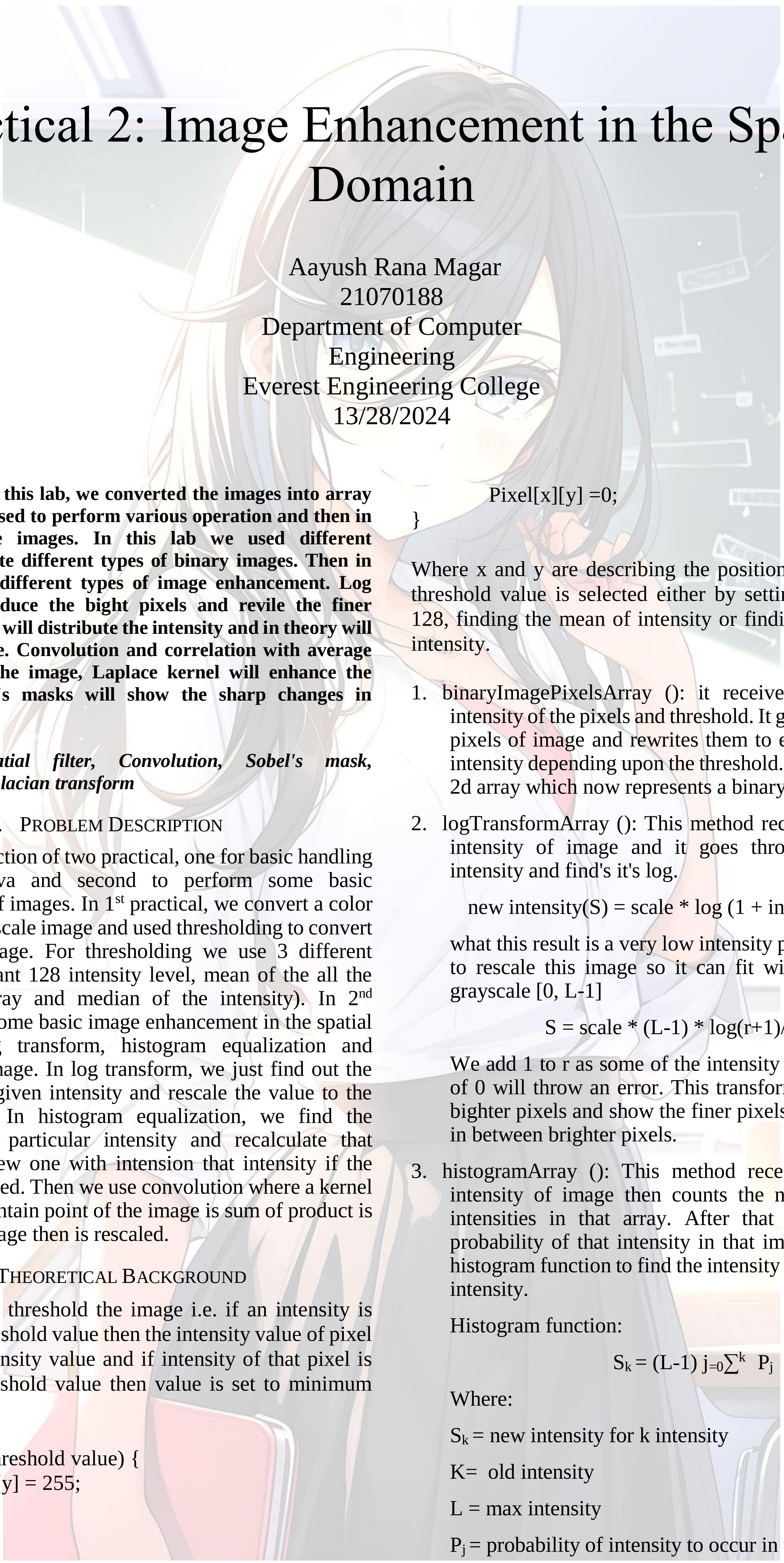


Practical 2: Image Enhancement in the Spatial Domain

Aayush Rana Magar
21070188
Department of Computer Engineering
Everest Engineering College
13/28/2024

***Summary*— In this lab, we converted the images into array and arrays were used to perform various operation and then in result manipulate images. In this lab we used different thresholds to create different types of binary images. Then in 2nd lab we used different types of image enhancement. Log transform will reduce the bight pixels and revile the finer details. Histogram will distribute the intensity and in theory will enhance the image. Convolution and correlation with average kernel will blur the image, Laplace kernel will enhance the image and Sobel's masks will show the sharp changes in intensities.**

Keywords—Spatial filter, Convolution, Sobel's mask, Enhancement, Laplacian transform

I. Problem Description

This lab is a collection of two practical, one for basic handling of image in java and second to perform some basic transformations of images. In 1st practical, we convert a color image into a grayscale image and used thresholding to convert into a binary image. For thresholding we use 3 different thresholds (constant 128 intensity level, mean of the all the pixels of the array and median of the intensity). In 2nd practical, we do some basic image enhancement in the spatial domain like log transform, histogram equalization and convolution of image. In log transform, we just find out the log value of the given intensity and rescale the value to the grayscale value. In histogram equalization, we find the probability of a particular intensity and recalculate that intensities to a new one with intension that intensity if the image be distributed. Then we use convolution where a kernel is selected and contain point of the image is sum of product is calculated and image then is rescaled.

II. Theoretical Background

In 1st lab we just threshold the image i.e. if an intensity is greater than a threshold value then the intensity value of pixel is set to max intensity value and if intensity of that pixel is less than the threshold value then value is set to minimum intensity value.

```
If(pixel[x][y] > threshold value) {
        Pixel[x][y] = 255;
}
Else {
        Pixel[x][y] =0;
}
```

Where x and y are describing the position of the pixel and threshold value is selected either by setting a constant i.e. 128, finding the mean of intensity or finding the median of intensity.

1. binaryImagePixelsArray (): it receives a 2d array of intensity of the pixels and threshold. It goes through every pixels of image and rewrites them to either max or min intensity depending upon the threshold. It then returns the 2d array which now represents a binary image.
2. logTransformArray (): This method receives 2d array of intensity of image and it goes through each pixel's intensity and find's it's log.

   new intensity(S) = scale * log (1 + input intensity(r))

   what this result is a very low intensity picture so we have to rescale this image so it can fit with in the rage of grayscale [0, L-1]

   $$S = scale * (L-1) * \log(r+1)/\log(L)$$

   We add 1 to r as some of the intensity may be 0 and log of 0 will throw an error. This transform will reduce the bighter pixels and show the finer pixels that were hidden in between brighter pixels.
3. histogramArray (): This method receives 2d array of intensity of image then counts the number of certain intensities in that array. After that it calculated the probability of that intensity in that image. Then it uses histogram function to find the intensity to replace that old intensity.

   Histogram function:

   $$S_k = (L-1) \sum_{j=0}^{k} P_j$$

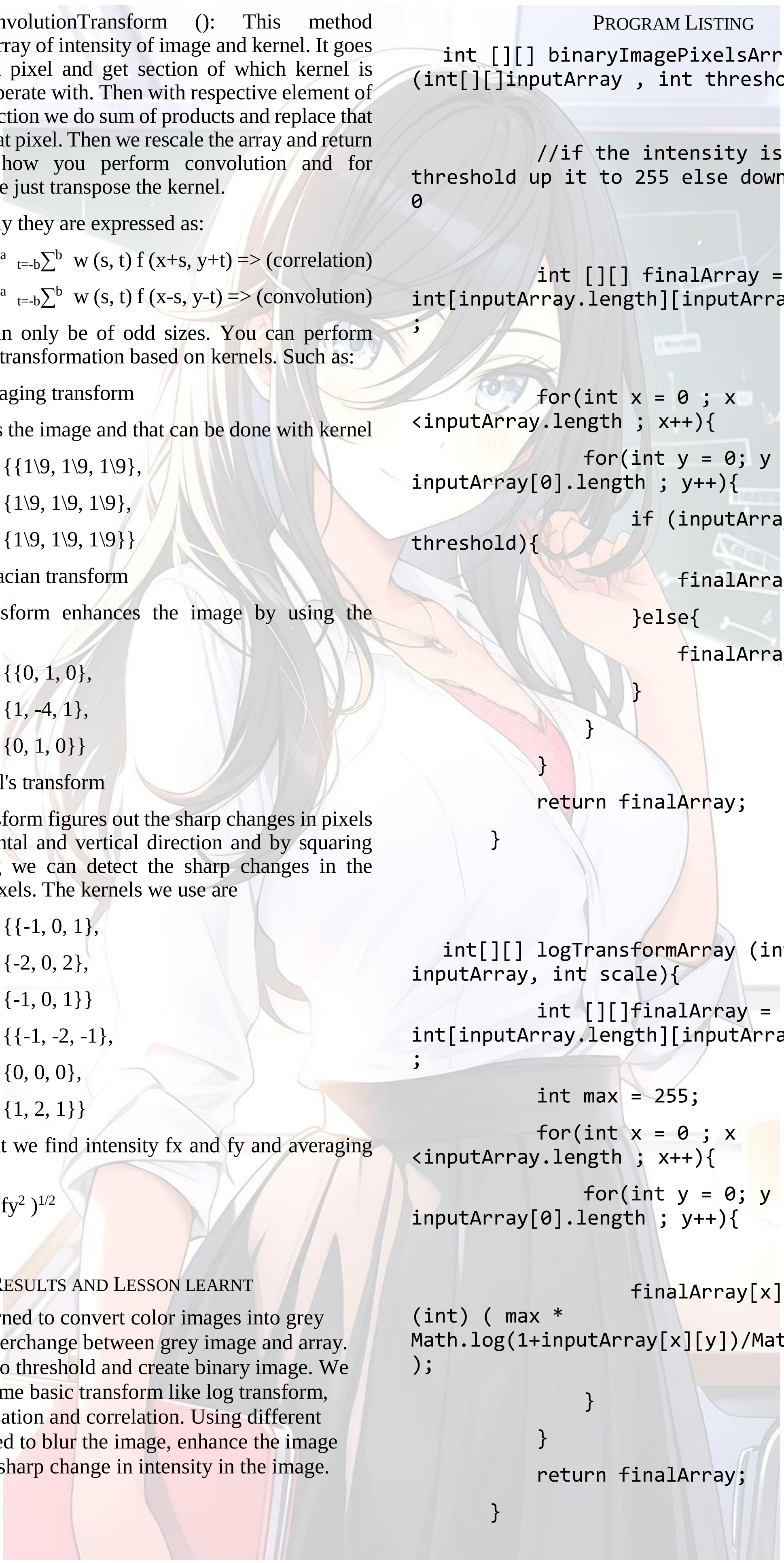
   Where:

   S_k = new intensity for k intensity

   K= old intensity

   L = max intensity

   P_j = probability of intensity to occur in that image



4. correlationConvolutionTransform (): This method receives 2d array of intensity of image and kernel. It goes through each pixel and get section of which kernel is required to operate with. Then with respective element of kernel and section we do sum of products and replace that value with that pixel. Then we rescale the array and return it. That is how you perform convolution and for correlation we just transpose the kernel.

Mathematically they are expressed as:

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

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

The kernel can only be of odd sizes. You can perform different types of transformation based on kernels. Such as:

a) Averaging transform

This blurs the image and that can be done with kernel

K = {{1\9, 1\9, 1\9},
{1\9, 1\9, 1\9},
{1\9, 1\9, 1\9}}

b) Laplacian transform

This transform enhances the image by using the kernel

K = {{0, 1, 0},
{1, -4, 1},
{0, 1, 0}}

c) Sobel's transform

This transform figures out the sharp changes in pixels in horizontal and vertical direction and by squaring averaging we can detect the sharp changes in the nearby pixels. The kernels we use are

Kx = {{-1, 0, 1},
{-2, 0, 2},
{-1, 0, 1}}

Ky = {{-1, -2, -1},
{0, 0, 0},
{1, 2, 1}}

Using that we find intensity fx and fy and averaging them by:

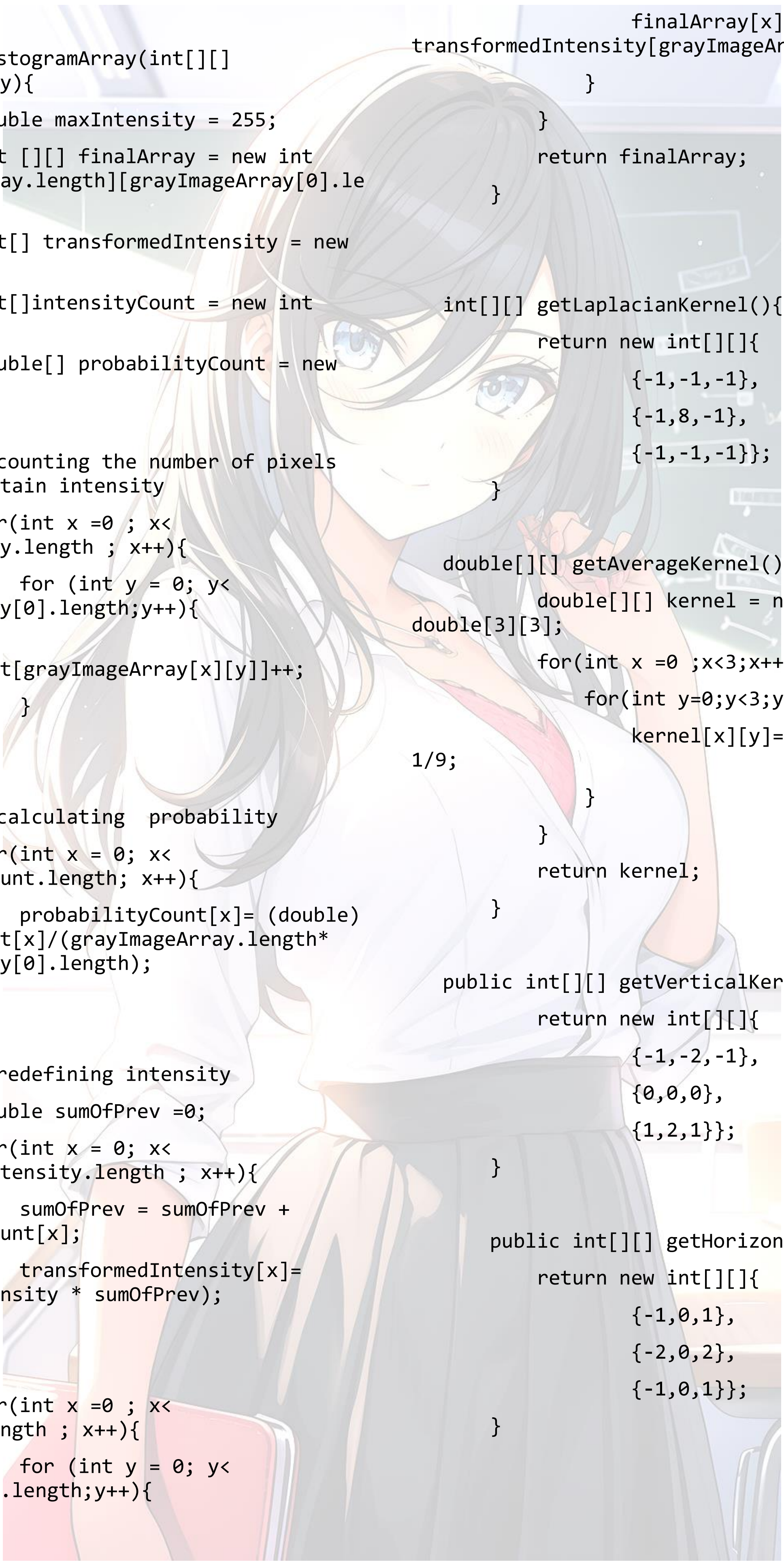
$F = (fx^2 + fy^2)^{1/2}$

III. Results and Lesson learnt

In this lab we learned to convert color images into grey image and can interchange between grey image and array. We learned how to threshold and create binary image. We also how to do some basic transform like log transform, histogram equalization and correlation. Using different kernels, we learned to blur the image, enhance the image and detected any sharp change in intensity in the image.

Program Listing

```
   int [][] binaryImagePixelsArray
(int[][]inputArray , int threshold){


           //if the intensity is greater than
threshold up it to 255 else downgrade it to
0


           int [][] finalArray = new
int[inputArray.length][inputArray[0].length]
;


           for(int x = 0 ; x
<inputArray.length ; x++){

               for(int y = 0; y <
inputArray[0].length ; y++){
                   if (inputArray[x][y] >
threshold){

                       finalArray[x][y] =255;

                   }else{

                       finalArray[x][y] = 0;

                   }

               }

           }

           return finalArray;

       }



   int[][] logTransformArray (int[][]
inputArray, int scale){

           int [][]finalArray = new
int[inputArray.length][inputArray[0].length]
;

           int max = 255;

           for(int x = 0 ; x
<inputArray.length ; x++){

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


                   finalArray[x][y] = scale *
(int) ( max *
Math.log(1+inputArray[x][y])/Math.log(1+max)
);

               }

           }

           return finalArray;

       }
```



```
   int[][] histogramArray(int[][]
grayImageArray){
           double maxIntensity = 255;
           int [][] finalArray = new int
[grayImageArray.length][grayImageArray[0].le
ngth];
           int[] transformedIntensity = new
int [256];
           int[]intensityCount = new int
[256];
           double[] probabilityCount = new
double [256];


           //counting the number of pixels
that have certain intensity
           for(int x =0 ; x<
grayImageArray.length ; x++){
               for (int y = 0; y<
grayImageArray[0].length;y++){

intensityCount[grayImageArray[x][y]]++;
               }
           }


           //calculating  probability
           for(int x = 0; x<
probabilityCount.length; x++){
               probabilityCount[x]= (double)
intensityCount[x]/(grayImageArray.length*
grayImageArray[0].length);
           }


           //redefining intensity
           double sumOfPrev =0;
           for(int x = 0; x<
transformedIntensity.length ; x++){
               sumOfPrev = sumOfPrev +
probabilityCount[x];
               transformedIntensity[x]=
(int)(maxIntensity * sumOfPrev);
           }


           for(int x =0 ; x<
finalArray.length ; x++){
               for (int y = 0; y<
finalArray[0].length;y++){
                   finalArray[x][y] =
transformedIntensity[grayImageArray[x][y]];
               }
           }
           return finalArray;
       }


   int[][] getLaplacianKernel(){
           return new int[][]{
                   {-1,-1,-1},
                   {-1,8,-1},
                   {-1,-1,-1}};
       }


   double[][] getAverageKernel(){
           double[][] kernel = new
double[3][3];
           for(int x =0 ;x<3;x++){
               for(int y=0;y<3;y++){
                   kernel[x][y]= (double)
1/9;
               }
           }
           return kernel;
       }


   public int[][] getVerticalKernel (){
           return new int[][]{
                   {-1,-2,-1},
                   {0,0,0},
                   {1,2,1}};
       }


       public int[][] getHorizontalKernel(){
           return new int[][]{
                   {-1,0,1},
                   {-2,0,2},
                   {-1,0,1}};
       }
```

```
	int[][]
getPadded2DSection(int[][]imageArray, int
xCord, int yCord , int sizeOfKernel){
		int[][] finalArray = new
int[sizeOfKernel][sizeOfKernel];


		int x = xCord - (sizeOfKernel /2);


		for(int i = 0; i< sizeOfKernel;
i++){
			int y = yCord - (sizeOfKernel
/2);
			for(int j = 0; j<
sizeOfKernel; j++){
				if(x<0 || y< 0 ||
x>=imageArray.length || y>=
imageArray[0].length){
					finalArray[i][j]=0;
				}else {
					finalArray[i][j] =
imageArray[x][y];
				}
				y++;
			}
			x++;
		}
		return finalArray;
	}


	int  sumOfDotMatrix(int[][]
sectionArray,int [][] kernel){
		int sum =0;
		for(int
x=0;x<sectionArray.length;x++){
			for(int
y=0;y<sectionArray[0].length;y++){
				sum = sum +
(sectionArray[x][y]*kernel[x][y]);
			}
		}
		return  sum;
	}


	public  int[][]
correlationConvolutionTransform(int[][]
imageArray,int[][]kernel){
		writeFiles console =new
writeFiles();
		int[][] unscaledArray = new
int[imageArray.length][imageArray[0].length]
;
		int[][] sectionArray ;


		for(int x=0; x< imageArray.length
; x++){
			for(int y=0; y<
imageArray[0].length;y++){
				sectionArray =
getPadded2DSection(imageArray,x,y,
kernel.length);
				unscaledArray[x][y]=
sumOfDotMatrix(sectionArray,kernel);
				console.writeConsole("padded
esction "+x+","+y,sectionArray);
			}
		}
		console.writeConsole("unscaeld
array",unscaledArray);


		return
rescaleImageArray(unscaledArray);
	}


	public  int[][]
imageLaplacianEnhancement(int[][]grayImageAr
ray, int[][]correlationArray,int[][]
kernel){
		/*
		The value of image Laplacian is
higher where there is sharp transition in
intensity, i.e. where the image is more
detailed.
		 Thus, addition of the Laplacian
to the original image enhances the details
in an image.
		 */
		int [][] unscaledArray = new int
[grayImageArray.length][grayImageArray[0].le
ngth];


		int centerRow = (kernel.length
/2);
		int centerColm = (kernel[0].length
/2);
		int constant;
```

```
        if(kernel[centerRow][centerColm] < 0 ){
            constant =-1;
        }else {
            constant =1;
        }

        for(int x =0; x< grayImageArray.length ; x++){
            for(int y=0 ; y< grayImageArray[0].length;y++){
                unscaledArray[x][y] = (grayImageArray[x][y]+ constant * correlationArray[x][y]) ;
            }
        }

        return rescaleImageArray(unscaledArray);
    }


    public int[][] getMagnitudeOfSobelOperator(int[][] grayImageArray,int[][]horizontalKernel, int[][] verticalKernel){

        int [][] unscaledArray = new int[grayImageArray.length][grayImageArray[0].length];
        int[][] sectionArray;
        int magnitudeX , magnitudeY;
        for(int x =0; x<grayImageArray.length;x++){
            for(int y =0 ; y<grayImageArray[0].length;y++){
                sectionArray = getPadded2DSection(grayImageArray ,x,y,horizontalKernel.length);
                magnitudeX = sumOfDotMatrix(sectionArray,horizontalKernel);
                magnitudeY = sumOfDotMatrix(sectionArray,verticalKernel);
                unscaledArray[x][y]= (int) Math.sqrt(magnitudeX * magnitudeX + magnitudeY * magnitudeY);
            }
        }
        return rescaleImageArray(unscaledArray);
    }

    public static int get93thPercentile(int [][]array ){
        // how we do is we flatten 1d array into 2d and sort the array
        //then if odd number of elements are odd then value in length/2 is median
        //if even we take out average of 2

        int [] oneDArray = new int[array.length * array[0].length];
        int index = 0;
        for (int[] ints : array) {
            for (int y = 0; y < array[0].length; y++) {
                oneDArray[index] = ints[y];
                index++;
            }
        }

        Arrays.sort(oneDArray);

        return oneDArray[oneDArray.length *93/100];
    }
```