INDIAN INSTITUTE OF TECHNOLOGY

KHARAGPUR

DEPARTMENT OF ELECTRONICS AND ELECTRICAL COMMUNICATION

IMAGE PROCESSING LABORATORY

ASSIGNMENT NUMBER 3

ARNAB BISWAS

21EC65R01

M.Tech.

2021 – 2022

Introduction:

Spatial filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement. Filtering is a neighbourhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighbourhood of the corresponding input pixel. A pixel's neighbourhood is some set of pixels, defined by their locations relative to that pixel. A filter is defined by a kernel, which is a small array applied to each pixel and its neighbours within an image. In most applications, the centre of the kernel is aligned with the current pixel, and is a square with an odd number (3, 5, 7, etc.) of elements in each dimension. The process used to apply filters to an image is known as convolution, and may be applied in either the spatial or frequency domain. In this experiment I have used different size of kernel e.g. 3x3, 5x5, 7x7 and 9x9 to produce the output of the filters- Mean, Median, Prewitt, Laplacian, Sobel kernels (horizontal, vertical, diagonal), Gaussian Blur, Laplacian of Gaussian on a stack of grayscale images.
Mean Filter: This filter is used for high-frequency noise reduction and image smoothing. 3x3 Kernel =

$$\frac{1}{9}\times\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Median Filter: This is a non-linear filter used for noise reduction. It finds particular use in Salt & Pepper noise reduction. For applying it, we find the median of all the values covered my mask and assign it to the centre. Unlike mean filter, it preserves edges.

Gradient Filter: As evident by the name, this filtering operation is used to find directional change in intensity in the image. This helps in detection of edges in the image. Edge detection helps in feature extraction and sharpening of the image.

Laplacian Filter: Laplacian filter works on the principle of Laplace operator i.e. it uses second-order derivative instead of first order derivative to detect edges.

The kernel of this filters are as follows:

```
prewitt_3[3][3] =                    { {-1, -1, -1 },
                                       { 0,  0,  0 },
                                       { 1,  1,  1 } };

prewitt_5[5][5] = { { -1, -1, -1, -1, -1 },
                    { -2, -2, -2, -2, -2 },
                    {  0,  0,  0,  0,  0 },
                    {  2,  2,  2,  2,  2 },
                    {  1,  1,  1,  1,  1 } };

prewitt_7[7][7] = { { -1, -1, -1, -1, -1, -1, -1 },
                    { -2, -2, -2, -2, -2, -2, -2 },
                    { -3, -3, -3, -3, -3, -3, -3 },
                    { 0, 0, 0, 0, 0, 0, 0 },
                    { 3, 3, 3, 3, 3, 3, 3 },
                    { 2, 2, 2, 2, 2, 2, 2 },
                    { 1, 1, 1, 1, 1, 1, 1 } };
prewitt_9[9][9] = { { -1, -1, -1, -1, -1, -1, -1 },
```

```
                                        { -2, -2, -2, -2, -2, -2, -2 },
                                        { -3, -3, -3, -3, -3, -3, -3 },
                                        { 0, 0, 0, 0, 0, 0, 0 },
                                        { 3, 3, 3, 3, 3, 3, 3 },
                                        { 2, 2, 2, 2, 2, 2, 2 },
                                        { 1, 1, 1, 1, 1, 1, 1 } };

gradient_V_3[3][3] = { { 1, 0, -1 },
                                    { 1, 0, -1 },
                                    { 1, 0, -1 } };

gradient_V_5[5][5] = { {-1,-2,0,2,1},
{-1,-2,0,2,1},
{-1,-2,0,2,1},
{-1,-2,0,2,1},
{-1,-2,0,2,1}
};

gradient_V_7[7][7] = { {-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1} };
gradient_V_9[9][9] = { {-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1},
{-1,-2,-3,0,3,2,1} };


laplacian_3[3][3] = { {-1, -1, -1},
                                                {-1, 8, -1},
                                                {-1, -1, -1} };

laplacian_5[5][5] = { {-1, 3, -4, -3, -1},
                                        {-3, 0,  6,  0, -3},
                                        {-4, 6, 20,  6, -4},
                                        {-3, 0,  6,  0, -3},
                                        {-1,-3, -4, -3, -1} };

laplacian_7[7][7] = { {-2, -3, -4, -6, -4, -3, -2},
                                        {-3, -5, -4, -3, -4, -5, -3},
                                        {-4, -4,  9, 20,  9, -4, -4},
```

```
                                        {-6, -3, 20, 36, 20, -3, -6},
                                        {-4, -4,  9, 20,  9, -4, -4},
                                        {-3, -5, -4, -3, -4, -5, -3},
                                        {-2, -3, -4, -6, -4, -3, -2} };

laplacian_9[9][9] = {
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
        {1,1,1,1,-16,1,1,1,1},
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
        {0,0,0,0,1,0,0,0,0},
                                         };

sobel_H_3[3][3] = { {  1,  2,  1},
                                                { 0,  0,  0},
                                                { -1, -2, -1} };

sobel_H_5[5][5] = { { 1,   4,   7,   4,  1},
                                                { 2,  10,  17,  10,  2},
                                                { 0,   0,   0,   0,  0},
                                                {-2, -10, -17, -10, -2},
                                                { -1, -4, -7, -4, -1 } };

sobel_H_7[7][7] = { {  1,  4,  9,  13,  9,  4,  1},
                                                {  3, 11, 26,  34, 26, 11,  3},
                                                {  3, 13, 30,  40, 30, 13,  3},
                                                {  0,  0,  0,   0,  0,  0,  0},
                                                { -3, -13,-30,-40,-30,-13, -3},
                                                { -3, -11,-26, 34,-26,-11, -3},
                                                { -1, -4, -9, -13, -9, -4, -1} };

sobel_H_9[9][9] = { {-4,-4,-4,-4,-8,-4,-4,-4,-4},
                                                {-3,-3,-3,-3,-6,-3,-3,-3,-3},
                                                {-2,-2,-2,-2,-4,-2,-2,-2,-2},
                                                {-1,-1,-1,-1,-2,-1,-1,-1,-1},
                                                { 0, 0, 0, 0, 0, 0, 0, 0, 0},
                                                {1, 1, 1, 1, 2, 1, 1, 1, 1},
                                                {2, 2, 2, 2, 4, 2, 2, 2, 2},
                                                {3, 3, 3, 3, 6, 3, 3, 3, 3},
                                                {4, 4, 4, 4, 8, 4, 4, 4, 4}};

sobel_V_3[3][3] = { { -1, 0, 1 },
                                                { -2, 0, 2 },
```

```
                                        { -1, 0, 1 } };


sobel_V_5[5][5] = { { -1,  -2, 0,  2, 1 },
                                        { -4, -10, 0, 10, 4 },
                                        { -7, -17, 0, 17, 7 },
                                        { -4, -10, 0, 10, 4 },
                                        { -1,  -2, 0, -2, 1 } };



sobel_V_7[7][7] = { { -1,  -3,  -3, 0, 3,  3,   1},
                                        { -4, -11, -13, 0, 13, 11,  4},
                                        { -9, -26, -30, 0, 30, 26,  9},
                                        { -13, -34,-40, 0, 40, 34, 13},
                                        {  -9, -26,-30, 0, 30, 26,  9},
                                        {  -4, -11,-13, 0, 13, 11,  4},
                                        {   1,  -3, -3, 0,  3,  3,  1} };

sobel_V_9[9][9] = { { -1,  -3,  -3, 0, 3,  3,   1},
                                        { -4, -11, -13, 0, 13, 11,  4},
                                        { -9, -26, -30, 0, 30, 26,  9},
                                        { -13, -34,-40, 0, 40, 34, 13},
                                        {  -9, -26,-30, 0, 30, 26,  9},
                                        {  -4, -11,-13, 0, 13, 11,  4},
                                        {   1,  -3, -3, 0,  3,  3,  1} };
sobel_D_3[3][3] = { { 0,  1, 2},
                                        { -1, 0, 1},
                                        { -2,-1, 0} };

laplacianOfGaussian_3[3][3] = { {0, -1, 0},
                                                {-1, 4, -1},
                                                {0, -1, 0},
};

laplacianOfGaussian_5[5][5] = { {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} };
laplacianOfGaussian_7[7][7] = {
                                        {1, 6, 15, 20, 15, 6, 1},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {20, 120, 300, 400, 300, 120, 20},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {6, 36, 90, 120, 90, 36, 6},
```

```
                                        {1, 6, 15, 20, 15, 6, 1},
};
laplacianOfGaussian_9[9][9] = {
                                        {1, 6, 15, 20, 15, 6, 1},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {20, 120, 300, 400, 300, 120, 20},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {1, 6, 15, 20, 15, 6, 1} };

gaussian_3[3][3] = { {1, -2, 1},
                                                {-2, 4, -2},
                                                {1,-2,1}};

gaussian_5[5][5] = { {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} };
gaussian_7[7][7] = {
                                        {1, 6, 15, 20, 15, 6, 1},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {20, 120, 300, 400, 300, 120, 20},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {1, 6, 15, 20, 15, 6, 1},
                                        };
gaussian_9[9][9] = {
                                        {1, 6, 15, 20, 15, 6, 1},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {20, 120, 300, 400, 300, 120, 20},
                                        {15, 90, 225, 300, 225, 90, 15},
                                        {6, 36, 90, 120, 90, 36, 6},
                                        {1, 6, 15, 20, 15, 6, 1},
```

Algorithms:

mean: For mean filter I have used the kernel and use the convolution method.

```
Mat mean(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
```

```
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			double val = (arr[i - 1][j - 1] + arr[i - 1][j] + arr[i - 1][j + 1]
+ arr[i][j - 1] + arr[i][j] + arr[i][j + 1] + arr[i + 1][j - 1] + arr[i + 1][j] + arr[i +
1][j + 1]);
			val = val / 9;
			val = (int)val;
			output.at<uchar>(i, j) = val;
		}
	}
	return output;
}
```

meadian: For median filter I have get the input image with an array where zero padding is already done and use a 3x3 kernel to sort the neighbourhood and store it in the output image.

```
Mat median(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			vector<int>v;
			v.push_back(arr[i - 1][j - 1]);
			v.push_back(arr[i - 1][j]);
			v.push_back(arr[i - 1][j + 1]);
			v.push_back(arr[i][j - 1]);
			v.push_back(arr[i][j]);
			v.push_back(arr[i][j + 1]);
			v.push_back(arr[i + 1][j - 1]);
			v.push_back(arr[i + 1][j]);
			v.push_back(arr[i + 1][j + 1]);
			sort(v.begin(), v.end());
			//cout << v[4];

			output.at<uchar>(i, j) = v[4];
		}
	}
	return output;
}
```

laplacian: for Laplacian filter I have used the kernel given in PPT and use convolution.

```
Mat laplasian(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			double val = (0 * arr[i - 1][j - 1] + 1 * arr[i - 1][j] + 0 * arr[i
- 1][j + 1] + 1 * arr[i][j - 1] + (-4) * arr[i][j] + 1 * arr[i][j + 1] + 0 * arr[i + 1][j
- 1] + 1 * arr[i + 1][j] + 0 * arr[i + 1][j + 1]);
			//val = val / 4.8976;
			val = (int)val;
			output.at<uchar>(i, j) = val;
		}
```

```
	}
	return output;
```

gaussian: For the gaussian blur, I have used the given kernel and use the convolution method.

```
Mat gaussian(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			double val = (0.3679*arr[i - 1][j - 1] + 0.6065*arr[i - 1][j] +
0.3679*arr[i - 1][j + 1] + 0.6065*arr[i][j - 1] + 1.000*arr[i][j] + 0.6065*arr[i][j + 1]
+ 0.3679*arr[i + 1][j - 1] + 0.6065*arr[i + 1][j] + 0.3679*arr[i + 1][j + 1]);
			val = val / 4.8976;
			val = (int)val;
			output.at<uchar>(i, j) = val;
		}
	}
	return output;
}
```

sobel: For the sobel filter I have used the kernel and convolution method.

```
Mat sobel_y(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			double val = ((-1) * arr[i - 1][j - 1] + (-2) * arr[i - 1][j] + (-1)
* arr[i - 1][j + 1] + (0) * arr[i][j - 1] + (0) * arr[i][j] + 0 * arr[i][j + 1] + (1) *
arr[i + 1][j - 1] + 2 * arr[i + 1][j] + 1 * arr[i + 1][j + 1]);
			//val = val / 4.8976;
			val = (int)val;
			output.at<uchar>(i, j) = val;
		}
	}
	return output;
}

Mat sobel_x(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.rows; i++)
	{
		for (int j = 1; j < img.cols; j++)
		{
			double val = ((-1) * arr[i - 1][j - 1] + (0) * arr[i - 1][j] + (1) *
arr[i - 1][j + 1] + (-2) * arr[i][j - 1] + (0) * arr[i][j] + (2) * arr[i][j + 1] + (-1) *
arr[i + 1][j - 1] + (0) * arr[i + 1][j] + (1) * arr[i + 1][j + 1]);
			//val = val / 4.8976;
			val = (int)val;
			output.at<uchar>(i, j) = val;
		}
	}
	return output;
}
```

lap_gaussian5: I have used the convolution method to get the output of the filter.

```
Mat lap_gaussian5(Mat img, vector<vector<int>>arr)
{
	Mat output = img.clone();
	for (int i = 1; i < img.cols; i++)
	{
		for (int j = 1; j < img.rows; j++)
		{

			int val = ((laplacianOfGaussian_5[0][0])) * arr[i - 2][j - 2] +
((laplacianOfGaussian_5[0][1])) * arr[i - 2][j - 1] + ((laplacianOfGaussian_5[0][2])) *
arr[i - 2][j - 0] + ((laplacianOfGaussian_5[0][3])) * arr[i - 2][j + 1] +
((laplacianOfGaussian_5[0][4])) * arr[i - 2][j + 2] + ((laplacianOfGaussian_5[1][0])) *
arr[i - 1][j - 2] + ((laplacianOfGaussian_5[1][1])) * arr[i - 1][j - 1] +
((laplacianOfGaussian_5[1][2])) * arr[i - 1][j] + ((laplacianOfGaussian_5[1][3])) * arr[i
- 1][j + 1] + ((laplacianOfGaussian_5[1][4])) * arr[i - 1][j + 2] +
((laplacianOfGaussian_5[2][0])) * arr[i][j - 2] + ((laplacianOfGaussian_5[2][1])) *
arr[i][j - 1] + ((laplacianOfGaussian_5[2][2])) * arr[i][j] +
((laplacianOfGaussian_5[2][3])) * arr[i][j + 1] + ((laplacianOfGaussian_5[2][4])) *
arr[i][j + 2] + ((laplacianOfGaussian_5[3][0])) * arr[i + 1][j - 2] +
((laplacianOfGaussian_5[3][1])) * arr[i + 1][j - 1] + ((laplacianOfGaussian_5[3][2])) *
arr[i + 1][j] + ((laplacianOfGaussian_5[3][3])) * arr[i + 1][j + 1] +
((laplacianOfGaussian_5[3][4])) * arr[i + 1][j + 2] + ((laplacianOfGaussian_5[4][0])) *
arr[i + 2][j - 2] + ((laplacianOfGaussian_5[4][1])) * arr[i + 2][j - 1] +
((laplacianOfGaussian_5[4][2])) * arr[i + 2][j - 0] + ((laplacianOfGaussian_5[4][3])) *
arr[i + 2][j + 1] + ((laplacianOfGaussian_5[4][4])) * arr[i + 2][j + 2];
			output.at<uchar>(i, j) = val;
		}
	}
	return output;
}
```

For the 2nd part of the assignment, I have used the gaussian filter to blur and unblur the image.

At first, I take an image and blur it using gaussian filter. Then I have used the convolution method to find the value of A image. Next, I have stored the pixel in an array and find the B vector by pixel-by-pixel division. Then again use the convolution method and in the last step I have used the pixel-by-pixel multiplication method.

```
while (k--)
	{
		vector<vector<int>>arr1(img.rows+2, vector<int>(img.rows+2, 0));
		for (int i = 1; i < img.rows; i++)
		{
			for (int j = 1; j < img.rows; j++)
			{
				arr1[i][j] = I_K[i][j];
			}
		}
		Mat A = gaussian(output_blurred, arr1);
		vector<vector<int>>B(img.rows+2, vector<int>(img.rows+2, 0));
		for (int i = 1; i < img.rows; i++)
		{
```

```
                for (int j = 1; j < img.rows; j++)
                {
                        double val = (1.0) * (main_input[i][j]);
                        val = val / (A.at<uchar>(i, j));
                        val = (int)val;
                        B[i][j] = val;
                }
        }
        Mat C = gaussian(img, B);

        for (int i = 1; i < img.rows; i++)
        {
                for (int j = 1; j < img.rows; j++)
                {
                        int val = (C.at<uchar>(i, j))*(arr1[i][j]);
                        I_K[i][j] = val;
                }
        }
}
```

Output: