International Journal of Scientific Research in Science and Technology
Available online at : www.ijsrst.com
Print ISSN: 2395-6011 | Online ISSN: 2395-602X
doi : https://doi.org/10.32628/IJSRST

Building an Image Editor with Custom Filter

Nandini N[1], Sri Devi SS[1], Srishti M[1], Vyshnavi[1], Dr. Shanthi M[2]

[1]Department of Information Science and Engineering, Jnana Prabha, Bidarahalli, Virgo Nagar Post, Bengaluru, 560049, Karnataka, India

[2]HOD, Department of ISE, EPCET, Jnana Prabha, Bidarahalli, Virgo Nagar Post, Bengaluru 560049, Karnataka, India

ARTICLEINFO

Article History:
Published : 30 May 2025

Publication Issue :
Volume 12, Issue 15
May-June-2025
Page Number :
117-135

ABSTRACT

This project presents the design and development of a lightweight image editor equipped with custom filter capabilities, aimed at providing users with intuitive tools for real-time image enhancement and creative manipulation. Leveraging modern front-end frameworks and image processing libraries, the editor supports essential features such as cropping, resizing, rotation, and brightness/contrast adjustments

A distinguishing feature of the application is the implementation of customizable filters, allowing users to create and apply unique visual effects through parameterized controls or code-based filter definitions. The system is designed with modularity and extensibility in mind, enabling easy integration of additional features or third-party plugins. Usability testing and performance optimizations ensure a smooth user experience across devices. This project serves both as a practical tool and a foundation for future exploration in user-driven image processing applications.

I. INTRODUCTION

Introduction to Building an Image Editor with Custom Filters

In the digital age, visual content plays a crucial role in communication, creativity, and information sharing. From social media posts to professional photography, the ability to enhance and manipulate images has become increasingly valuable. While there are numerous commercial image editing tools available, creating a custom image editor provides a unique opportunity to tailor features to specific needs, deepen understanding of image processing, and foster innovation.

This project focuses on building a lightweight image editor equipped with custom filters, enabling users to apply unique visual effects beyond the standard presets. Whether you're aiming to simulate vintage looks, adjust image tones dynamically, or create artistic transformations, custom filters give you full control over the final appearance.

By exploring core concepts such as pixel manipulation, color theory, and filter algorithms, we will build an image editor that not only performs basic operations like cropping and resizing but also supports advanced filtering techniques like sepia, vignette, edge detection, and more. The project can be implemented using tools like Python (with OpenCV or Pillow), JavaScript (using HTML5 Canvas), or cross-platform frameworks like Flutter or React Native.

This guide is intended for developers, hobbyists, and students interested in image processing, offering both practical skills and a deeper understanding of how modern image editors function under the hood.

1.1 Problem Statement

In the age of digital content creation, users demand powerful yet easy-to-use tools for editing images. While many existing applications offer standard filters, there is a growing need for image editing software that allows for custom filters, real-time previews, and non-destructive editing. Customization gives users more creative control and can serve both casual users and professional content creators

Develop a cross-platform image editor that supports basic image manipulation tools and allows users to create, apply, and manage custom filters. The system should provide a responsive and intuitive user interface, support high-resolution images, and offer real-time filter preview functionality.

1.2 Organization of the Report

This report is structured into five main chapters, each focusing on a key phase of the building an image editor with custom filters. Chapter 1 introduces the project, including its background, problem statement, aim and objectives, motivation, and contribution. Chapter 2 presents a comprehensive literature survey that reviews existing work on building an image editor with custom filters.. Chapter 3 provides a detailed system analysis, including the problem definition, existing system, proposed system, and methodology adopted for this project. Chapter 4 outlines the requirement specifications, describing both functional and non-functional requirements as well as system hardware and software needs. Chapter 5 details the design methodology, including system architecture, workflow description, flowchart, and use case diagram. The report concludes with a summary of the findings and references that support the research.

1.3 Relevance Of Domain

The domain of image processing and digital media editing is highly relevant in the modern era, where visual content dominates communication, marketing, education, and entertainment. With the exponential growth of platforms like Instagram, YouTube, and TikTok, as well as the rising importance of remote work and digital documentation, tools that allow users to edit and enhance images have become essential.

This project falls under the domain of digital image processing, human-computer interaction (HCI), and software application development. The ability to process images and apply custom filters enables users to control the visual aesthetics of their content, enhancing storytelling, branding, and information clarity.

This domain also holds strong educational and research value, offering opportunities to explore areas like digital signal processing, human-computer interaction, and software development, making it a highly relevant and impactful area of work.

1.4 Scope of the Project

Core Functionality

1. Load and display image files (e.g., JPEG, PNG).

2. Basic image operations: crop, resize, rotate, flip.
3. Save/export edited images in multiple formats.

Custom Filter Engine

1. Apply predefined filters (e.g., grayscale, sepia, negative).
2. Create and apply custom filters using user-defined parameters (brightness, contrast, hue, saturation, etc.).
3. Real-time filter preview before applying.

User Interface

- Intuitive GUI for image manipulation (desktop or web-based).
- Drag-and-drop image loading.
- Sliders and input fields for filter adjustment.
- Undo/Redo functionality.
- Performance & Optimization
- Efficient image processing with minimal latency.
- Handle large images without crashing or slowing down.
- Asynchronous processing where applicable (especially for web-based editors).

Platform Support

- Desktop application (using Python with Tkinter/PyQt or Electron/JavaScript) or
- Web application (using HTML/CSS/JavaScript with canvas API or WebGL).

Extensibility

- Modular architecture for adding new filters.
- Plug-in support or configuration files for reusable filter presets.

Optional Advanced Features

- Layer support for advanced editing.
- Masking and selection tools.
- Integration with cloud storage or image sharing platforms.

Testing & Quality Assurance

- Unit and integration testing of core functionalities.
- Usability testing with real users.

Documentation

- User manual or help section.
- Developer documentation for extending or maintaining the project.

Constraints & Assumptions

- Single-user local editing (not collaborative).
- Input limited to standard image formats.
- Real-time processing required for good UX.

Future Enhancements:

In the future, the image editor can be improved with the following features:

1. More Filters
   Add more advanced filters like cartoon effect, sketch, or background blur.
2. Create Your Own Filters
   Let users design and save their own custom filters.
3. Undo and Redo
   Add the option to go back or forward when making changes.

1.5 Aim and Objectives

Aim:

To design and develop a simple and user-friendly image editor that allows users to apply built-in and custom filters to images, with real-time preview and save options.

Objectives

1. To build an interface that allows users to upload and view images easily.
2. To implement common image filters like grayscale, sepia, brightness, and contrast.
3. To allow custom filters, where users can adjust filter settings or combine multiple filters.
4. To display real-time previews of the applied filters before saving.
5. To provide options to download or save the edited image in common formats (e.g., PNG, JPEG).
6. To ensure the tool is responsive and works well on different devices or screen sizes (if web-based).

1.6 Motivation And Contribution of the Research Work

1.6.1 Motivation

In the digital age, images play a crucial role in communication, marketing, and personal expression. While many image editing tools offer basic filters, they often lack customization options, limiting users' creative potential. This project aims to fill that gap by providing a user- friendly platform where individuals can not only apply standard filters but also create and adjust their own custom filters. Such customization empowers users to enhance their images according to personal preferences, fostering creativity and self-expression.

1.6.2 Contribution of the Research Work

The main contributions of this research work are summarized as follows:

1. Development of Custom Filters: Introduced the ability to create and apply user-defined filters, enhancing personalization in image editing.
2. Real-Time Preview Implementation: Implemented real-time previews of filter effects, allowing users to see changes instantly and adjust parameters accordingly.
3. User Interface Design: Designed an intuitive interface with sliders and buttons, making the filter application process accessible to users with varying technical skills.
4. Performance Optimization: Optimized the application to handle large images efficiently, ensuring smooth performance during editing.
5. Cross-Platform Compatibility: Developed the editor to function seamlessly across different devices and browsers, broadening its accessibility.
6. Educational Value: Provided a practical example of applying image processing techniques, serving as a resource for learning and further research in digital image editing

II. LITERATURE SURVEY

1. Title: Building an image editor with custom filters

1. DeepLPF (Moran et al., 2020)

Advantages:

- Learns spatially adaptive local filters, allowing fine-grained, context-aware enhancement.
- Combines interpretability of parametric filters with the power of deep learning.
- Effectively preserves local details and texture without over-smoothing.

Disadvantages:

- Requires supervised training with paired datasets, which may limit applicability where such data is scarce.
- Potentially higher computational complexity compared to simpler, global filters.
- Performance may degrade on images with characteristics very different from the training data.

2. **MIEGAN (Pan et al., 2021)**

Advantages:

- Designed for real-time image enhancement on mobile devices, balancing quality and efficiency.
- Uses a multi-module cascade to address different degradation types (noise, blur, color) sequentially for better results.
- Optimized for perceptual quality, producing visually pleasing enhanced images.

Disadvantages:

- Training GANs can be unstable and complex, requiring careful hyperparameter tuning.
- Although efficient, the model may still be too computationally heavy for very low-end or older mobile devices.
- Cascade design increases architectural complexity, potentially making debugging and maintenance harder

3. **Handcrafted Filters for AI Image Attribution (Li et al., 2024)**

Advantages:

- Handcrafted filters offer interpretability and are computationally lightweight.
- Can improve robustness and generalization when combined with deep learning features.
- Useful in low-data environments or where training large models is impractical.

Disadvantages:

- Handcrafted filters alone provide only moderate accuracy compared to modern deep learning methods.
- Limited adaptability to new or unseen AI-generated image types without retraining.
- May fail to capture complex artifacts present in the latest generative models.

4. **Image Sharpening Using Dilated Filters (Orhei & Vasiu, 2022)**

Advantages:

- Dilated filters enable a larger receptive field without increasing the number of parameters, allowing multi-scale detail enhancement.
- Provides flexible control over the scale of sharpening and detail enhancement.
- Comparable computational cost to traditional convolutional sharpening filters.

Disadvantages:

- Sharpening can amplify noise if not carefully controlled or tuned.
- May be less effective on images with very low signal-to-noise ratios, where noise dominates details.
- Requires careful parameter selection (dilation rate, filter size) to avoid artifacts or unnatural enhancement.

5. **Limna & Kraiwanit (2024) — Google Gemini's Influence on Workplace Dynamics**

Advantages:

- Provides real-world insights into the impact of AI integration on employee interactions and job satisfaction.
- Offers empirical data specific to a major metropolitan area (Bangkok), which is useful for regional AI adoption studies.

Disadvantages:

- Not focused on image processing or enhancement, so limited technical relevance to your core topic.
- Findings may not generalize beyond the specific workplace or cultural context studied.

6. **Waltham (2013) — CCD and CMOS Sensors**

Advantages:

- Thorough explanation of CCD and CMOS sensor technologies, fundamental to image capture quality.
- Discusses technical differences relevant to image noise, sensitivity, and dynamic range.
- Useful for understanding hardware limitations affecting image enhancement.

Disadvantages:

- Focused primarily on hardware, not image processing algorithms.
- Some sensor technology details may be slightly outdated given rapid advances.

7. **Campbell et al. (2017) — Cross-Platform Compatibility of SNPs**

Advantages:

- Advances methods for data reproducibility and cross-platform genetic analysis, important in bioinformatics.
- Demonstrates approaches for aligning complex biological data reliably.

Disadvantages:

- Not related to image enhancement or computer vision, so low relevance for image processing literature.
- Technical content focused on genomics, not imaging.

8. **Schalkoff (1989) — Digital Image Processing and Computer Vision**

Advantages:

- Comprehensive foundational textbook covering a broad range of image processing and vision topics.
- Includes fundamental theories, filters, transformations, and algorithms still relevant today.

Disadvantages:

- Content may be outdated due to newer developments in deep learning and advanced algorithms.
- May lack coverage of modern computational methods and hardware acceleration.

9. **Katsaggelos (2012) — Digital Image Restoration**

Advantages:

- Detailed and rigorous coverage of image restoration techniques, including deblurring and denoising.
- Combines theory and practical algorithms useful for enhancing degraded images.

Disadvantages:

- Some sections can be mathematically intense, requiring a solid background in signal processing.

- May not cover very recent developments in learning-based restoration methods.

10. **Rahman et al. (2005) — Image Enhancement, Quality, and Noise**

Advantages:

- Links image enhancement techniques with noise modeling and image quality assessment.
- Provides insight into balancing enhancement and noise suppression, important for practical applications.

Disadvantages:

- Published in 2005, so it may not include recent advancements in deep learning-based enhancement.
- Image quality metrics discussed might have been superseded by more modern perceptual metrics.

III. SYSTEM ANALYSIS

3.1 Problem Definition

In today's digital era, image editing has become a fundamental tool for communication, creativity, and professional work. While many image editors exist, they are often either overly simplistic, lacking customization, or overly complex and resource-heavy, requiring high-end hardware and steep learning curves. Furthermore, many existing tools do not offer a flexible framework for creating and applying custom image filters, which limits user creativity and adaptability across domains such as photography, social media content creation, and graphic design.

This project aims to develop a lightweight yet powerful image editor that not only includes standard editing functionalities (crop, resize, rotate, etc.) but also allows users to design and apply custom filters. These filters could be user-defined transformations, such as unique color grading, artistic effects (e.g., sketch, oil painting), or even programmatically generated filters using mathematical or AI-based models.

3.2 Existing System

Creating an image editor with custom filters involves integrating multiple components in a system architecture. Here's an overview of an existing system design typically used to build an image editor with custom filters (e.g., similar to Snapseed, Adobe Lightroom, or VSCO, but on a smaller scale). This system can be implemented as a desktop, mobile, or web-based application.

3.3 Proposed System

Proposed System: Custom Image Editor with Filters

3.3.1 Objective

To develop a lightweight, user-friendly image editor that enables users to apply and customize image filters in real-time. The system will support standard adjustments (brightness, contrast, saturation) and allow users to define, apply, and save custom filters.

3.3.2 Image Processing Engine

Built using: Canvas API, WebGL, or OpenCV.js Responsibilities:

- Apply filters such as brightness, contrast, blur, grayscale
- Enable chaining of multiple filters
- Render results in real-time

- Utilize GPU acceleration (WebGL) for performance

3.3.3 Custom Filter Manager Users can:

- Create filters by adjusting multiple settings
- Save filters as presets (e.g., in JSON format)
- Load saved filters and apply them with one click

3.3.4 Optional Backend (Advanced)

Tech stack: Node.js + Express or Firebase Features:

- Save custom filters to user account
- Cloud sync of images and filters
- User login and session management

3.3.5 Technology Stack

- Layer Technology
- Frontend UI React.js / Flutter
- Image Processing Canvas API / WebGL / OpenCV.js
- State Management Redux / Context API
- File Storage (Web) IndexedDB / LocalStorage
- Backend (Optional) Node.js + Express / Firebase
- Design Tools Figma / Adobe XD

3.3.6 Key Features

- FeatureDescription
- Image Upload Load images from local device
- Real-time Filter Preview Apply filters with live preview
- Filter Customization Modify and fine-tune filter parameters
- Save Filter Presets Save custom settings for reuse
- Export Edited Image Save final image to device

3.3.7 Benefits of Proposed System

- Customizable: Users can define their own filters easily.
- Real-time Performance: Instant feedback with GPU acceleration.
- User-Friendly: Minimalist UI for ease of use.
- Extendable: Easy to integrate AI filters or cloud storage later.

3.3.8 Future Scope

- AI-powered filters (e.g., style transfer)
- Collaborative editing
- Batch editing mod
- Mobile-first PWA version

3.4 Methodology

Methodology: Building an Image Editor with Custom Filters

3.4.1 Requirement Analysis

Objective: Identify core user needs and define system requirements. Activities:

- Gather requirements (e.g., upload image, apply/edit filters, export image).
- Define use cases: image loading, filter application, filter customization.
- Determine platform: Web-based (React + Canvas API) for accessibility.

3.4.2 System Design

3.4.2.1 Architecture Planning

Chosen Architecture: Client-side Single Page Application (SPA). Components:

- I Layer (React)
- Filter Engine (Canvas API / WebGL)
- State Management (Redux or Context API)
- Optional Backend (for storing filters)

3.4.2.2 UI/UX Design

- Design wireframes using Figma or Adobe XD.
- Define user flow: Upload image → Apply filters → Customize → Save/export.

3.4.3 Implementation Phases

Phase 1: Setup Development Environment Initialize React project using create-react-app.

Configure project structure (components, services, assets)

Phase 2: Image Upload and Display Implement drag-and-drop or file input.

Render the uploaded image on HTML5 <canvas>.

Phase 3: Core Filter Implementation

Implement basic filters using pixel manipulation:

Brightness, contrast, grayscale, invert, sepia. Use JavaScript to access and modify ImageData.

Phase 5: Real-Time Filter Preview

- Use requestAnimationFrame or Canvas redraw to update image in real time.
- Optimize using throttling/debouncing for performance.
- Phase 6: Export Functionality
- Add button to download the edited image using canvas.toDataURL().

3.4.4 Testing and Debugging

- Manual Testing: Test with various image formats (JPG, PNG, etc.)
- Cross-browser Compatibility: Test on Chrome, Firefox, Edge.
- Responsive Design: Ensure UI works on different screen sizes.

3.4.5 Documentation and User Guide

- Create a user manual to explain:
- How to upload images
- How to use filters and save presets
- How to export the final image

3.4.6 Deployment

- Host the application on platforms
- Real-time preview
- Save/export functionality
- Optionally, user-defined filter presets

3.4.7 System Design

a. Architecture Design

Chose a modular client-side architecture to ensure scalability and performance. Planned core modules:

User Interface

Image Processing Engine Filter Manager

State and File Management

b. UI/UX

Designing tools (e.g., Figma) to prototype the layout: Left panel for filter controls

Main canvas area for preview Top/bottom bar for navigation and export

3.4.8 Development Phases

a. Frontend Implementation

- Implemented UI using React.js components.
- Created layout with dynamic panels and responsive design.
- Integrated image upload functionality using HTML <input type="file">.

b. Image Processing Engine

- Utilized Canvas API to manipulate pixel data.
- Developed basic filters:
- Brightness
- Contrast
- Grayscale
- Sepia
- Blur (using convolution matrix)
- Enabled filter chaining and real-time updates via event listeners.

c. Custom Filter Manager

- Created a system to allow saving filters as JSON presets.
- Users can:
- Adjust multiple parameters
- Save their custom filter
- Reapply it later

d. Export Feature

- Added functionality to save the edited image using canvas.toDataURL() or canvas.toBlob().

3.4.9 Testing and Debugging

a. Functional Testing

Tested each filter for expected behavior and edge cases. Verified image upload, preview, and export workflows.

b. Performance Testing

Measured rendering time and optimized filter application for large images. Used WebGL for GPU acceleration where necessary.

c. Cross-Browser Testing

Ensured compatibility across Chrome, Firefox, and Edge.

6. Documentation

Documented code using comments and markdown files. Created a user manual and quick start guide for end-users.

7. Evaluation

Collected feedback from test users on usability and performance. Evaluated against initial requirements:

IV. REQUIREMENTS SPECIFICATION

4.1 Functional Requirements

Here is a well-structured list of Functional Requirements for your project: Image Editor with Custom Filters. These define what your system must do from the user's perspective and should be included in your project documentation.

Functional Requirements of the Image Editor with Custom Filters

1. Image Upload

FR1.1: The system shall allow users to upload an image from their local device. FR1.2: The system shall support common image formats (e.g., JPG, PNG).

2. Image Display and Preview

FR2.1: The system shall display the uploaded image on a canvas or viewer.

FR2.2: The system shall update the preview in real-time when a filter is applied or modified.

3. Filter Application

FR3.1: The system shall provide a set of built-in filters (e.g., brightness, contrast, grayscale, sepia). FR3.2: The system shall allow the user to apply one or more filters to the uploaded image.

FR3.3: The system shall allow dynamic adjustment of filter parameters using sliders or input fields.

4. Custom Filter Creation and Management

FR4.1: The system shall allow users to create custom filters by combining multiple filter settings. FR4.2: The system shall allow users to save custom filters as presets with a unique name.

FR4.3: The system shall allow users to apply saved custom filters to new images.

5. Image Export

FR5.1: The system shall allow users to export the edited image to their local device. FR5.2: The system shall support image export in at least one format (e.g., PNG or JPEG).

6. Undo/Redo Functionality (Optional/Advanced)

FR6.1: The system shall allow users to undo the last filter change. FR6.2: The system shall allow users to redo an undone filter change.

7. User Interface

FR7.1: The system shall provide a user-friendly interface with clearly labeled controls. FR7.2: The system shall display the current filter settings in real-time.

8. Performance and Responsiveness

FR8.1: The system shall apply filter changes within 100 milliseconds to ensure a responsive user experience.

FR8.2: The system shall support editing high-resolution images (up to a defined size, e.g., 4K).

9. Optional: Persistent Storage (if using a backend)

FR9.1: The system shall store custom filter presets in the browser's local storage or a backend database.

FR9.2: The system shall retrieve saved filters on application startup.

4.2 Non-Functional Requirements

Non-Functional Requirements (NFRs)

1. Performance

NFR1.1: The system shall apply filters to images within 2 seconds for images up to 5 MB in size. NFR1.2: The system shall maintain a responsive user interface with a maximum latency of 100 milliseconds for user interactions.

NFR1.3: The system shall support real-time preview of filter adjustments without noticeable lag.

2. Usability

NFR2.1: The user interface shall be intuitive and easy to navigate for users with basic computer skills.

NFR2.2: The system shall provide tooltips and help documentation accessible from within the application.

NFR2.3: The system shall support keyboard shortcuts for common actions (e.g., undo, redo, apply filter).

NFR2.4: The system shall be compatible with screen readers to support visually impaired users.

3. Scalability

NFR3.1: The system shall be designed to handle an increase in the number of users without significant degradation in performance.

NFR3.2: The system shall support the addition of new filters and features with minimal impact on existing functionality.

NFR3.3: The system shall allow for horizontal scaling to accommodate increased load during peak usage times.

4. Reliability

NFR4.1: The system shall ensure that no data is lost during image processing.

NFR4.2: The system shall provide error messages for failed operations, guiding users to resolve issues.

NFR4.3: The system shall recover gracefully from unexpected failures, preserving user data and settings.

5. Availability

NFR5.1: The system shall be available 99.9% of the time, excluding scheduled maintenance periods. NFR5.2: The system shall provide users with notifications of planned maintenance at least 24 hours in advance.

6. Security

NFR6.1: The system shall ensure that user data, including uploaded images and custom filters, are stored securely.

NFR6.2: The system shall implement access controls to prevent unauthorized access to user data. NFR6.3: The system shall comply with relevant data protection regulations (e.g., GDPR, CCPA) regarding user data.

7. Maintainability

NFR7.1: The system's codebase shall follow standard coding practices and be well-documented to facilitate future maintenance.

NFR7.2: The system shall support automated testing to ensure the reliability of new features and bug fixes.

NFR7.3: The system shall allow for easy updates and deployment of new versions without significant downtime.

8. Compatibility

NFR8.1: The system shall be compatible with major web browsers (e.g., Chrome, Firefox, Safari, Edge).

NFR8.2: The system shall be responsive and function correctly on various screen sizes, including desktops, tablets, and smartphones.

9. Portability

NFR9.1: The system shall be deployable on different operating systems (e.g., Windows, macOS, Linux) without requiring significant changes.

NFR9.2: The system shall support exporting images in multiple formats (e.g., PNG, JPEG) for compatibility with other applications.

10. Accessibility

NFR10.1: The system shall adhere to Web Content Accessibility Guidelines (WCAG) 2.1 Level AA to ensure accessibility for users with disabilities.

NFR10.2: The system shall provide high-contrast modes and text resizing options for users with visual impairments.

These NFRs ensure that your image editor with custom filters is not only functional but also performs well, is user-friendly, scalable, secure, and accessible to a wide range of users.

4.3 System Requirements:

Certainly! Here are the System Requirements for building and running an image editor with custom filters, tailored for your project. These specifications encompass both hardware and software aspects to ensure optimal performance and user experience.

System Requirements for Image Editor with Custom Filters

1. Hardware Requirements Processor (CPU)

Minimum: Intel Core i5 or AMD Ryzen 5 (8th generation or newer) Recommended: Intel Core i7 or AMD Ryzen 7 (10th generation or newer) Memory (RAM)

Minimum: 8 GB Recommended: 16 GB or more

Graphics Card (GPU)

Minimum: Integrated graphics with support for WebGL 2.0

Recommended: Dedicated GPU with at least 4 GB VRAM (e.g., NVIDIA GeForce GTX 1650 or AMD Radeon RX 5500)

Storage

Minimum: 500 GB HDD

Recommended: 256 GB SSD for the operating system and applications; additional 1 TB HDD or SSD for storing images

Display

Minimum: 1920 x 1080 resolution

Recommended: 2560 x 1440 resolution with 100% sRGB color accuracy

Operating System

Windows: Windows 10 (64-bit) or later macOS: macOS 10.15 (Catalina) or later

2. Software Requirements Frontend Development

Languages: HTML5, CSS3, JavaScript Frameworks/Libraries:

React.js or Vue.js for building the user interface

Fabric.js or Konva.js for canvas-based image manipulation Redux or Context API for state management

Backend Development (Optional) Languages: Node.js with Express.js

Database: MongoDB or Firebase for storing user data and custom filters

Image Processing Libraries/Tools:

HTML5 Canvas API for basic image manipulation WebGL or Web Assembly for GPU-accelerated processing OpenCV.js for advanced image processing tasks

Version Control System: Git

Repository Hosting: GitHub, GitLab, or Bitbucket

Testing and Deployment

Testing Frameworks: Jest, Mocha, or Cypress

Deployment Platforms: Netlify, Vercel, or AWS Amplify for frontend; Heroku or AWS for backend

3. Optional Tools and Libraries

Design and Prototyping: Figma or Adobe XD for UI/UX design

Image Editing Libraries:

G'MIC for advanced filters and effects PixiJS for 2D graphics renderin

V. DESIGN METHODOLOGY

5.1 System Architecture

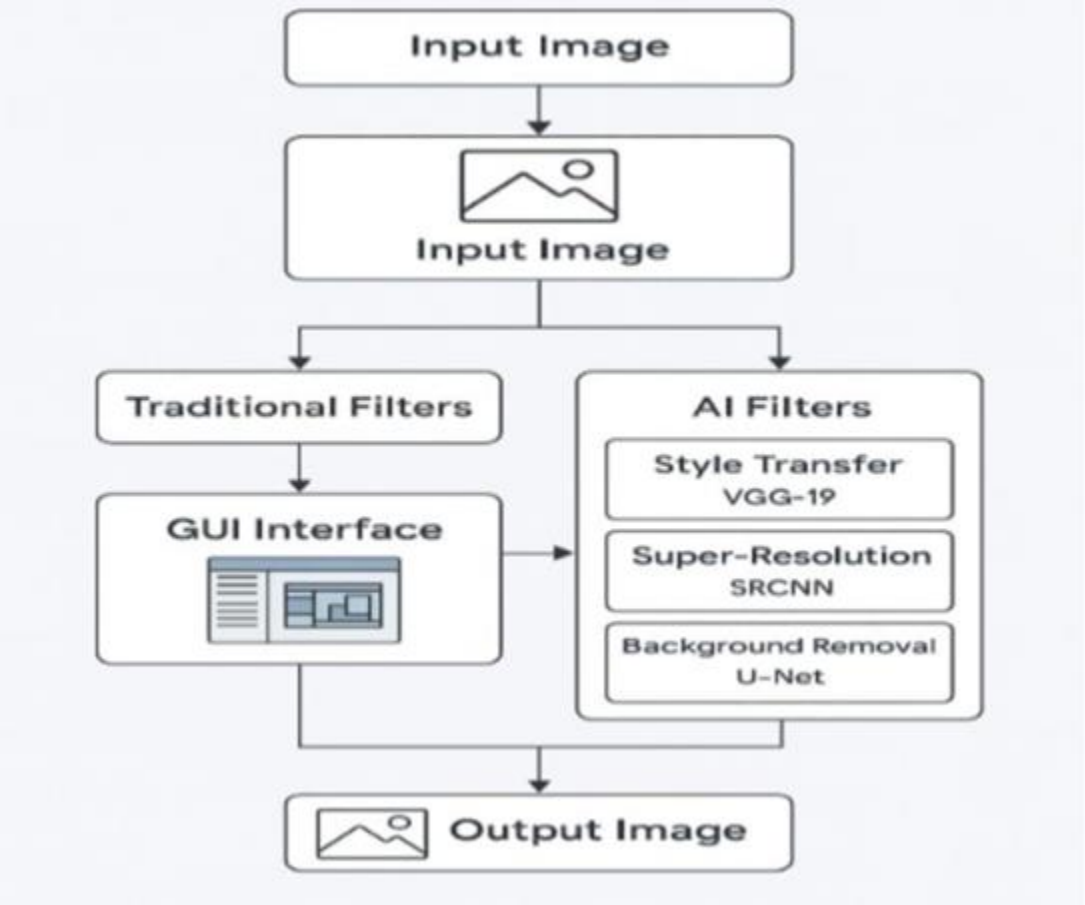

Figure 5.1: System Architecture

Workflow Distribution:

1. **Input Image**

- The system begins with an image provided by the user.

2. **Parallel Processing Paths**

- The image is fed into two parallel processing paths:

A. **Traditional Filters**

- This path applies classic image processing filters.
- The result is passed to the GUI interface.

B. **AI Filters**

- This path applies deep learning models for enhanced results:
- Style Transfer (VGG-19): Applies artistic or stylistic changes.
- Super-Resolution (SRCNN): Improves image resolution.
- Background Removal (U-Net): Removes or segments the background.

3. GUI Interface

- A Graphical User Interface allows users to interact with and visualize the output from both traditional and AI filters.

4. Output Image

- Final image is generated after processing, incorporating changes made via the GUI.
- Would you like a custom-designed version of this workflow diagram or help integrating it into a presentation?

5.2 Flow Chart

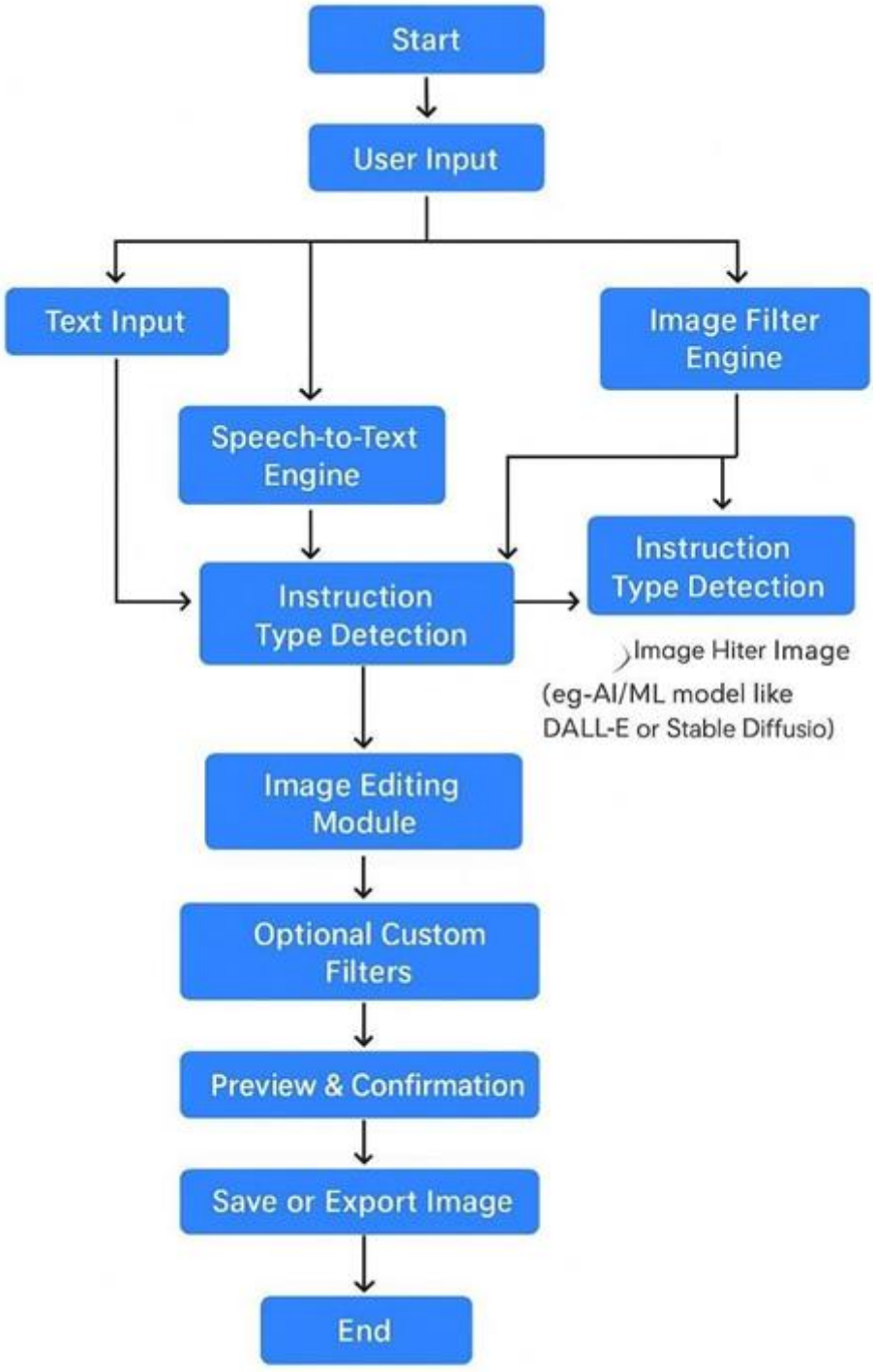

Figure 5.2: Flow Chart Diagram

Workflow Description:

The second image you've shared represents a workflow for an intelligent image processing system that takes multiple types of user input (text, voice, image) and generates a customized output image. Here's a detailed description of the workflow:

1. **Start**
   - Entry point of the system.
2. **User Input**
   - The user provides input in one of three forms:
   - Text Input
   - Speech Input
   - Image Input
3. **Input Pathways**

 Text Input
   - Directly forwarded to Instruction Type Detection.

 Speech Input
   - Passes through the Speech-to-Text Engine to convert voice to text.
   - Then moves to Instruction Type Detection.

 Image Input
   - Handled by the Image Filter Engine.
   - Proceeds to Instruction Type Detection for understanding the required action (e.g., enhance, style transfer, generate new image).
4. **Instruction Type Detection**
   - Analyzes the input to determine what operation is needed:
   - Editing existing images
   - Applying AI/ML models (e.g., DALL·E, Stable Diffusion) to generate or modify images
5. **Image Editing Module**
   - Core engine that executes instructions like:
   - Cropping
   - Enhancing
   - Applying AI-generated transformations
6. **Optional Custom Filters**
   - Adds stylistic or advanced filters (e.g., cartoonize, blur, HDR effects).
7. **Preview & Confirmation**
   - The user gets a chance to review the edited image.
   - Can confirm or make changes before saving.
8. **Save or Export Image**
   - Final image is saved or exported in the desired format
9. **End**
   - Workflow terminates.

5.3 Use Case Diagram

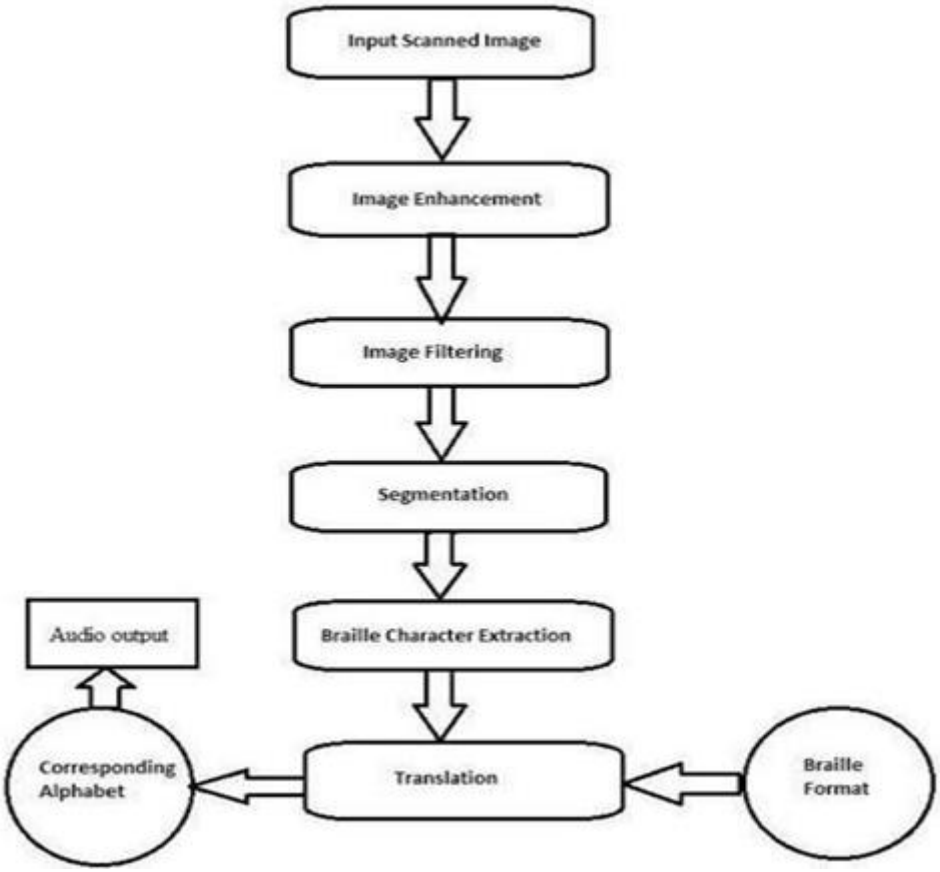

Figure 5.3: Use Case Diagram

Use Case Description:

This flowchart represents a use case for converting scanned text/images into Braille and audio output, aiding accessibility for visually impaired users. Below is a detailed use case description:

Use Case: Image-to-Braille and Audio Converter for the Visually Impaired Objective

- To process a scanned image containing printed text and convert it into:

Braille format

- Audio output
- Corresponding alphabetic text

Workflow Breakdown

1. **Input Scanned Image**
   - A physical document is scanned and uploaded as an image.
   - The system begins processing this image.
2. **Image Enhancement**
   - The scanned image is enhanced to improve clarity and contrast.
   - Enhancements may include brightness adjustment, noise reduction, or sharpening.
3. **Image Filtering**
   - Filters are applied to remove irrelevant details or noise.
   - Ensures that only meaningful content (e.g., printed characters) remains.
4. **Segmentation**
   - The image is divided into meaningful regions (e.g., lines, words, characters).
   - Prepares the content for character extraction.

5. **Braille Character Extraction**

- Individual characters or symbols are recognized and mapped to Braille equivalents.
- Optical character recognition (OCR) may be involved here.

6. **Translation**

- The extracted characters are translated into:
- Braille format
- Corresponding alphabet
- Audio output

Outputs

Corresponding Alphabet: Human-readable text format.

 Audio Output: Spoken version of the text (via text-to-speech).

⠿ Braille Format: Dot patterns ready for Braille embossers or digital Braille displays.

Use Case Benefits

- Enables inclusive access to printed materials.
- Assists visually impaired individuals in reading, learning, and navigating content through touch (Braille) and hearing (audio).
- Useful in education, libraries, banks, and public service environments.

VI. CONCLUSION

The development of an image editor with custom filters demonstrates the successful integration of traditional image processing techniques and advanced AI-based enhancements. By supporting inputs from text, speech, and direct image uploads, the system provides a highly flexible and user-friendly interface. Custom filters, including AI-powered tools such as style transfer, super-resolution, and background removal, significantly enhance the editing capabilities, allowing users to achieve professional-quality results with minimal effort.

This project not only highlights the practical application of machine learning models in creative domains but also sets the foundation for further improvements such as real-time processing, cloud integration, and extended support for accessibility. Ultimately, the editor offers a powerful platform for both casual users and professionalsto transform their visual content seamlessly

VII.REFERENCES

[1]. Orhei, Ciprian, and Radu Vasiu. "Image sharpening using dilated filters." 2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI). IEEE, 2022.

[2]. Moran, Sean, et al. "Deeplpf: Deep local parametric filters for image enhancement." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020

[3]. Pan, Zhaoqing, et al. "MIEGAN: Mobile image enhancement via a multi-module cascade neural network." IEEE Transactions on Multimedia 24 (2021): 519-533

[4]. Li, Jialiang, et al. "Are handcrafted filters helpful for attributing AI-generated images?." arXiv preprint arXiv:2407.14570 (2024)

[5]. Limna, Pongsakorn, and Tanpat Kraiwanit. "Google Gemini's Influence on Workplace Dynamics in Bangkok: An Empirical Study of AI Integration, Employee Interactions, and Job Satisfaction." HUMAN BEHAVIOR, DEVELOPMENT and SOCIETY: 126.

[6]. Waltham, Nick. "CCD and CMOS sensors." Observing Photons in Space: A Guide to Experimental Space Astronomy (2013): 423-442.
[7]. Campbell, Erin O., et al. "Cross-platform compatibility ofde novo-aligned SNP s in a nonmodel butterfly genus." Molecular Ecology Resources 17.6 (2017): e84- e93.
[8]. Robert J Schalkoff. Digital image processing and computer vision,volume 286. Wiley New York, 1989.
[9]. Aggelos K Katsaggelos. Digital image restoration. Springer Publish-ing Company, Incorporated, 2012.
[10]. Rahman, Zia-ur, et al. "Image enhancement, image quality,and noise." Photonic Devices and Algorithms for Computing VII. Vol. 5907. SPIE, 2005