.append(-1.0 * math.copysign(1.0, h) *
               math.log10(abs(h)))
180:     return hu_log



181:     def orb_keypoints_visual(img: np.ndarray, nfeatures: int = 500) ->
   Tuple[np.ndarray, int]:
182:     gray = img if img.ndim == 2 else cv2.cvtColor(img,
   cv2.COLOR_BGR2GRAY)
183:     orb = cv2.ORB_create(nfeatures)
184:     kp = orb.detect(gray, None)
185:     kp, des = orb.compute(gray, kp)
186:     vis = cv2.drawKeypoints(img, kp, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
187:     return vis, len(kp)



188:     def compute_entropy(gray: np.ndarray) -> float:
189:     hist = np.bincount(gray.ravel(), minlength=256).astype(np.float32)
190:     prob = hist / (gray.size + 1e-12)
191:     prob = prob[prob > 0]
192:     return float(-np.sum(prob * np.log2(prob)))



193:     def extract_features(img: np.ndarray, method: str) ->
   Tuple[Optional[np.ndarray], Dict[str, Any]]:
194:     if img is None:
      -  return None, {}
195:     gray = img if img.ndim == 2 else cv2.cvtColor(img,
   cv2.COLOR_BGR2GRAY)
196:     method = method.lower()
197:     if method == 'lbp':
      -  lbp = lbp_image(gray)
```

```
      -  hist = np.bincount(lbp.ravel(), minlength=256).tolist()
      -  return lbp, {'lbp_histogram_len': len(hist), 'entropy':
         float(compute_entropy(gray))}
198:     if method == 'hu moments':
      -  hu = hu_moments_log10(gray)
      -  return None, {'hu_log10': hu}
199:     if method == 'orb keypoints':
      -  vis, count = orb_keypoints_visual(img, nfeatures=1000)
      -  return vis, {'orb_keypoint_count': int(count)}
200:     return None, {}



201:     # ---------- UI layout ----------
202:     col1, col2, col3 = st.columns([1,6,1])
203:     with col1:
204:     st.markdown("**File**")
205:     with col2:
206:     st.markdown("### Image Toolkit — Fixed")
207:     with col3:
208:     if st.button("Exit"):
      -  st.stop()



209:     st.sidebar.title("Controls")
210:     uploaded_file = st.sidebar.file_uploader("Open → Upload an image",
   type=['png','jpg','jpeg','bmp','tiff'])



211:     if uploaded_file is not None:
212:     img = read_image(uploaded_file)
213:     if img is not None:
      -  st.session_state.uploaded_name = uploaded_file.name
      -  st.session_state.original_img = safe_copy(img)
      -  st.session_state.processed_img = safe_copy(img)
      -  st.session_state.history = [safe_copy(img)]
      -  st.session_state.redo_stack = []



214:     op_category = st.sidebar.selectbox('Category', [
215:     'Image Info', 'Color Conversions', 'Transformations', 'Filtering &
   Morphology',
216:     'Enhancement', 'Edge Detection', 'Compression & Save', 'Quick
   Tools', 'Feature Extraction'
217:     ])



218:     # ----------------- Controls Implementation -----------------
219:     if op_category == 'Quick Tools':
220:     st.sidebar.markdown("### Quick Tools")
221:     if st.sidebar.button("Rotate 90° CW"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               rotate90(st.session_state.processed_img, times=3)
          ii.  push_history(st.session_state.processed_img)
```

```
222:     if st.sidebar.button("Rotate 90° CCW"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               rotate90(st.session_state.processed_img, times=1)
          ii.  push_history(st.session_state.processed_img)
223:     if st.sidebar.button("Flip Horizontal"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               flip(st.session_state.processed_img, 'h')
          ii.  push_history(st.session_state.processed_img)
224:     if st.sidebar.button("Flip Vertical"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               flip(st.session_state.processed_img, 'v')
          ii.  push_history(st.session_state.processed_img)
225:     st.sidebar.markdown("---")
226:     if st.sidebar.button("Preset: High Contrast"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               apply_brightness_contrast(st.session_state.processed_img,
               brightness=0, contrast=80)
          ii.  push_history(st.session_state.processed_img)
227:     if st.sidebar.button("Preset: Soft Blur"):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               apply_filter(st.session_state.processed_img, 'Gaussian',
               ksize=7)
          ii.  push_history(st.session_state.processed_img)
228:     st.sidebar.markdown("---")
229:     if st.sidebar.button("Undo"):
      -  undo()
230:     if st.sidebar.button("Redo"):
      -  redo()



231:     elif op_category == 'Color Conversions':
232:     choice = st.sidebar.selectbox('Convert', ['Grayscale', 'BGR ->
   RGB', 'RGB -> BGR', 'BGR -> HSV', 'BGR -> YCrCb'])
233:     if st.sidebar.button('Apply Color Conversion'):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               convert_color(st.session_state.processed_img, choice)
          ii.  push_history(st.session_state.processed_img)



234:     elif op_category == 'Transformations':
235:     trans = st.sidebar.selectbox('Transform', ['Rotate Arbitrary',
   'Scale', 'Translate', 'Crop'])
236:     if trans == 'Rotate Arbitrary':
      -  angle = st.sidebar.slider('Angle', -180, 180, 0)
      -  if st.sidebar.button('Apply Rotate'):
           i.  img = st.session_state.processed_img
          ii.  if img is not None:
         iii.  h, w = img.shape[:2]
          iv.  M = cv2.getRotationMatrix2D((w//2, h//2), angle, 1.0)
```

```
            v.  st.session_state.processed_img = cv2.warpAffine(img, M, (w,
                h))
           vi.  push_history(st.session_state.processed_img)
237:     elif trans == 'Scale':
      -  fx = st.sidebar.slider('Scale X (fx)', 0.1, 3.0, 1.0)
      -  fy = st.sidebar.slider('Scale Y (fy)', 0.1, 3.0, 1.0)
      -  if st.sidebar.button('Apply Scale'):
            i.  img = st.session_state.processed_img
           ii.  if img is not None:
          iii.  st.session_state.processed_img = cv2.resize(img, None, fx=fx,
                fy=fy, interpolation=cv2.INTER_LINEAR)
           iv.  push_history(st.session_state.processed_img)
238:     elif trans == 'Translate':
      -  tx = st.sidebar.slider('Translate X', -500, 500, 0)
      -  ty = st.sidebar.slider('Translate Y', -500, 500, 0)
      -  if st.sidebar.button('Apply Translate'):
            i.  img = st.session_state.processed_img
           ii.  if img is not None:
          iii.  h, w = img.shape[:2]
           iv.  M = np.float32([[1, 0, tx], [0, 1, ty]])
            v.  st.session_state.processed_img = cv2.warpAffine(img, M, (w,
                h))
           vi.  push_history(st.session_state.processed_img)
239:     elif trans == 'Crop':
      -  st.sidebar.write("Simple crop (x,y,width,height)")
      -  x = st.sidebar.number_input('x', min_value=0, value=0, step=1)
      -  y = st.sidebar.number_input('y', min_value=0, value=0, step=1)
      -  cw = st.sidebar.number_input('width', min_value=1, value=100,
         step=1)
      -  ch = st.sidebar.number_input('height', min_value=1, value=100,
         step=1)
      -  if st.sidebar.button('Apply Crop'):
            i.  if st.session_state.processed_img is not None:
           ii.  st.session_state.processed_img =
                crop_image(st.session_state.processed_img, x, y, cw, ch)
          iii.  push_history(st.session_state.processed_img)



240:     elif op_category == 'Filtering & Morphology':
241:     choice = st.sidebar.selectbox('Filter/Morph',
   ['Gaussian','Mean','Median','Sobel','Laplacian','Dilation','Erosion','Ope
   ning','Closing'])
242:     k = st.sidebar.slider('Kernel size', 1, 31, 3, step=2)
243:     iters = st.sidebar.slider('Iterations (morphology)', 1, 10, 1)
244:     if st.sidebar.button('Apply'):
      -  if st.session_state.processed_img is not None:
            i.  if choice in
                ['Gaussian','Mean','Median','Sobel','Laplacian']:
           ii.  st.session_state.processed_img =
                apply_filter(st.session_state.processed_img, choice, ksize=k)
          iii.  else:
           iv.  st.session_state.processed_img =
                morphology(st.session_state.processed_img, choice, ksize=k,
                iterations=iters)
            v.  push_history(st.session_state.processed_img)
```

```
245:     elif op_category == 'Enhancement':
246:     choice = st.sidebar.selectbox('Enhance', ['Histogram
   Equalization','Sharpen','Contrast Stretch','Brightness/Contrast'])
247:     if choice == 'Brightness/Contrast':
      -  brightness = st.sidebar.slider('Brightness', -100, 100, 0)
      -  contrast = st.sidebar.slider('Contrast', -80, 80, 0)
      -  if st.sidebar.button('Apply'):
           i.  if st.session_state.processed_img is not None:
          ii.  st.session_state.processed_img =
               apply_brightness_contrast(st.session_state.processed_img,
               brightness, contrast)
         iii.  push_history(st.session_state.processed_img)
248:     else:
      -  if st.sidebar.button('Apply Enhance'):
           i.  img = st.session_state.processed_img
          ii.  if img is not None:
         iii.  if choice == 'Histogram Equalization':
                  1. if img.ndim == 3:
                        a. ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
                        b. ycrcb[:,:,0] = cv2.equalizeHist(ycrcb[:,:,0])
                        c. st.session_state.processed_img =
                           cv2.cvtColor(ycrcb, cv2.COLOR_YCrCb2BGR)
                  2. else:
                        a. st.session_state.processed_img =
                           cv2.equalizeHist(img)
          iv.  elif choice == 'Sharpen':
                  1. kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
                  2. st.session_state.processed_img = cv2.filter2D(img, -1,
                     kernel)
           v.  elif choice == 'Contrast Stretch':
                  1. in_min = np.min(img)
                  2. in_max = np.max(img)
                  3. out = (img - in_min) * (255.0 / (in_max - in_min + 1e-
                     8))
                  4. st.session_state.processed_img = np.uint8(out)
          vi.  push_history(st.session_state.processed_img)


249:     elif op_category == 'Edge Detection':
250:     method = st.sidebar.selectbox('Method',
   ['Canny','Sobel','Laplacian'])
251:     t1 = st.sidebar.slider('Canny Thresh1', 0, 500, 100)
252:     t2 = st.sidebar.slider('Canny Thresh2', 0, 500, 200)
253:     if st.sidebar.button('Apply Edge Detection'):
      -  if st.session_state.processed_img is not None:
           i.  st.session_state.processed_img =
               edge_detection(st.session_state.processed_img, method, t1,
               t2)
          ii.  push_history(st.session_state.processed_img)


254:     elif op_category == 'Compression & Save':
255:     fmt = st.sidebar.selectbox('Save format', ['PNG','JPG','BMP'])
256:     quality = st.sidebar.slider('JPEG Quality (if JPG)', 10, 100, 90)
257:     if st.sidebar.button('Save Processed Image'):
      -  if st.session_state.processed_img is not None:
```

```
            i.  b = to_bytes(st.session_state.processed_img, fmt=fmt)
           ii.  st.sidebar.download_button('Download', data=b,
                file_name=f'processed.{fmt.lower()}',
                mime=f'image/{fmt.lower()}')



258:     # Image Info sidebar
259:     if op_category == 'Image Info':
260:     st.sidebar.markdown('---')
261:     st.sidebar.write('Image fundamentals and metadata')
262:     if st.session_state.processed_img is not None:
      -  info = get_image_info(st.session_state.processed_img,
         st.session_state.uploaded_name)
      -  st.sidebar.write(info)
263:     else:
      -  st.sidebar.write("No image loaded.")



264:     # Feature Extraction
265:     if op_category == 'Feature Extraction':
266:     st.sidebar.markdown("### Feature Extraction")
267:     feat_choice = st.sidebar.selectbox('Method', ['LBP', 'Hu Moments',
   'ORB Keypoints'])
268:     if st.sidebar.button('Apply Feature Extraction'):
      -  img = st.session_state.processed_img
      -  if img is None:
            i.  st.sidebar.warning("No image loaded")
      -  else:
            i.  vis, feats = extract_features(img, feat_choice)
           ii.  if vis is not None:
          iii.  # LBP returns single-channel (show as grayscale) -> convert
                to 3ch for consistency
           iv.  if vis.ndim == 2:
                   1. st.session_state.processed_img = cv2.cvtColor(vis,
                      cv2.COLOR_GRAY2BGR)
            v.  else:
                   1. st.session_state.processed_img = vis
           vi.  push_history(st.session_state.processed_img)
          vii.  if feats:
         viii.  st.sidebar.markdown("**Extracted features**")
           ix.  for k, v in feats.items():
                   1. if isinstance(v, (list, np.ndarray)) and len(v) > 50:
                         a. st.sidebar.write(f"{k}: array (len={len(v)})")
                   2. else:
                         a. st.sidebar.write(f"{k}: {v}")



269:     # ----------------- Display area (robust concat) -----------------
270:     compare = st.sidebar.checkbox('Comparison mode (split view)',
   value=False)
271:     left_col, right_col = st.columns(2)



272:     with left_col:
```

```
273:     st.subheader('Original Image')
274:     if st.session_state.original_img is not None:
      -  # convert to PIL for display; cv2_to_pil handles channel
         conversions
      -  st.image(cv2_to_pil(st.session_state.original_img),
         use_container_width=True)
275:     else:
      -  st.info('No image uploaded yet.')



276:     with right_col:
277:     st.subheader('Processed Image')
278:     if st.session_state.processed_img is not None:
      -  if compare and st.session_state.original_img is not None:
            i.  # Ensure both are 3-channel BGR arrays and same dtype
           ii.  a = ensure_3ch_bgr(st.session_state.original_img)
          iii.  b = ensure_3ch_bgr(st.session_state.processed_img)
           iv.  # crop to min height/width
            v.  h = min(a.shape[0], b.shape[0])
           vi.  w = min(a.shape[1], b.shape[1])
          vii.  left_half = a[:h, :w].copy()
         viii.  right_half = b[:h, :w].copy()
           ix.  # if either is still single-channel (shouldn't), convert
            x.  if left_half.ndim == 2:
           xi.  left_half = cv2.cvtColor(left_half, cv2.COLOR_GRAY2BGR)
          xii.  if right_half.ndim == 2:
         xiii.  right_half = cv2.cvtColor(right_half, cv2.COLOR_GRAY2BGR)
          xiv.  # ensure both have same number of channels (3)
           xv.  if left_half.shape[2] != right_half.shape[2]:
          xvi.  left_half = ensure_3ch_bgr(left_half)
         xvii.  right_half = ensure_3ch_bgr(right_half)
        xviii.  # build split: left image left half + right image right half
          xix.  mid = w // 2
           xx.  left_piece = left_half[:, :mid]
          xxi.  right_piece = right_half[:, mid:]
         xxii.  # If widths are mismatched due to odd widths, adjust
        xxiii.  if left_piece.shape[1] != right_piece.shape[1]:
         xxiv.  # pad the smaller to match width
          xxv.  target_w = max(left_piece.shape[1], right_piece.shape[1])
         xxvi.  def pad_to_width(img_arr, target_width):
                   1. h, w0 = img_arr.shape[:2]
                   2. if w0 >= target_width:
                         a. return img_arr
                   3. pad_w = target_width - w0
                   4. pad = np.zeros((h, pad_w, img_arr.shape[2]),
                      dtype=img_arr.dtype)
                   5. return np.concatenate([img_arr, pad], axis=1)
        xxvii.  left_piece = pad_to_width(left_piece, target_w)
       xxviii.  right_piece = pad_to_width(right_piece, target_w)
         xxix.  split = np.concatenate([left_piece, right_piece], axis=1)
          xxx.  st.image(cv2_to_pil(split), use_container_width=True)
      -  else:
            i.  st.image(cv2_to_pil(st.session_state.processed_img),
                use_container_width=True)
279:     else:
      -  st.info('No processed image available.')
```

```
280:     # Footer / download / reset
281:     st.markdown('---')
282:     if st.session_state.processed_img is not None:
283:     info = get_image_info(st.session_state.processed_img,
   st.session_state.uploaded_name)
284:     size_bytes = len(to_bytes(st.session_state.processed_img,
   fmt='PNG'))
285:     st.caption(f"Dimensions: {info.get('Height')} x {info.get('Width')}
   | Channels: {info.get('Channels')} | File: {info.get('Filename')} |
   Estimated size (PNG): {size_bytes} bytes")
286:     else:
287:     st.caption('No image loaded')



288:     col_a, col_b, col_c = st.columns([1,1,1])
289:     with col_a:
290:     if st.button('Reset to Original') and st.session_state.original_img
   is not None:
      -  st.session_state.processed_img =
         safe_copy(st.session_state.original_img)
      -  st.session_state.history =
         [safe_copy(st.session_state.original_img)]
      -  st.session_state.redo_stack = []
291:     with col_b:
292:     if st.button('Save as PNG') and st.session_state.processed_img is
   not None:
      -  b = to_bytes(st.session_state.processed_img, fmt='PNG')
      -  st.download_button('Download PNG', data=b,
         file_name='processed.png', mime='image/png')
293:     with col_c:
294:     if st.button('Save as JPG') and st.session_state.processed_img is
   not None:
      -  b = to_bytes(st.session_state.processed_img, fmt='JPEG')
      -  st.download_button('Download JPG', data=b,
         file_name='processed.jpg', mime='image/jpeg')
```