

10/22/2015

Image Processing Simulator

**CS 3042 – IMAGE PROCESSING
FINAL PHASE**

YEAR OF STUDY : 3rd Year

DATE OF SUBMISSION: 22nd October, 2015

INDEX NUMBER : 120126K

NAME : A.Y Dissanayake

TABLE OF CONTENTS

1. Introduction	02
2. Tools and Techniques Used	05
3. Snap Shots of the system	06
4. Main Classes	08

1. INTRODUCTION

This project has 3 phases and each phase added the new functionality to the system. Those add functions are as below:

1.1 PHASE I

1. GUI framework
2. Image loading
3. Save and Save As operations
4. Image cloning (clone image to another window)
5. RGB Channel Splitting

1.2 PHASE II

1. Re-sampling and scaling by any given factor (user should be able to select the sampling method)
2. Compute and display histogram
3. Negative, Brightness and contrast adjustments
4. Contrast stretching and normalization

1.3 PHASE III

1. Bit-plane based run length coding of an image
2. Huffman coding of gray level images (8bpp) (convert any RGB image into gray scale. Save code book in as image header)
3. LOG filter of an image (5x5 and 7x7 mask)
4. Filter an image based on a user specified 3x3 mask

2. TOOLS AND TECHNIQUES USED

For overall system was build on Netbeans IDE and as a language it has been used JAVA. At the 1st step for the assigns tasks (except RGB channel splitting since it is a additional function) I never use any framework for JUI graphics I used jFreechart.jar for make UI more attractive. And for RGB Splitting part, there are 2 options one for RGB channel split in color space and other one is intensity level Splitting. For the second part only I used ImageJ framework.

At the 2nd and 3rd step I never use any special frameworks .

3. SNAPSHOTS OF THE SYSTEM

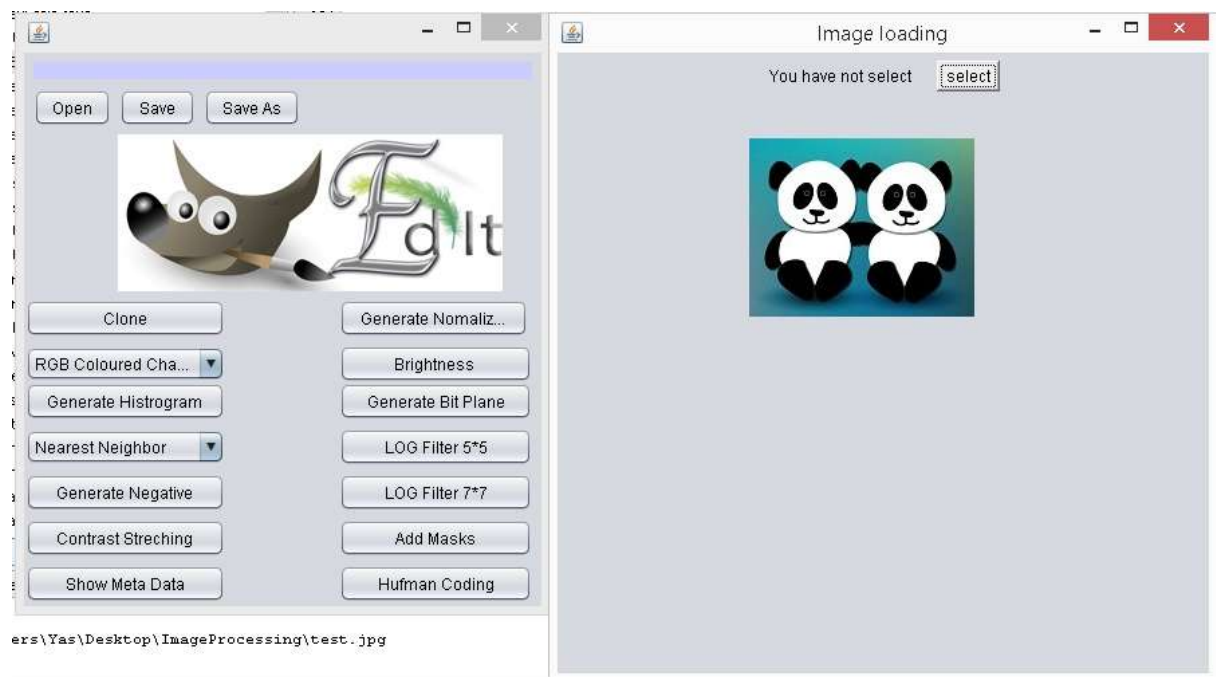
3.1 PHASE I

a. GUI of the system

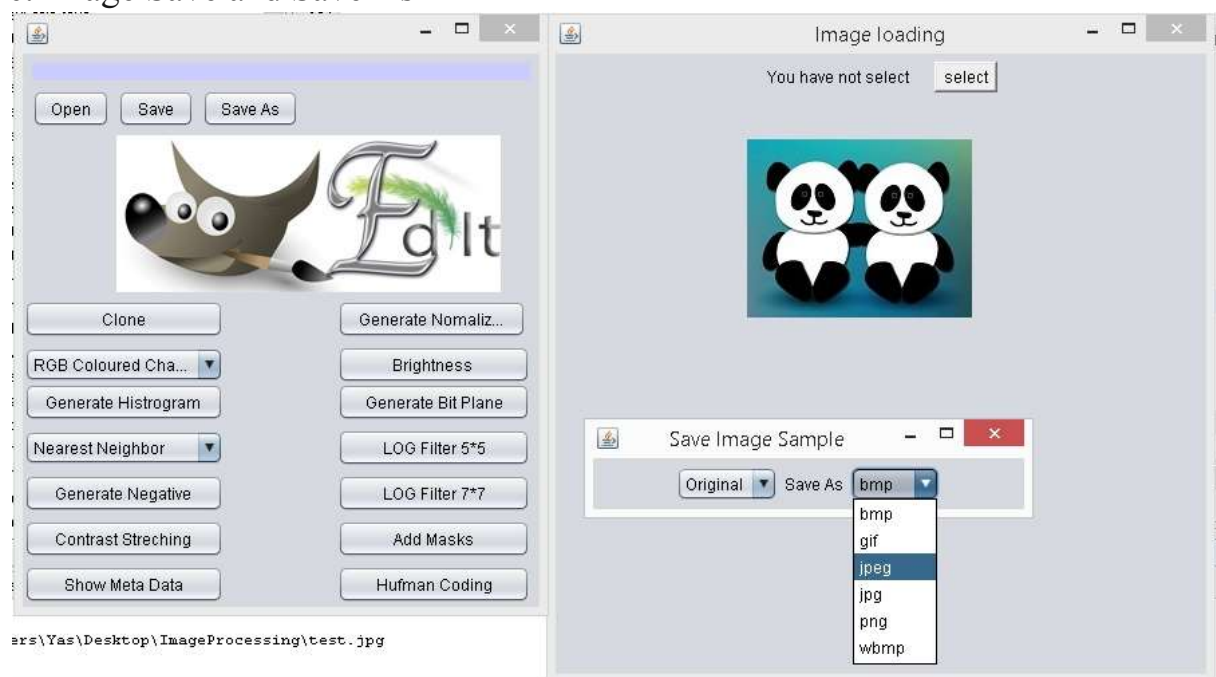


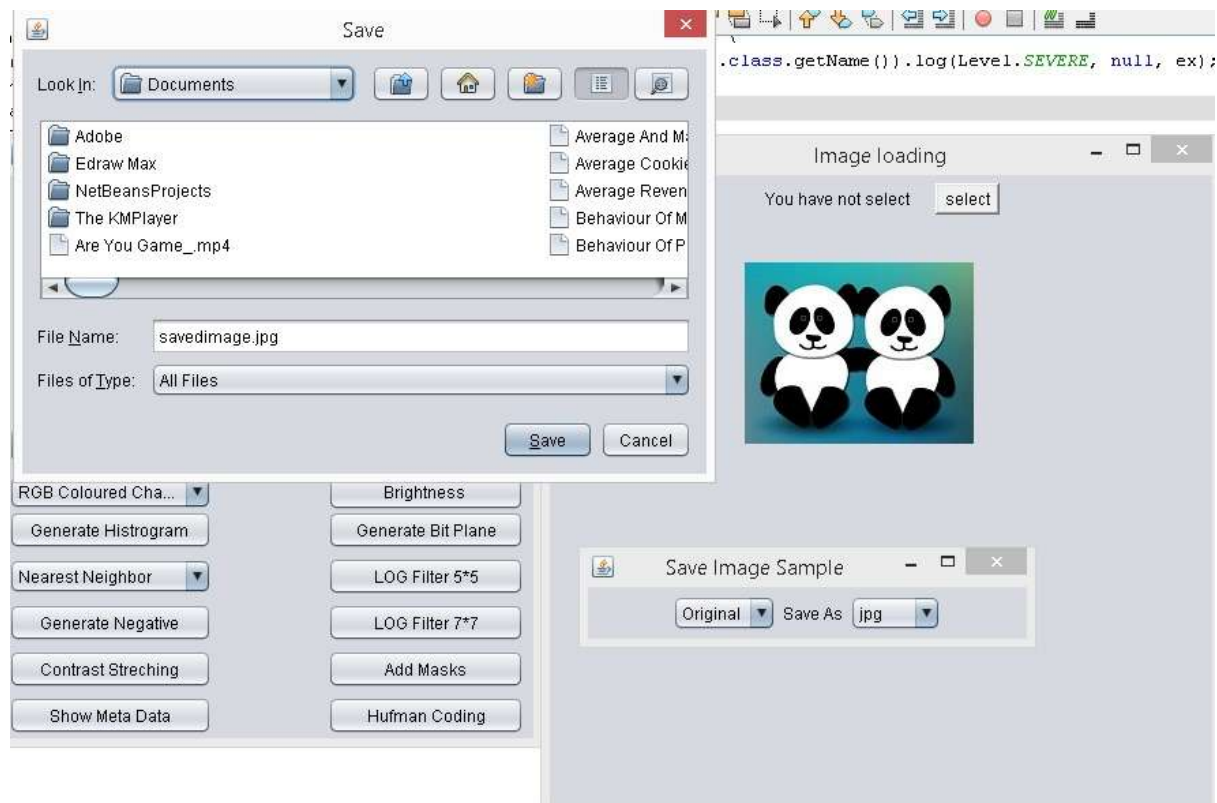
b. Image Loading





c. Image Save and Save As

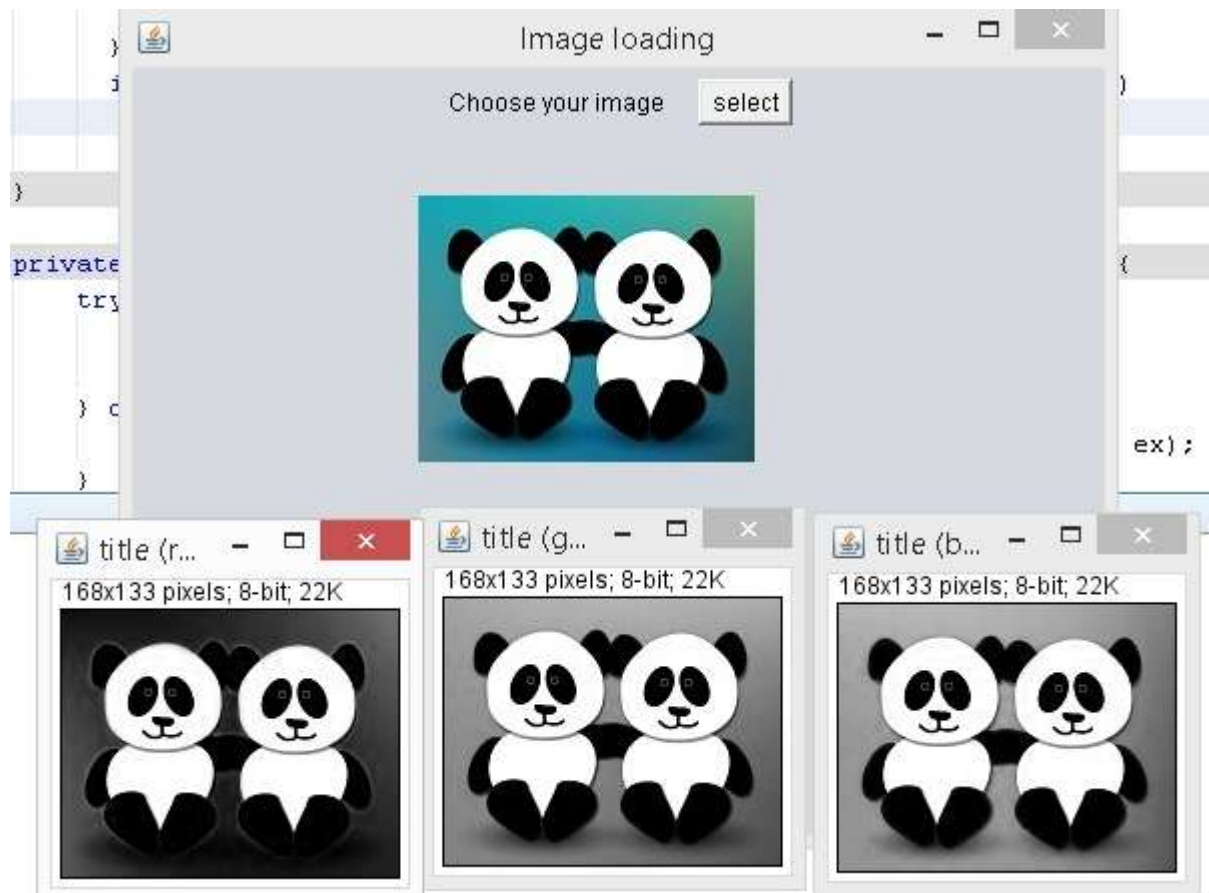




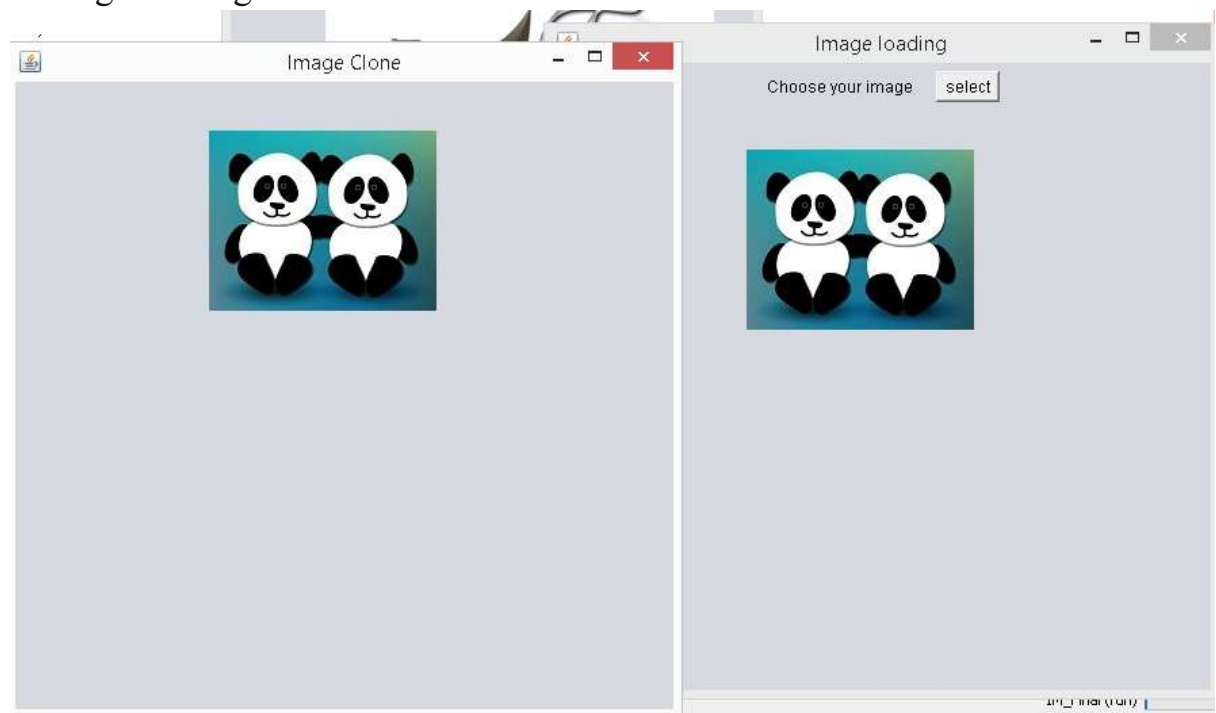
d. RGB color space Channel Splitting



e. RGB intensity level channel splitting

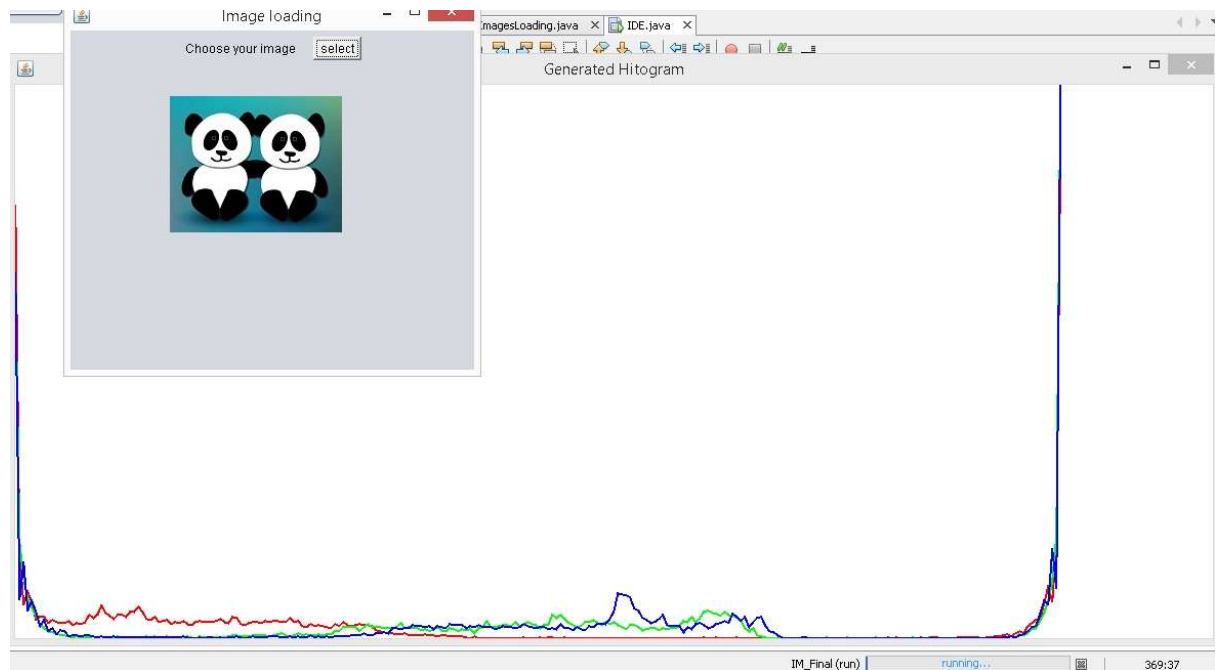


f. Image cloning

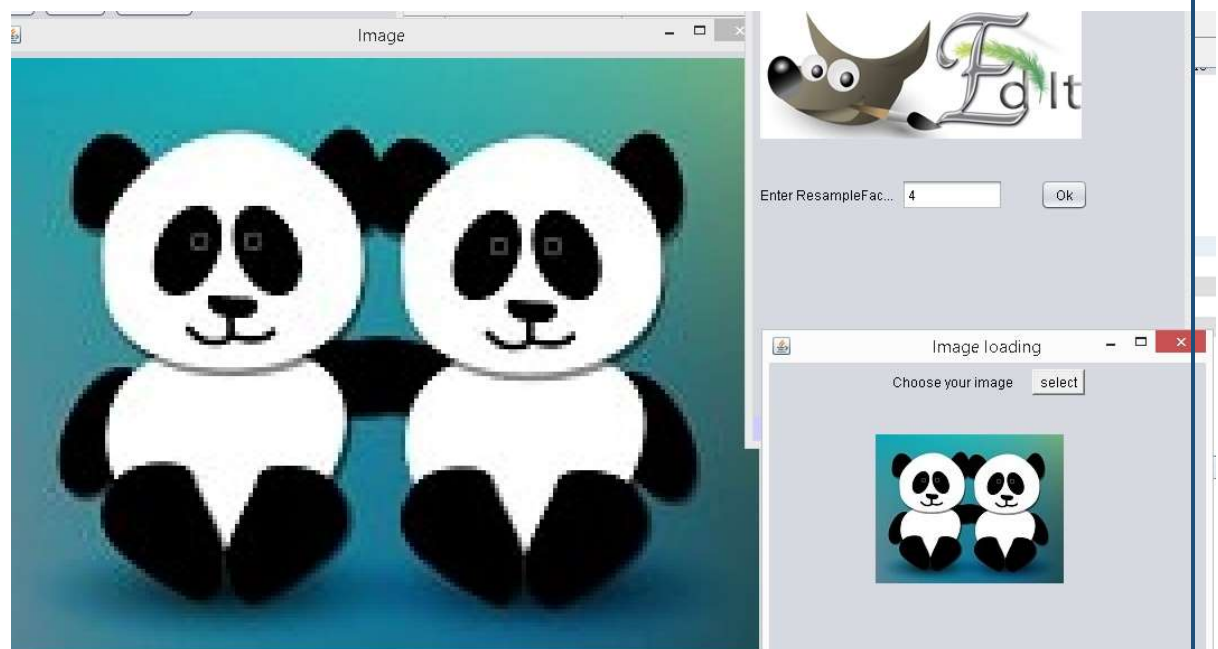


3.2 PHASE II

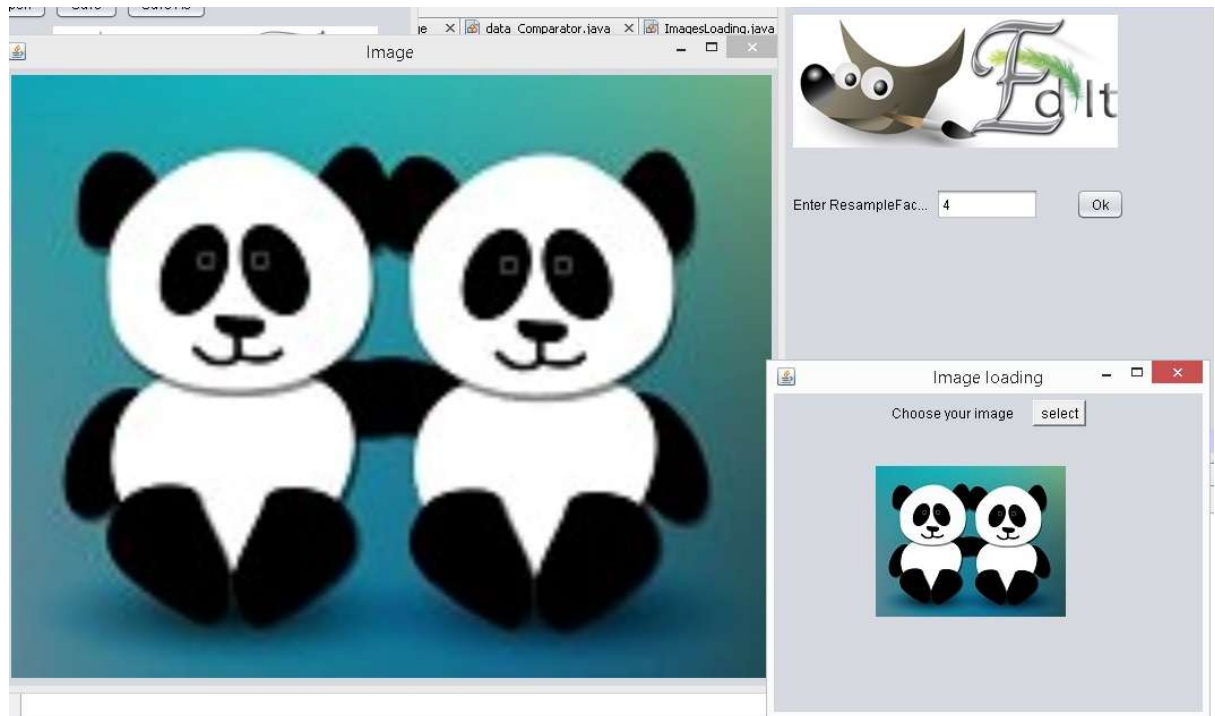
a. Generate Hitrogram



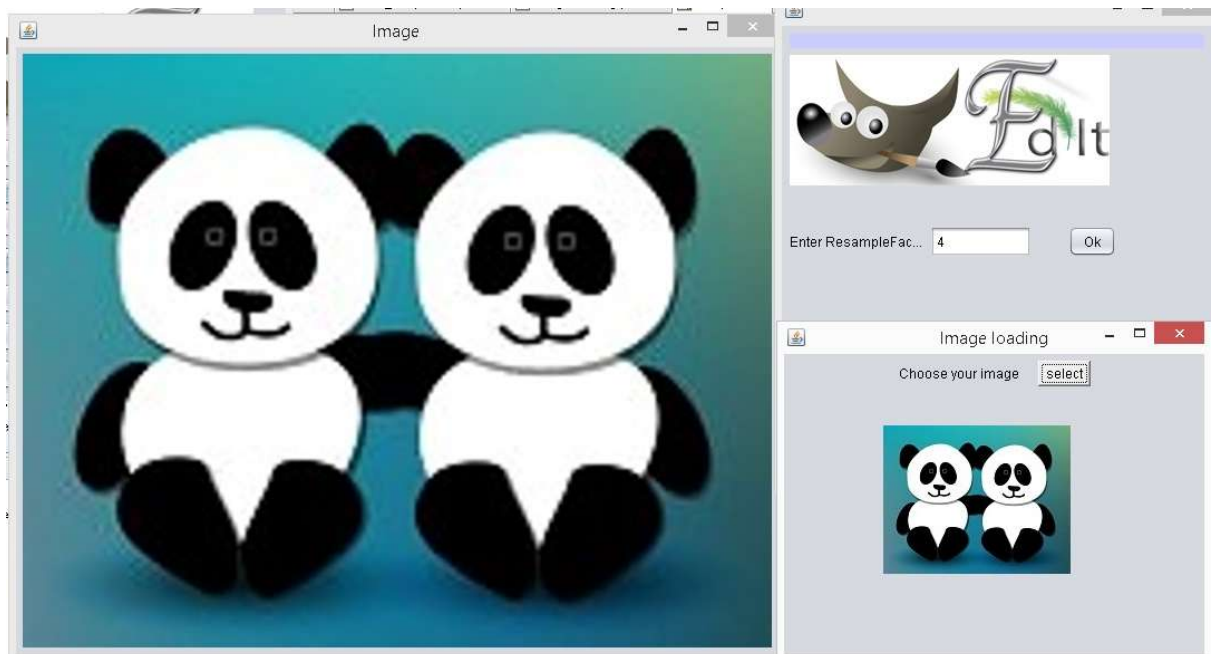
b. Re- sampling by factor 4 – NEARESTNEIGHBOUR



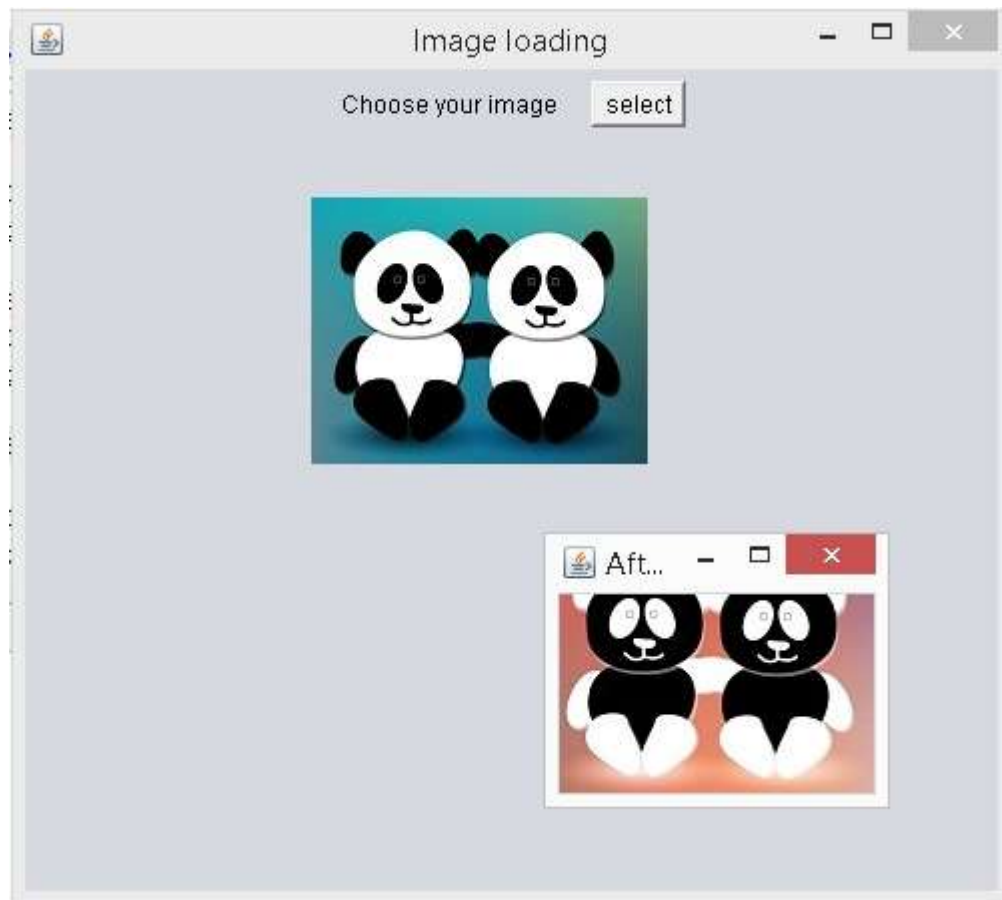
c. Re- sampling by factor 4 –BI-LINEAR



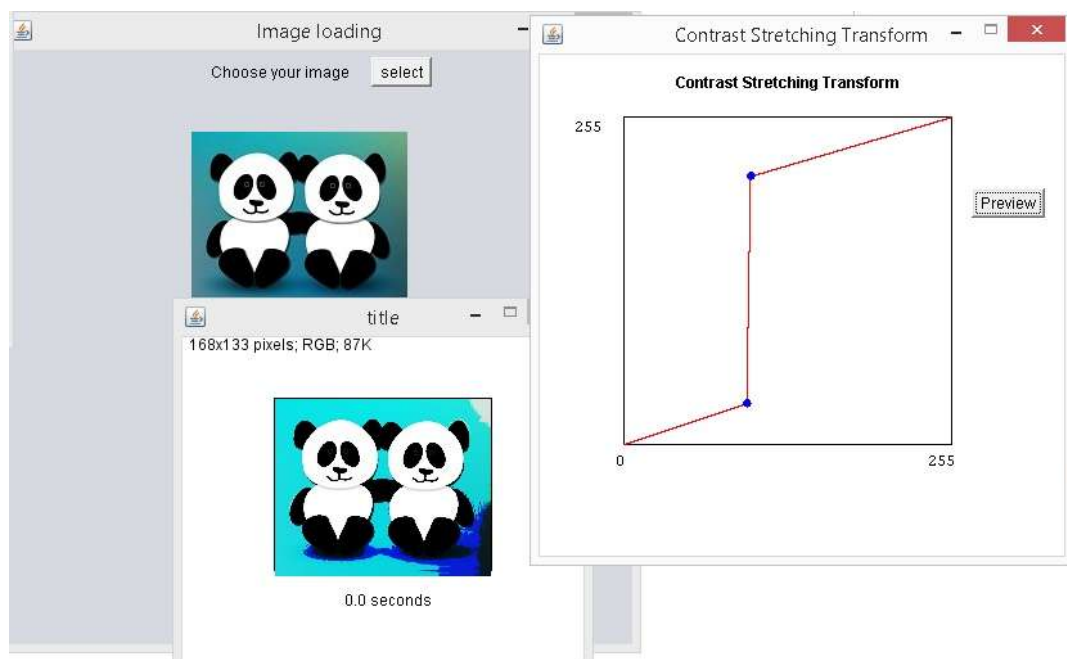
d. Re- sampling by factor 4 – BI-CUBIC



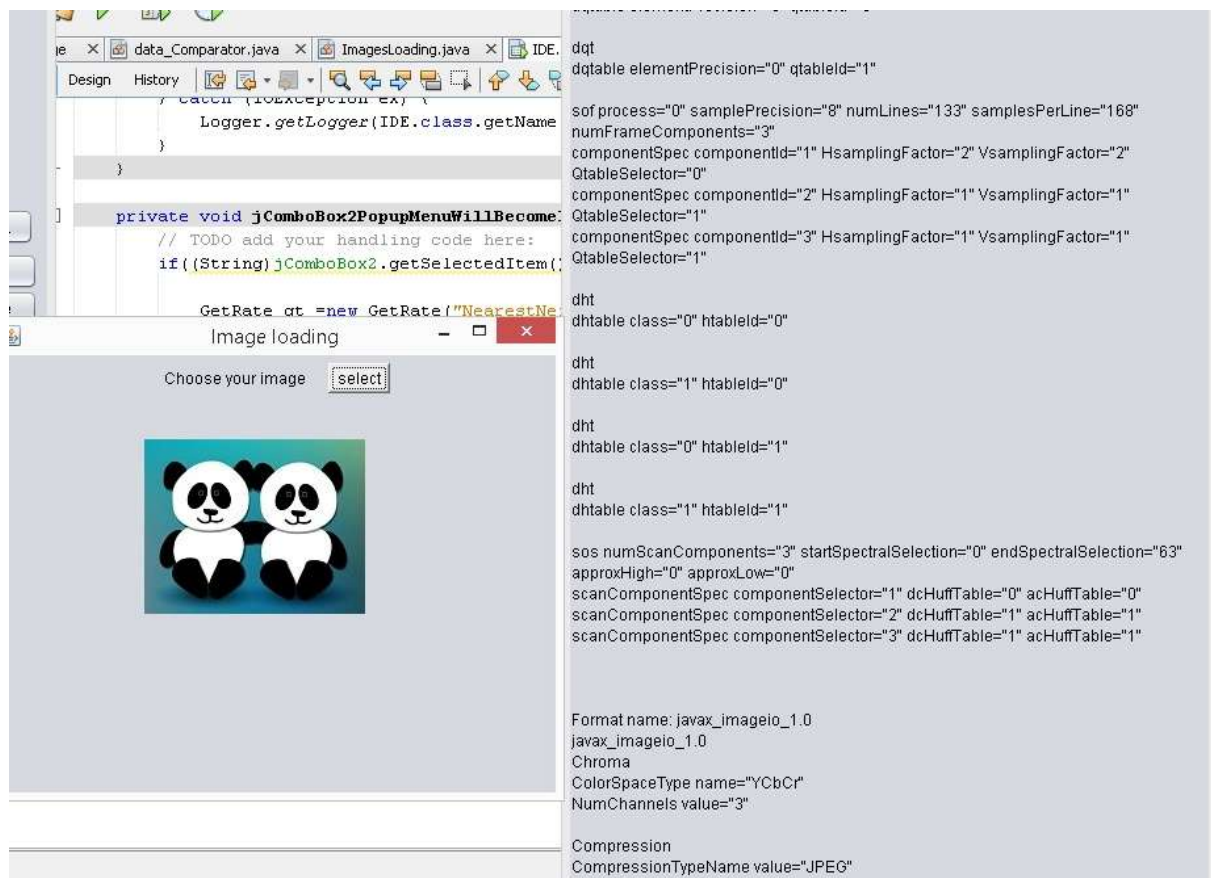
e. Generate Negative



f. Contrast Stretching



g. Image meta data absorbing



The screenshot shows an IDE with two windows. The top window, titled 'ImagesLoading.java', contains the following Java code:

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class ImagesLoading {
    private static Logger logger = Logger.getLogger(ImagesLoading.class.getName());

    private void jComboBox2PopupMenuWillBecomeInvisible() {
        // TODO add your handling code here:
        if ((String) jComboBox2.getSelectedItem() != null) {
            GetRate qt = new GetRate("NearestNeighbor");
            Image loading = new Image(jComboBox2.getSelectedItem().toString());
        }
    }
}
```

The bottom window, titled 'Image loading', displays a 'Choose your image' button and a 'select' button. Below these buttons is a small image of two pandas. The console window on the right displays the following metadata:

```
dqt
dqttable elementPrecision="0" qtableId="1"

sof process="0" samplePrecision="8" numLines="133" samplesPerLine="168"
numFrameComponents="3"
componentSpec componentId="1" HsamplingFactor="2" VsamplingFactor="2"
QtableSelector="0"
componentSpec componentId="2" HsamplingFactor="1" VsamplingFactor="1"
QtableSelector="1"
componentSpec componentId="3" HsamplingFactor="1" VsamplingFactor="1"
QtableSelector="1"

dht
dhttable class="0" htableId="0"

dht
dhttable class="1" htableId="0"

dht
dhttable class="0" htableId="1"

dht
dhttable class="1" htableId="1"

sos numScanComponents="3" startSpectralSelection="0" endSpectralSelection="63"
approxHigh="0" approxLow="0"
scanComponentSpec componentSelector="1" dcHuffTable="0" acHuffTable="0"
scanComponentSpec componentSelector="2" dcHuffTable="1" acHuffTable="1"
scanComponentSpec componentSelector="3" dcHuffTable="1" acHuffTable="1"

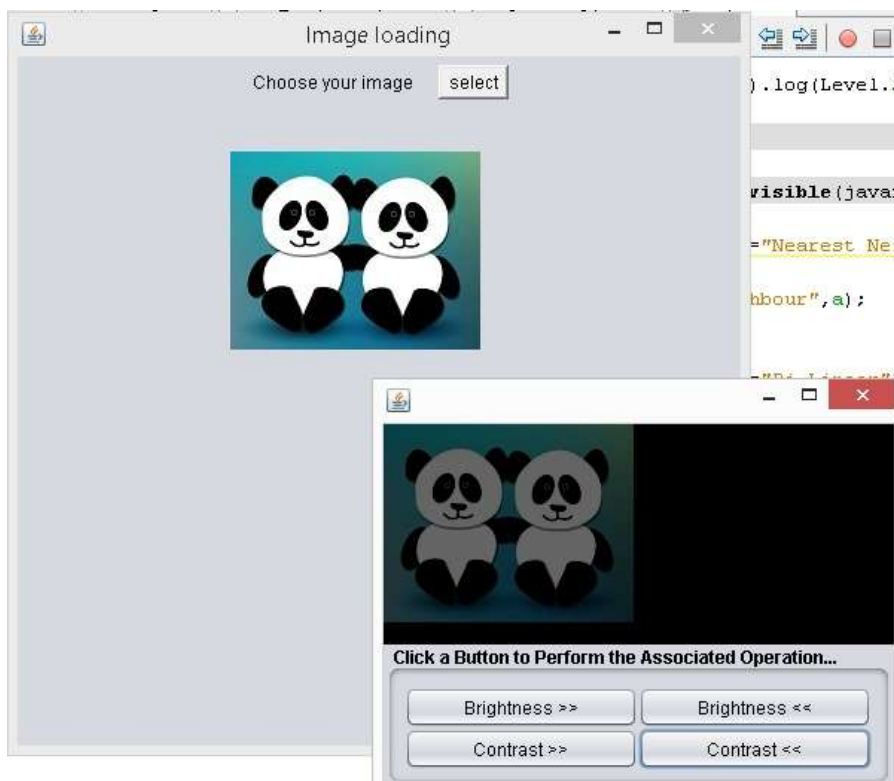
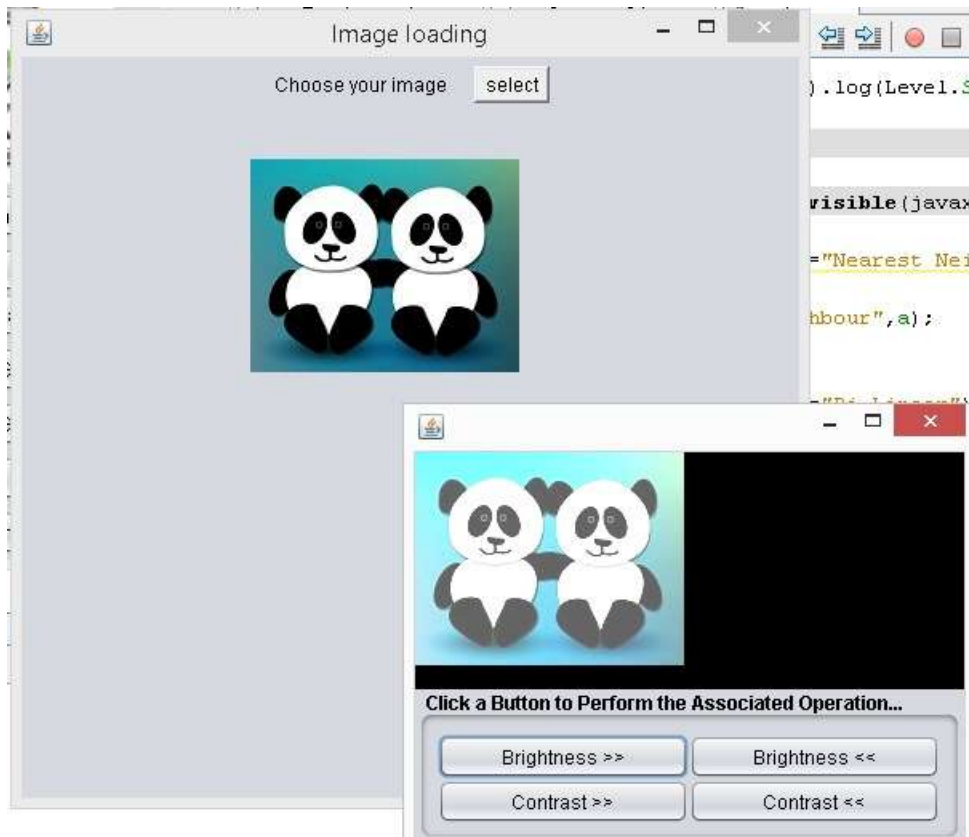
Format name: javax_imageio_1.0
javax_imageio_1.0
Chroma
ColorSpaceType name="YCbCr"
NumChannels value="3"

Compression
CompressionTypeName value="JPEG"
```

h. Image Normalizing

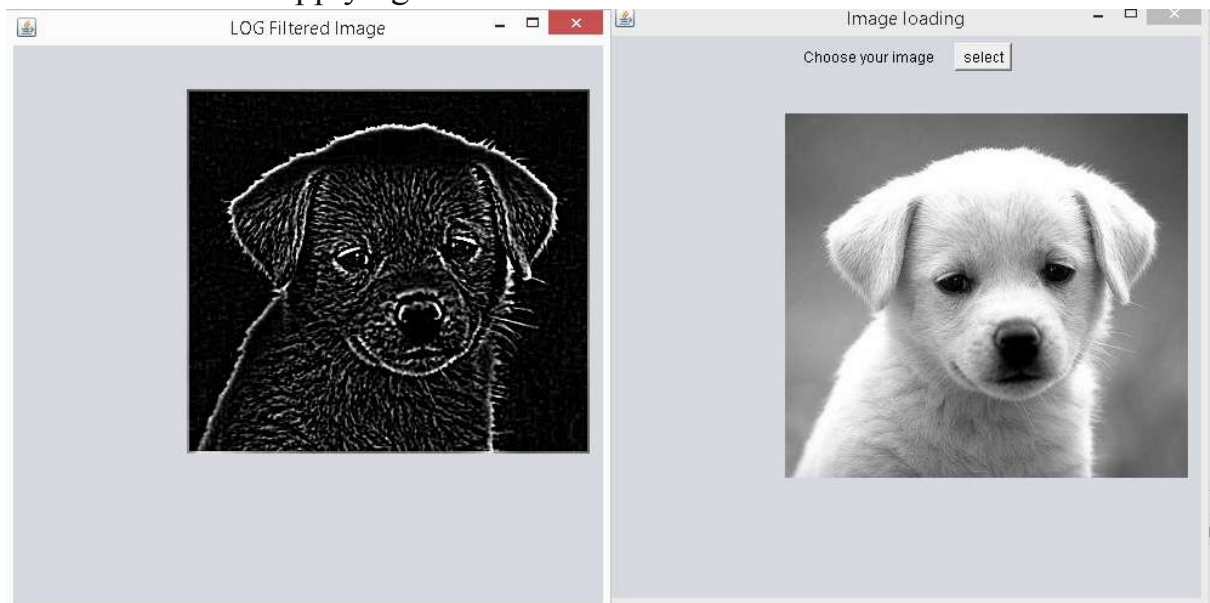


I. Brightness and Contrast Adjusting



3.3 PHASE III

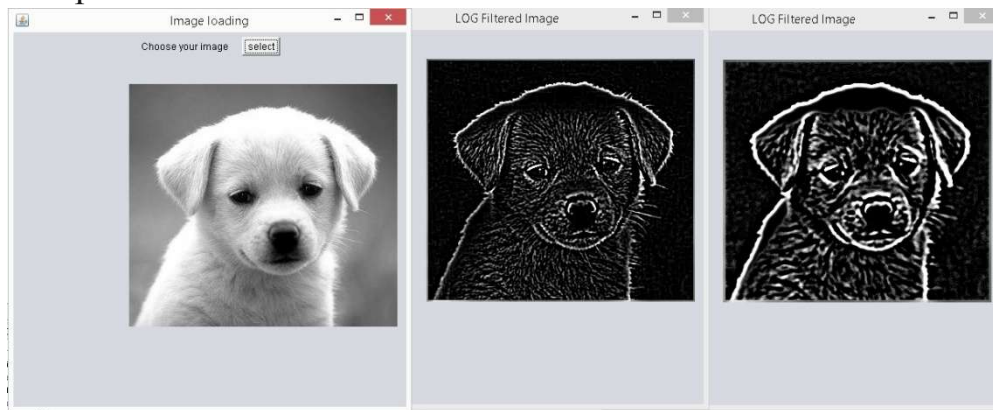
a. 5*5 LOG Filter applying



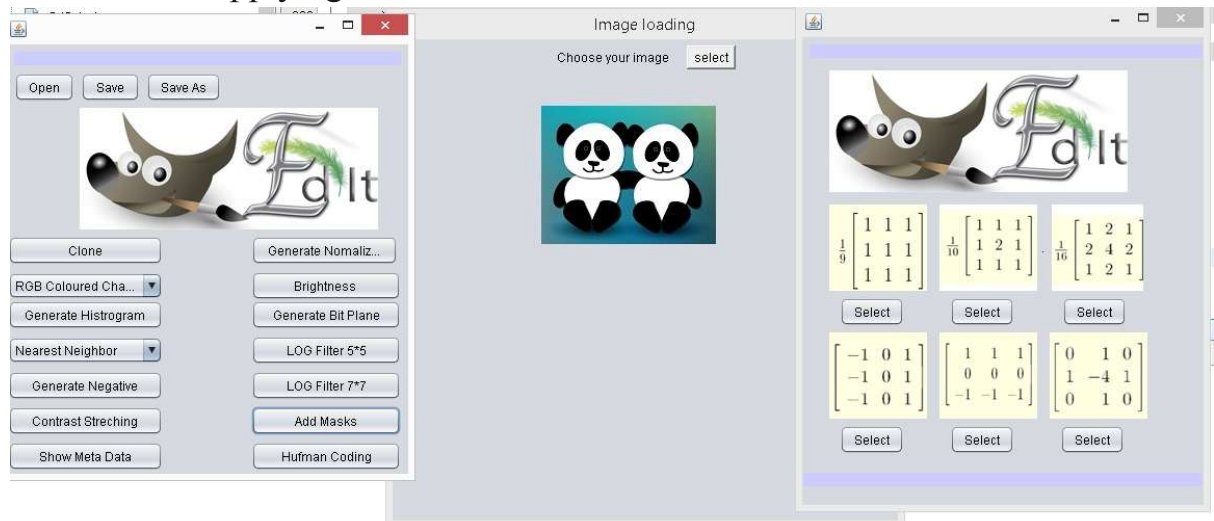
b. 7*7 LOG Filter applying



Comparison of window 5*5 and 7*7



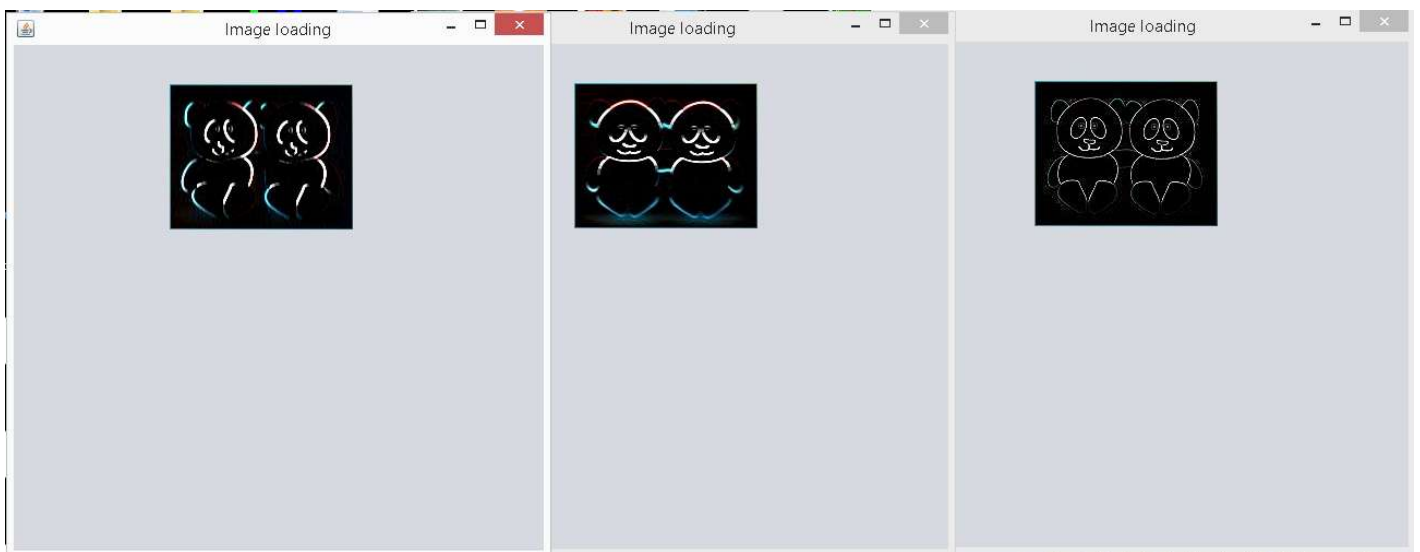
c. 3*3 Masks applying as user choose



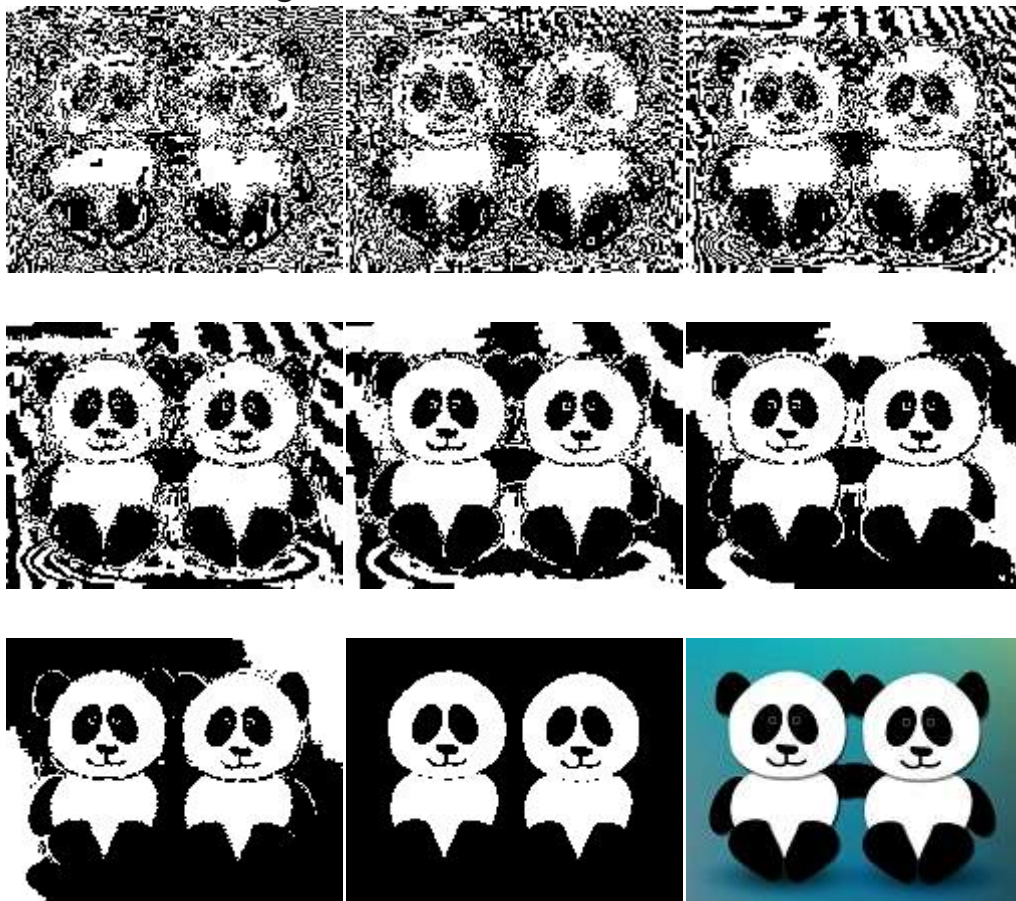
(1st, 2nd and 3rd Mask applying)



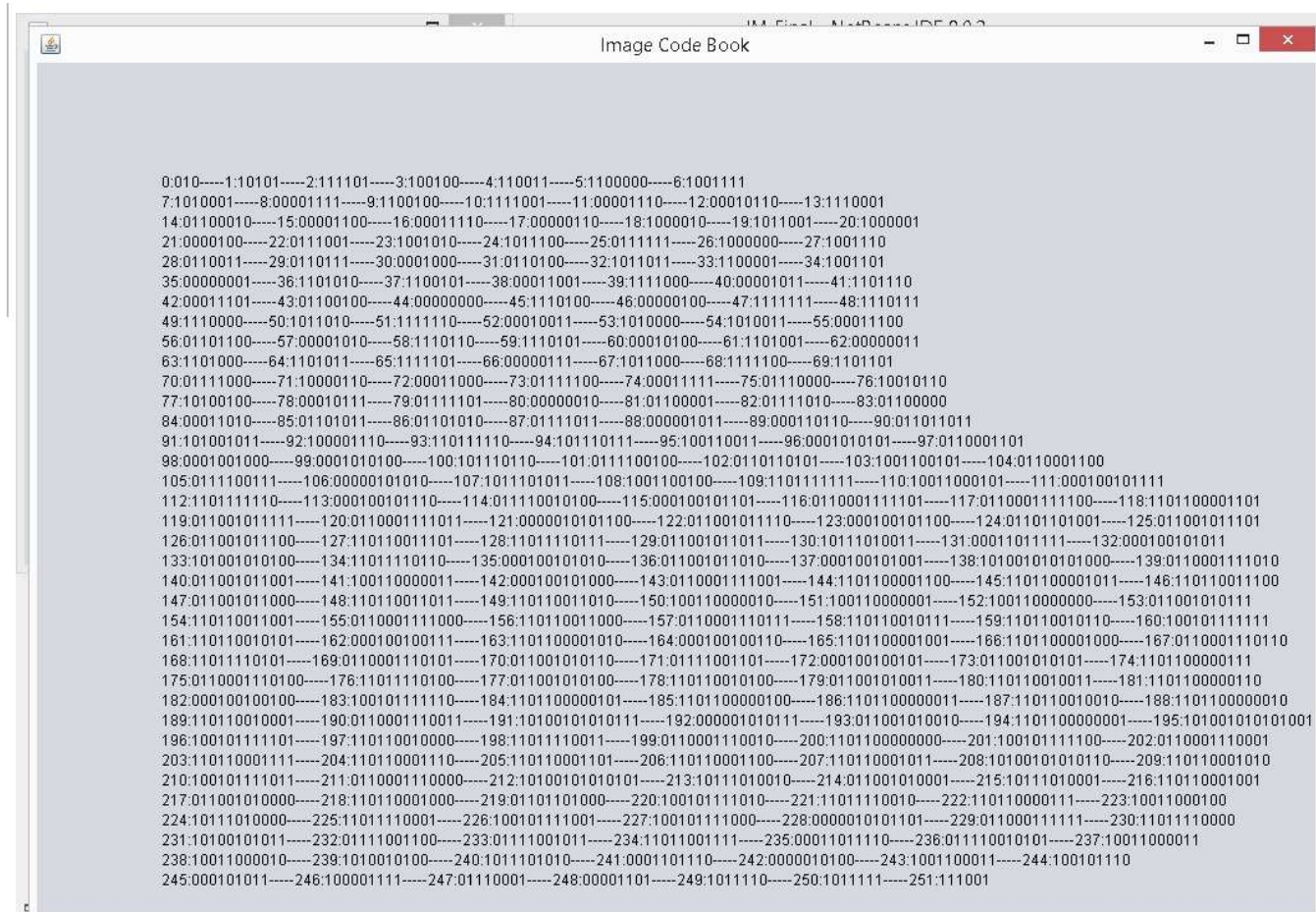
(4th , 5th and 6th Mask Appling)



c. Bit Plane Coding



g. Huffman Coding



```

turn:
d is C:\Users\Yas\Desktop\ImageProcessing\test.jpg
0.011680989: 110011
0.023540996: 11001
0.047574293: 1100
0.09358217: 110
0.18148048: 11
0.40668637: 1

-----CODEBOOK-----
0:010,1:10101,2:111101,3:100100,4:110011,5:1100000,6:1001111,7:1010001,8:00001111,9:1100100,10:1111001,11:00001110,12:00010110,13:1110001,14:01100010,15:00001100,16:00011110,17:00000

-----Compressed Image-----
00001100000011000000110000001100000011000000110000001100111000111000101100010011000100000110000011000001010000101000101011001100000110000011000001

```

4. PHASE III IMPORTANT CODES

a. Bit-plane based run length coding of an image

```
import java.awt.image.BufferedImage;
import java.awt.image.MemoryImageSource;
import java.awt.image.Raster;

public class BitPlane{
    public BitPlane(String path){
        BufferedImage img = null;

        try{
            img = ImageIO.read(new File(path));
        }catch (IOException e){
            e.printStackTrace();
        }

        int m = img.getWidth();
        int n = img.getHeight();

        double[][] matrix = new double[n][m];
        int [][]pixels = new int[n*m][8];

        String s=null;
        int cnt=0;

        for (int row = 0; row < n; ++row)
        {
            for (int col = 0; col < m; ++col)
            {
                Raster rst = img.getRaster();

                int grayLevel = rst.getSample(col,row,0);

                matrix[row][col] = grayLevel;

                s=Integer.toBinaryString((int)matrix[row][col]);

                int len = s.length()-1;

                boolean b = false;
```

```

int len2 = 7;

if(len<7)

b = true;

for(int i=len;i>=0;i--){

char c = s.charAt(i);

int x = Character.getNumericValue(c);

if(b==false)

pixels[cnt][i] = x;

else if(b==true)

{pixels[cnt][len2] = x;

len2--;}}

cnt++;}}

int bit0[],bit1[],bit2[],bit3[],bit4[],bit5[],bit6[],bit7[];

bit0 = new int[(n*m)];

bit1 = new int[(n*m)];

bit2 = new int[(n*m)];

bit3 = new int[(n*m)];

bit4 = new int[(n*m)];

bit5 = new int[(n*m)];

bit6 = new int[(n*m)];

bit7 = new int[(n*m)];

// BIT 7

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

bit7[i] = pixels[i][0];

}

//BIT 6

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

bit6[i] = pixels[i][1];

}

//BIT 5

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

bit5[i] = pixels[i][2];

}

```

```

//BIT 5
for(int i=0;i<n*m;i++){
    bit5[i] = pixels[i][2];
}

//BIT 4
for(int i=0;i<n*m;i++){
    bit4[i] = pixels[i][3];
}

//BIT 3
for(int i=0;i<n*m;i++){
    bit3[i] = pixels[i][4];
}

//BIT 2
for(int i=0;i<n*m;i++){
    bit2[i] = pixels[i][5];
}

//BIT 1
for(int i=0;i<n*m;i++){
    bit1[i] = pixels[i][6];
}

//BIT 0
for(int i=0;i<n*m;i++){
    bit0[i] = pixels[i][7];
}

getImageFromArray(bit7,n,m,"Bit 7");
getImageFromArray(bit6,n,m,"Bit 6");
getImageFromArray(bit5,n,m,"Bit 5");
getImageFromArray(bit4,n,m,"Bit 4");
getImageFromArray(bit3,n,m,"Bit 3");
getImageFromArray(bit2,n,m,"Bit 2");
getImageFromArray(bit1,n,m,"Bit 1");
getImageFromArray(bit0,n,m,"Bit 0");

```

```

public void getImageFromArray(int[] pixels, int height, int width,String name){

BufferedImage im = new BufferedImage(width, height, BufferedImage.TYPE_BYTE_BINARY);

int k=0;

int data[]=new int[width*height];

int imData[][]= new int[height][width];

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

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

imData[i][j] = pixels[k];

k++;}

}

k=0;

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

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

try{

if(imData[i][j]==1){

int r = 255;

int g = 255;

int b = 255;

data[k] = (255 << 24) | (r << 16) | (g << 8) | b; //WHITE

}

else if(imData[i][j]==0)

{

int r = 0;

int g = 0;

int b = 0;

data[k] = (255 << 24) | (r << 16) | (g << 8) | b; //BLACK

}

k++;

}

catch(Exception e){}

}

}
}

```

```
MemoryImageSource mis = new MemoryImageSource(width, height, data, 0, width);  
Toolkit tk = Toolkit.getDefaultToolkit();  
Image imgOut = tk.createImage(mis);  
Graphics2D g = im.createGraphics();  
g.drawImage(imgOut, 0,0,null);  
try  
{ImageIO.write(im, "jpg", new File(name+".jpg"));  
}  
catch(IOException e ){  
System.out.println("saveJpeg "+e);  
}  
}  
}
```

b. Huffman coding of gray level images (8bpp) (convert any RGB image into gray scale. Save code book in as image header)

```
public class Haffman {
    BufferedImage image;

    public Haffman(BufferedImage image) {
        this.image=image;
        //Compress();
    }

    public String getCodeBook() throws IOException{
        String CdBk=Compress();
        return CdBk;
    }

    public String Compress() throws IOException{
        int grayLevelCounts[]=new int[256];
        long pixelCount=0;
        for (int i=0;i<image.getWidth();i++){
            for (int j=0;j<image.getHeight();j++){
                Color c=new Color(image.getRGB(i,j));
                grayLevelCounts[c.getRed()]++;
                pixelCount++;
            }
        }

        ArrayList<GrayLevel> levels = new ArrayList<>();
        for (int i=0;i<grayLevelCounts.length;i++){
            GrayLevel gl=new GrayLevel((float)grayLevelCounts[i]/(float)pixelCount,i);
            levels.add(gl);
        }

        ArrayList<GrayLevel> originals = new ArrayList<>(levels);

        while (levels.size() > 2) {
            Collections.sort(levels);
        }
    }
}
```

```

public class Haffman {
    BufferedImage image;

    public Haffman(BufferedImage image) {
        this.image=image;
        //Compress();
    }

    public String getCodeBook() throws IOException{
        String CdBk=Compress();
        return CdBk;
    }

    public String Compress() throws IOException{
        int grayLevelCounts[]=new int[256];
        long pixelCount=0;
        for (int i=0;i<image.getWidth();i++){
            for (int j=0;j<image.getHeight();j++){
                Color c=new Color(image.getRGB(i,j));
                grayLevelCounts[c.getRed()]++;
                pixelCount++;
            }
        }

        ArrayList<GrayLevel> levels = new ArrayList<>();
        for (int i=0;i<grayLevelCounts.length;i++){
            GrayLevel gl=new GrayLevel((float)grayLevelCounts[i]/(float)pixelCount,i);
            levels.add(gl);
        }

        ArrayList<GrayLevel> originals = new ArrayList<>(levels);

        while (levels.size() > 2) {
            Collections.sort(levels);

```



```

GrayLevel lowest1 = levels.remove(0);
GrayLevel lowest2 = levels.remove(0);

GrayLevel newLevel = new GrayLevel(lowest1.probability + lowest2.probability);

lowest1.setParent(newLevel);
lowest1.setLeft(true);

lowest2.setParent(newLevel);
lowest2.setLeft(false);

levels.add(newLevel);
}

BufferedWriter WriteHand=new BufferedWriter(new FileWriter("table.txt"));
BufferedWriter WriteHand2=new BufferedWriter(new FileWriter("comp.txt"));
//WriteHand.write(mydata_coded);
//WriteHand.flush();
//WriteHand.close();
GrayLevel p = originals.get(4);
while (p != null) {
    System.out.println(p.probability + ": " + p.getBlock());
    WriteHand.write(p.probability + ": " + p.getBlock()+" "+"\\n");
    p = p.parent;
}
WriteHand.flush();
WriteHand.close();
Collections.sort(levels);
levels.get(0).setLeft(true);
levels.get(1).setLeft(false);
//System.out.println(levels);
String codeBook="";
for (GrayLevel level : originals) {
    if (level.getLevel() >= 0) {

```

```

        codeBook+=level.getLevel()+":"+level.getBlock()+",";
    }
}
System.out.println("");
System.out.println("-----CODEBOOK-----");
System.out.println(codeBook);
String coded="";
for (int i=0;i<image.getHeight();i++){
    for (int j=0;j<image.getWidth();j++){
        Color c=new Color(image.getRGB(j,i));
        String blk=originals.get(c.getRed()).getBlock();
        coded+=blk;
    }
    coded+=";";
}
System.out.println("");
System.out.println("-----Copressed Image-----");
System.out.println(coded);
WriteHand2.write(coded);
WriteHand2.flush();
WriteHand2.close();
String output = codeBook+"\n\r"+coded;
return output;
}
}

```

```

public class GrayLevel implements Comparable<GrayLevel> {

    Float probability;

    int level = -1;

    GrayLevel parent;

    boolean left = true;

    public void setLeft(boolean left) {

        this.left = left;

    }

    public void setParent(GrayLevel parent) {

        this.parent = parent;

    }

    public void setLevel(int level) {

        this.level = level;

    }

    public int getLevel() {

        return level;

    }

    public String getBlock() {

        return (parent != null ? parent.getBlock() : "") + (left ? "1" : "0");

    }

    public GrayLevel(float probability) {

        this.probability = probability;

    }

    public GrayLevel(float probability, int level) {

        this.probability = probability;

        this.level = level;

    }

    @Override

    public String toString() {

        return this.level + ":" + this.probability + ":" + this.getBlock();

    }

    @Override

```

```

public class MaskImage {
//extends Component implements ActionListener

String descs[] = {
    "Original",
    "Convolve : LowPass",
    "Convolve : Sharpen",
    "LookupOp",
};

int opIndex;
private BufferedImage bi, biFiltered;
int w, h;

public static final float[] SHARPEN3x3 = { // sharpening filter kernel
    0.f, 1.f, 0.f,
    1.f, -4.f, 1.f,
    0.f, 1.f, 0.f
};

public static final float[] Mask13x3 = { // sharpening filter kernel
    1.f, 1.f, 1.f,
    1.f, 1.f, 1.f,
    1.f, 1.f, 1.f
};

public static final float[] Mask23x3 = { // sharpening filter kernel
    1.f, 1.f, 1.f,
    1.f, 2.f, 1.f,
    1.f, 1.f, 1.f
};

public static final float[] Mask33x3 = { // sharpening filter kernel
    1.f, 2.f, 1.f,
    2.f, 4.f, 2.f,

```

```
public static final float[] Mask63x3 = { // sharpening filter kernel
    0.f, 1.f, 0.f,
    1.f, -4.f, 1.f,
    0.f, 1.f, 0.f
};
```

```
public static final float[] BLUR3x3 = {
    0.1f, 0.1f, 0.1f, // low-pass filter kernel
    0.1f, 0.2f, 0.1f,
    0.1f, 0.1f, 0.1f
};
```

```
String d;
```

```
public MaskImage(String Path) {
    d=Path;
    System.out.println("Mask Called");
    try {
        bi = ImageIO.read(new File(Path));
        w = bi.getWidth(null);
        h = bi.getHeight(null);
        if (bi.getType() != BufferedImage.TYPE_INT_RGB) {
            BufferedImage bi2 =
                new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
            Graphics big = bi2.getGraphics();
            big.drawImage(bi, 0, 0, null);
            biFiltered = bi = bi2;        }
    } catch (IOException e) {
        System.out.println("Image could not be read");
        System.exit(1);
    }
}

public Dimension getPreferredSize() {
    return new Dimension(w, h);
}
```

```

String[] getDescriptions() {
    return desc; }
void setOpIndex(int i) {
    opIndex = i;}
public void paint(Graphics g) {
    filterImage(1);
    g.drawImage(biFiltered, 0, 0, null); }
int lastOp;
public void filterImage(int OPi) {
    System.out.println("Filtered Image");
    BufferedImageOp op = null;
    opIndex = OPi;
    float[] data;
    System.out.println(OPi);
    switch (opIndex) {
    case 0: biFiltered = bi; /* original */
        return;
    case 1: /* low pass filter */
        data=Mask13x3 ;
        op = new ConvolveOp(new Kernel(3, 3, data),
            ConvolveOp.EDGE_NO_OP,
            null);
        break;
    case 2: /* sharpen */
        data=Mask23x3 ;
        op = new ConvolveOp(new Kernel(3, 3, data),
            ConvolveOp.EDGE_NO_OP,
            null);
        break;
    case 3: /* low pass filter */

```

```

data=Mask33x3 ;

op = new ConvolveOp(new Kernel(3, 3, data),
    ConvolveOp.EDGE_NO_OP,
    null);

break;

case 4: /* sharpen */
    data=Mask43x3 ;
    op = new ConvolveOp(new Kernel(3, 3, data),
        ConvolveOp.EDGE_NO_OP,
        null);

    break;

case 5: /* low pass filter */
    data=Mask53x3 ;
    op = new ConvolveOp(new Kernel(3, 3, data),
        ConvolveOp.EDGE_NO_OP,
        null);

    break;

case 6: /* sharpen */
    data=Mask63x3 ;
    op = new ConvolveOp(new Kernel(3, 3, data),
        ConvolveOp.EDGE_NO_OP,
        null);

    break;

case 7 : /* lookup */
    byte lut[] = new byte[256];
    for (int j=0; j<256; j++) {
        lut[j] = (byte)(256-j);
    }
    ByteLookupTable blut = new ByteLookupTable(0, lut);
    op = new LookupOp(blut, null);

    break;
,

```

```

        biFiltered = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        op.filter(bi, biFiltered);
        ImLoad();
    }

    public void ImLoad(){
        System.out.println("Load Image");
        JFrame fr = new JFrame ("Image loading");
        imageLoad Canvas3;
        fr.setSize(500,500);
        fr.setLocation(50,200);
        fr.setBackground(Color.lightGray);
        fr.setLayout(new FlowLayout());
        Canvas3 = new imageLoad(null);
        Canvas3.setSize(1000,1000);
        fr.add(Canvas3);
        Canvas3.setImage(biFiltered);
        Canvas3.repaint();
        fr.show();

    }
}

```


e. LOG filter of an image (5x5)

```
public class LOGFilter5 {
//extends Component implements ActionListener
    int opIndex;
    private BufferedImage bi, biFiltered, biFiltered2;
    int w, h;
    public static final float[] MaskGaussian2= { // sharpening filter kernel
        2.f, 7.f, 12.f,7.f, 2.f,
        7.f, 31.f, 52.f,31.f, 7.f,
        12.f, 52.f, 127.f, 52.f,12.f,
        7.f, 31.f, 52.f,31.f, 7.f,
        2.f, 7.f, 12.f,7.f, 2.f
    };
    public static final float[] MaskLaplacianHigh= { // sharpening filter kernel
        -1.f, -1.f, -1.f,-1.f, -1.f,
        -1.f, -1.f, -1.f,-1.f, -1.f,
        -1.f, -1.f, 24.f,-1.f, -1.f,
        -1.f, -1.f, -1.f,-1.f, -1.f,
        -1.f, -1.f, -1.f,-1.f, -1.f
    };
    String d;
    public LOGFilter5(String Path) {
        d=Path;
        try {
            bi = ImageIO.read(new File(Path));
            w = bi.getWidth(null);
            h = bi.getHeight(null);
            if (bi.getType() != BufferedImage.TYPE_INT_RGB) {
```

```

w = bi.getWidth(null);
h = bi.getHeight(null);
if (bi.getType() != BufferedImage.TYPE_INT_RGB) {
    BufferedImage bi2 =
        new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    Graphics big = bi2.getGraphics();
    big.drawImage(bi, 0, 0, null);
    biFiltered = bi = bi2;
}
} catch (IOException e) {
    System.out.println("Image could not be read");
    System.exit(1);
}
}

```

```

public Dimension getPreferredSize() {
    return new Dimension(w, h);
}

```

```

int lastOp;
public void filterImage() {
    System.out.println("Filtered Image");
    BufferedImageOp op1 = null;
    BufferedImageOp op2 = null;
    float[] data;
    data=MaskLaplacianHigh;
    op1 = new ConvolveOp(new Kernel(5, 5, data),
        ConvolveOp.EDGE_NO_OP,
        null);
    data=MaskGaussian2 ;
    for (int i = 0; i < data.length; i++) {
        data[i]=data[i]/571;
    }
}

```

```

        op2 = new ConvolveOp(new Kernel(5, 5, data),
                               ConvolveOp.EDGE_NO_OP,
                               null);

        biFiltered = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        biFiltered2 = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        op2.filter(bi, biFiltered);
        op1.filter(biFiltered, biFiltered2);

        ImLoad();
    }

    public void ImLoad(){
        JFrame fr = new JFrame ("LOG Filtered Image");
        imageLoad Canvas3;
        fr.setSize(500,500);
        fr.setLocation(50,200);
        fr.setBackground(Color.lightGray);
        fr.setLayout(new FlowLayout());
        Canvas3 = new imageLoad(null);
        Canvas3.setSize(1000,1000);
        fr.add(Canvas3);
        Canvas3.setImage(biFiltered2);
        Canvas3.repaint();
        fr.show();

    }

```

f. LOG filter of an image (7x7)

```
public class LOGFilter7 {

//extends Component implements ActionListener

    int opIndex;

    private BufferedImage bi, biFiltered, biFiltered2;

    int w, h;


    public static final float[] MaskGaussian2= { // sharpening filter kernel

        1.f, 1.f, 2.f,2.f, 2.f,1.f,1.f,

        1.f, 3.f, 4.f,5.f, 4.f,3.f,1.f,

        2.f, 4.f, 7.f, 8.f,7.f,4.f,2.f,

        2.f, 5.f, 8.f,10.f, 8.f,5.f,5.f,

        2.f, 4.f, 7.f, 8.f,7.f,4.f,2.f,

        1.f, 3.f, 4.f,5.f, 4.f,3.f,1.f,

        1.f, 1.f, 2.f,2.f, 2.f,1.f,1.f

    };

    public static final float[] MaskLaplacianHigh= { // sharpening filter kernel

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f,

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f,

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f,

        -1.f, -1.f, -1.f, 48.f, -1.f, -1.f,-1.f,

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f,

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f,

        -1.f, -1.f, -1.f,-1.f, -1.f,-1.f, -1.f

    };

    String d;

    public LOGFilter7(String Path) {

        d=Path;

        System.out.println("Mask Called");

        try {

            bi = ImageIO.read(new File(Path));
```

```

w = bi.getWidth(null);
h = bi.getHeight(null);
if (bi.getType() != BufferedImage.TYPE_INT_RGB) {
    BufferedImage bi2 =
        new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    Graphics big = bi2.getGraphics();
    big.drawImage(bi, 0, 0, null);
    biFiltered = bi = bi2;
}
} catch (IOException e) {
    System.out.println("Image could not be read");
    System.exit(1);
}
}

public Dimension getPreferredSize() {
    return new Dimension(w, h);
}

int lastOp;

public void filterImage() {
    System.out.println("Filtered Image");
    BufferedImageOp op1 = null;
    BufferedImageOp op2 = null;
    float[] data;
    data=MaskLaplacianHigh;
    op1 = new ConvolveOp(new Kernel(7, 7, data),
        ConvolveOp.EDGE_NO_OP,
        null);
    data=MaskGaussian2 ;
    for (int i = 0; i < data.length; i++) {
        data[i]=data[i]/170;
    }
    op2 = new ConvolveOp(new Kernel(7, 7, data),
        ConvolveOp.EDGE_NO_OP,
        null);

```

```

        biFiltered = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

        biFiltered2 = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

        op2.filter(bi, biFiltered);

        op1.filter(biFiltered, biFiltered2);


        ImLoad();
    }

    public void ImLoad(){
        JFrame fr = new JFrame ("LOG Filtered Image");

        imageLoad Canvas3;

        fr.setSize(500,500);

        fr.setLocation(50,200);

        fr.setBackground(Color.lightGray);

        fr.setLayout(new FlowLayout());

        Canvas3 = new imageLoad(null);

        Canvas3.setSize(1000,1000);

        fr.add(Canvas3);

        Canvas3.setImage(biFiltered2);

        Canvas3.repaint();

        fr.show();
    }

    JFrame fr = new JFrame ("Image Preview");

    imageLoad Canvas3;

    fr.setSize(500,500);

    fr.setLocation(50,200);

    fr.setBackground(Color.lightGray);

    fr.setLayout(new FlowLayout());

    Canvas3 = new imageLoad(null);

    Canvas3.setSize(1000,1000);

    fr.add(Canvas3);

    BufferedImage bi;

    try {

        bi = ImageIO.read(new File("output2.jpg"));

        Canvas3.setImage(bi);

        Canvas3.repaint();
    }

```