“SharpView – Advance Low Light Image Enhancement Tool using C++”

Project Report

Submitted in completion of the

requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Name	Roll No.
Sagar Thapliyal	R2142220936
Arshdeep Kaur	R2142220447
Aviral Khanna	R2142221156
Saksham Siwach	R2142220937

Under the guidance of

Mrs. Gaytri

School of Computer Science

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, Uttarakhand

(November 2024)

CANDIDATE'S DECLARATION

We hereby certify that the project work entitled **"SharpView: Advance Low Light Image Enhancement using C++"** in fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning and submitted to the Department of AIML, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from August 2024 to November 2024, under the supervision of **Mrs. Gaytri**, **Associate Professor, SOCS.**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other university.

Sagar Thapliyal – R2142220936

Arshdeep Kaur – R2142220447

Aviral Khanna – R2142221156

Saksham Siwach – R2142220937

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: 27.11.2024

Mrs. Gaytri

Project Guide

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide, **Mrs. Gaytri**, for all advice, encouragement and constant support she has given us throughout our project work. This work would not have been possible without her support and valuable suggestions.

We sincerely thank most respected **Dr. Anil Kumar (Head), Department of SoCS,** for his great support in doing our project in **"SharpView: Advance Low Light Image Enhancement Tool in C++"**.

We are also grateful to Dean SoCS, UPES for giving us the necessary facilities to carry out our project work successfully. We also thanks to our Course Coordinator, **Mr. Sandeep Chand Kumain** and our Activity Coordinator, **Mr. Suneet Gupta** for providing timely support and information during the completion of this project.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

NAME	**Sagar Thapliyal**	**Arshdeep Kaur**	**Aviral Khanna**	**Saksham Siwach**
ROLL	**R2142220936**	**R2142220447**	**R2142221156**	**R2142220937**

ABSTRACT

This project is based on implementing a high-performance image enhancement toolkit from scratch in the C++ programming language. The main objective is to enhance the visual quality of still images captured under adverse conditions, such as low light, blur, or noise, without dependency on external libraries or frameworks. This light tool applies a series of advanced image enhancement techniques, such as brightness control and adjustment, contrast, gamma correction, Gaussian smoothing, and gray-world color balancing. Together, these approaches remediate many deficits in degraded images, thus improving the overall resolution, sharpness, and color balance of images.

Unlike regular image processing tools that will invariably be based on resource-intensive libraries, this project focuses on developing a compact, self-contained implementation in C++. The toolkit is optimized for low-end systems and resource-constrained devices, making it suitable for environments such as embedded systems, mobile devices, and other limited-computation platforms. The toolkit converts input images to a structured data format (PPM) and processes them through a pipeline of algorithms, hence achieving efficient and real-time enhancements with compatibility in diverse use cases.

The modular design of the system provides flexibility for future expansions, such as adding new enhancement techniques or enabling real-time processing. Moreover, the program incorporates user feedback mechanisms to enable interactive adjustments of the image parameters, including brightness and contrast, making it adaptable for various user requirements. The application targets a wide range of users, from general users trying to improve their own personal images to professionals like photographers or researchers working to optimize images and law enforcement agencies that require better illumination for low-light surveillance images.

This toolkit, with its lightweight design, high efficiency, and robust image processing capabilities, is ideal for educational and professional environments. It could be used as a useful teaching tool, since the foundational concepts of enhancement in image are explained and applied in computer vision. Finally, it provides practical benefits to industries where enhanced image quality plays a critical role. This project is a testament to the power of simplicity and efficiency in software design, creating a gap between academic principles and real-world applications.

CONTENTS

Introduction

1.1 History

This project has a basis in the need to find common problems to be solved concerning low-light image enhancement without using an external library or framework. Traditional approaches demanded a high number of computational resources and dependence on third-party software, rendering them less possible for use in low-end devices or resource-constrained environments. To overcome this limitation, this project was conceptualized toward developing a lightweight and standalone tool using C++. The project initially studied datasets ExDark and enhancement techniques CLAHE and SA-DWT, but these were later improved for better performance and simplicity. In the final design, optimized algorithms in gamma correction, brightness adjustment, and Gaussian smoothing have been added to the database, balancing efficiency with output quality.

1.2 Purpose of the Project

The purpose of this project is to develop a standalone, light image enhancement tool in C++. It aims to improve the visual quality of the low-light and degraded images by techniques such as brightness adjustment, contrast enhancement, and gamma correction. Designed to run efficiently on devices with limited resources, the tool can be applied to education, security, and industries requiring high-quality image processing without any dependency on external libraries.

1.3 Requirement Analysis

The project demands a standalone light-weight C++ application to enhance the pictures taking in low-light conditions. It must support brightness, contrast, gamma correction, and Gaussian smoothing and able to deal with usual image formats in the converted PPM format. Its performance should be optimized for lower-end devices while focusing on having real-time feedback from the user on changes made in the system. Additionally, it must ensure compatibility across platforms like Windows and Linux, with minimal memory usage and efficient processing pipelines.

1.4 Main Objectives

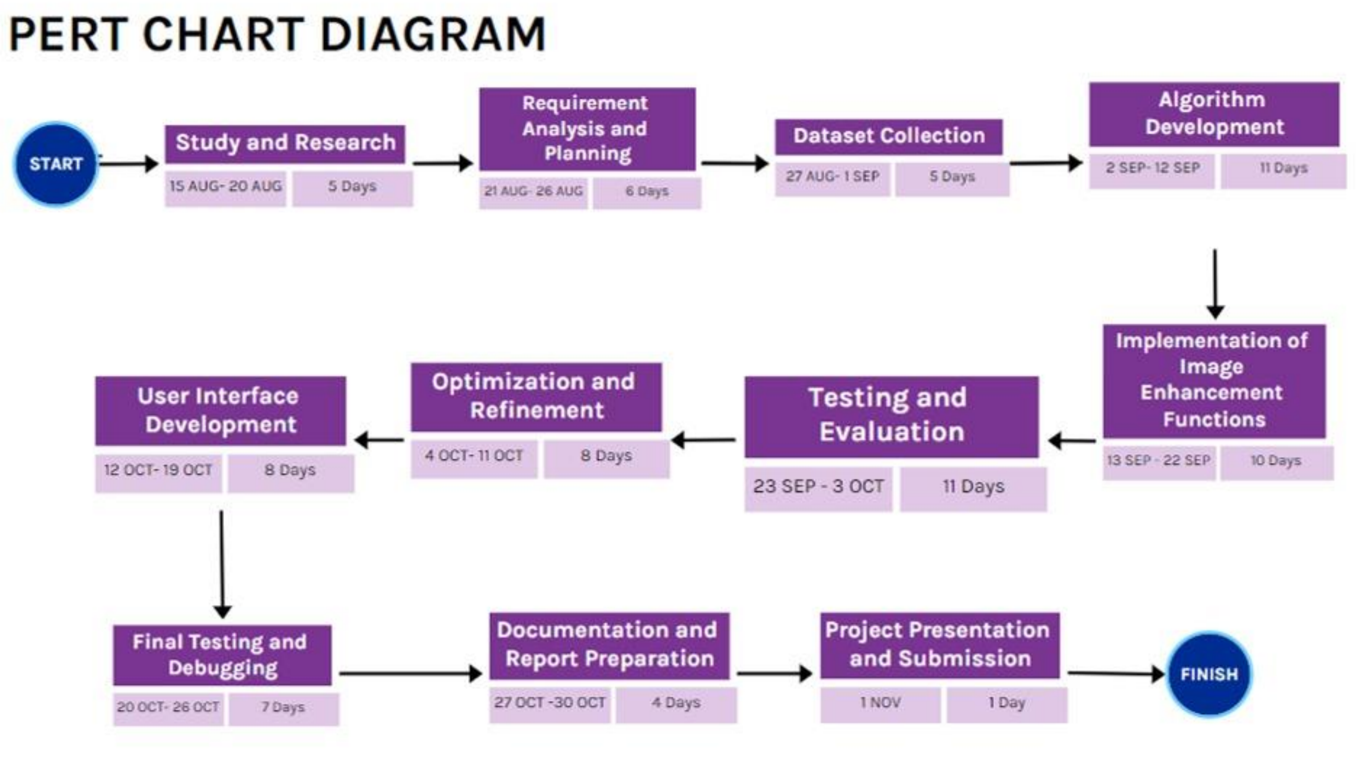
To design and implement a lightweight and robust C++ program that performs a variety of image enhancement techniques, which aims to improve the visual quality of degraded or distorted images(containing noise/ blur) captured in low-light conditions, that too without using any external libraries, and ensuring it is suitable for low-end devices (resource-constrained systems).

1.5 Sub Objectives

- Enhance the quality of low-quality or noisy security footage to identify the subjects in the image, such as a likely thief.
- Designs algorithms capable of enhancing specific regions in images which could be face or movement, for better recognition.
- Optimizing the enhancement pipeline such that it would allow for real-time processing for enabling real-time detection of suspicious activity or people.

1.6 Pert Chart Legend

A PERT (Program Evaluation Review Technique) chart is a project management tool used to visualize and schedule tasks.



System Analysis

2.1 Existing Systems

Most existing image enhancement systems rely on quite resource-intensive libraries like OpenCV or MATLAB. This makes these systems unsuitable for resource-scarce, low-end devices. They offer more advanced features but lack real-time processing efficiency for environments that are highly resource constrained. Moreover, the demands they make on computational power usually lock them out from being used in applications for embedded systems, mobile platforms, or users with limited hardware capabilities.

2.2 Motivation

- **Real-Life Use Case** – Can be used to enhance images captured at night or in low-light conditions, which can be very beneficial in terms of safety and security.
- **For Low-End Devices** – As it uses no external libraries or dependencies, it has very less overhead on the machine and eventually it has a fast-processing pipeline. As a result, can also be used easily on a low-end device.
- **Deeper Understanding of Image Enhancement Concepts** – As we are implementing each technique from scratch, we will gradually develop a greater understanding of each technique, and it would benefit us in longer run to work on other image related projects

2.3 Proposed System

The proposed system is a free-standing, lightweight C++ tool that enhances low-light and degraded images without the need for any external library dependencies. The key techniques involved include brightness adjustment, contrast enhancement, gamma correction, and Gaussian smoothing. Optimized on resource-constrained devices, it provides easy processing on any given platform with real-time feedback support for changing parameters as required by the user. Modularity makes way for future growth and adaptation.

2.4 Modules

2.4.1 Gaussian Filtering

Noise reduction and image smoothing with a Gaussian filter to provide a finer clarity without morphing important information details.

2.4.2 Gamma Correction

Images are corrected based on brightness and tone using non-linear pixel intensity transformation to achieve eye appeal.

2.4.3 Gray World Color Balance

The RGB channels are balanced to neutralize color casts resulting in image colors appearing more natural and accurate.

2.4.4 Brightness Adjustment

It increases or decreases pixel intensity depending on user-defined requirements to give enhanced clarity of images in visibility.

2.4.5 Contrast Enhancement

Better definition of light and dark areas due to more prominent details within the balanced visual background.

Design

3.1 Image Enhancement Modelling

Image enhancement modeling encompasses a structured pipeline for visually enhancing image quality. It first loads an input image and then applies a series of enhancement techniques, including brightness adjustment, contrast enhancement, gamma correction, Gaussian smoothing, and color balancing. Each of these techniques is designed to target specific aspects of the image, such as visibility in low light, noise, and distortion in color. The model ensures that these operations are done efficiently, especially on low-resource devices, without compromising the integrity and clarity of the image in the enhancement process.

3.2 Image Processing Pipeline

1. Gaussian Smoothing

The first one is a Gaussian filter that reduces noise in an image, keeping important features while smoothing out unwanted distortions.

2. Gamma Correction

This step adjusts brightness using a non-linear transformation to enhance visibility in either the dark or overly bright regions by correcting the pixel intensity.

3. Gray World Color Balance

The gray world algorithm normalizes the color channels (RGB), expecting average color to be gray, thus compensating for color casts from lighting.

4. Iterative Improvement Section:

4.1 Brightness Correction

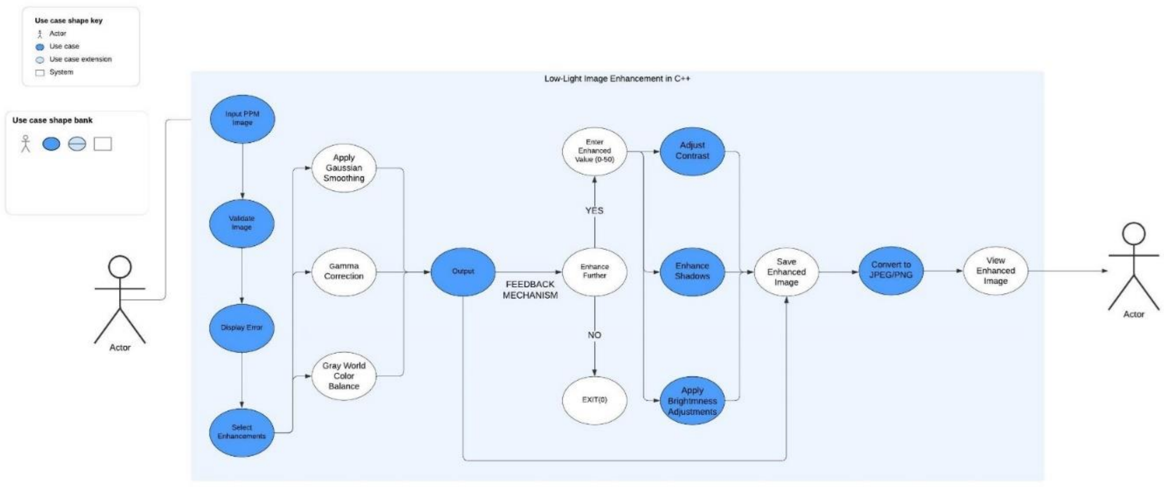
Increases or decreases the overall brightness of the image, depending on an individual's preference, darkening or lightening certain areas.

4.2 Contrast Improvement

Boosts the contrast of the image; it helps differentiate between bright and dark regions more prominently for easier clarity and details.

3.3 Use Case Model

A Use Case Model visually represents the interactions between users (or external systems) and the system itself.



3.4 Design Model

The Design Model describes the design architecture and structure of the image improvement system. It focuses on how all components and modules are meant to interact in the system towards achieving the required functionality. Below is an outline of the key components in the design:

3.4.1 System Architecture

- Frontend (User Interface): Developed using HTML, CSS, and JavaScript, the frontend is responsible for user interaction. Users can upload images, select enhancement techniques, preview results, and save final enhanced images.

- Backend (Flask Application): Developed with Flask, the backend manages all requests from users, invokes the C++ image processing code, and sends back the processed images to the frontend for display.
- C++ Image Processing Engine: The core of the system, where all image enhancement algorithms are implemented (e.g., Gaussian smoothing, brightness adjustment, etc.). This is compiled into an executable that is invoked by the Flask app as a subprocess for image processing.
- Communication Between Frontend and Backend: The frontend interacts with the backend through HTTP requests, while the backend invokes the C++ image processing engine to apply the selected enhancements.

3.4.2 Key Components

Image Data Handling: The image is initially read into a defined format, say PPM. Once read, it becomes a 2D matrix of pixel values and then input to work upon.

3.4.3 Modules

- Gaussian Smoothing: Removes noise from images and enhances detail quality.
- Gamma Correction: It corrects the brightness and tones of the image.
- Gray World Color Balancing: It corrects color aberrations based on the assumption that the average color of a scene is gray.
- Brightness & Contrast Adjustment: Brightness and contrast are adjusted to enhance the image's clarity and detail.
- Iterative Processing: Enhancement modules are available for iterated application, so the user can change parameters to fine-tune the output of the enhancement module.

3.4.4 Workflow

- Step 1: User uploads an image through the UI
- Step 2: The backend receives that image and picks enhancement techniques
- Step 3: The Flask app then sends the image along with enhancement parameters as a subprocess to the C++ executable
- Step 4: The C++ engine then processes that image and hands over the enhanced result.
- Step 5: The user previews the enhanced image.
- Step 6: The user can modify parameters, additional enhancement, or save the final image.

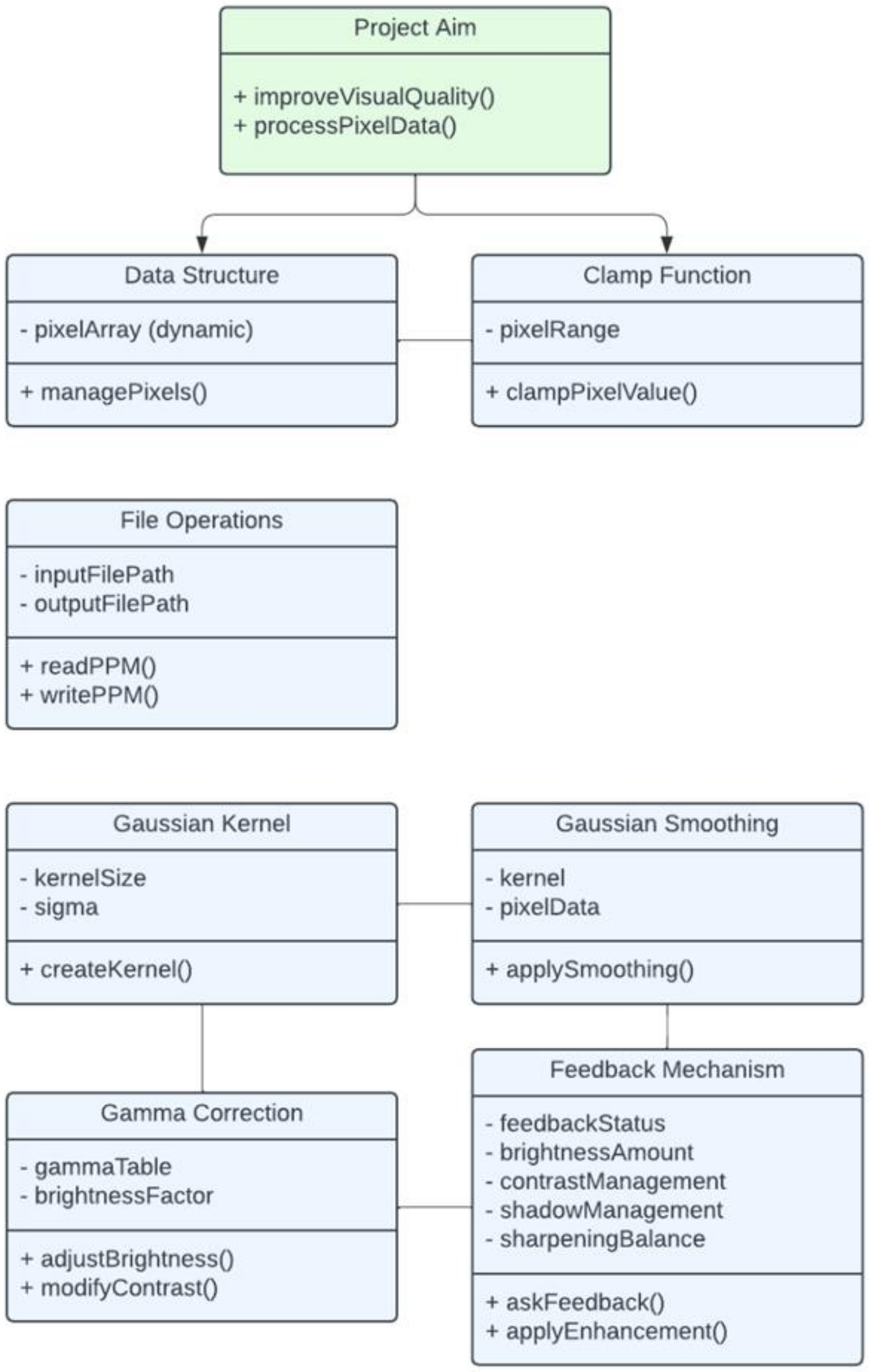
3.4.5 Data Flow

- Input: The original image (PPM format) and enhancement parameters.
- Processing: Applies image enhancement techniques sequentially.
- Output: The enhanced image in the desired format.

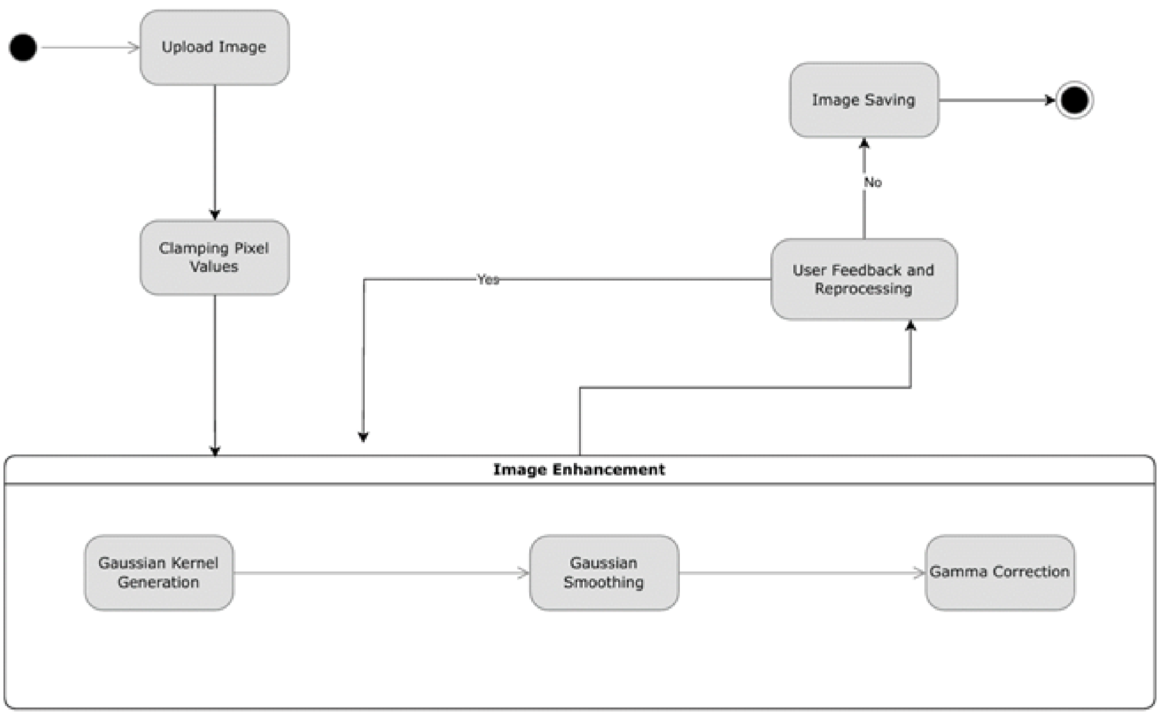
3.4.6 Modularity and Extensibility

- The system is designed with modularity in mind, allowing easy addition of new enhancement techniques in the future.
- Every enhancement technique is implemented as a separate C++ function and can be independently updated or extended.
- This design model ensures the system would be efficient and scalable and that there would be room for future improvements and enhancements.

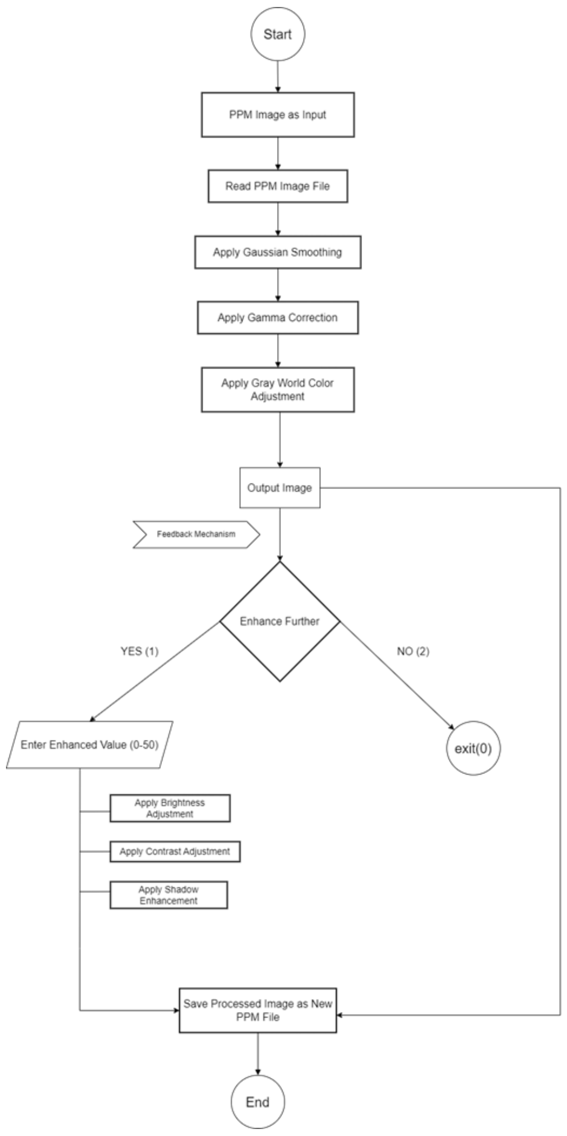
3.5 Class Diagram



3.6 State Transition Diagram



3.7 Activity Diagram



SharpView Image Enhancement System

4.1 Overview

The SharpView Image Enhancement System is a high-level solution designed to enhance images using advanced algorithms and improve clarity, reduce noise, adjust the brightness and contrast of visual attributes, and hence the details from an image. This system is well suited for applications in photography and video surveillance, medical imaging, and much more, guaranteeing not only outstanding image quality but also reliability.

4.2 System Components

- Input Module: Capture and upload the image for processing.
- Processing Unit: Execution of enhancement algorithms; noise reduction, sharpening, and contrast.
- Control Interface: Allows users to specify enhancement parameters.
- Output Module: Presents the enhanced images in the desired format.
- Data Storage: Stores original and processed images for later use.
- System Hardware: Contains all computational resources required to function effectively.

4.3 Technical Specifications

- Supported File Formats: JPEG, PNG, BMP, TIFF.
- Resolution Support: Full HD (1920 x 1080)
- Processing Speed: Image enhancement within 3-5 seconds on low-end devices.
- Enhancement Techniques:
  - Basic noise reduction - Median filtering.
  - Simple edge sharpening.
  - Contrast and brightness adjustments.
- User Interface: Lightweight and optimized for minimal system resources.
- Hardware Requirements:
- Minimum:
  - Dual-core processor. 1.5 GHz or above
  - 2GB of RAM
  - Integrated graphics support
- Recommended:
  - Quad-core processor.
  - 4GB of RAM
  - Basic GPU will be preferable.
- Compatibility: Optimized to be perfectly compatible with low-end devices running Windows, Linux, or Android systems.
- Output Quality: Optimized to have acceptable clarity while prioritizing speed and efficiency on resource-constrained hardware.

4.4 Features

- Light-Weight: Designed for low-end devices with minimal hardware requirements.
- Multi-Format Support: Compatible with popular image formats, including JPEG, PNG, BMP, and TIFF.

- Real-time enhancement: Enhancement speeds of 3-5 seconds with extremely low computations . User-friendly adjustment interface for brightness, contrast, and sharpening.
- Noise Reduction: Gets rid of graininess; increases the clarity of images.
- Edge sharpening : Improves detail and sharpness without over-processing, while maintaining a wide dynamic contrast.
- Cross-Platform Compatibility: Works flawlessly under Windows, Linux, and Android.
- Energy Efficient: Built to work efficiently on low-power devices, increasing their battery life.
- Storage Friendly: Processes and saves optimized images without considerable file size increases.

4.5 Architecture & Memory Organization

- **System Architecture:** Modular Design: Divided into independent functional modules such as input processing, image enhancement, and output generation for flexibility and scalability.
- **Low-Resource Optimization:** Tailored for low-end devices with focus on computational efficiency and minimal memory usage.
- **Client-Server Model (optional):** Supports local or remote processing to offload resource-intensive tasks to a server.
- **Memory Organization:** Input Buffer: Temporary container used to store the raw image data during upload for use in processing.
- **Memory Processing:** Dynamic memory allocation for executing operations on images such as filtering and adjustment with low latency.
- **Cache Memory:** Frequently used enhancement algorithms and settings which avoid redundant computations.
- **Output Buffer:** Holds processed image data before saving it or displaying the output. Ensures smooth transitions.
- **Memory Footprint:** Product may run in 2GB RAM on a minimum configuration. Efficient management of memory avoids overflow or bottlenecks.

This architecture ensures efficient use of system resources without compromising the quality of output, thus making it quite suitable for low-end devices.

Implementation

5.1 Algorithms for Image Enhancement

5.1.1 Gamma Correction Algorithm - Adjust the brightness of the image based on a non-linear transformation of pixel intensities.

- **STEPS:**
  i. Load the input images as a 2D matrix of pixel values.
  ii. Normalize pixel values to the range [0,1] by dividing each intensity value by 255.
  iii. Apply the Gamma correction formula:

$$\mathbf{I_{output} = I^{\gamma}_{output}}$$

where $\gamma>1$, reduces brightness(darker images), and $0 < \gamma < 1$ increases brightness(lighter images)

  iv. Rescale the normalized values back to [0, 255].
  v. Save the processed pixel values in the output image.

- **KEY FEATURES** – Efficient for improving brightness and contrast in specific regions, ensuring natural tone balance.

5.1.2 Gaussian Smoothing Algorithm – Reduce noise and smooth the image while preserving edges.

- **STEPS:**
   i. Define a Gaussian Kernel (a 2D matrix) using the formula:
   ii. Slide the kernel over the image, calculating the weighted sum of pixel intensities for each position.
   iii. Replace the central pixel of the kernel's position with the computed weighted sum.
   iv. Repeat for all pixels in the image.

- **KEY FEATURES** – Eliminates random noise and artificats, making the image appear smoother without significant distortion.

5.1.3 Gray World Algorithm – Correct the color balance of an image by assuming that the average color in a scene should be gray.

- **STEPS:**
   i. Calculate the average intensity of each color channel (Red, Green, Blue).
   ii. Determine the scaling factor for each channel:

Sc = [Average Intensity of All Channels/ Average Intensity of Channel c]

   iii. Adjust each pixel's color values:

$$[\ I'_c = I_c \cdot S_c\ ]$$

   iv. Clip values to the range [0, 255] to ensure valid pixel intensities.

- **KEY FEATURES** – Removes color casts caused by lighting conditions, producing a more natural appearance.

5.1.4 Brightness Adjustment Algorithm – Increase or decrease the overall brightness of an image based on user input.

- **STEPS:**
   i. Take the user-defined brightness factor(B) as input.
   ii. For each pixel, add the brightness factor to all RGB channels:

$$[\ I'_c = I_c + B\ ]$$

   iii. Clip pixel values to the range [0, 255] to avoid overflow or underflow.

- **KEY FEATURES –** Useful for enhancing visibility in underexposed or overexposed areas.

5.1.5 Contrast Adjustment Algorithm – Improve the difference between light and dark areas in the image, making details more prominent.

- **STEPS:**
  i. Compute the average intensity of the image (I_{avg}) across all pixels.
  ii. For each pixel, adjust its intensity based on a contrast factor (C):

$$[\ I'_c = I_{avg} + C.\ (I_C - I_{avg})\ ]$$

where C>1 increases contrast, and 0 < C < 1 decreases contrast.

  iii. Clip pixel values to the range [0, 255].

- **KEY FEATURES –** Enhances fine details and sharpens images by boosting the distinction between bright and dark regions.

These algorithms give a strong enhancement pipeline for images to meet challenges of brightness, contrast, noise, and color correction toward high-quality image creation.

5.2 User Interface Integration

5.2.1 Frontend: HTML, CSS and Vanilla JS

The frontend of the image enhancement tool is designed as a user-friendly interface for interaction. It is implemented using HTML, CSS, and vanilla JavaScript to maintain simplicity and responsiveness while ensuring efficient performance on resource-constrained devices.

❖ **Key Components of the Frontend:**

I. HTML (Structure)

– Upload Section of the Image: Lets the user upload an image to be enhanced.
– Enhancement Options: Drop-downs, sliders, or check boxes for choosing and applying enhancement methods (for example: brightness, contrast).
– Preview Window: Show both the original and enhanced images simultaneously.
– Action Buttons: Such as "Apply Enhancements", "Reset", "Download Enhanced Image".

II. CSS (Styling)

– The interface should be aesthetically pleasing and user-friendly:
– Responsive Design : Should make it responsive for use on desktop and notebook computers, tablets, as well as mobile phones.
– Theme: This uses minimalist and clean styles with design focus that keeps the emphasis on functionality.
– Animations: Adds soft animations within preview or save image for smooth transitions

III. JavaScript(Interactivity)

The JavaScript component of the frontend enhances user interactivity, enabling a seamless experience for uploading, enhancing, and previewing images. It ensures dynamic and responsive behaviour throughout the workflow.

- **Key Functionalities**

1. File Upload Handling

– Displays the name of the selected image file for better user awareness.

– Validates that a file has been uploaded before proceeding with any action.

2. Viewing the Uploaded Image

– Displays the uploaded image in a modal for preview before applying enhancements.

– Uses URL.createObjectURL to dynamically render the image within the browser without uploading it to the server.

3. Image Enhancement Process

– Sends the uploaded image to the backend (/enhance endpoint) via a POST request using the Fetch API.

– Handles backend responses to dynamically display the enhanced images returned by the Flask server.

– Supports multiple enhancement techniques (e.g., Gamma Correction, Gaussian Smoothing) and labels them appropriately for user clarity.

4. Iterative Enhancement

– Prompts the user for additional adjustments, such as brightness, allowing for fine-tuning of the enhanced image.

– Sends iterative enhancement requests to the backend (/iterative_enhance endpoint) with user-specified parameters.

– Updates the display dynamically with each iteration for real-time feedback.

5. Modal and Enhanced Image Box

– Implements modal functionality to display images (both original and enhanced) in an organized, user-friendly way.

– Includes options to close the modal or enhanced image box by clicking a close button or outside the modal area.

6. Real-Time Interaction

– Uses setTimeout to introduce a delay before prompting for further enhancement, allowing the user to assess the current output.

– Implements recursive calls for iterative enhancements, ensuring a continuous and responsive process based on user inputs.

▪ **Code Highlights**

1. **Dynamic Image Display**

– Renders uploaded and enhanced images dynamically within a modal or designated section on the page.

– Assigns descriptive labels (e.g., "Gamma Image," "Gaussian Image") to help users understand the applied techniques.

2. **Fetch API for Backend Communication**

– Sends image files and enhancement parameters to the Flask backend and processes JSON responses.

– Handles both success and error scenarios gracefully, providing feedback to the user.

3. **User Prompts and Feedback**

- Prompts the user for brightness values during iterative enhancement, ensuring a customized experience.
- Alerts users about errors, such as missing files or failed enhancements, for better clarity.

4. **Dynamic Updating**

- Updates the preview of enhanced images in real-time, allowing users to assess and adjust enhancements without reloading the page.

5. **Closing Modal and Enhanced Box**

- Provides intuitive close options to dismiss modals or enhanced boxes, improving usability.

▪ **Advantages**

- Enables a responsive and dynamic user interface.
- Simplifies the enhancement process with intuitive prompts and real-time feedback.
- Reduces user frustration with clear validation, error handling, and interactive previews.

This JavaScript implementation ensures smooth interactivity, effectively bridging the frontend and backend functionalities for a user-friendly image enhancement application.

5.2.2 Backend: Flask Application

The backend of the system is implemented using Flask, providing essential functionality for image processing and managing user interactions. It handles the upload of images, initiates the enhancement process by invoking Python scripts and C++ subprocesses, and returns enhanced images to the frontend.

❖ **Key Features**

1. File Handling

- Uploaded images are stored in a dedicated uploads folder.
- Enhanced images are saved in the enhanced folder, ensuring proper organization and accessibility.
- The backend supports cleanup mechanisms to clear old files before new enhancements.

2. Image Enhancement

- Receives an uploaded image from the frontend.
- Saves the image and triggers the automation.py script using subprocess.run, passing the file path as input.
- Collects enhanced images from the enhanced folder and sends them back to the frontend.

3. Iterative Enhancement

- Takes brightness adjustments as query parameters from the frontend.
- Passes these parameters, along with the current enhanced image, to iterativeEnhance.py for further processing.
- Converts the output from PPM to JPEG using the Python Imaging Library (PIL).
- Returns the updated image to the frontend for preview.

4. Static File Management

- Serves frontend files (HTML, CSS, JavaScript) through defined routes, allowing seamless integration between the frontend and backend.
- Provides a cache-free delivery of enhanced images to ensure users always see the latest versions.

5. Error Handling

- Checks for missing files or invalid inputs before processing.
- Handles subprocess errors (e.g., failed execution of Python scripts or C++ code) gracefully and sends meaningful error messages to the frontend.

6. Real-Time Feedback Support

- Enables iterative enhancements by dynamically adjusting parameters like brightness and contrast.
- Facilitates on-the-fly updates to the processed image for a responsive user experience.

❖ **Workflow**

1. Initial Enhancement

- The user uploads an image via the frontend.
- Flask saves the image in the uploads folder.
- The automation.py script is executed as a subprocess to apply all core enhancement techniques (e.g., Gaussian smoothing, gamma correction).
- Enhanced images are saved in the enhanced folder and returned to the frontend.

2. **Iterative Enhancement**

- The user modifies parameters such as brightness.
- The updated values and the most recent enhanced image are passed to the iterativeEnhance.py script.
- The backend processes the image iteratively and sends the updated version to the frontend for preview or download.

3. **Static File Handling**

- Frontend files (HTML, CSS, JavaScript) are served from the Frontend directory.
- Enhanced images are served dynamically from the enhanced folder with proper caching disabled.

5.2.3 C++ Integration with Backend

The image enhancement system integrates the C++ engine with the Python backend using Python's subprocess module. This approach leverages the computational efficiency of C++ for image processing while maintaining flexibility in communication through Python and Flask.

❖ **Key Integration Details**

1. **C++ Executable**

   - The core image enhancement algorithms (e.g., Gaussian smoothing, gamma correction, brightness adjustment) are implemented in C++ for high performance.
   - The C++ code is compiled into an executable file, which is invoked by the Python backend to process images.

2. **Python Subprocess Module**

- The Python subprocess module is used to run the C++ executable as a separate process.
- Input parameters (e.g., image paths and enhancement settings) are passed as command-line arguments to the executable.
- Output files generated by the C++ process (e.g., enhanced images) are captured and returned to the Flask backend for further handling.

RESULTS & OUTPUTS

The image enhancement system successfully improves the visual quality of low-light and degraded images. Outputs include enhanced versions showcasing:

1. **Gamma Correction**: Improved brightness and tonal balance.
2. **Gaussian Smoothing**: Reduced noise and sharper edges.
3. **Gray World Color Balance**: Corrected color tones for natural visuals.
4. **Brightness and Contrast Adjustments**: Enhanced visibility and detail clarity.

The results demonstrate the system's ability to process images efficiently, with real-time adjustments and iterative enhancements providing user-controlled outputs. Visual comparisons (before and after) validate the effectiveness of the implemented techniques.

CONCLUSION

The project successfully delivers a lightweight, standalone image enhancement tool using C++, addressing challenges in low-light and degraded images. By employing techniques like gamma correction, Gaussian smoothing, and brightness adjustment, it achieves high-quality enhancements suitable for resource-constrained devices. The integration with a Flask backend and a user-friendly interface ensures accessibility and interactivity. This tool is ideal for both educational and practical applications, providing a scalable solution for real-time image processing.

APPENDICES

Appendix A: Glossary of Terms

1. Gamma Correction: Adjusting image brightness non-linearly.
2. Gaussian Smoothing: Reducing noise using a weighted kernel.
3. Gray World Color Balance: Correcting color casts by balancing RGB channels.
4. Brightness Adjustment: Modifying pixel intensity for better visibility.
5. Contrast Adjustment: Enhancing the difference between light and dark areas.

Appendix B: Software & Hardware Specifications

1. **Software**:
   - Languages: C++, Python (Flask), HTML, CSS, JavaScript.
   - Libraries: PIL (Python Imaging Library).
   - Tools: GCC, Python 3.x, Flask framework.

2. **Hardware**:

   – Processor: Minimum Dual-Core.
   – RAM: Minimum 4 GB.
   – Disk Space: At least 1 GB for image storage and processing.

SPECIFICATIONS

The specifications define the technical requirements and features of the image enhancement system to ensure its efficient operation and usability.

9.1 Software Specifications

- **Programming Languages**:
  - C++: Core image processing algorithms.
  - Python: Flask backend for communication and execution.
  - HTML, CSS, JavaScript: Frontend for user interface.
- **Frameworks and Libraries**:
  - Flask: Backend framework for managing requests and serving files.
  - PIL (Python Imaging Library): Used for image format conversions.
- **Image Formats**:
  - Input: JPEG, PNG (converted to PPM for processing).
  - Output: Enhanced images saved as JPEG or PNG.

9.2 Hardware Specifications

- **Processor**: Minimum Dual-Core, recommended Quad-Core for faster processing.
- **RAM**: Minimum 4 GB, recommended 8 GB for high-resolution images.
- **Storage**: At least 1 GB free space for image storage and temporary files.
- **Graphics Support**: Not required; relies on CPU for processing.

9.3 Functional Specifications

- **Input Handling**:
  - Accepts images through a web-based interface.
  - Provides validation for file type and size.
- **Image Enhancement Techniques**:
  - Gamma Correction, Gaussian Smoothing, Gray World Color Balance.
  - Brightness and Contrast Adjustment.
- **Iterative Enhancement**:
  - Supports real-time adjustments based on user feedback.
- **Output**:

– Saves enhanced images in user-friendly formats with consistent quality.

9.4 Non-Functional Specifications

- **Efficiency**: Processes moderate-sized images (few MBs) within seconds.
- **Scalability**: Designed for extensibility, allowing additional techniques to be integrated.
- **Portability**: Compatible with Windows and Linux operating systems.
- **User Accessibility**: Simple and intuitive web-based interface for non-technical users.

REFERENCES

– [1] A Dynamic Histogram Equalization for Image Contrast Enhancement M. Abdullah-Al-Wadud, Md. Hasanul Kabir, M. Ali Akber Dewan, and Oksam Chae, Member, IEEE

– [2] Low-Light Image Enhancement: A Comparative Review and Prospects WONJUN KIM , (Member, IEEE) Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, South Korea

– [3] Gray World based Color Correction and Intensity Preservation for Image Enhancement N.M. Kwoka, D. Wanga, X. Jiab, S.Y. Chenc, G. Fangd and Q.P. Hae aSchool of Mechanical and Manufacturing Engineering The University of New South Wales, Australia bSchool of Information Technology