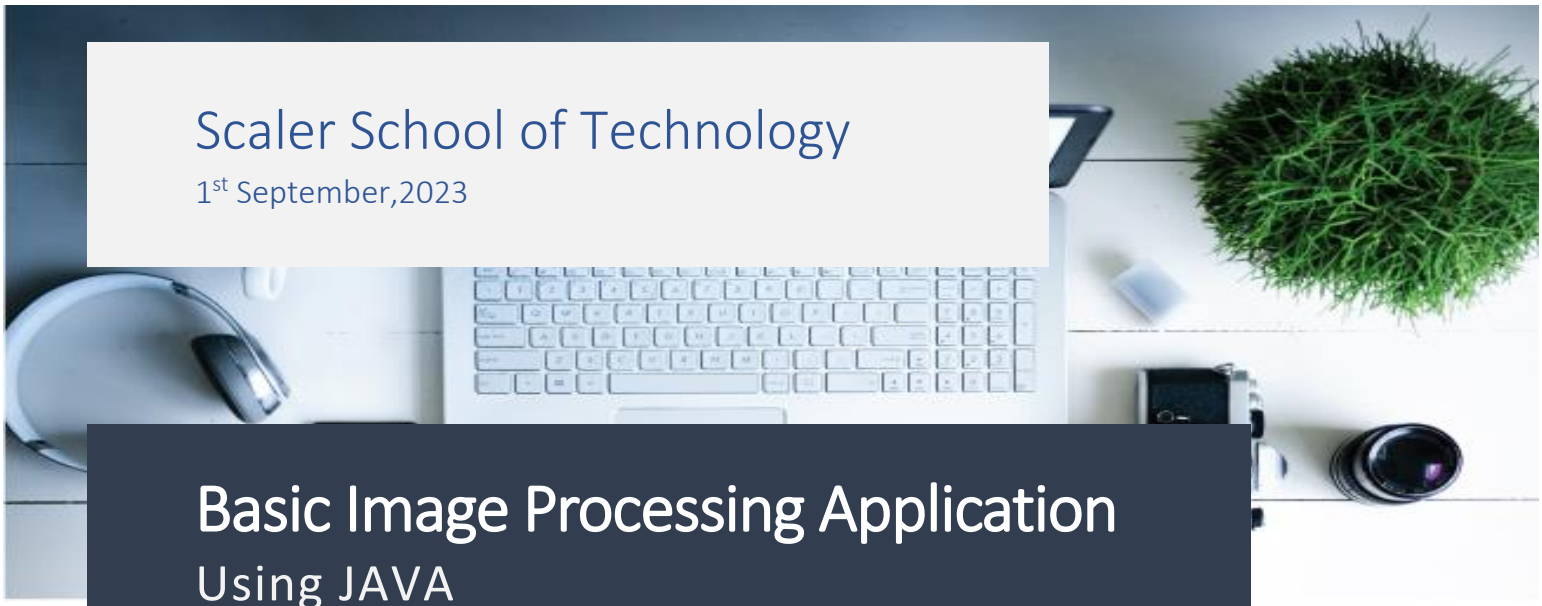


Scaler School of Technology

1st September,2023

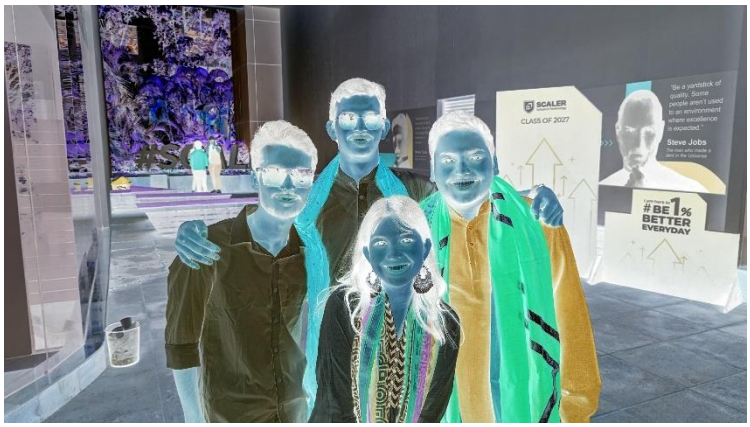
Basic Image Processing Application

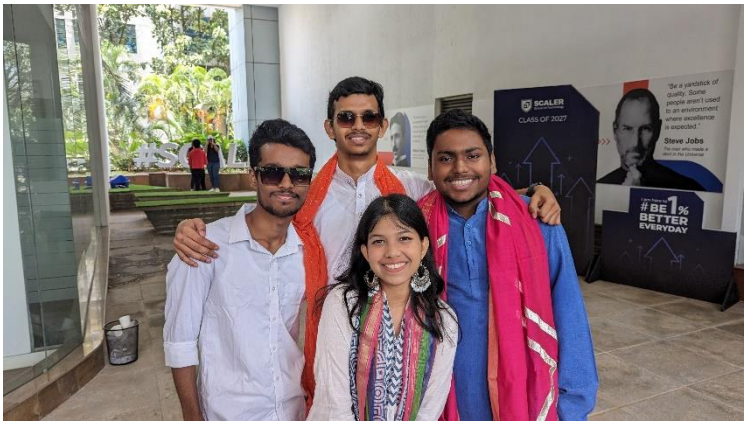
Using JAVA

Table of Contents:

1. Introduction

In the era of digital media and photography, image processing plays a vital role in enhancing and manipulating images. This project aims to develop an Image Processing Application that allows users to apply various image processing operations to input images. The application provides options for adjusting brightness, converting images to grayscale, creating negative images, rotating images, flipping images horizontally, and applying Gaussian and pixelated blurs.





2. Problem Statement

The project addresses the need for a versatile image processing tool that can be utilized for various purposes, from basic image enhancements to creative transformations.

3. Methodology

The methodology section outlines the algorithms and techniques employed in the Image Processing Application to achieve various image transformation operations.

3.1 Brightness Adjustment

Algorithm: The brightness adjustment function allows users to modify the brightness of an image. It employs the following algorithm:

```
newPixel = originalPixel + (originalPixel * brightnessFactor)
```

This algorithm scales each pixel's color values by the specified brightness factor while ensuring that resulting values stay within the valid range of 0-255.

3.2 Grayscale Conversion

Algorithm: The grayscale conversion function transforms color images into grayscale using the following algorithm:

```
gray = (red + green + blue) / 3
```

For each pixel, it computes the average of the red, green, and blue color channels to derive the grayscale value. This results in a grayscale image with varying shades of gray.

3.3 Negative Image Transformation

Algorithm: The negative image transformation function inverts colors within the input image using the following algorithm:

```
newRed = 255 - originalRed
newGreen = 255 - originalGreen
newBlue = 255 - originalBlue
```

By subtracting each color channel value from 255, the algorithm creates a negative image with colors inverted.

3.4 Image Rotation

Algorithm: The image rotation function offers three rotation options: clockwise by 90 degrees, counterclockwise by 90 degrees, and a 180-degree rotation. These rotations are achieved through pixel rearrangement without altering the color values:

- Clockwise 90 degrees: **outputPixel(x, y) = inputPixel(height - y - 1, x)**
- Counterclockwise 90 degrees: **outputPixel(x, y) = inputPixel(y, width - x - 1)**
- 180-degree rotation: **outputPixel(x, y) = inputPixel(width - x - 1, height - y - 1)**

3.5 Horizontal Image Flip

Algorithm: The horizontal image flip function mirrors the image horizontally by swapping pixel values between the left and right sides:

```
outputPixel(x, y) = inputPixel(width - x - 1, y)
```

This simple operation effectively flips the image along its vertical axis.

3.6 Gaussian Blur

Algorithm: The Gaussian blur function applies a blur effect to the image using a convolution kernel. The algorithm calculates a weighted average of pixel values within a specified radius, effectively smoothing the image. While the provided code demonstrates a basic blur algorithm, a more optimized version can employ a Gaussian kernel for smoother results.

3.7 Pixelated Blur

Algorithm: The pixelated blur function divides the image into blocks and calculates the average color for each block. The user can specify the block size. The algorithm follows these steps:

1. Divide the image into blocks of the specified size.
2. Calculate the average color within each block.
3. Replace all pixels within a block with the average color.

This operation pixelates the image by reducing details within each block.

These algorithms collectively enable the Image Processing Application to offer a range of image transformation capabilities, from simple adjustments like brightness and grayscale conversion to more complex operations like rotation and blur.

4. Implementation

The project is implemented in Java and leverages the Java AWT and ImageIO libraries for image processing. User input is handled using the Scanner class. Each function is encapsulated in a separate method to ensure modularity and maintainability.

5. Results

The application successfully processes input images according to the chosen operations, generating output images with the desired transformations.

6. Conclusion

In conclusion, the Image Processing Application provides a user-friendly interface for performing various image processing tasks. It offers versatility and ease of use, making it suitable for both novice and experienced users. The project demonstrates the capability of Java for image processing tasks and lays the foundation for potential enhancements and future improvements.

7. Future Improvements

The project can be enhanced in several ways:

- **User Interface**: Develop a graphical user interface (GUI) for a more intuitive user experience.
- **Performance Optimization**: Optimize image processing algorithms for faster execution.
- **Additional Features**: Add more advanced image processing features such as edge detection and sharpening.
- **File Format Support**: Expand support for different image file formats.
- **Error Handling**: Implement robust error handling and validation for user inputs.

These improvements will enhance the application's functionality and performance, making it a more powerful tool for image processing tasks.

8. References

- Java AWT Documentation
- Java ImageIO Documentation

9. Acknowledgement

I extend my sincere thanks to our instructors, Mr. Kshitij Mishra and Mr. Pragy Agrawal, for their invaluable guidance and resources that made this project possible. I'm also grateful to my friends, Sandip Das and Sourashis Sarkar, for their collaboration and support throughout this project.

Their contributions and collaborative spirit greatly enriched this learning experience.