EXPERIMENT - 3

Report Submission

<u>Topic</u> : Designing Spatial Filters (Mean, Median, Prewitt, Laplacian, Sobel, Gaussian, Laplacian of Gaussian) for enhancing grayscale images

GROUP - 18

Name	Sumegh Roychowdhury	Arun Sammit Pandey
Roll Number	17EC35033	17EC35006

Table of Contents

Introduction:

We use spatial filters for image enhancement. There are broadly two types of spatial filters, Smoothing and Sharpening. Smoothing filters are used for blurring and reducing noise. Sharpening filters are mainly used for highlighting the transitions in intensity. Generally for smoothing we replace each pixel of image with some form of average value of its neighbourhood pixels. We only consider filters with odd kernel size to avoid the ambiguity incase of even sized kernels.
The common smoothing spatial filters that we use in this assignment are

1. Arithmetic Mean Filter [2]
2. Median Filter (Order-Statistic Filter) [2]
3. Gaussian Filter [1]

The common sharpening spatial filters that we use in this assignment are

1. Prewitt Filter[2]
2. Laplacian Filter [2]
3. Sobel Filter[2]
4. Laplacian of Gaussian (LoG) [2]

Mean Filter:
The resultant pixel is the average value of the neighbour pixels including the pixel itself. This helps in reducing the intensity of outlier pixel values. The kernel size can be increased in case more smoothening is desired. This filter can be implemented using the below equation where we take a window of size $m \times n$ centered at the point (x,y). $g(s,t)$ is corrupted image and $f(x,y)$ is the intensity value of enhanced image at point (x,y).

$$f(x,y) = 1/(mn) * \sum_{(s,t)\in S_{xy}} g(s,t)$$

Median Filter
In Median Filter we replaces the value of a pixel by the median of the intensity values in the neighborhood of that pixel .We also include the original value of the pixel in the computation of the median. They are quite effective for certain types of random noises like impulse noise or salt and pepper noise.

$$f(x,y) = median\{g(s,t)\} \ \forall (s,t) \ \epsilon S_{xy}$$

Gaussian Filter
In Gaussian Filter we use a filter mask that has values sampled from a gaussian function of two variables which has a form like:

$$h(x,y) = e^{-(x^2+y^2)/(2*\sigma^2)}$$

Here σ is standard deviation. To generate a 3×3 mask we can take $w_1 = h(-1,-1),\ w_2 = h(-1,0)\ ,\ ...,\ w_9 = h(1,1)$. Similarly we can generate masks for other dimensions.

Prewitt and Sobel Filter

In these filters, we use horizontal and vertical prewitt/sobel operators to calculate the gradients in x and y direction respectively. After applying the horizontal and vertical operator we can calculate the magnitude of the two gradients by using the equation

$$M(x,y) = \sqrt{{g_x}^2 + {g_y}^2}$$

Then, $M(x,y)$ will tell us the location of edges.

The 3×3 prewitt operators are [2]

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

The 3×3 sobel operators are [2]

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel

We can also have other variants of Prewitt and Sobel filters so that they have their strongest response along diagonal direction. [2]

0	1	1	−1	−1	0
−1	0	1	−1	0	1
−1	−1	0	0	1	1

Prewitt

0	1	2	−2	−1	0
−1	0	1	−1	0	1
−2	−1	0	0	1	2

Sobel

Laplacian Filter

It is the simplest possible isotropic derivative operator, which can be defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

We can express it in discrete form by using finite difference in place of derivatives to get the kernel as

0	1	0
1	−4	1
0	1	0

To incorporate the diagonal directions in this operator we can also use a modified kernel, which is

1	1	1
1	−8	1
1	1	1

Laplacian of Gaussian (LoG) filter

It is an edge detection filter in which we (theoretically) apply the gaussian operator and then the laplacian operator. The main advantage of this operator is that a LoG operator of large kernel size can be used to detect blurry edges and LoG operators of small kernel size can be used to detect sharply focused fine details. The analytical expression of LoG operator is

$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Masks of arbitrary size can be generated from above equation by sampling above equation and then scaling the coefficients so that they sum to zero

Algorithm:

We use correlation with the filter masks of desired sizes to calculate the output image in each case. For example for an image at each pixel location (except for boundary pixels), given the kernel, we assign the pixel value as follows:

```
while (x < width)
{
    y = 0;
    while (y < width)
    {
        int tmp =
            input.at<uchar>(i + x - width / 2, j + y - width / 2)
                * kernel[x][y];
        value += tmp;
        y++;
    }
    x++;
}
```

Here width is the kernel size.

For some filters like Laplacian of Gradient and Diagonal Sobel we have hardcoded the values of kernel for the given sizes (3×3 for sobel and 7×7 for Laplacian of Gradient).

For other linear filters we have calculated the kernels based on the width of the kernel given/required.

For example while calculating horizontal kernel for sobel filter, we defined two vectors, p and q.

Where we calculated $p[i]$ as

$p[i] = exp(-r/(2 * sigma * sigma)) * k$

Here $k = 1/(sqrt(2.0 * M\_PI) * sigma)$ and $r = (i - width/2) * (i - width/2)$

And $q[i]$ as:

```
if (i < width / 2)
    q[i] = -1;
else if (i > width / 2)
    q[i] = 1;
else
    q[i] = 0;
i++;
```

After calculating $q[i]$ and $p[i]$, we found out the kernel by using the formulae:

$kernel[i][j] = p[i] * q[j]$

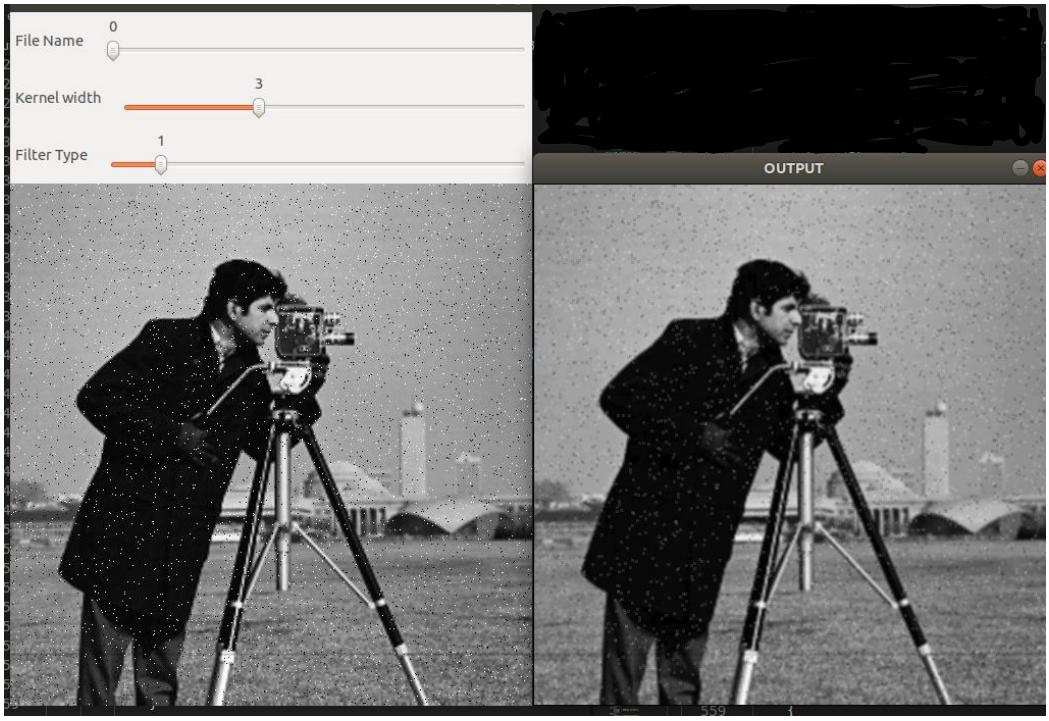
Since Median Filter is a non linear filter, we calculated the output image response by explicitly finding the median corresponding to the given neighbourhood width. For this, we used a

temporary array in which we took all the required neighbourhood values of the current pixel, sorted the temporary array and took the value at middle index as the output intensity value for output image

Results :

For each subsection, we have an output image on the right side and input image on the left side. The kernel width is also shown in the figure itself.

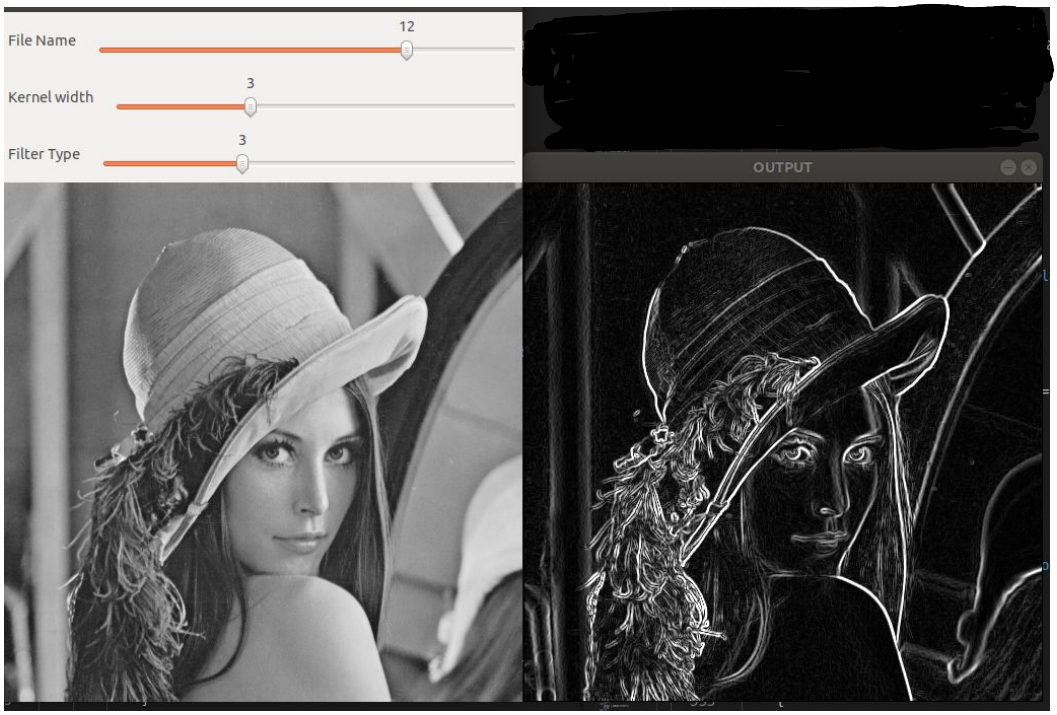
Arithmetic Mean Filter:



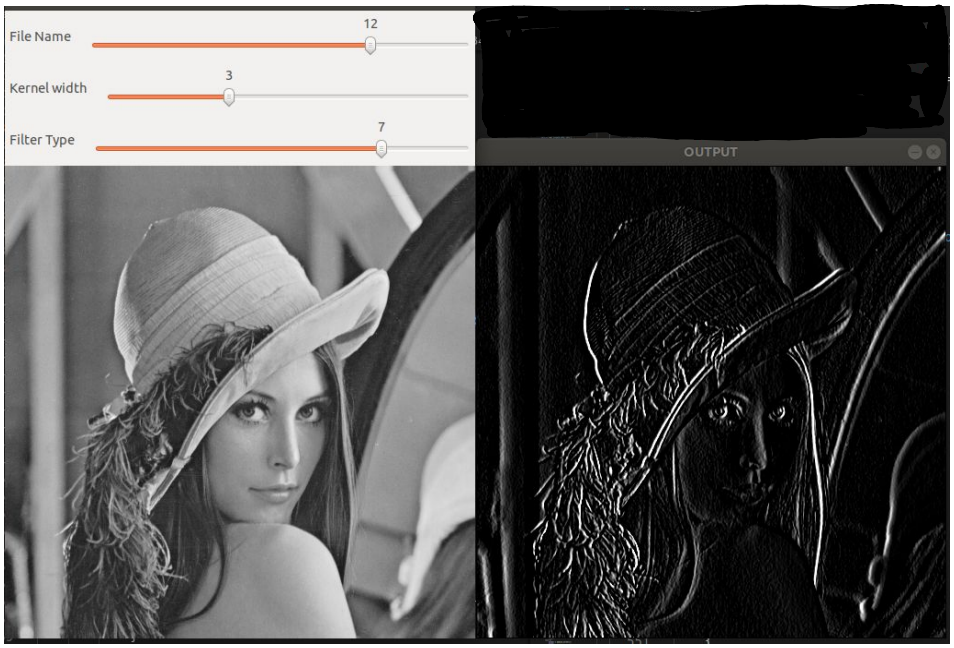
Median Filter

Gaussian Filter

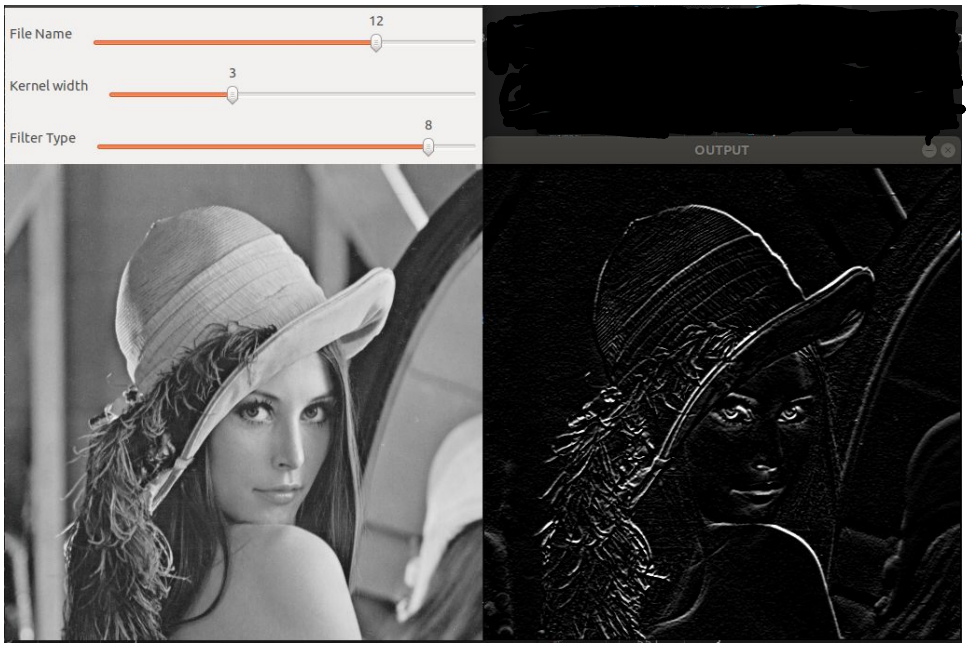
Prewitt Filter (Magnitude only):



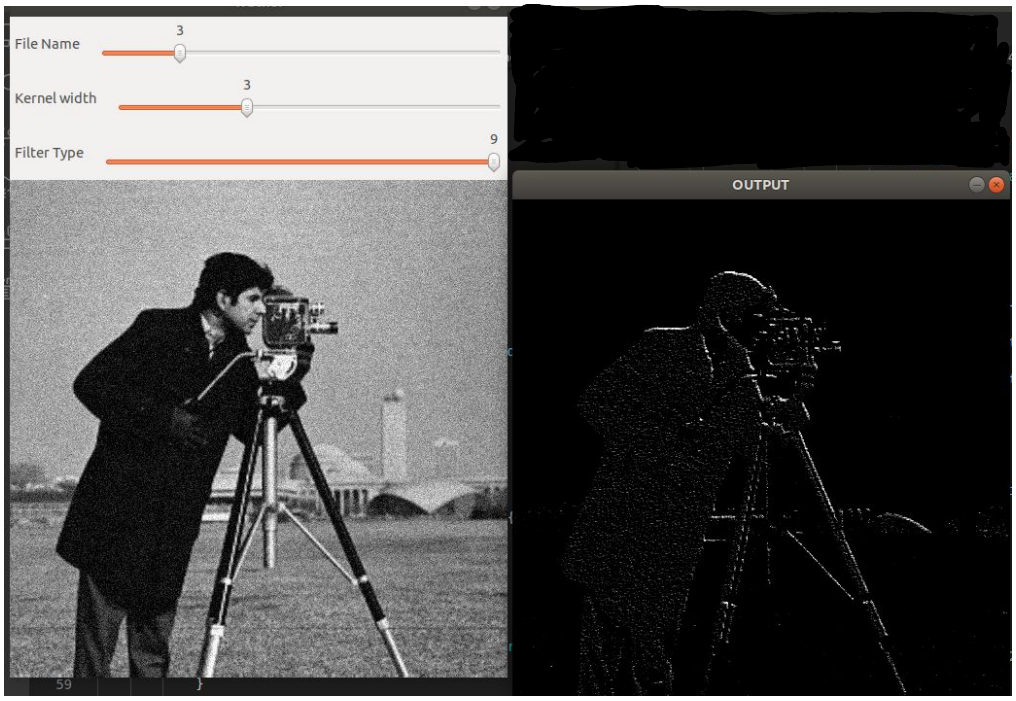
Sobel Filter (Vertical):



Sobel Filter (Horizontal):

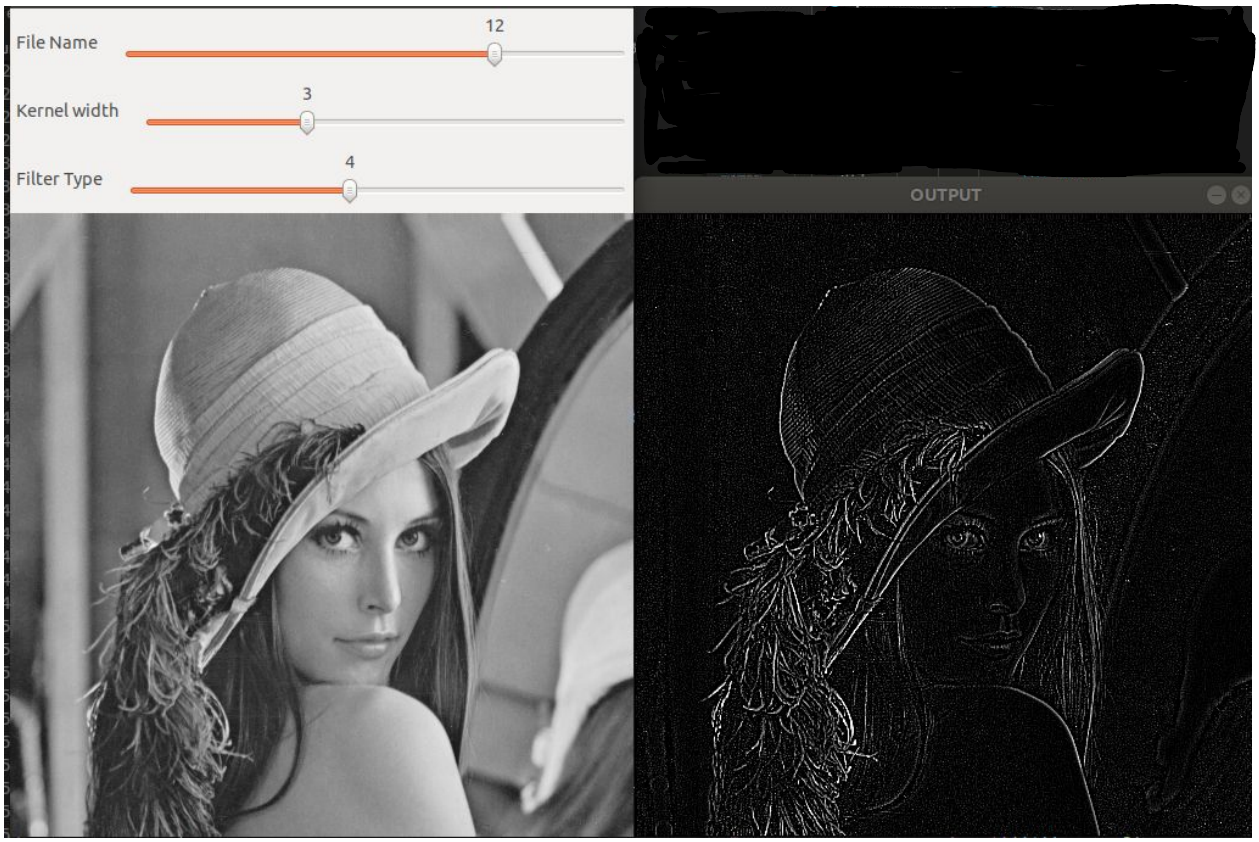


Sobel Filter(Diagonal):



Laplacian of Gaussian:

Laplacian



Analysis :

1. From the output images, we can see that arithmetic mean filter is not effective in removing the salt and pepper or impulse noises.
2. It is also apparent that Median Filter performs quite well in case of salt and pepper or impulse noises
3. We also found out that the Gaussian filter performs quite well in removing gaussian noises from images.
4. From the output images we can also see that the horizontal sobel filters primarily detects the edges which are parallel to x axis (horizontal axis) and vertical sobel filters primarily detects the edges which are parallel to y axis (vertical axis).
5. It is also apparent from the output images that the diagonal sobel filters primarily detects the diagonal edges.
6. The Laplacian of Gaussian is most effective in detecting prominent edges and neglecting the structures (including noise) which are much smaller than the standard deviation used for the Gaussian part.
7. The prewitt filter (magnitude) also works great for detecting the edges but it has a limitation that it involves the computation of square root of the two pixel intensity squared.
8. The Laplacian filter is better than the prewitt filter (magnitude) as it is not as computationally expensive as prewitt filter (magnitude). However we don't get the direction of change of pixel intensity in laplacian, whereas we get that information in case of prewitt filter.
9. For the sobel kernel, we used gaussian and gradient row and column matrix decomposition.

Contributions:

Sumegh: Arithmetic Mean Filter, Median Filter (Order-Statistic Filter), Gaussian Filter, Prewitt Filter
Arun: Laplacian, Sobel (horizontal, vertical, diagonal), Laplacian of Gaussian,

References

[1] https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
[2]https://www.pearson.com/us/higher-education/program/Gonzalez-Digital-Image-Processing-4th-Edition/PGM241219.html
[3]https://en.wikipedia.org/wiki/Prewitt_operator
[4]https://docs.opencv.org/