Lab 2: Image Enhancement in the Spatial Domain

Kamala Rai Danuwar

13 [20070291]

Bachelors of Engineering in Computer

Everest Enginee0ring College

16/02/2024

Summary

In this lab we learnt about basic grayscale image-enhancement methods in the Spatial Domain.

I solved different problems comprising of log-transformation, histogram equalization, convolution and Laplacian, averaging and Sobel’s masks.

The questions I solved are as follows:

1) implementation of log-transform in an image.
2) implementation of algorithm of histogram equalization .
3) finding Laplacian of the image and enhancing image using Laplacian using convolution operation.
4) using averaging to blur the image.
5) finding the image gradient using Sobel’s masks.

I. PROBLEM DESCRIPTION

This practical is entirely focused on grayscale image processing and basic grayscale image-enhancement methods which uses simple transformations of the form s=t(r) where r and s are pixel intensities of the original and transformed images respectively. We used different concepts and algorithms of equalization, correlation and convolution, spatial filter to solve the questions of this practical. In this practical I solved 5 problems: 1) implementation of log-transform 2) implementation of histogram equalization 3) finding Laplacian of the image and enhancing imaging using Laplacian 4) using averaging to blur the image 5) finding the image gradient using Sobel's masks.

II. THEORETICAL BACKGROUND

Log-transform is defined as s = logr.

Another transformation of importance is histogram equalization. Histogram equalization is achieved by the following transformation function.

$$s_k = (L - 1) \sum_{j=0}^{k} Pj$$

Similarly other important transformations are correlation and convolution in the spatial domain and they can be defined as follows:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \quad \text{(Correlation)}$$

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \quad \text{(Convolution)}$$

The important methods used in this practical are:

- logTransform(int[][] f): this method first finds the minimum and maximum pixel of the image and applies the logarithmic function ‘`F[x][y] = (int) Math.log(f[x][y] + 1)`’ and normalizes the transformed image ‘F’ to the range [0,255] and further scales the pixel values to ensure that they fall in the range [0,255] and the transformed image ‘F’ is returned.
- calculateHistogran(int[][] image) : this method calculates the histogram of the input image.
- applyHistogramEqualization(int[][] image, int[] mapping) : this method applies histogram equalization to the input image using the mapping function generated by ‘calculateMapping’.
- convolution(int[][] f) : this method performs convolution operation on an input image ‘f’ using the specified kernel(kernel for the Laplacian) over each pixel of the input image. Then it is normalized and scaled to the range [0,255].
- averaging(int[][] f, int filterSize): filtersize is input from the user using the ‘getFilterSizeFromUser()’ method. Then this method applies an averaging filter to the input image ‘f’ using that filter size.
- imageGradient(int[][] f) : this method computes the gradient magnitude of the input image ‘f’ using the Sobel’s masks. The computed gradient magnitude is checked to ensure that it falls withing the range[0,255] and final image ‘F’ is returned.

III. RESULTS AND LESSON LEARNT

To test my program I compared the original image and the converted image side by sides and then concluded using the methods theoretical significances.

I learnt that the pixels may go out of the color range [0,255] and need to be rescaled to ensure that the pixels of the transformed image falls in the range [0,255].

PROGRAM LISTING

Function for implementing log transform

```
public static int[][] logTransform (int[][]f) {
            int [][] F= new int [f.length][f[0].length];
            int min = f[0][0];
            int max = f[0][0];

            for(int x=0;x<f.length;x++) {
                for(int y=0;y<f[0].length;y++) {

                    F[x][y] = (int) Math.log(f[x][y] +1);

                    min = Math.min(min, F[x][y]);
                    max = Math.max(max, F[x][y]);
                }
            }



            for(int x=0;x<f.length;x++) {
                for(int y=0;y<f[0].length;y++) {

                    F[x][y] = (int)(((F[x][y] - min)/ (double) ( max -
min)) * 255); //normalization of transformed image in [0,255]
                }
            }


            double scale = 255.0 / Math.log(1 + max); // Adjust the
scaling factor
            for (int x = 0; x < f.length; x++) {
                for (int y = 0; y < f[0].length; y++) {
                    F[x][y] = (int) (scale * F[x][y]);
                }
            }
            System.out.println("Minimum Pixel Value (After Log
Transformation): " + min);
            System.out.println("Maximum Pixel Value (After Log
Transformation): " + max);

            return F;
        }
```

Function for implementing histogram equalization

```
// Calculate histogram
public static int[] calculateHistogram(int[][] image) {
    int[] histogram = new int[256];
    for (int x = 0; x < image.length; x++) {
        for (int y = 0; y < image[0].length; y++) {
            histogram[image[x][y]]++;
        }
    }
    return histogram;
}

// Calculate cumulative distribution function (CDF)
public static int[] calculateCumulativeHistogram(int[] histogram) {
    int[] cumulativeHistogram = new int[256];
    cumulativeHistogram[0] = histogram[0];
```

```java
    for (int i = 1; i < 256; i++) {
        cumulativeHistogram[i] = cumulativeHistogram[i - 1] +
histogram[i];
    }
    return cumulativeHistogram;
}

// Calculate mapping function for histogram equalization
public static int[] calculateMapping(int[] cumulativeHistogram, int
totalPixels) {
    int[] mapping = new int[256];
    for (int i = 0; i < 256; i++) {
        mapping[i] = (int) (255.0 * cumulativeHistogram[i] /
totalPixels);
    }
    return mapping;
}

// Apply histogram equalization to the image
public static int[][] applyHistogramEqualization(int[][] image, int[]
mapping) {
    int[][] equalizedImage = new int[image.length][image[0].length];
    for (int x = 0; x < image.length; x++) {
        for (int y = 0; y < image[0].length; y++) {
            equalizedImage[x][y] = mapping[image[x][y]];
        }
    }
    return equalizedImage;
}
```

Function for finding the Laplacian of the image

////convolution

```
        public static int [][] convolution (int [][]f) {
            int [][] F = new int [f.length][f[0].length];

            //int [][] w = {{0,1,0},{1,-4,1},{0,1,0}};

            int [][] w = {{0,1,0},{1,-4,1},{0,1,0}};  //kernel for
laplacian

            int a = (w.length - 1)/2;
            int b = (w[0].length - 1)/2;

            int[][] f_padded = new int[2*a+f.length][2*b+f[0].length];

            for(int x=0;x<f.length;x++)
                for(int y=0;y<f[0].length;y++)
                    f_padded[a+x][b+y] = f[x][y];

            for (int x=0; x<f.length ; x++)
                for (int y=0; y<f[0].length; y++)
                    for (int s=-a;s<=a;s++)
                    for(int t=-b;t<b;t++){
                        int v = w[s+a][t+b];
                        F[x][y] = F[x][y] + v*f_padded[(a+x)-s][(b+y)-
t];
                    }

            //re-scaling
            int min = f[0][0];
            int max = f[0][0];

            for(int x=0;x<f.length;x++) {
                for(int y=0;y<f[0].length;y++) {

                    F[x][y] = (int) Math.log(f[x][y] +1);

                    min = Math.min(min, F[x][y]);
                    max = Math.max(max, F[x][y]);
                }
            }

            for(int x=0;x<f.length;x++) {
                for(int y=0;y<f[0].length;y++) {

                    F[x][y] = (int)(((F[x][y] - min)/ (double) ( max - 
min)) * 255);
                }
            }

            double scale = 255.0 / Math.log(1 + max); // Adjust the 
scaling factor
            for (int x = 0; x < f.length; x++) {
                for (int y = 0; y < f[0].length; y++) {
                    F[x][y] = (int) (scale * F[x][y]);
                }
            }

            return F;
```

```
        }
```

Function for bluring the image using averaging

```
        //function to get filter size from user
        public static int getFilterSizeFromUser() {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter the filter size for blurring (an 
integer): ");
            return scanner.nextInt();
        }

        //averaging filter
        public static int [][] averaging (int [][]f, int filterSize){
            int [][] F = new int [f.length][f[0].length];

            int filterArea = filterSize * filterSize;

            for (int x=0; x<f.length; x++) {
                for (int y=0; y<f[0].length; y++) {
                    int sum =0;
                    for (int i= -filterSize/2; i<=filterSize/2; i++) {
                        for (int j = -filterSize/2; i<=filterSize/2; 
i++) {
                            int newX = Math.max(0, Math.min(f.length - 
1, x+i));
                            int newY = Math.max(0, 
Math.min(f[0].length - 1, y+j));
                            sum += f[newX][newY];
                        }
                    }
                    F[x][y] = sum / filterArea;
                }
            }
            return F;
        }
```

image Gradient using Sobel's masks

```
    public static int [][] imageGradient(int [][] f) {

        int [][] F = new int [f.length][f[0].length];

        int[][] sobelX = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
        int[][] sobelY = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

        for (int x = 1; x < f.length - 1; x++) {
            for (int y = 1; y < f[0].length - 1; y++) {
                int gradientX = 0;
                int gradientY = 0;

                for (int i = 0; i < 3; i++) {
                    for (int j = 0; j < 3; j++) {
                        gradientX += sobelX[i][j] * f[x - 1 + i][y
- 1 + j];
                        gradientY += sobelY[i][j] * f[x - 1 + i][y
- 1 + j];
                    }
                }
```

```
                    // Calculate gradient magnitude
                    F[x][y] = (int) Math.sqrt(gradientX * gradientX +
gradientY * gradientY);

                    // Clamp values to be in the valid color range
                    F[x][y] = Math.min(255, Math.max(0, F[x][y]));
                }
            }
            return F;
        }
```

main

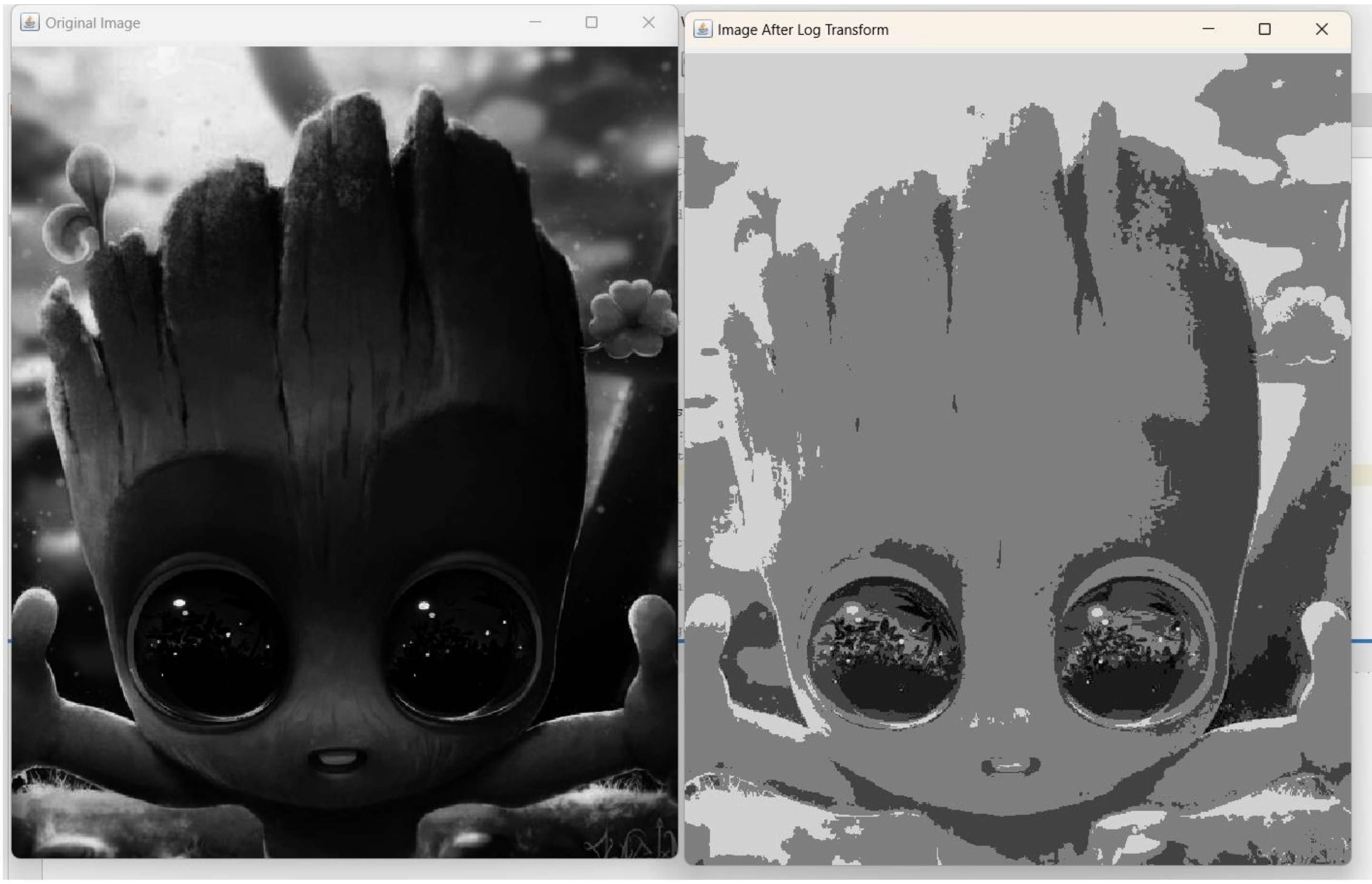
```
public static void main(String[] args) {

            Image img = new Image("C:\\Users\\Kamala Rai
Danuwar\\Pictures\\Iam grut.jpg");
            int[][] f = img.getPixelArray();

// log transform
           /*
            int [][] F = logTransform(f);
            Image.display(f, "Original Image");
            Image.display(F, "Image After Log Transform");
*/
```

```
           //histogram equalization'


 /*
            int[] histogram = calculateHistogram(f);
            int[] cumulativeHistogram =
calculateCumulativeHistogram(histogram);
            int[] mapping = calculateMapping(cumulativeHistogram,
f.length * f[0].length);


            int[][] equalizedImage = applyHistogramEqualization(f,
mapping);


            Image.display(f, "Original Image");
            Image.display(equalizedImage, "Image After Histogram
Equalization");
*/



           //convolution and correlation


           /*
            int [][] F = convolution(f);


            Image.display(f, "Original Image");
            Image.display(F, "Laplacian of Image");


*/
```

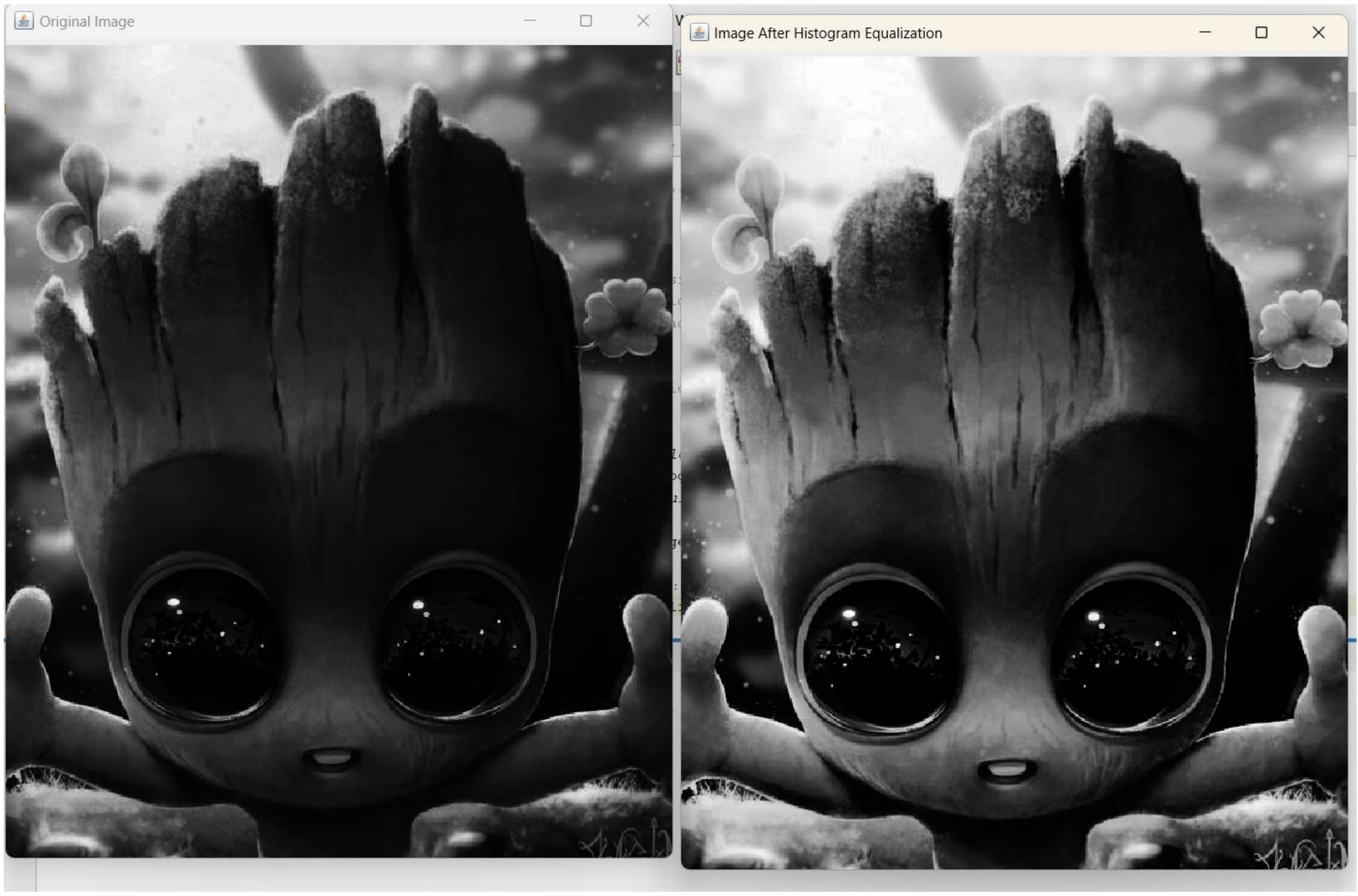
```
        //blurring
        /*
        int filterSize = img.getFilterSizeFromUser();
        int [][] F = averaging(f, filterSize);
        Image.display(f,"Original Image");
        Image.display(F,"Blurred Image");
         */




        //image gradient using Sobel's mask
        /*
        int [][] F = imageGradient(f);
        Image.display(f, "Original Image");
        Image.display(F, "Gradient Image");
        */


}
```

OUTPUTS

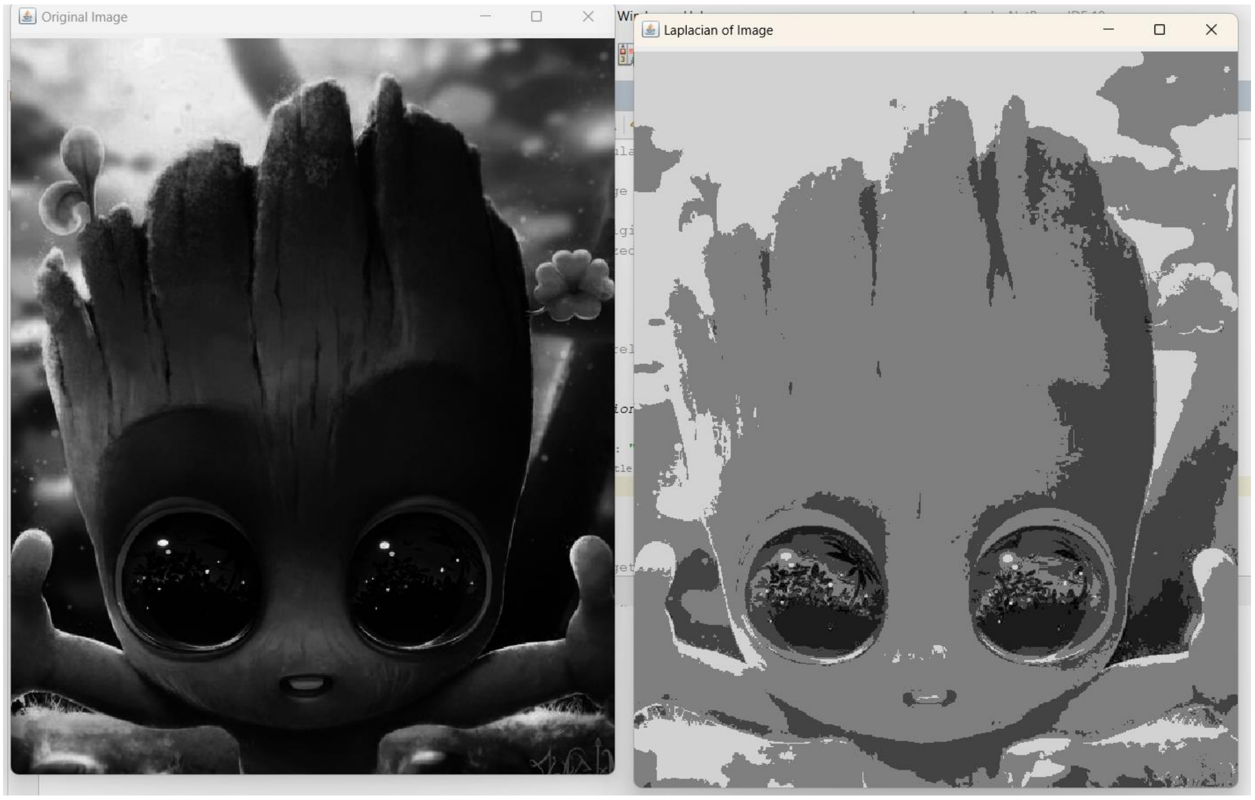
1) Log transformation on the image



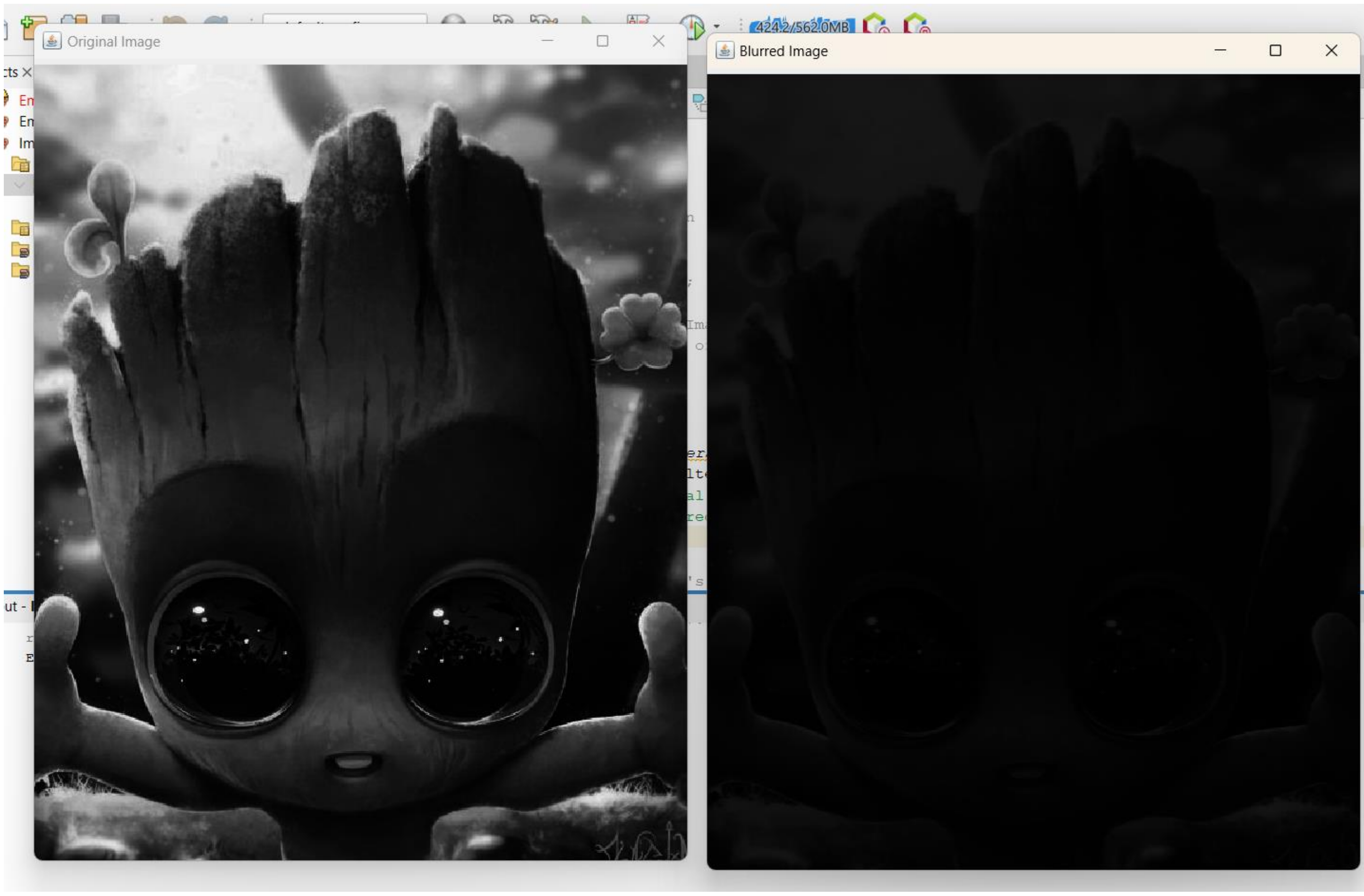
2) Histogram equalization



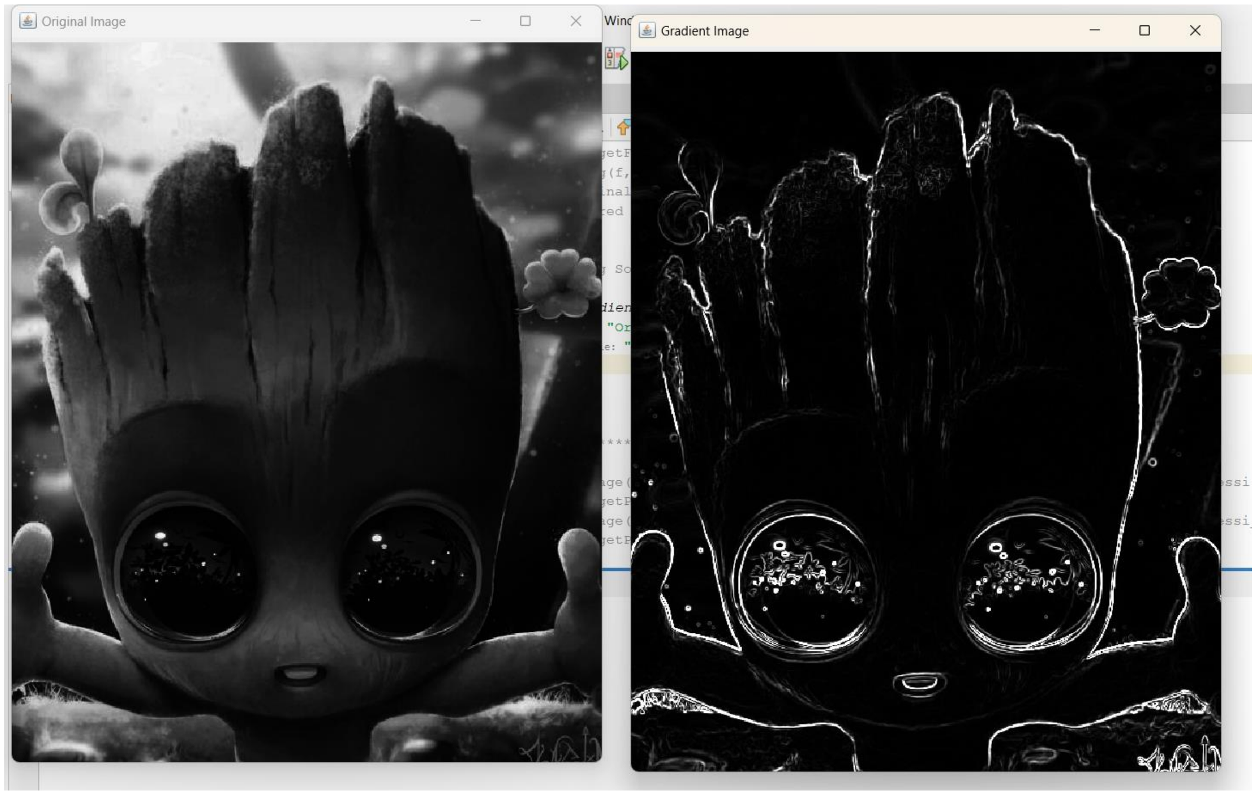
3) Laplacian



4) averaging



5) image gradient using Sobel’s masks


Original Image
Gradient Image