Neighborhood operations processing

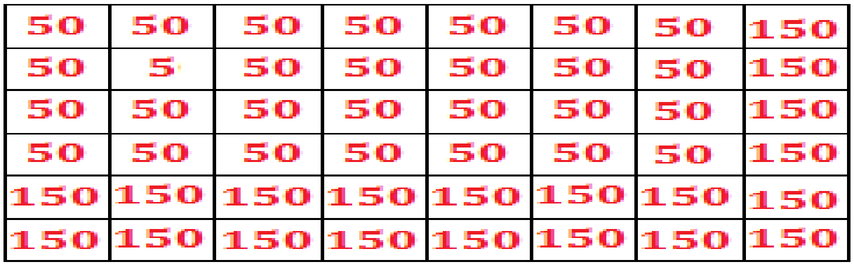
- Neighborhood processing enhances the image in a way doesn't possible by point processing such as:
  - Smoothing →linear low pass filter
  - Salt and pepper noise → order statistical filter
  - Edge detection (sharpening)→ linear high pass filter

Order-Statistics Filter:

- Idea → it replacing the value of the center pixel with the value determined by the ranking result.
- Types → Min, Max or Median filter
- Advantages →remove salt and pepper noise while preserved the Edges

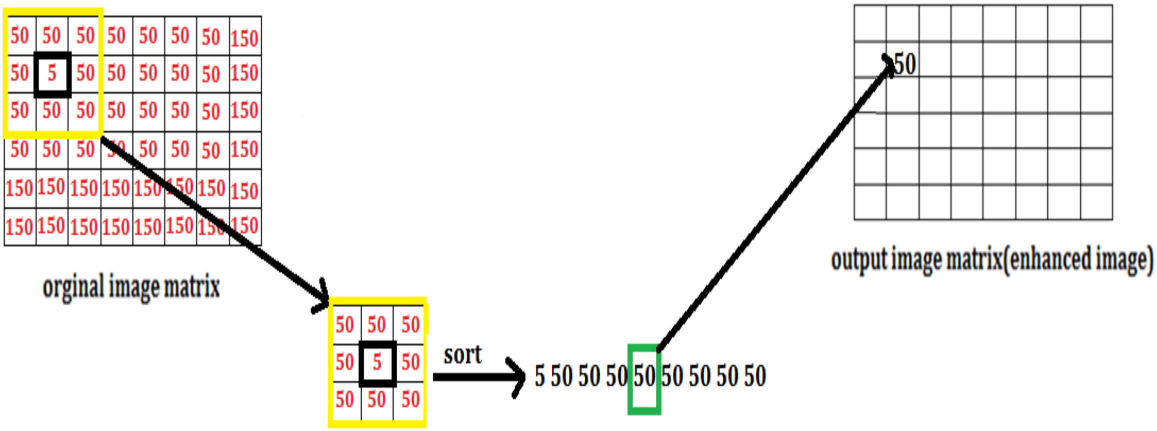
Example

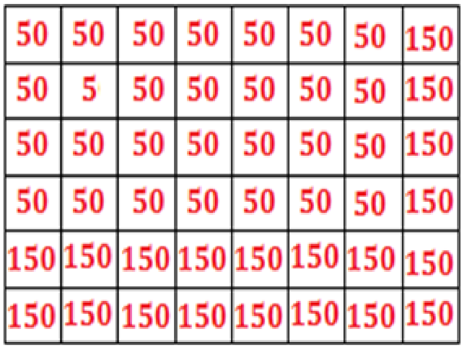
for the following image matrix apply 3*3 Median filter, once by **ignoring padding then after zero padding, comment on the result?**



50	50	50	50	50	50	50	150
50	5	50	50	50	50	50	150
50	50	50	50	50	50	50	150
50	50	50	50	50	50	50	150
150	150	150	150	150	150	150	150
150	150	150	150	150	150	150	150

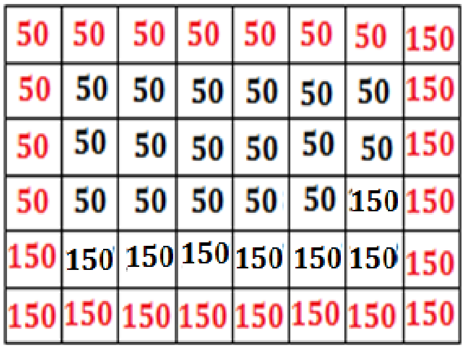
- Ignoring padding





50	50	50	50	50	50	50	150
50	5	50	50	50	50	50	150
50	50	50	50	50	50	50	150
50	50	50	50	50	50	50	150
150	150	150	150	150	150	150	150
150	150	150	150	150	150	150	150

orginal image matrix

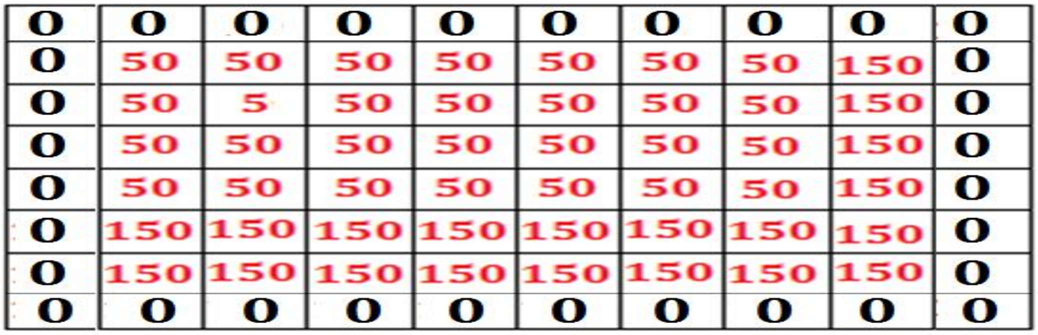


50	50	50	50	50	50	50	150
50	50	50	50	50	50	50	150
50	50	50	50	50	50	50	150
50	50	50	50	50	50	150	150
150	150	150	150	150	150	150	150
150	150	150	150	150	150	150	150

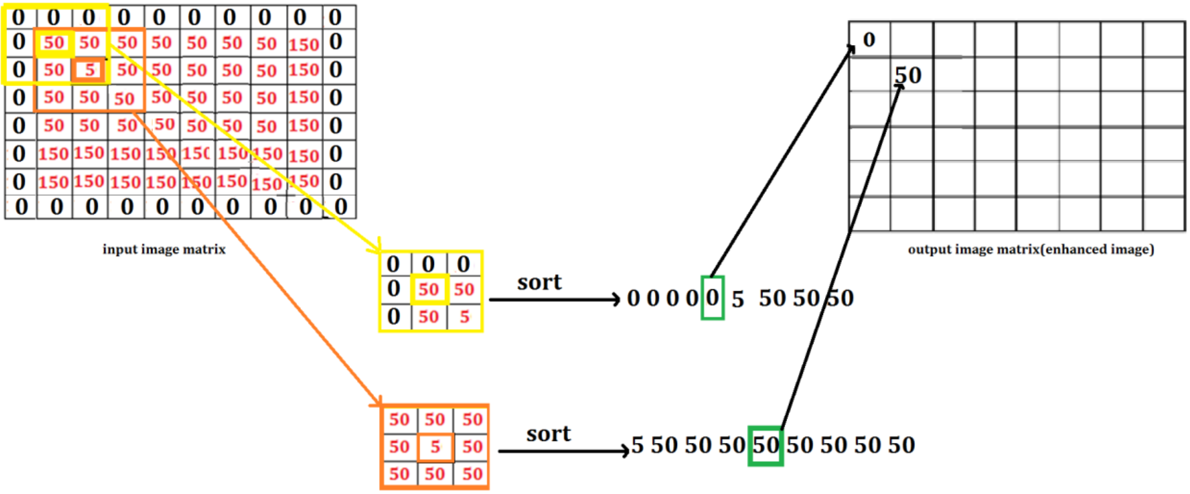
output image matrix(enhanced image)

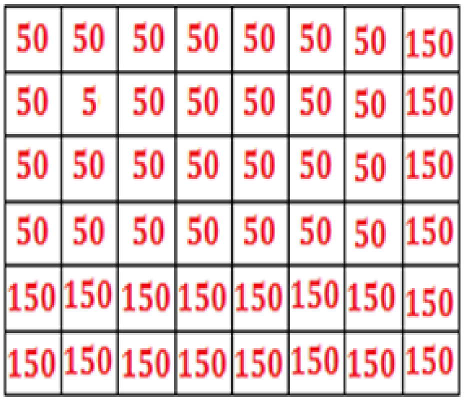
Median filter not only enhanced the spark noise at P(1,1) but also preserve the edges

- **After zero padding**
  - After zero padding (Null values) The matrix rows and columns are extended out with zeros.



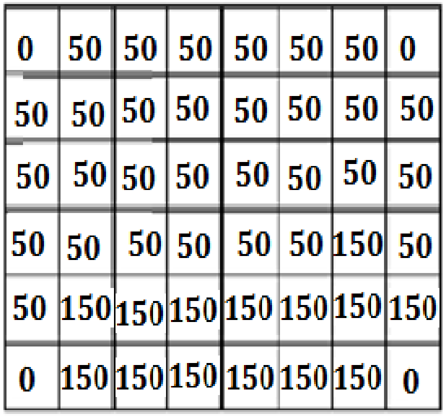
0	0	0	0	0	0	0	0	0	0
0	50	50	50	50	50	50	50	150	0
0	50	5	50	50	50	50	50	150	0
0	50	50	50	50	50	50	50	150	0
0	50	50	50	50	50	50	50	150	0
0	150	150	150	150	150	150	150	150	0
0	150	150	150	150	150	150	150	150	0
0	0	0	0	0	0	0	0	0	0





50	50	50	50	50	50	50	150
50	5	50	50	50	50	50	150
50	50	50	50	50	50	50	150
50	50	50	50	50	50	50	150
150	150	150	150	150	150	150	150
150	150	150	150	150	150	150	150

orginal image matrix



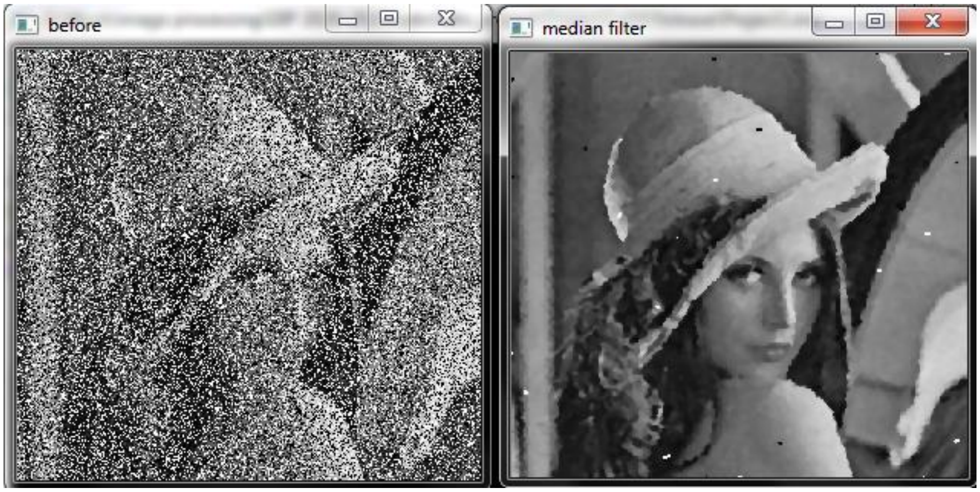
0	50	50	50	50	50	50	0
50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50
50	50	50	50	50	50	150	50
50	150	150	150	150	150	150	150
0	150	150	150	150	150	150	0

output image matrix(enhanced image)

//medianBlur (src, dst, ksize);

- **Src** → A Mat object representing input image for this operation.
- **dst** → A Mat object representing output image for this operation.
- **ksize**→ representing the kernel size.

```
#include<opencv2/opencv.hpp>
#include"iostream"
using namespace std;
using namespace cv;
int main()
{
  Mat srcM, dst;
  srcM = imread("med.jpg", 0);
  namedWindow("before median filter", 0);
  imshow("before median filter", srcM);
  waitKey(0);
  medianBlur(srcM, dst, 3);
  namedWindow("median filter", 0);
  imshow("median filter", dst);
  waitKey(0);

}
```



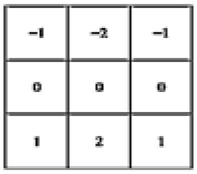
Linear high pass filter:

- It’s called Derivative as it produces values only if there are a change so all masks coefficients sum to zero.
- It try to find edge by finding sharp changes in intensities
- Disadvantages → it sensitive to the noise (increase spark noise)

  So usually it needs a smooth filter before detecting edge

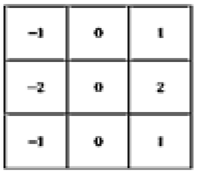
Filters kernels:

- Sobel operators→



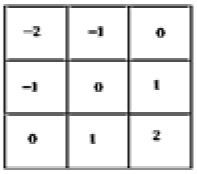
-1	-2	-1
0	0	0
1	2	1

Gy



-1	0	1
-2	0	2
-1	0	1

Gx

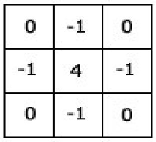


-2	-1	0
-1	0	1
0	1	2

D

$$\text{mag} = \sqrt{Gx^2 + Gy^2 + D^2}$$

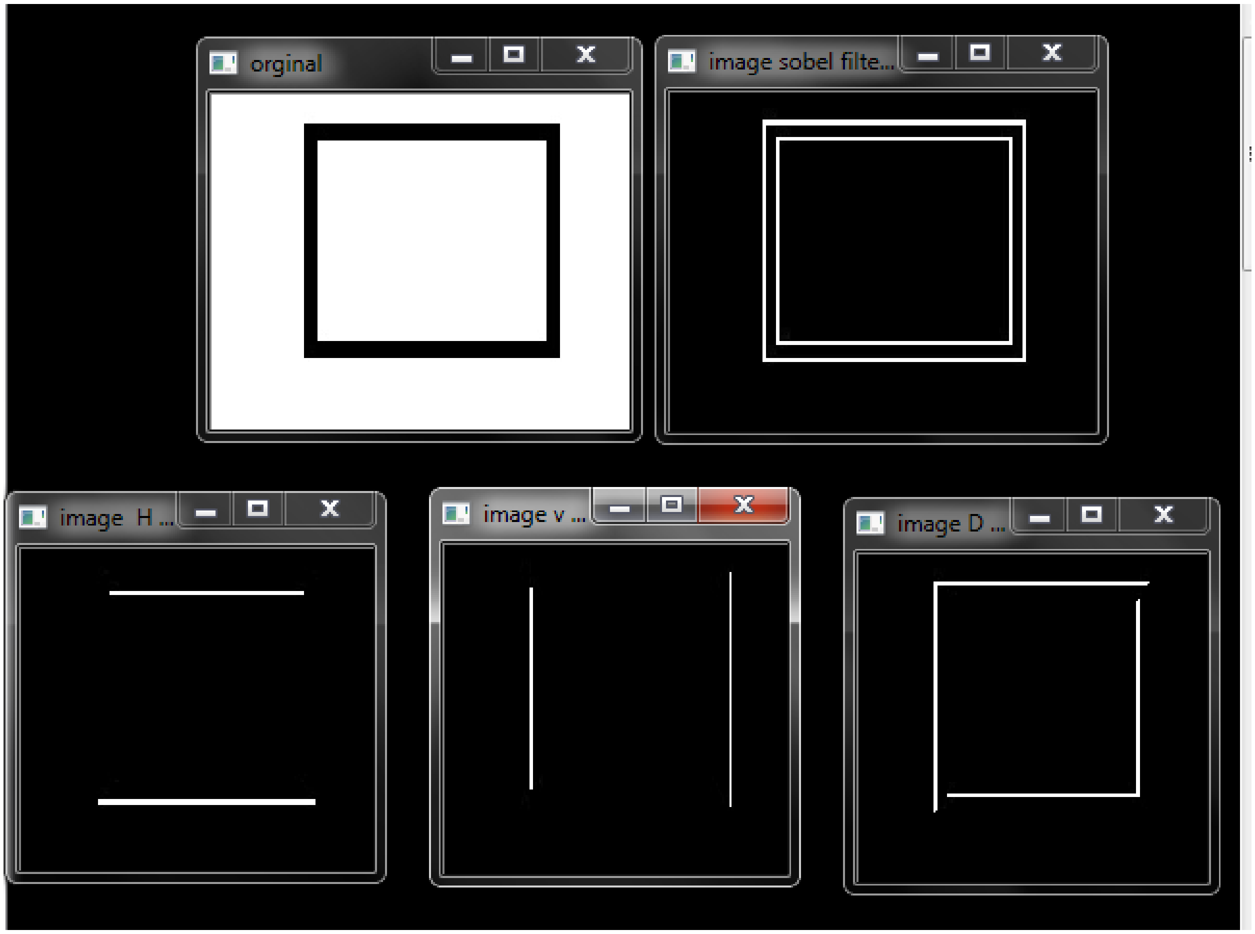
- Laplacian operator →



0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

```
#include<opencv2/opencv.hpp>
#include"iostream"
using namespace std;
using namespace cv;
int main()
{
	////////////// linear high pass filter_ sobel filter////////////////////
	Mat src12, dst12_2, dst12_3, dst12_4, dst12_5, dst12_6;
	src12 = imread("xsobel.jpg", 0);
	namedWindow("orginal", 0);
	imshow("orginal", src12);

	Mat kernel_TH = (Mat_<int>(3, 3) << -1, -2, -1, 0,0,0, 1, 2, 1);
	filter2D(src12, dst12_2, CV_8UC1, kernel_TH);
	namedWindow("image  H filter", 0);
	imshow("image  H filter", dst12_2);

	Mat kernel_TV = (Mat_<int>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
	filter2D(src12, dst12_3, CV_8UC1, kernel_TV);
	namedWindow("image v filter", 0);
	imshow("image v filter", dst12_3);

	Mat kernel_Td= (Mat_<int>(3, 3) << 2, 1, 0, 1, 0, -1, 0, -1, -2);
	filter2D(src12, dst12_4, CV_8UC1, kernel_Td);
	namedWindow("image D filter", 0);
	imshow("image D filter", dst12_4);

	addWeighted(dst12_2, 1, dst12_3, 1, 0, dst12_5);

	addWeighted(dst12_4, 1, dst12_5, 1, 0, dst12_6);
	namedWindow("image sobel filter_f", 0);
	imshow("image sobel filter_f", dst12_6);
	waitKey(0);

	////////////// linear high pass filter laplacian_filter///////////////////
	Mat  dst12g, dst12_L;
	Mat src122 = imread("xsobel.jpg", 0);
	namedWindow("orginal", 0);
	imshow("orginal", src122);
	waitKey(0);
	Mat kernel_L = (Mat_<int>(3, 3) << 0,-1,0,-1,4,-1,0,-1,0);
	filter2D(src122, dst12_L, CV_8UC1, kernel_L);
	Laplacian(src122, dst12_L, CV_8UC1);
	namedWindow("Laplacian", 0);
	imshow("Laplacian", dst12_L);
	waitKey(0);
}
```


orginal
image sobel filte...
image H ...
image v ...
image D ...

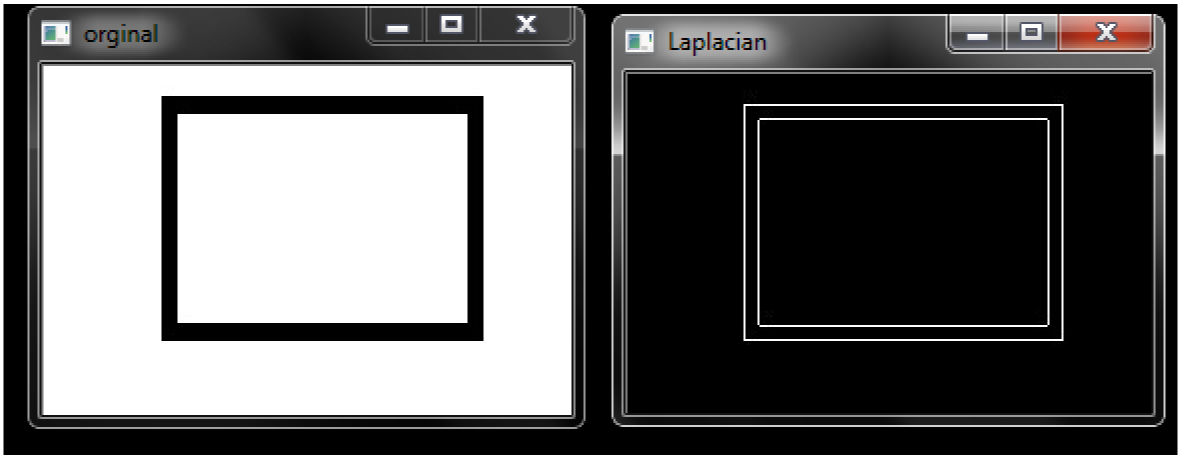

orginal
Laplacian

Image segmentation:

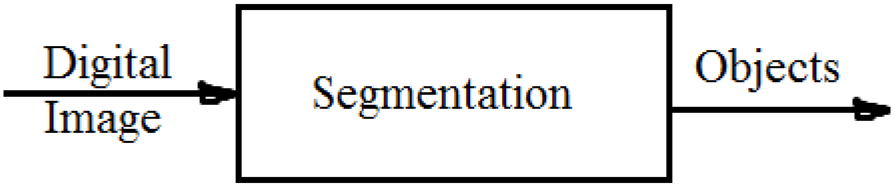


Image segmentation process: separates the objects from the background and also separation between objects

Segmentation Methodologies:

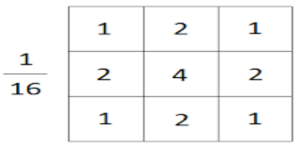
1) Thresholding: we choose a specific value and make it the Threshold and what is before it is considered object
2) Edge Based: use derivative filter, makes a high pass filter to detect the edges to get the object off the image
3) Region Based

Global Thresholding (object & background)

$$g(x,y)=\begin{cases}255 & if \quad f(x,y)>T\\ 0 & if \quad f(x,y)\le T\end{cases}$$

Edge Based:

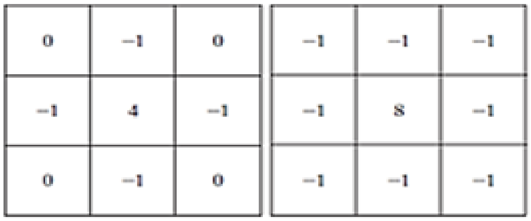
Smoothing → Derivative filters → edge detection



$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

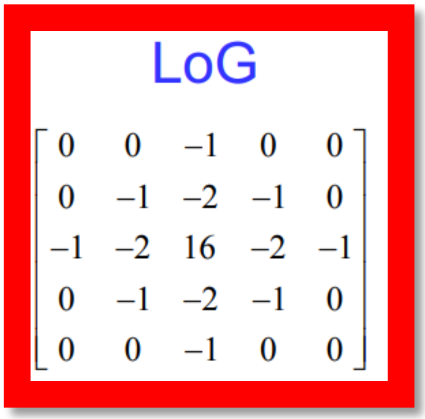
3 x 3 Gaussian Kernel



0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

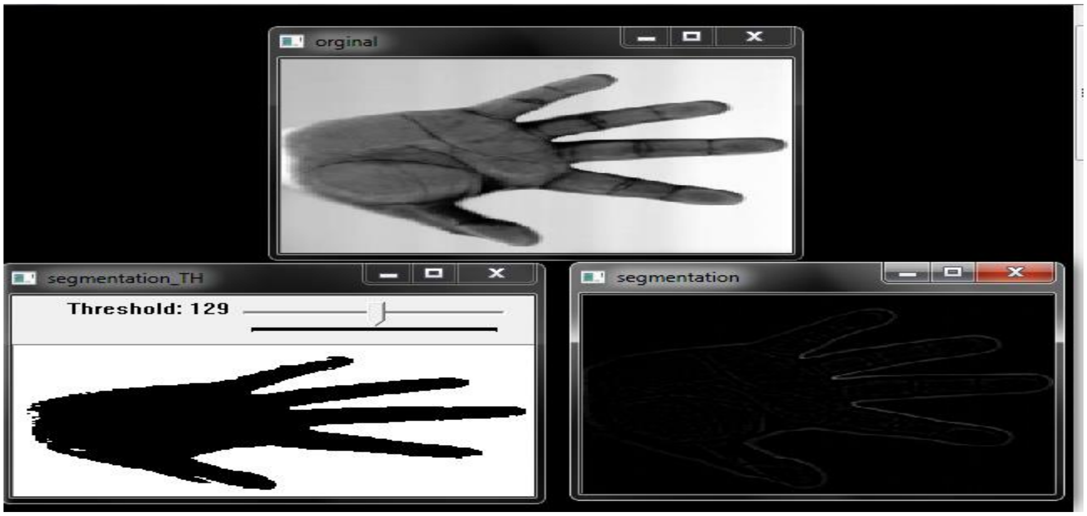
Filter mask used to implement the digital Laplacian.



LoG

$$\begin{bmatrix}0 & 0 & -1 & 0 & 0\\ 0 & -1 & -2 & -1 & 0\\ -1 & -2 & 16 & -2 & -1\\ 0 & -1 & -2 & -1 & 0\\ 0 & 0 & -1 & 0 & 0\end{bmatrix}$$

```
#include<opencv2/opencv.hpp>
#include"iostream"
#include"math.h"
using namespace std;
using namespace cv;
int main()
{
                //////////////////////////image segmentation//////////
        ////////////////////////////////thresholding method/////////////////
                Mat src,dst;
                src = imread("seg.jpg", 0);
                int th = 10;
                for (int x = 0; x < 3; x++)
                {
                        threshold(src, dst, th, 255, THRESH_BINARY);
                        namedWindow("segmentation_TH", 0);
                        createTrackbar("Threshold", "segmentation_TH", &th, 255);
                        imshow("segmentation_TH", dst);
                        waitKey(0);
                }
        //////////////////////////////////////////////edge based method///////////////////
        Mat  dst12g, dst12_L;
        Mat src122 = imread("seg.jpg", 0);
        namedWindow("orginal", 0);
        imshow("orginal", src122);
        waitKey(0);
        ////////////////////////gaussian////////////////
        Mat kernel_G= (Mat_<float>(3, 3) << 1,2,1,2,4,2,1,2,1);
        kernel_G=kernel_G/16;
        filter2D(src122, dst12g, CV_8UC1, kernel_G);
        //GaussianBlur(src122, dst12g, Size(3, 3), 0);
        namedWindow("gaussian", 0);
        imshow("gaussian", dst12g);
        waitKey(0);
        //////////////////////////laplacian////////////
        Mat kernel_L = (Mat_<int>(3, 3) << 0,-1,0,-1,4,-1,0,-1,0);
        filter2D(dst12g, dst12_L, CV_8UC1, kernel_L);
        //Laplacian(dst12g, dst12_L, CV_8UC1);
        namedWindow("segmentation", 0);
        imshow("segmentation", dst12_L);
        waitKey(0);
}
```


orginal
segmentation_TH
Threshold: 129
segmentation