Project Title:

Image to Grayscale Converter using NumPy and PyQt

Anurag Paul (22MID0080), Kinjal Ghosh (22MID00331), Pratik A Shah (22MID0281), Dhanushvanth VH (22MID0063)

1. Abstract

This project implements a simple yet effective **desktop application** that converts any given colour image into its grayscale equivalent using **NumPy-based pixel-level computation**.
The application is developed with **PyQt5** for the graphical user interface (GUI) and relies solely on **NumPy** for image data processing — without using any specialized image-processing libraries such as OpenCV or Pillow.

The program supports both **file selection** and **drag-and-drop** image loading, displays the **original** and **grayscale** images side by side, and allows the user to **save** the generated grayscale image.
The objective of this mini project is to demonstrate how basic **matrix and numerical operations** can be effectively utilized for image manipulation.

2. Project Objectives

- To build an intuitive GUI application for converting colour images to grayscale.
- To perform grayscale conversion purely using **NumPy operations** (no OpenCV/Pillow).
- To enable user interaction through **PyQt** with image loading, visualization, and saving features.
- To understand how pixel data can be accessed and manipulated at the **array level** using NumPy.

3. Tools and Technologies Used

Component	**Technology / Library**
Programming Language	Python 3
GUI Framework	PyQt5
Image Processing	NumPy
Development Environment	VS Code / PyCharm / IDLE
Supported Formats	PNG, JPG, JPEG, BMP, GIF

4. Project Approach

The project integrates GUI design with numerical computation using the following layered approach:

1. **Frontend (User Interaction):**
    - Developed using **PyQt5**, providing buttons and drag-and-drop functionality.
    - Displays two image panels: one for the original image and another for the grayscale version.
    - Includes buttons for “Load Image” and “Save Grayscale”.
2. **Backend (Image Conversion Logic):**
    - The selected image is loaded as a **QImage** object.
    - Its raw pixel buffer is accessed and converted into a **NumPy array**.
    - The RGB pixel values are processed mathematically to compute grayscale intensities using the formula: Gray = [0.299*R + 0.587*G + 0.114*B]
    - The resultant grayscale matrix is then converted back into a **QImage** for display.
3. **Output Handling:**
    - The grayscale image is displayed beside the original.
    - Users can save the processed image to a local directory.

5. Methodology

Step 1: Image Loading

- The user selects or drags an image into the application.
- The QImage class reads the file into memory.
- The pixel buffer is accessed using bits() and converted into a NumPy array.

Step 2: Image Conversion

- Each pixel’s RGB values are extracted from the NumPy array.
- A linear transformation (dot product) is applied using the weighted coefficients [0.299, 0.587, 0.114].
- The output is cast to uint8 type to form a valid grayscale image.

Step 3: Display and Save

- The grayscale matrix is transformed back into a QImage object for rendering in PyQt.
- The user can optionally save the grayscale image in PNG or JPEG format.\

6. Observations and Results

- The grayscale image produced is visually accurate and matches expected luminance distribution.
- The vertical line artifacts were initially observed due to **byte padding** in QImage data; resolved by adjusting for **bytesPerLine**.
- The NumPy-based pixel manipulation is efficient and fast even for large images.
- The PyQt interface remains responsive, supporting drag-and-drop for better usability.
- Memory consumption remains minimal since only array operations are used.

7. Advantages

- Lightweight and dependency-free (uses only NumPy + PyQt).
- Demonstrates direct pixel manipulation using NumPy arrays.
- Educational value in understanding color-to-grayscale transformation.
- Platform-independent and easy to extend (can add filters or color effects).

8. Limitations

- Does not currently handle alpha transparency (translucent pixels are ignored).
- Limited to grayscale conversion — no advanced filters yet.
- Processing speed may vary slightly for very large images due to NumPy array reshaping.

9. Future Enhancements

- Add support for other image effects (sepia, blur, contrast adjustment, etc.).
- Include histogram visualization of grayscale pixel intensities.
- Integrate real-time preview with sliders for brightness and contrast.
- Extend to batch image processing.
- Provide export options for different formats and compression levels.

10. Conclusion

This mini project demonstrates how a simple concept like **grayscale conversion** can be implemented efficiently using **NumPy's matrix operations** integrated within a **PyQt GUI framework**.
It successfully bridges theoretical image-processing formulas with practical, interactive visualization — making it an ideal example of numerical computing applied to real-world graphics.
The project reinforces understanding of how **RGB data is stored and manipulated**, and provides a foundational base for more advanced image-processing projects.

11. Output:

Initial App:

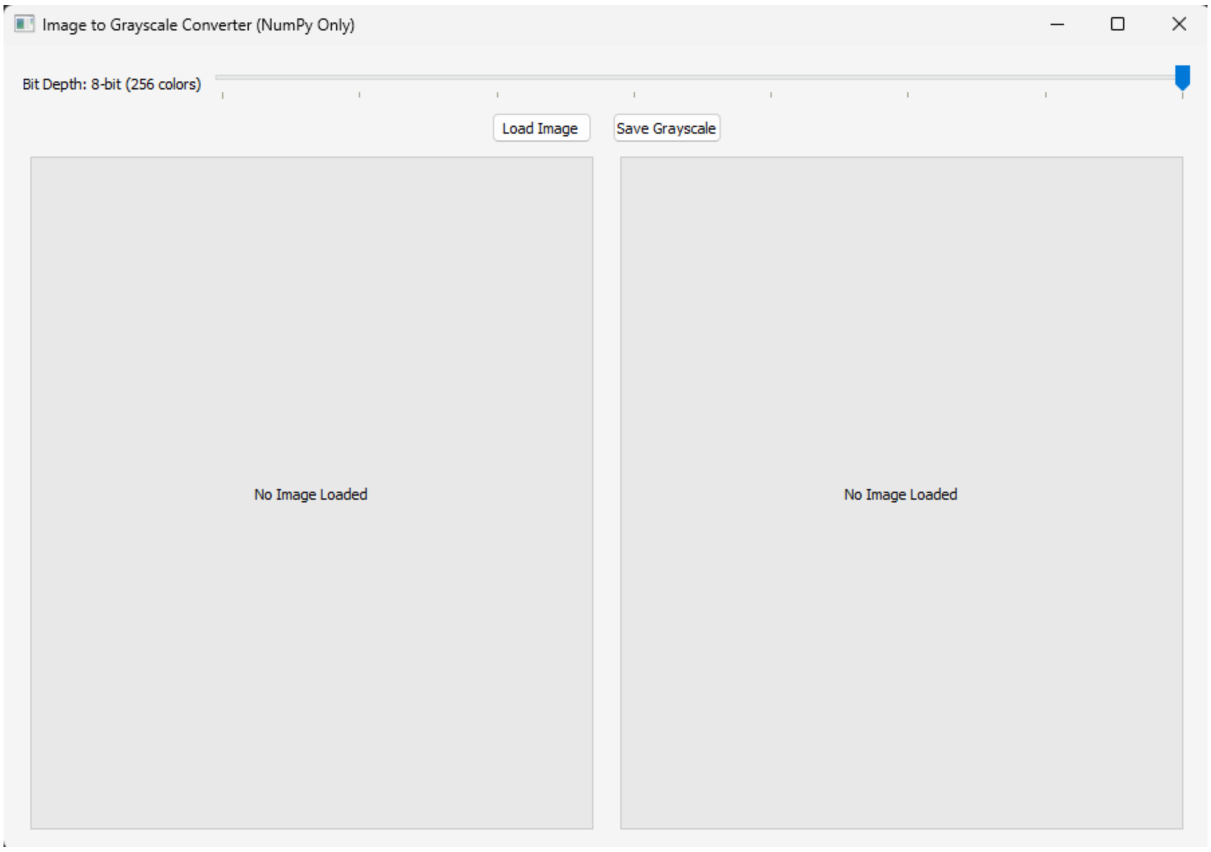


Image to Grayscale Converter (NumPy Only)
Bit Depth: 8-bit (256 colors)
Load Image
Save Grayscale
No Image Loaded
No Image Loaded

Greyscale Conversion:

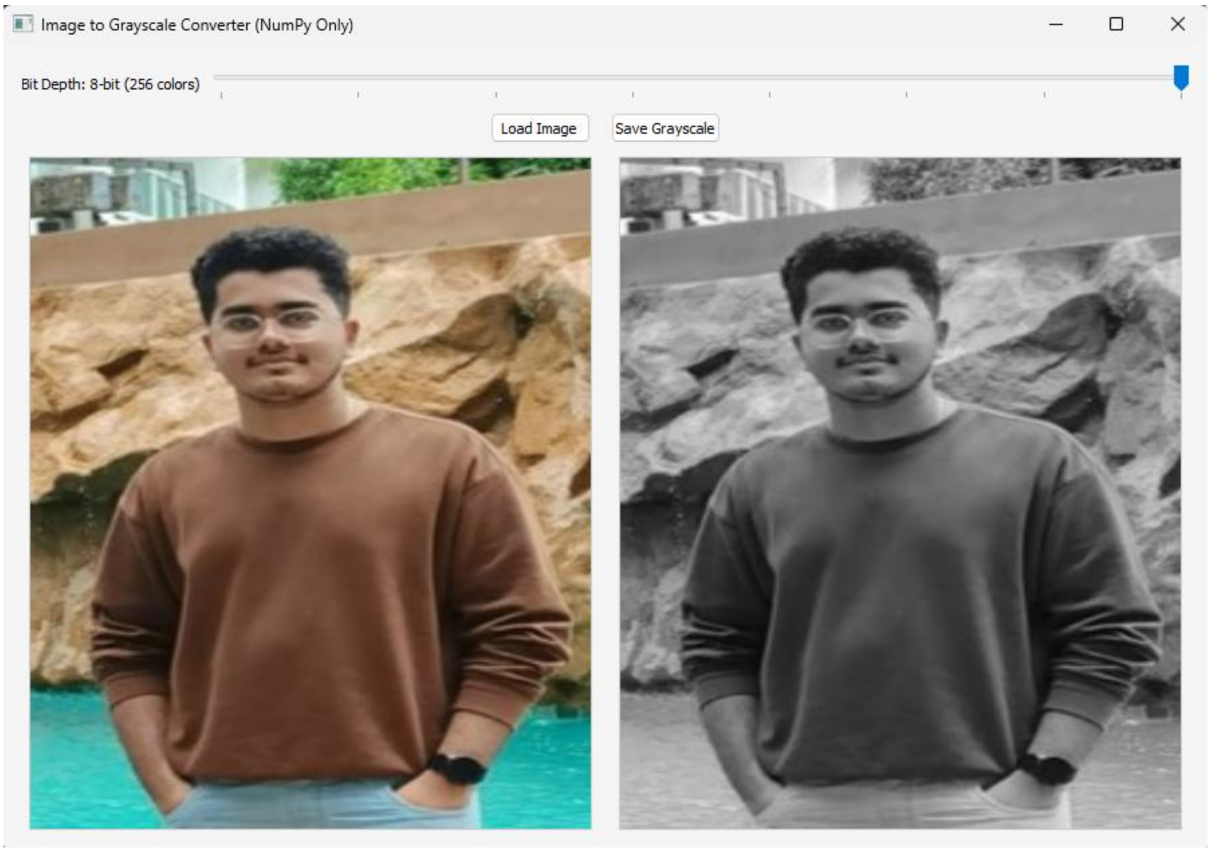
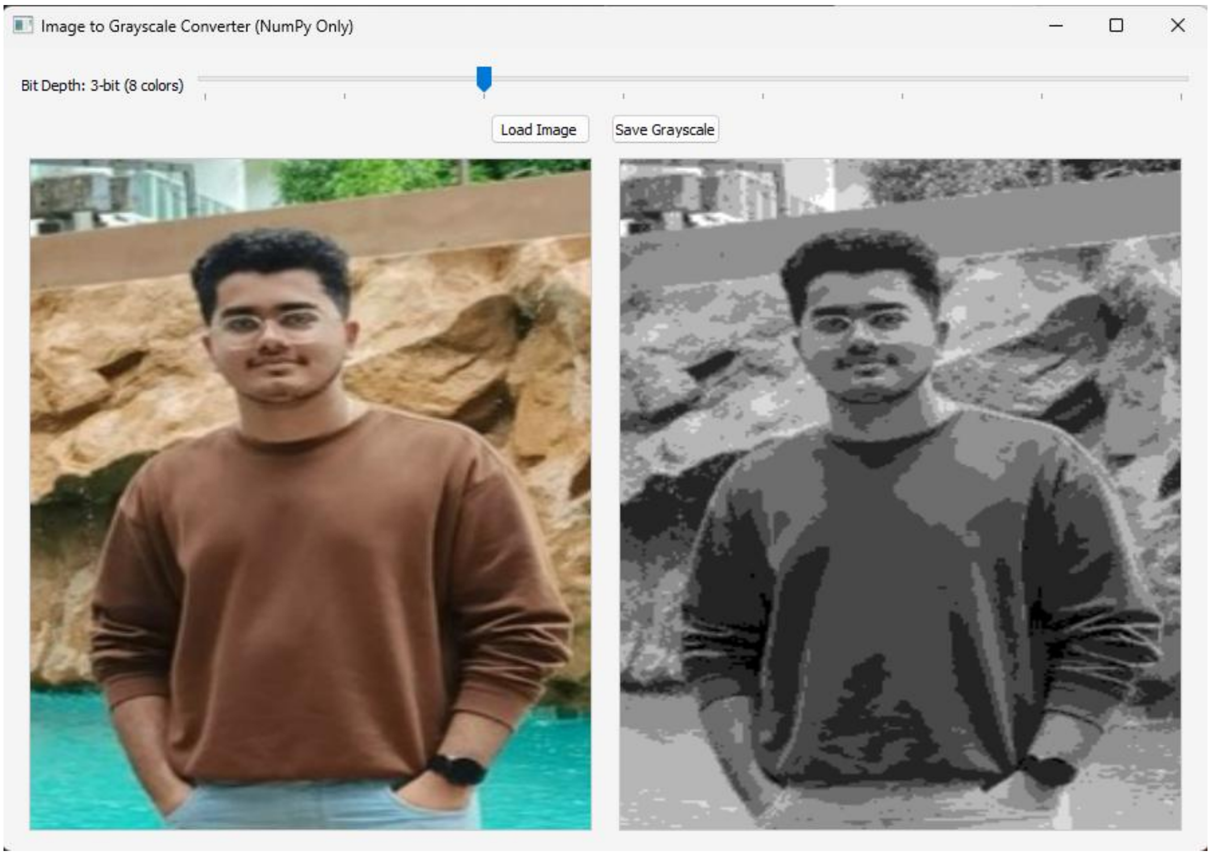


Image to Grayscale Converter (NumPy Only)
Bit Depth: 8-bit (256 colors)
Load Image
Save Grayscale

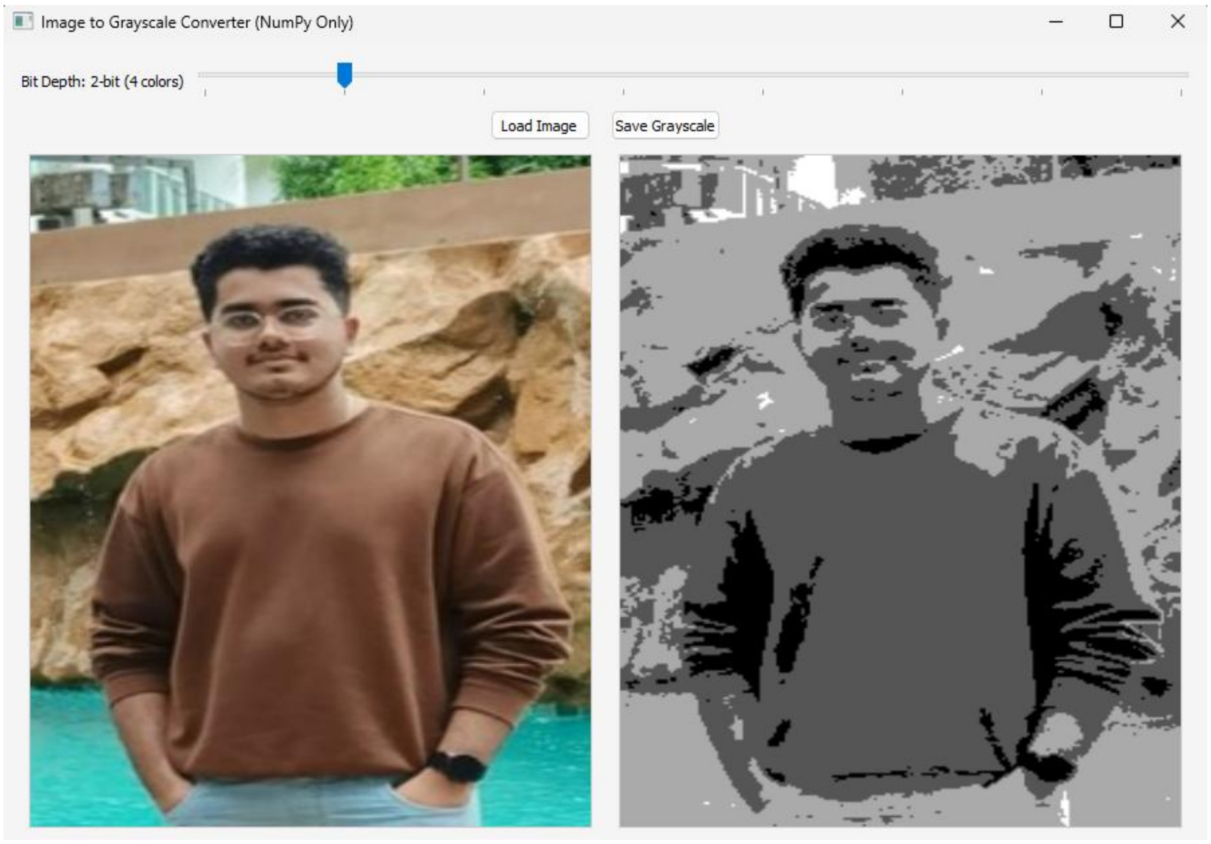
Greyscale with Quantization:

Level 8 Quantization:



Bit Depth: 3-bit (8 colors)

Level 4 Quantization:



Bit Depth: 2-bit (4 colors)

Project GitHub Link:

Link: GrayScale-Inverter-With-Quantization/Greyscale_Inverter/src/main.py at main · Anurag30032004/GrayScale-Inverter-With-Quantization