Image Editor

Name: Rudra Chauhan

Date: 02/09/2023

Roll No: 23BCS10137

Contents |

1. Introduction
2. Printing Pixel Value
3. Grayscale Function
4. Brightness Function
5. Fliping Function
6. Blur Function
7. Image Rotation
8. Main Function

Introduction

I created this cool Image Editor project using the Buffered Image class and some 2D array concepts. It's a program that you can use in your terminal and it offers a bunch of functions to edit your images. The functions are as follows -

Functions

1. **Grayscale Function**: The Grayscale function allows users to convert color images into grayscale, effectively removing color information and producing monochromatic versions of the original image.
2. **Brightness Function**: Users can adjust the brightness of their images with this feature. By increasing or decreasing the brightness, images can be made either brighter or darker.
3. **Horizontal Flip:** This function enables users to horizontally flip an image, creating a mirror image of the original.
4. **Vertical Flip**: Similar to horizontal flipping, the Vertical Flip function allows users to create a vertically flipped version of their images.
5. **Blur Function**: The Blur function consists of two types of blur - Mossiac Blur and Gaussian Blur.
6. **Image Rotation**: Users can rotate their images both clockwise and anti-clockwise.

Print Pixel Value

```java
// This function prints the value of all the pixels in the terminal
public static void printpixel(BufferedImage inputImage)
{
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            Color pixel = new Color(inputImage.getRGB(j, i));
            System.out.print(" ("+pixel.getRed()+" "+pixel.getGreen()+" "+pixel.getBlue()+")");
        }
        System.out.println();
    }
}
```

This function essentially processes each pixel in a given image (represented as a `BufferedImage`), extracts its RGB values using the Color class, and prints them to the console in a structured format, providing a visual representation of the image's color content. Here is an example.

TERMINAL

```
(207 132 49) (206 131 48) (206 131 48) (205 130 47)
(204 129 46) (204 129 46) (204 129 46) (204 129 46)
(209 134 51) (212 137 54) (214 139 56) (217 143 58)
(233 163 75) (231 164 75) (231 164 75) (231 164 75)
(233 168 78) (231 167 77) (231 167 77) (232 168 78)
(235 171 81) (235 171 81) (235 171 81) (235 171 81)
(231 167 77) (230 166 76) (230 166 76) (232 166 79)
(231 165 78) (230 164 77) (230 164 77) (230 164 77)
(232 166 80) (232 166 80) (233 167 81) (232 166 80)
(225 155 69) (224 154 69) (224 154 69) (222 154 69)
(222 151 69) (223 151 69) (223 151 69) (223 151 69)
(212 136 61) (212 136 61) (211 135 60) (210 134 59)
(209 131 56) (220 142 67) (231 155 77) (240 165 84)
```

Grayscale

```java
//This function is going to convert the given inmage into a greyscale image
public static BufferedImage convertToGrayscale(BufferedImage inputImage)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_BYTE_GRAY);
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            outputImage.setRGB(j,i,inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

This function converts a color image into a grayscale image by copying the RGB values of each pixel from the input image to a new image with a grayscale color model. The result is a monochromatic image where each pixel represents a shade of gray, with black for the darkest shades and white for the lightest shades. Below is an example.

Brightness

```java
//Function to increase/decrease the brightness of an image by a provided value
public static BufferedImage changeBrightness(BufferedImage inputImage, int increase)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            Color pixel = new Color(inputImage.getRGB(j, i));
            int red = pixel.getRed(), green = pixel.getGreen(), blue = pixel.getBlue();
            red = red+((red*increase)/100);
            green = green+((green*increase)/100);
            blue = blue+((blue*increase)/100);
            if(red>255)
                red = 255;
            if(green>255)
                green = 255;
            if(blue>255)
                blue = 255;
            if(red<0)
                red = 0;
            if(red<0)
                green = 0;
            if(red<0)
                blue = 0;
            Color newPixel = new Color(red, green, blue);
            outputImage.setRGB(j, i, newPixel.getRGB());
        }
    }
    return outputImage;
}
```

The ‘`changeBrightness`’ function is designed to modify the brightness of an input image. It takes two parameters: ‘`inputImage`’, which is the original image to be adjusted, and ‘`increase`’, an integer that represents the percentage by which the brightness should be changed.

In this function, we iterate over each pixel of the image. For each pixel at position `(j, i)` in the input image, it retrieves the color information using the ‘`Color`’ class which allows us to access the individual red, green, and blue components of the pixel.

It changes the brightness by increasing or decreasing each component based on the `increase` parameter. For example, ‘`red = red + ((red * increase) / 100);`’ increases the red component by a percentage specified by ‘`increase`’. If ‘`increase`’ is positive, it brightens the pixel; if negative, it darkens the pixel. To ensure that the adjusted RGB values stay within the valid range of 0 to 255. Below is an example of a brightened image with '`increase = 50`'

Fliping Function

```java
//This function inverts the image horizontally
public static BufferedImage invertHorizontal(BufferedImage inputImage)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            outputImage.setRGB((width-1)-j, i, inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

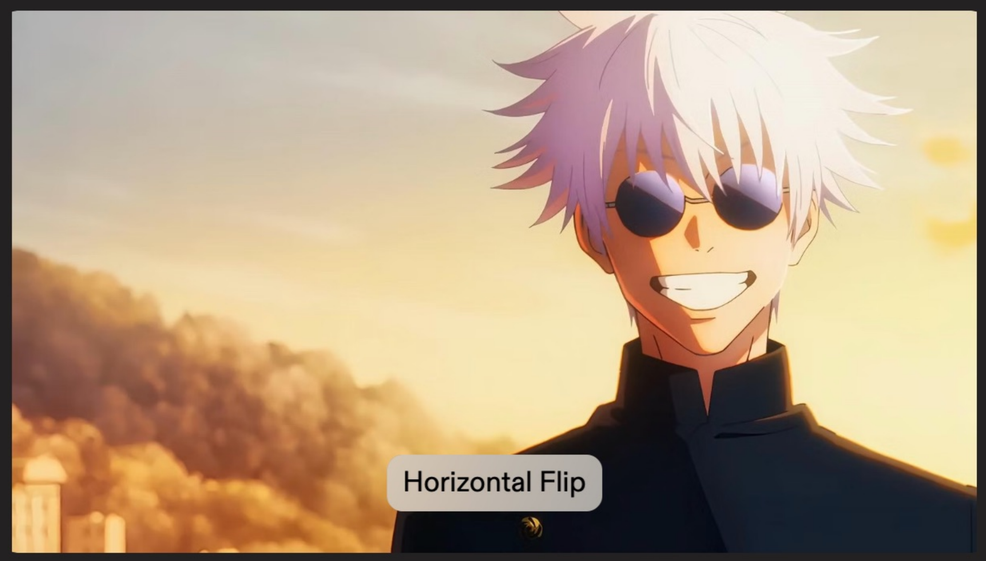
```java
//This function inverts the image horizontally
public static BufferedImage invertvertical(BufferedImage inputImage)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedIma
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            outputImage.setRGB(j, (height-1)-i, inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

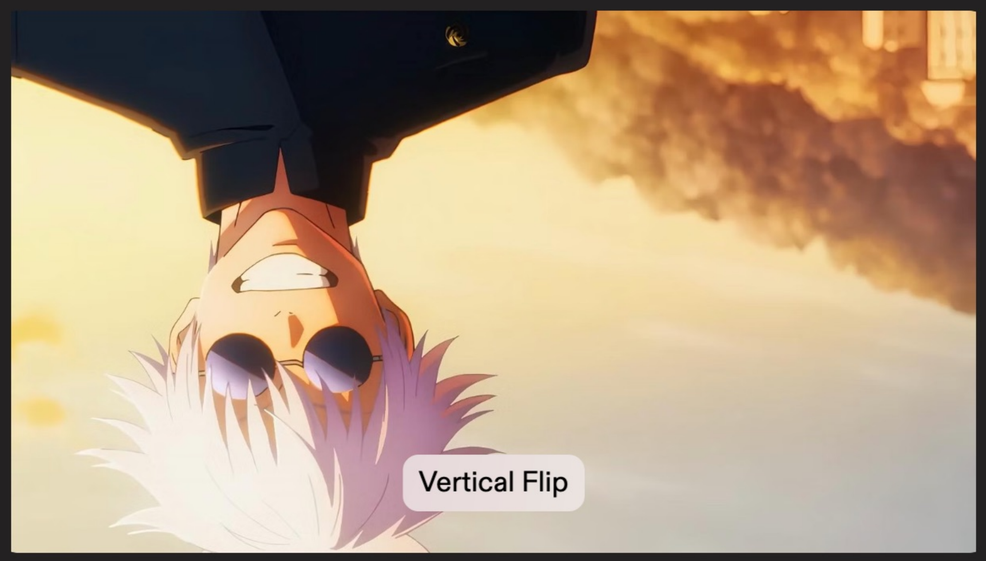
1. invertHorizontal:

`'invertHorizontal'` flips the input image horizontally, creating a mirror image. It iterates through each pixel in the input image and copies it to the output image with its horizontal position inverted. The result is an image that appears as if it has been flipped left to right.

2. invertVertical:

`‘invertVertical’` flips the input image vertically, turning it upside down. Similar to `‘invertHorizontal’`, it iterates through each pixel but copies it to the output image with its vertical position inverted. The result is an image that appears as if it has been flipped from top to bottom.





Blur

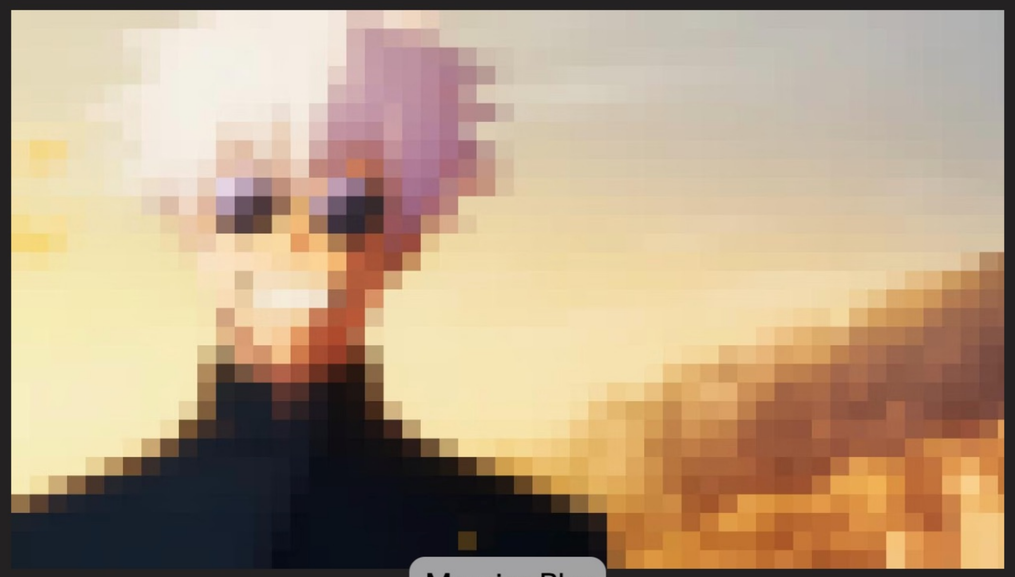
```java
//This function blurs the image
public static BufferedImage blurImage(BufferedImage inputImage, int strength)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage((width),(height),BufferedImag
    for(int i=0; i<height;i+=strength)
    {
        for(int j=0;j<width;j+=strength)
        {
            int red=0, green=0,blue=0,count=0;
            for(int m=i;m<(i+strength) && (m<height);m++)
            {
                for(int n=j;n<(j+strength) && (n<width);n++)
                {
                    Color pixel = new Color(inputImage.getRGB(n, m));
                    red = red + pixel.getRed();
                    green = green + pixel.getGreen();
                    blue = blue + pixel.getBlue();
                    count++;
                }
            }
            red = red/count;
            green = green/count;
            blue = blue/count;
            Color new_pixel = new Color(red, green, blue);
            for(int m=i;m<(i+strength) && (m<height);m++)
            {
                for(int n=j;n<(j+strength) && (n<width);n++)
                {
                    outputImage.setRGB(n, m, new_pixel.getRGB());
                }
            }
        }
    }
    return outputImage;
```

The ‘`blurImage`’ function takes an input image and a '`strength'` parameter as arguments.

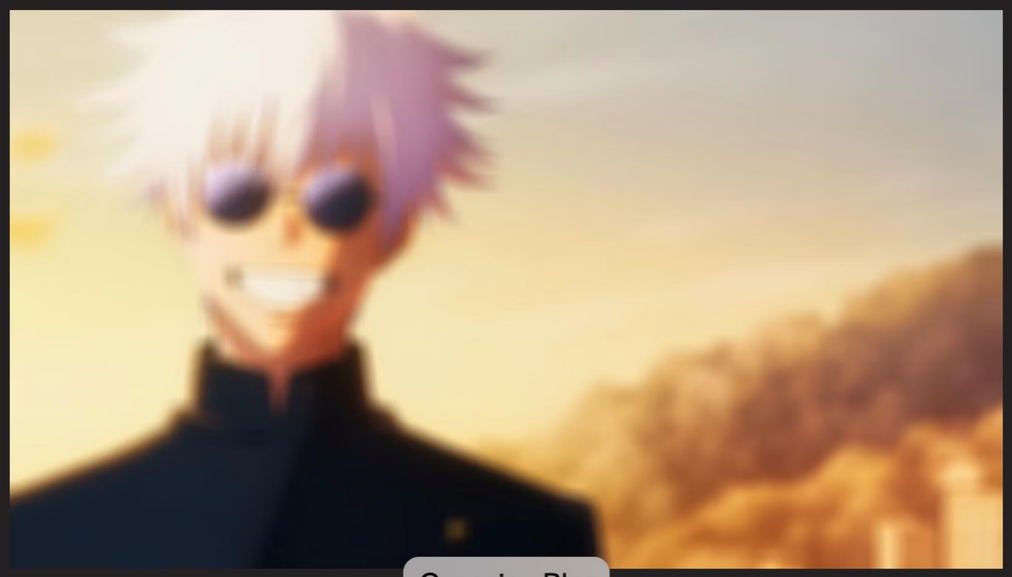
The function iterates through the pixels of the input image in blocks of size determined by the ‘`strength`’.

Within each block, it calculates the average RGB values of the pixels and creates a new pixel with those averaged values.

It then sets all the pixels within the block to the averaged color, effectively blurring the image. This results in a Mossiac blur or a Pixalated blur. For Gaussian Blur we have to fill the average color to that particular pixel only and do this for all the pixels.



Mossiac Blur



Gaussian Blur

Image Rotation

```java
//This function makes a tranpose image
public static BufferedImage transposeImg(BufferedImage inputImage)
{
    int width = inputImage.getWidth();
    int height = inputImage.getHeight();
    BufferedImage outputImage = new BufferedImage(height, width, BufferedImage.TYPE_3BYTE_BGR);
    for(int i=0;i<height;i++)
    {
        for(int j=0;j<width;j++)
        {
            outputImage.setRGB(i, j, inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

The `transposeImg` function takes an input image and returns its transpose. It swaps the width and height dimensions to create a new `BufferedImage`. Then, it iterates through each pixel of the `inputImage`, copying the pixel's color to the `outputImage` with swapped coordinates. The result is an image where rows become columns and vice versa.

Then for Clockwise rotation, we simply flip it horizontally using `'invertHorizontal'` function. And for Anticlockwise rotation, we have to flip it vertically using `‘invertVertical’` function.

Main function

This `main` function allows users to perform a variety of image editing tasks interactively, providing a user-friendly interface to apply these operations to an input image and save the results as separate image files.

It begins by reading an input image into a BufferedImage called `'inputImage'`.

It presents a menu of image editing options to the user, allowing them to choose from various operations. Then by using switch-case, depending on the user's choice, the program applies the selected image editing operation to the input image and saves the resulting image with an appropriate filename. Below is an example:

```
        ********IMAGE EDITOR********
Press 1 : To create a greyscale image
Press 2 : Change the brightness of the image
Press 3 : To Invert the image horizontally
Press 4 : To Invert the image vertically
Press 5 : Mossiac Blur
Press 6 : Gaussian Blur
Press 7 : Transpose the image
Press 8 : To rotate the image 90 degree clockwise
Press 9 : To rotate the image 90 degree anticlockwise
Press 10 : Exit

Enter the file name or path: image.jpeg

Enter your choice: 1

The image was succesfully created.
```

```java
case 1:
    BufferedImage grayscale = convertToGrayscale(inputImage);
    File grayscaleImage = new File(pathname:"grayscaleImage.jpeg");
    ImageIO.write(grayscale, formatName:"jpeg", grayscaleImage);
    break;
```

Thank you

rudrachauhan2888@gmail.com
rudrachauhan