PRODUCE LOW-PASS AND HIGH-PASS IMAGE FILTER IN JAVA

Omeed Kamal Khorsheed
School of Engineering, Dep of s\w Engineering
Koya University, Iraq - Erbil

ABSTRACT

Image processing is a computers procedures to convert an image to digital data and implement a lot of operations on the converted data, Processing to enhanced the image details or to extract some information from it, Low-Pass and High-Pass are the two famous filter in image enhanced processing, In this paper we will try to review and compare between two types of image filtering algorithm Low-Pass Filter and High-Pass Filter and how we can implement them using java, Low-Pass and High-Pass are contradictory Image filtering with the same executive step but the result is opposite, Low-Pass will Produce a Gaussian smoothing blur image, in the other hand High-Pass filter will increases the contrast between bright and dark pixel to produce a sharpen image.

KEYWORDS: High-Pass, Low-Pass, Image Processing, Pixel, Gaussian, Convolution Kernel, Smoothing, Blur, Sharpening, Java.

I. INTRODUCTION

In Image processing and filtering the most important technique is convolution, convolution is an expression which means to convolve digital Image original information with convolution kernel as dimensional convolution. [1]

When we take a digital photo with camera or scanner, often images will be noisy. Noise always changes rapidly from pixel to pixel because each pixel generates its own independent noise. So we need to improve the image detail, this improve require many image filter effects [2]. Filtering effects can be used to:

- Sharpen image.
- Blur or smoothing image.
- Image noise reduction.
- Detect and enhance edges.
- Alter the contrast of the image.
- Alter the brightness of the image.

In this paper we will discuss two type of image filter Low-Pass Filter and High-Pass filter, the two Filters have the same Convolution technique with deferent convolution kernel.

II. SURVEY OF LOW-PASS FILTER

Low-Pass filter is known as (smoothing or blurring) filter, Low-pass used for image smoothing and noise reduction. Low-pass effect is to calculate the average of a pixel and all of its eight neighbors [3].Low- Pass filtering is convolution that attenuates high frequency of an image while allowing low frequency passing (leaving) [4].High frequency introduced noise in image that decrease image quality, The Low-Pass will smooth the image and reduction noise [2].

2.1 Low-Pass Types

1. Ideal low-pass filters (ILPF)
ILPF is the most simple low-pass filter, it remove all frequencies higher than the cut-off (a specified distance) frequency and leaves smaller frequencies unchanged. 

**ILPF transfer response equation is:**

\[
H(u, v) = \begin{cases} 
1 & \text{if } \sqrt{((u - M/2)^2 + (v - N/2)^2)} \leq \text{Cutoff} \\
0 & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (1)

Where:

- M is number of rows in image
- N is number of columns in image
- \((M/2, N/2)\) is center of frequency rectangle

\[\sqrt{((u - M/2)^2 + (v - N/2)^2)}\]

Is the transform point \((u, v)\) from the center

**ILPF frequency Curve:**

![ILPF Frequency Curve](image)

**Figure 1:** Ideal low-pass filters frequency

ILPF transfer response placed the (one) inside and the zero is placed outside in this case we will get a blurred smoothing image in the same time the image edge content will be reduced.

2. **Butterworth low-pass filters (BLPF)**

BLPF keeps frequencies inside radius cut-off and discards values outside, the transfer equation of order \((n)\) with cut-off frequency defined as:

\[
D(u, v) = \sqrt{((u - M/2)^2 + (v - N/2)^2)}
\]

\[
H(u, v) = \frac{1}{1 + \left[D(u, v) / \text{Cutoff}\right]^{2n}}
\]  \hspace{1cm} (2)

(3)

Where:

- M is number of rows in image
- N is number of columns in image
- \((M/2, N/2)\) is center of frequency rectangle

\[\sqrt{((u - M/2)^2 + (v - N/2)^2)}\]

Is the transform point \((u, v)\) from the center

N is the transfer order

**BLPF frequency Curve:**
3. **Gaussian low-pass filters (GLPF)**

GLPF is highly effective in blurring and removing Gaussian noise from the image. The transfer equation of GLPF is defined as:

\[
D^2(u, v) = (u - M/2)^2 + (v - N/2)^2
\]

\[
H(u, v) = e^{-D^2(u, v) / 2 Cutoff^2}
\]

Where:
- \(M\) is number of rows in image
- \(N\) is number of columns in image
- \((M/2, N/2)\) is center of frequency rectangle
- \(D(u, v)\) is the transform point \((u, v)\) from the center

**GLPF frequency Curve:**

![Figure 2: Butterworth low-pass filter frequency](image)

![Figure 3: Gaussian low-pass filter frequency](image)

### III. SURVEY OF HIGH-PASS FILTER

High-Pass filter is known as (Sharpening) filter, High-pass used for image Sharpening and emphasize Image details. High-Pass same as Low-pass calculate the average of a pixel and all of its eight neighbors [3] but High-Pass convolution attenuates low frequency of an image by allowing high frequency passing and changing the Low frequency with convolution average [4].

3.1 High-Pass Types:

1. **Ideal High-pass filters (IHPF)**
IHHPF is the simplest High-pass filter. It removes all frequencies below the cut-off frequency and leaves higher frequencies unchanged.

**HLPF transfer response equation** is:

\[
H(u, v) = \begin{cases} 
0 & \text{if } \sqrt{(u - M/2)^2 + (v - N/2)^2} \leq \text{Cutoff} \\
1 & \text{otherwise}
\end{cases}
\]  

(6)

Where:
- \(M\) is the number of rows in the image.
- \(N\) is the number of columns in the image.
- \((M/2, N/2)\) is the center of the frequency rectangle.

\[
\sqrt{(u - M/2)^2 + (v - N/2)^2}
\]

Is the transform point \((u, v)\) from the center.

**IHHPF frequency Curve**:

![Ideal High-pass filters frequency curve](image)

**Figure 4:** Ideal High-pass filters frequency

### 2. Butterworth High-pass filters (BHPF)

BHPF keeps frequencies outside radius cut-off and discards values inside. The transfer equation of order \((n)\) with cut-off frequency defined as:

\[
D_0 = \text{Cutoff}
\]  

(8)

\[
D(u, v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}
\]

(9)

\[
H(u, v) = \frac{1}{1 + \left[D_0 / D(u, v)\right]^{2n}}
\]

(10)

Where:
- \(M\) is the number of rows in the image.
- \(N\) is the number of columns in the image.
- \((M/2, N/2)\) is the center of the frequency rectangle.
Is the transform point $(u,v)$ from the center
N is the transfer order
D0 is Cutoff

**BHPF frequency Curve:**

![Butterworth High-pass filters frequency]

**Figure 5:** Butterworth High-pass filters frequency

### 3. Gaussian High-pass filters

GHPF has the same concept of ideal high pass filter, but the transition is smoother. GHPF is defined as:

GHPF transfer response equation is:

\[
D^2(u,v) = \left((u - M/2)^2 + (v - N/2)^2\right)
\]

\[
H(u, v) = 1 - e^{-D^2(u,v) / 2 \text{Cutoff}^2}
\]

Where:
- $M$ is number of rows in image
- $N$ is number of columns in image
- $(M/2, N/2)$ is center of frequency rectangle
- $D(u,v)$ is the transform point $(u,v)$ from the center

**GHPF frequency Curve:**

![Gaussian low-pass filter frequency]

**Figure 6:** Gaussian low-pass filter frequency

### 4. Convolution Kernel Method
Convolution is mathematical operation used in image processing operators. Convolution is a way of multiplying together two arrays of numbers with the same size or in different sizes producing a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values [5,4,8].

Kernel is a synonym for a matrix of numbers that is used in image convolutions, Kernels can have different size matrix to present different convolution numbers pattern. The kernel multiplying neighborhood pixels to get the average, and after we get the average each pixel is replaced by the average.

To understand the convolution kernel let us look to this example:
We have the 3*3 (h) kernel and f(x,y) as original Image element:

Now the convolution will be equation:

\[ H[f(x,y)] = \left\{ k1.f(x-1,y-1)+ k2.f(x-1,y)+ k3.f(x-1,y)+ k4.f(x-1,y)+ k5.f(x,y)+ k6.f(x,y+1)+ k7.f(x+y-1)+ k8.f(x+y, y1)+ k9.f(x+y, y+1) \right\} \]

\[ H[f(x, y)] \] it’s the convolution of original Image element (pixel).

Now we can get the pixel average by divide \[ H[f(x, y)] \] by 9

The most general Convolution average equation is:

\[ C_{xy} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \times m_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} m_{ij}} \]  

(m is the kernel ,p is the image element).
(i is row and j is column).

4.1 convolution Algorithm Step:

1. Convert the image to array.
2. Reads the (x, y) value of each pixel in the array
3. Multiplies (x, y) value by appropriate weight in kernel
4. Sums the products of (x, y value x weight) for the nine pixels, and divides sum by 9 (3*3 Kernel)
5. Derived value applied to center cell of array
6. Filter moves one pixel to right, and operation is repeated, pixel by pixel, line by line

4.2 Convolution Kernel Values:
The Kernel Values are not specific and not isotropic ,we can change this values dependence on the purpose of convolution ,in fact we can use the same kernel to smooth image or to sharpen image [8]. For example this Kernel is to detect isolated points from background.
Table 1: detect isolated Kernel

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

And this kernel is for smoothing image:

Table 2: smoothing Kernel

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

And this kernel is for harshly smooth image:

Table 3: harshly smooth Kernel

| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

If we went subtle smooth effect we can use this kernel

Table 4: subtle smoothing Kernel

|  0 | 1/8 |  0 |
| 1/8 | 1/2 | 1/8 |
|  0 | 1/8 |  0 |

Briefly By choosing different kernel, we can create filter has enough noise smoothing, without blurring the image too much. If any one of the pixels in the neighborhood has a noisy value, this noise will be besmeared over nine pixels and we will get blur image. Wherefore we must use a median convolution kernel.

5. Convolution Implementation in Java:

5.1. Java Kernel Class

The Java Kernel class defines a matrix that describes how a spatial domain pixel and its neighborhood pixels affect the average value computed for the pixel's position in the output image of a filtering operation. The X origin and Y origin indicate the kernel matrix element that corresponds to the pixel position for which an output value is being computed [6, 4].

5.1.1 Kernel Constructors:

Kernel(int width, int height, float[] data)

Where:

Width Convolution Kernel width
Height is Convolution Kernel height
Data is the Convolution Kernel matrix of float numbers.

*To use image kernel we must import java.awt.image.Kernel

5.2 java ConvolveOp Class

Java ConvolveOp class coded to execute the image convolution from original image to the filtering output image. Convolution applies a convolution kernel in a spatial domain operation that computes the average value for the output pixel, the multiplying operation take in repetitions step an input pixel and multiply by the kernel matrix. Convolution mathematically affected the output pixel with its immediate neighborhood according to the kernel values.

ConvolveOp class invoke Buffered Image object to present the image pixels data. Pixels data is the RGB color components multiplied with alpha component. If the Source Image has an alpha component, and the color components are not pre-multiplied with the alpha component, then the data are pre-multiplied before being convolved. If the Destination has color components which are not pre-multiplied, then alpha is divided out before storing into the Destination (if alpha is 0, the color...
components are set to 0). If the Destination has no alpha component, then the resulting alpha is discarded after first dividing it out of the color components [7,4].

5.2.1 filter method

Filter Method Performs a convolution on BufferedImage. Each component of the source image will be convolved (including the alpha component, if present). If the color model in the source image is not the same as that in the destination image, the pixels will be converted in the destination. If the destination image is null, a BufferedImage will be created with the source ColorModel. The IllegalArgumentException may be thrown if the source is the same as the destination [7,4].

5.3 Convolution Coding Example

In this example we will work on a noisy low quality image like the following baby image

![Figure 8: noisy low quality image](image)

Now we will Create the Convolution kernel to smoothing the baby image

<table>
<thead>
<tr>
<th>Table 5: smoothing Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/9</td>
</tr>
<tr>
<td>1/9</td>
</tr>
<tr>
<td>1/9</td>
</tr>
</tbody>
</table>

float val=1f/9f;
float[] data={ val, val, val, val, val, val, val, val, val };
Kernel kernel = new Kernel(3, 3, data);
Now we can load the original image using bufferedImage like this:
BufferedImage buff_original;
buff_original = ImageIO.read(new File("Baby.jpg");
It’s the time to use the BufferedImageOp class to create the ConvolveOp object which hold and buffer the a convolution kernel.
BufferedImageOp ConOp = new ConvolveOp(kernel);
The last step is to call the filter method:
buff_original = ConOp.filter(buff_original, null);
The Convolution result will be:
If we change the convolution Kernel to detect isolated points from background like this kernel:

**Table 6: detect isolated Kernel**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

We will get this result

**Figure 9: Convolution smooth image**

**Figure 10: Convolution Detect isolated point image**

5.4 Convolution Test Class:

```java
import java.awt.image.BufferedImage;
import java.awt.image.BufferedImageOp;
import java.awt.image.ConvolveOp;
```
import java.awt.image.Kernel;
import java.awt.Image;
import javax.swing.ImageIcon;
import java.awt.*;
import javax.swing.JPanel;
import java.awt.event.*;
import java.io.*;
import javax.imageio.*;

public class Convolution {
    public static void main(String[] argv) throws Exception {
        try {
            BufferedImage buff_original = ImageIO.read(new File("Baby.jpg"));
            float val = 1f / 9f;
            float[] data = {val, val, val, val, val, val, val, val, val};
            Kernel kernel = new Kernel(3, 3, data);
            BufferedImageOp ConOp = new ConvolveOp(kernel);
            buff_original = ConOp.filter(buff_original, null);
            JPanel content = new JPanel();
            content.setLayout(new FlowLayout());
            // label to load image
            content.add(new JLabel(new ImageIcon(buff_original)));
            JFrame f = new JFrame("Convolution Image ");
            f.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
            f.setContentPane(content);
            f.pack();
            f.setVisible(true);
        } catch (IOException e) {
            // catch (IOException e) {
        }
    }
}

IV. CONCLUSION

In this paper, we discussed the Low-Pass and High-Pass Image filtering, after that we explained the Convolution Kernel and how to use java utilities to implement the convolution technique in easy way. In convolution operation we can produce both filters (Low and High) with the same step but there is a substantial deferent in the kernel, each type of filter and it’s sub type need deferent kernel data, in fact even with the same type of filter we need various data kernel that suitability the original image frequency domains data, by other word the Kernel must be adjustable in easy compatible way to adjust the filtering purpose that fit the original image. Smoothing image without low-pass filter or sharpening image without high-pass filter is not that easy. But fortunately Java provide us image filtering utilities which can process and smoothing or sharpening image in very simple easy way. All required from us to create the kernel matrix and let java ConvolveOp to complete the complex convolution lopping steps to image frequency domains and replace the pixels with average.

V. Future Work

For future work we will implement the convolution java technique in tangible application with user friendly interface which help the user to suggests and modify the kernel array data in easy way to fit
the filtering purpose, with this application user can select the original image from his pc and to save the filtered image. The next future research will be about image encryption and decryption algorithms.

REFERENCES

[3] Image Processing - Laboratory 9: Image filtering in the spatial and frequency domains , Technical University of Cluj-Napoca
http://docs.oracle.com/javase/7/docs/api/java/awt/image/Kernel.html
http://docs.oracle.com/javase/7/docs/api/java/awt/image/ConvolveOp.html

AUTHORS BIOGRAPHY

O. K. Khorsheed born in Baghdad in 1974. BSC computer science from al-mustansiriya University. Higher diploma degree from Informatics Institute of Higher Studies. Master degree from Jordan (Arab academic for banking and financial science) school of (IT) computer information system. Lecturer in koya University since 2005 in school of engineering SiW engineering department.