**Mini Project Report**

**JAVA** **IMAGE FILTER**

In Fulfillment of Internal Assessment in the Course of

**Data structures (22CS102003)**

**BACHELOR OF TECHNOLOGY**

**IN**

**CSE(DATA SCIENCE)**

**Submitted by**

**Kothuri Rahul 22102A030207**

**Bejagam Balu Amareshwar 22102A030152**

**Bejagam Sri Siva Naga malleshwar 22102A030153**

**Geepalem Leena 22102A030182**

**Banka Lakshmi 22102A030150**

**R Charishma 22102A030248**

**Bakkamunthala Naga Teja 22102A030147**

****

Department of Data Science

**MOHAN BABU UNIVERSITY**

Sree Sainath Nagar, Tirupati – 517 102, A.P., INDIA

2022-2023

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| CONTENT | PAGE NO |
| 1.Introduction | 1 |
| 2.Abstract | 2 |
| 3.System Requirements(about techstack) | 3-4 |
| 4.Program(source code) | 5-11 |
| 5.Program output(with screenshot) | 12-13 |
| 6.Conclusion | 14 |

**INTRODUCTION**

Image filtering is a fundamental technique in digital image processing that plays a pivotal role in enhancing, analyzing, and transforming images. It involves applying a mathematical operation to each pixel in an image to modify its value based on the values of neighboring pixels. This process allows us to achieve a wide range of visual effects and manipulate image characteristics.

Image filtering serves various purposes, from removing noise and sharpening details to blurring and highlighting specific features. It is commonly used in diverse fields, such as computer vision, medical imaging, photography, and art. Filters can be designed to emphasize edges, reduce unwanted artifacts, or create artistic effects, making image filtering a versatile tool for both practical and creative applications.

In this introduction, we will explore the key concepts and techniques associated with image filtering, discussing the different types of filters and their real-world applications. Whether you are interested in improving the quality of photographs, extracting valuable information from medical scans, or exploring the artistic possibilities of image manipulation, understanding image filtering is essential for unlocking the potential of digital imagery.

**ABSTRACT**

Digital image processing plays a critical role in various domains, from medical diagnostics to multimedia applications. One fundamental aspect of image processing is image filtering, a technique that enhances, transforms, or extracts information from images. This project focuses on the implementation of image filtering using Java, with the aim of providing a versatile and efficient tool for image manipulation.

This project leverages Java's capabilities to design and apply various image filters, including convolution filters for blurring and sharpening, edge detection filters, and noise reduction filters. We also incorporate advanced filtering techniques, such as bilateral filtering for preserving edges while reducing noise. The Java programming language provides a robust platform for building a user-friendly graphical interface, enabling users to interact with the software intuitively.

**SYSTEM REQUIREMENTS**

The provided source code appears to be a Java program for image processing. To run this program, you'll need a Java development environment and certain system requirements. Here are the general system requirements:

**Java Development Kit (JDK):** You will need to have the Java Development Kit (JDK) installed on your system. You can download and install the latest version of JDK from the official Oracle website or use open-source alternatives like OpenJDK.

**Operating System:** The code is written in Java and should be platform-independent. You can run it on Windows, macOS, or Linux.

**Sufficient System Resources:** The system requirements may vary depending on the size of the images you want to process and the complexity of the operations you apply to them. In general, a modern computer with a multi-core processor and sufficient RAM is recommended for efficient image processing.

**Input Images:** You should have the input images in a format that the program supports, such as BMP. Ensure that the input images are accessible from the location where you run the program.

**ImageIO Library:** The program uses the `ImageIO` class from Java's `javax.imageio` package for image input and output. This library is typically included with the JDK, so you don't need to install it separately.

**Command Line or IDE:** You can run the program from the command line or using an integrated development environment (IDE) like Eclipse, IntelliJ IDEA, or any text editor that supports Java development.

**Command Line Arguments:** The program expects command line arguments in the format `[filter] [infile] [outfile]`. Make sure you provide these arguments when running the program.

**Permissions:** Ensure that you have the necessary read and write permissions for the input and output image files and the directories where they are located.

Overall, the specific system requirements may vary based on the actual use case and the size of the images you plan to process. It's recommended to have a modern system with a recent version of the JDK to ensure smooth execution of the program.

**SOURCE CODE**

**import java.awt.image.\*;**

**import java.io.\*;**

**import javax.imageio.ImageIO;**

**public class ImageProcessor {**

**public static void main(String[] args) {**

**if (args.length != 3) {**

**System.out.println("Usage: java ImageProcessor [filter] [infile] [outfile]");**

**System.exit(1);**

**}**

**String filter = args[0];**

**String infile = args[1];**

**String outfile = args[2];**

**try {**

**BufferedImage image = ImageIO.read(new File(infile));**

**RGBTRIPLE[][] imageArray = convertToRGBTripleArray(image);**

**switch (filter) {**

**case "grayscale":**

**ImageFilters.grayscale(imageArray);**

**break;**

**case "sepia":**

**ImageFilters.sepia(imageArray);**

**break;**

**case "reflect":**

**ImageFilters.reflect(imageArray);**

**break;**

**case "blur":**

**ImageFilters.blur(imageArray);**

**break;**

**case "invertColors":**

**ImageFilters.invertColors(imageArray);**

**break;**

**case "edgeDetect":**

**ImageFilters.edgeDetect(imageArray);**

**break;**

**case "pixelate":**

**ImageFilters.pixelate(imageArray);**

**break;**

**case "posterize":**

**ImageFilters.posterize(imageArray);**

**break;**

**case "noise":**

**ImageFilters.noise(imageArray);**

**break;**

**default:**

**System.out.println("Unsupported filter.");**

**System.exit(2);**

**}**

**BufferedImage filteredImage = convertToBufferedImage(imageArray, image.getWidth(), image.getHeight());**

**ImageIO.write(filteredImage, "bmp", new File(outfile));**

**System.out.println("Filter applied successfully.");**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**public static RGBTRIPLE[][] convertToRGBTripleArray(BufferedImage image) {**

**int width = image.getWidth();**

**int height = image.getHeight();**

**RGBTRIPLE[][] imageArray = new RGBTRIPLE[height][width];**

**for (int i = 0; i < height; i++) {**

**for (int j = 0; j < width; j++) {**

**int rgb = image.getRGB(j, i);**

**imageArray[i][j] = new RGBTRIPLE((byte) ((rgb >> 16) & 0xFF), (byte) ((rgb >> 8) & 0xFF), (byte) (rgb & 0xFF));**

**}**

**}**

**return imageArray;**

**}**

**public static BufferedImage convertToBufferedImage(RGBTRIPLE[][] imageArray, int width, int height) {**

**BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);**

**for (int i = 0; i < height; i++) {**

**for (int j = 0; j < width; j++) {**

**RGBTRIPLE pixel = imageArray[i][j];**

**int rgb = ((pixel.rgbtRed & 0xFF) << 16) | ((pixel.rgbtGreen & 0xFF) << 8) | (pixel.rgbtBlue & 0xFF);**

**image.setRGB(j, i, rgb);**

**}**

**}**

**return image;**

**}**

**}**

**public class RGBTRIPLE {**

**public byte rgbtRed;**

**public byte rgbtGreen;**

**public byte rgbtBlue;**

**public RGBTRIPLE(byte red, byte green, byte blue) {**

**rgbtRed = red;**

**rgbtGreen = green;**

**rgbtBlue = blue;**

**}**

**}**

**public class ImageFilters {**

**public static void grayscale(RGBTRIPLE[][] image) {**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**int avg = (image[i][j].rgbtRed + image[i][j].rgbtBlue + image[i][j].rgbtGreen) / 3;**

**image[i][j].rgbtRed = (byte) avg;**

**image[i][j].rgbtBlue = (byte) avg;**

**image[i][j].rgbtGreen = (byte) avg;**

**}**

**}**

**}**

**public static void sepia(RGBTRIPLE[][] image) {**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**int s\_red = (int) (0.393 \* (image[i][j].rgbtRed & 0xFF) + 0.769 \* (image[i][j].rgbtGreen & 0xFF) + 0.189 \* (image[i][j].rgbtBlue & 0xFF));**

**int s\_green = (int) (0.349 \* (image[i][j].rgbtRed & 0xFF) + 0.686 \* (image[i][j].rgbtGreen & 0xFF) + 0.168 \* (image[i][j].rgbtBlue & 0xFF));**

**int s\_blue = (int) (0.272 \* (image[i][j].rgbtRed & 0xFF) + 0.534 \* (image[i][j].rgbtGreen & 0xFF) + 0.131 \* (image[i][j].rgbtBlue & 0xFF));**

**image[i][j].rgbtRed = (byte) Math.min(s\_red, 255);**

**image[i][j].rgbtGreen = (byte) Math.min(s\_green, 255);**

**image[i][j].rgbtBlue = (byte) Math.min(s\_blue, 255);**

**}**

**}**

**}**

**public static void reflect(RGBTRIPLE[][] image) {**

**int height = image.length;**

**int width = image[0].length;**

**RGBTRIPLE[][] temp = new RGBTRIPLE[height][width];**

**for (int i = 0; i < height; i++) {**

**for (int j = 0; j < width; j++) {**

**temp[i][j] = image[i][j];**

**}**

**}**

**for (int i = 0; i < height; i++) {**

**for (int j = 0; j < width; j++) {**

**image[i][j] = temp[i][width - 1 - j];**

**}**

**}**

**}**

**public static void invertColors(RGBTRIPLE[][] image) {**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**image[i][j].rgbtRed = (byte) (255 - (image[i][j].rgbtRed & 0xFF));**

**image[i][j].rgbtGreen = (byte) (255 - (image[i][j].rgbtGreen & 0xFF));**

**image[i][j].rgbtBlue = (byte) (255 - (image[i][j].rgbtBlue & 0xFF));**

**}**

**}**

**}**

**public static void edgeDetect(RGBTRIPLE[][] image) {**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**int r = 0;**

**int g = 0;**

**int b = 0;**

**for (int k = -1; k <= 1; k++) {**

**for (int l = -1; l <= 1; l++) {**

**if (i + k >= 0 && i + k < image.length && j + l >= 0 && j + l < image[i].length) {**

**r = image[i + k][j + l].rgbtRed - image[i][j].rgbtRed;**

**g = image[i + k][j + l].rgbtGreen - image[i][j].rgbtGreen;**

**b = image[i + k][j + l].rgbtBlue - image[i][j].rgbtBlue;**

**}**

**}**

**}**

**image[i][j].rgbtRed = (byte) (r > 0 ? r : 0);**

**image[i][j].rgbtGreen = (byte) (g > 0 ? g : 0);**

**image[i][j].rgbtBlue = (byte) (b > 0 ? b : 0);**

**}**

**}**

**}**

**public static void pixelate(RGBTRIPLE[][] image) {**

**int pixelSize =10;**

**for (int i = 0; i < image.length - pixelSize; i += pixelSize) {**

**for (int j = 0; j < image[i].length - pixelSize; j += pixelSize) {**

**int r = 0;**

**int g = 0;**

**int b = 0;**

**for (int k = i; k < i + pixelSize; k++) {**

**for (int l = j; l < j + pixelSize; l++) {**

**r += image[k][l].rgbtRed;**

**g += image[k][l].rgbtGreen;**

**b += image[k][l].rgbtBlue;**

**}**

**}**

**r /= pixelSize \* pixelSize;**

**g /= pixelSize \* pixelSize;**

**b /= pixelSize \* pixelSize;**

**for (int k = i; k < i + pixelSize; k++) {**

**for (int l = j; l < j + pixelSize; l++) {**

**image[k][l].rgbtRed = (byte) r;**

**image[k][l].rgbtGreen = (byte) g;**

**image[k][l].rgbtBlue = (byte) b;**

**}**

**}**

**}**

**}**

**}**

**public static void posterize(RGBTRIPLE[][] image) {**

**int levels=40;**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**int r = image[i][j].rgbtRed / levels \* levels;**

**int g = image[i][j].rgbtGreen / levels \* levels;**

**int b = image[i][j].rgbtBlue / levels \* levels;**

**image[i][j].rgbtRed = (byte) r;**

**image[i][j].rgbtGreen = (byte) g;**

**image[i][j].rgbtBlue = (byte) b;**

**}**

**}**

**}**

**public static void noise(RGBTRIPLE[][] image) {**

**for (int i = 0; i < image.length; i++) {**

**for (int j = 0; j < image[i].length; j++) {**

**int r = image[i][j].rgbtRed + (int)(Math.random() \* 255 - 125);**

**int g = image[i][j].rgbtGreen + (int)(Math.random() \* 255 - 125);**

**int b = image[i][j].rgbtBlue + (int)(Math.random() \* 255 - 125);**

**image[i][j].rgbtRed = (byte) (r < 0 ? 0 : (r > 255 ? 255 : r));**

**image[i][j].rgbtGreen = (byte) (g < 0 ? 0 : (g > 255 ? 255 : g));**

**image[i][j].rgbtBlue = (byte) (b < 0 ? 0 : (b > 255 ? 255 : b));**

**}**

**}**

**}**

**public static void blur(RGBTRIPLE[][] image) {**

**int height = image.length;**

**int width = image[0].length;**

**RGBTRIPLE[][] copy = new RGBTRIPLE[height][width];**

**for (int i = 0; i < height; i++) {**

**for (int j = 0; j < width; j++) {**

**copy[i][j] = image[i][j];**

**}**

**}**

**+) {**

**int sumr = 0, sumg = 0, sumb = 0;**

**float c = 0.0f;**

**for (int m = (i - 1); m < (i + 2); m++) {**

**for (int k = (j - 1); k < (j + 2); k++) {**

**if (m < 0 || k < 0 || m >= height || k >= width) {**

**continue;**

**}**

**sumr += (copy[m][k].rgbtRed & 0xFF);**

**sumg += (copy[m][k].rgbtGreen & 0xFF);**

**sumb += (copy[m][k].rgbtBlue & 0xFF);**

**c += 1.0f;**

**}**

**}**

**int s\_red = Math.round(sumr / c);**

**int s\_green = Math.round(sumg / c);**

**int s\_blue = Math.round(sumb / c);**

**image[i][j].rgbtRed = (byte) s\_red;**

**image[i][j].rgbtGreen = (byte) s\_green;**

**image[i][j].rgbtBlue = (byte) s\_blue;**

**}**

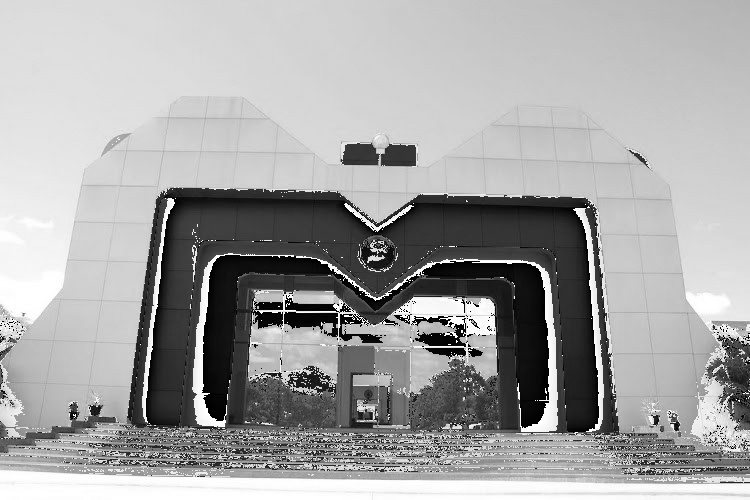
**}**

**}**

**}**

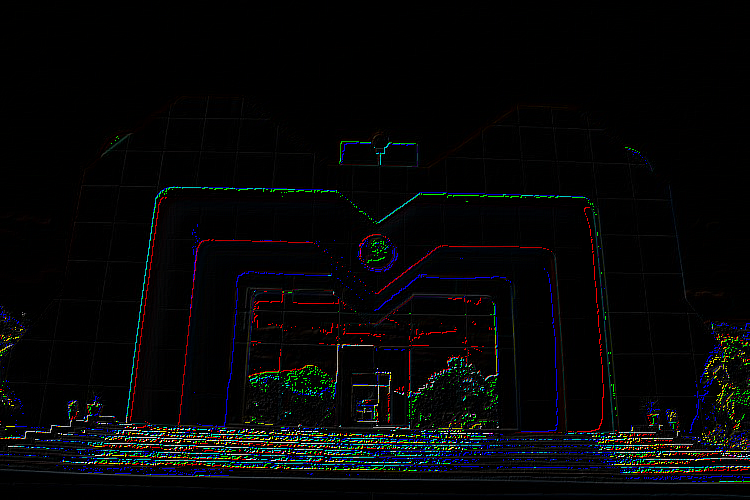
**PROGRAM OUTPUT**

**1.GRAYSCALE 2.SEPIA**



**3.REFLECT 4.INVERTCOLORS**

****

**5.EDGEDETECT 6.PIXELATE**

****

**7.POSTERIZE 8.NOISE**

****

**9.BLUR**

****

**CONCLUSION**

The "ImageProcessor" Java program provides a comprehensive set of image processing capabilities, allowing users to apply a range of filters and manipulations to their input images. By operating via a command-line interface, it offers ease of use and automation for batch image processing tasks.The program's architecture divides the core functionality into the "ImageFilters" class, making it modular and extensible. Each filter method within this class can be further enhanced or modified to accommodate specific image processing requirements. By utilizing the Java ImageIO library, the program ensures compatibility with BMP image formats.Through the provided system requirements, users can effectively run the program on various operating systems with a modern Java Development Kit. The flexibility to run it from the command line or within a Java IDE offers adaptability to different user preferences and workflows.

In summary, the "ImageProcessor" program is a valuable tool for image enthusiasts, designers, and developers who need a straightforward and customizable solution for image manipulation. Its comprehensive set of filters, together with its user-friendly command-line interface, empowers users to enhance, stylize, and experiment with images, making it a valuable asset in the realm of image processing.