# Gaussian and Mean Filters for Noise Removal

Harshal Vaze
harshalvaze11@gmail.com MatriculationNo. - 1269879

***Abstract*— In this paper, an efficient algorithm has been used that is capable to remove image noise. This complete noise removal procedure depends on Gaussian and Mean filter algorithm that has less computational complexity. A novel algorithm for Gaussian noise estimation and removal is proposed by using 3x3, 5x5, 7x7 and 9x9 sub windows in which the test pixel appears. The algorithm initially estimates the amount of noise corruption from the noise corrupted image. In the second stage, the center pixel is replaced by the mean value of the some of the surrounding pixels based on a threshold value. Noise removing with edge preservation and computational complexity are two conflicting parameters. Experimental results show the superior performance of Gaussian and Mean filter algorithm in .NET Core. This method removes Gaussian noise and the edges are better preserved with less computational complexity and this aspect makes it easy to implement in hardware.**

***Keywords*—Noise Removal, Gaussian Filter, Mean Filter, Image Processing, Convolution Filter**

## I. INTRODUCTION

Image noise is random variation of brightness or colour information in images, and is usually an aspect of electronic noise. It can be produced by the sensor and circuitry of a scanner or digital camera. Noise having Gaussian-like distribution is very often encountered in acquired data. Gaussian noise is characterized by adding to each image pixel a value from a zero-mean Gaussian distribution. The zero-mean property of the distribution allows such noise to be removed by locally averaging pixel values. Conventional linear filters such as arithmetic mean filter and Gaussian filter smooth noises effectively but blur edges. The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense, it is like the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. Since the goal of the filtering action is to cancel noise while preserving the integrity of edge and detail information, nonlinear approaches generally provide more satisfactory results than linear techniques. This information is necessary to perform the optimal choice of parameter values and/or threshold selections.

In the field of image processing, there have been many attempts to construct digital filters which have the qualities of noise attenuation and detail preservation. The Gaussian noise filtering technique it not a sufficient method to noise remove but a successful basic filtering method. Removing Gaussian noise involves smoothing the inside distinct region of an image. For this classical linear filter such as the Gaussian filter reduces noise efficiently but blur the edges significantly. Several researchers have attempted to generalize others standard filters those are seldom suitable for removing Gaussian noise. A nonlinear diffusion equations called as an anisotropic diffusion algorithm have been proposed for Gaussian noise removal.

## II. METHODOLOGY

In the sample application Kernel Size is the physical size dimensions of the kernel/matrix used in convolution. When higher values are specified in setting the Kernel Size, the resulting output image will reflect a greater degree of blurring. Kernel Sizes being specified as lower values result in the output image reflecting a lesser degree of blurring.

A. The Gaussian distribution in 1-D has the form

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. We have also assumed that the distribution has a mean of zero (i.e. it is centered on the line x=0). The distribution is illustrated in Figure 2.1.

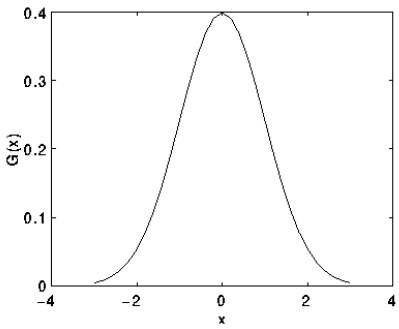



Fig-2.1: 1-D Gaussian distribution with mean 0 & σ =1

B. The Isotropic Gaussian distribution in 2-D has the form

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

This distribution is shown in Figure 2.2.

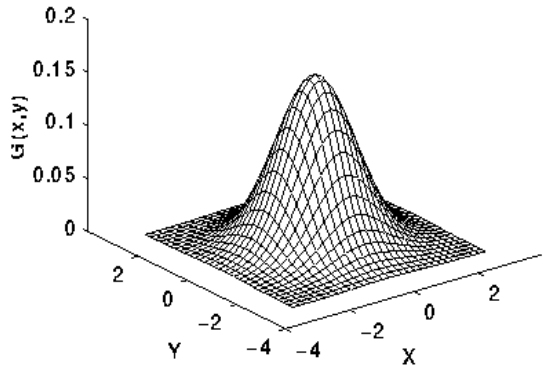



Fig-2.2: 2-D Gaussian distribution with mean (0,0) and σ =1

The idea of Gaussian smoothing is to use this 2-D distribution as a 'point-spread' function and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean and so we can truncate the kernel at this point. Figure 2.3 shows a suitable integer-valued convolution kernel of window size 5x5, that approximates a Gaussian with a σ of 1.0. It is not obvious how to pick the values of the mask to approximate a Gaussian. One could use the value of the Gaussian at the center of a pixel in the mask, but this is not accurate because the value of the Gaussian varies non-linearly across the pixel. We integrated the value of the Gaussian over the whole pixel (by summing the Gaussian at 0.001 increments). The integrals are not integers: we rescaled the array so that the corners had the value 1. Finally, the 273 is the sum of all the values in the mask.[1]

$\frac{1}{273}$

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 7 | 4 | 1 |
| 4 | 16 | 26 | 16 | 4 |
| 7 | 26 | 41 | 26 | 7 |
| 4 | 16 | 26 | 16 | 4 |
| 1 | 4 | 7 | 4 | 1 |

Fig-2.3: Discrete approximation to Gaussian function with σ =1.0

C. Remove Gaussian Noise

If the pixel is found corrupted then a filter is invoked. The corrupted pixel is replaced with a new value obtained from the following formula.

$$X_{new}(i, j) = [\ \mu - 0.5 \times \sigma_{avg}\ ]$$

where $X_{new}$ is the new value for the pixel position represented by (i, j), μ is the mean of the 3x3 central sub window, $\sigma_{avg}$ is the average standard deviation defined in the previous section.

Once a suitable kernel has been calculated, then the Gaussian smoothing can be performed using standard convolution methods. The convolution can in fact be performed quickly since the equation for the 2-D isotropic Gaussian shown above is separable into x and y components. Thus the 2-D convolution can be performed by first convolving with a 1-D Gaussian in the x-direction, and then convolving with another 1-D Gaussian in the y-direction. (The Gaussian is in fact the only completely circularly symmetric operator which can be decomposed in such a way.) Figure 2.4 shows the 1-D x-component kernel that would be used to produce the full kernel shown in Figure 2.3 (after scaling by 273, rounding and truncating one row of pixels around the boundary because they mostly have the value 0. This reduces the 7x7 matrix to the 5x5). The y-component is the same but is oriented vertically.[6]

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 7 | 4 | 1 |

Fig-2.4: 1-D x-component Gaussian Kernel

A further way to compute a Gaussian smoothing with a large standard deviation is to convolve an image several times with a smaller Gaussian. While this is computationally complex, it can have applicability if the processing is carried out using a hardware pipeline.

Following image has been used to demonstrate Gaussian filtering process.

which has been corrupted by 1% salt and pepper noise (i.e. individual bits have been flipped with probability 1%). Image became as follows, which

shows the result of Gaussian smoothing (using the same convolution as above). Compare this with the original

Notice that much of the noise still exists and that, although it has decreased in magnitude somewhat, it has been smeared out over a larger spatial region. Increasing the standard deviation continues to reduce/blur the intensity of the noise, but also attenuates high frequency detail (e.g. edges) significantly, as shown in pictures.[4]

D. Mean Filter

The mean filter is a simple sliding-window spatial filter that replaces the center value in the window with the average (mean) of all the pixel values in the window. The window or kernel, is usually square but can be any shape. An example of mean filtering of a single 3x3 window of values is shown below.

| | | |
|---|---|---|
| 5 | 3 | 6 |
| 2 | 1 | 9 |
| 8 | 4 | 7 |

Fig-2.5: Unfiltered values

Summation = 5 + 3 + 6 + 2 + 1 + 9 + 8 + 4 + 7 = 45
Mean = 45 / 9 = 5

| | | |
|---|---|---|
| * | * | * |
| * | 5 | * |
| * | * | * |

Fig-2.6: Mean filtered

The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbours, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown in Figure 2.6, although larger kernels (*e.g.* 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once to produce a similar but not identical effect as a single pass with a large kernel.) [2]

Image smoothing refers to any image-to-image transformation designed to smoothen or flatten an image by reducing the rapid pixel-to-pixel variation in grey levels. Smoothing may be accomplished by applying an averaging mask that computes a weighted sum of the pixel grey levels in a neighbourhood and replaces the center pixel with that grey level. The image is blurred and its brightness retained as the mask coefficients are all-

positive and sum to one. The mean filter is one of the most basic smoothing filters. Mean filtering is usually thought of as a convolution operation as the mask is successively moved across the image until every pixel has been covered. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Larger kernels are used when more severe smoothing is required.[2]

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{n^2}\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & \cdots & 1\end{bmatrix}$$

(a) (b)

Fig-2.7: (a) 3×3 Mean Filter; (b) General n×n Mean Filter

Fig-2.8: Effect of mean filtering: (a) original image; (b) Blurred image with n = 7 and (c) n = 11.

Fig-2.8 (a) shows a mean mask for a 3 × 3 window, while a more general n × n mask is shown in Fig-2.8 (b). Variations on the mean filter include threshold averaging, wherein smoothing is applied subject to the condition that the center pixel grey level is changed only if the difference between its original value and the average value is greater than a preset threshold. This causes the noise to be smoothed with a less blurring in image detail. It must be noted here that the smoothing operation is equivalent to low-pass filtering as it eliminates edges and regions of sudden grey level change by replacing the center pixel grey level by the neighbourhood average. It effectively eliminates pixel grey levels that are unrepresentative of their surroundings. Noise, due to its spatial de-correlations, generally has a higher spatial frequency spectrum than the normal image components. Hence, a simple low-pass filter can be very effective in noise cleaning. Smoothing filters thus find extensive use in blurring and noise removal. Blurring is usually a pre-processing step bridging gaps in lines or curves, helping remove small unwanted detail before the extraction of relevant larger objects. Figs-2.8 (b) and (c) show the effect of averaging for various window sizes n = 7 and n = 11. The mean filter is an important image processing tool and finds use in Gaussian noise reduction, blurring before thresholding to eliminate small detail, bridging gaps in broken characters for improved machine perception in Optical Character Readers, cosmetic processing of human faces images to reduce fine skin lines and blemishes, etc. The mean is also used as a derived or texture feature in image segmentation process.[2]

## III. WORKING ALGORITHM

A. Gaussian Kernel calculation of 2-D convolution

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where, *G(x,y)* - A value calculated using the Gaussian Kernel formula. This value forms part of a Kernel, representing a single element.
*x,y* – The variables referenced as x and y relate to pixel coordinates within an image. y Representing the vertical offset or row and x represents the horizontal offset or column.
σ - Standard deviation of distribution

When calculating the kernel elements, the coordinate values expressed by *x* and *y* should reflect the distance in pixels from the middle pixel. All coordinate values must be greater than zero.
To create a *3×3* kernel and with standard deviation of *5.5* our calculations can start off as indicated by the following illustration:

$$\begin{pmatrix} \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+1^2}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{0^2+1}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+1^2}{2(5.5)^2}} \\ \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+0^2}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{0^2+0^2}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+0^2}{2(5.5)^2}} \\ \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+1^2}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{0^2+1^2}{2(5.5)^2}} & \frac{1}{2\pi(5.5)^2}e^{-\frac{1^2+1^2}{2(5.5)^2}} \end{pmatrix}$$

Fig-3.1: Calculating Kernel values with standard deviation 5.5

The formula has been implement on each element forming part of the kernel, 9 values in total. Coordinate values have now been replaced with actual values, differing for each position/element. Calculating zero to the power of two equates to zero. In the scenario above indicating zeros which express exponential values might help to ease initial understanding, as opposed to providing simplified values and potentially causing confusing scenarios. The figure 3.2 illustrates the calculated values of each kernel element:

| | | |
|---|---|---|
| 0.005090 | 0.005175 | 0.005090 |
| 0.005175 | 0.005261 | 0.005175 |
| 0.005090 | 0.005175 | 0.005090 |

Fig-3.2: Kernel values

An important requirement to take note of at this point being that the sum total of all the elements contained as part of a kernel/matrix must equate to one. Looking at calculated results that is not the case., hence the kernel needs to be modified in order to satisfy the requirement of having a sum total value of 1 when adding together all the elements of the kernel. Rearranging element values to whole numbers. At this point the sum total of the kernel equates to 16 as shown in figure 3.3. Hence to correct the kernel values, ensuring the sum total of all kernel elements equate to 1 the kernel values should be updated by multiplying each element by one divided by the current kernel sum.[8]

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

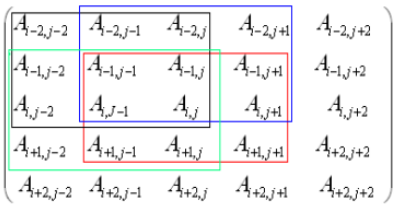
Fig-3.3: 3x3 Gaussian Filter

B. Running Kernel over Test Image

1. Consider a 5 x 5 test window $A_T$ from the noisy image as:

$$A_T = \begin{pmatrix} A_{i-2,j-2} & A_{i-2,j-1} & A_{i-2,j} & A_{i-2,j+1} & A_{i-2,j+2} \\ A_{i-1,j-2} & A_{i-1,j-1} & A_{i-1,j} & A_{i-1,j+1} & A_{i-1,j+2} \\ A_{i,j-2} & A_{i,J-1} & A_{i,j} & A_{i,j+1} & A_{i,j+2} \\ A_{i+1,j-2} & A_{i+1,j-1} & A_{i+1,j} & A_{i+1,j+1} & A_{i+2,j+2} \\ A_{i+2,j-2} & A_{i+2,j-1} & A_{i+2,j} & A_{i+2,j+1} & A_{i+2,j+2} \end{pmatrix}$$

2. Divide this window into 3 x 3 sub-windows such that the test pixel should appear in each of the sub-windows. Nine such sub-windows are possible and four of them as shown below.



3. For each 3x3 sub-window calculate the standard deviation ($\sigma_i$), i=1, 2 …. N where N is maximum number of the sub-windows.
4. Set reference standard deviation ($\sigma_{ref}$), as median of $\sigma_i$, i=1, 2 …. N.
5. Set $\sigma_{min}$ = k1 x $\sigma_{ref}$.
6. Set $\sigma_{max}$ = k2 x $\sigma_{ref}$.
7. Calculate average ($\sigma_{avg}$) of the standard deviations $\sigma_i$, i=1, 2 ……N whose standard deviation lies in the range [$\sigma_{min}$, $\sigma_{max}$].
8. This $\sigma_{avg}$ is used as a parameter to decide whether the test pixel is corrupted or not.

The above process is repeated by sliding 5x5 window one step forward row wise and then column wise to cover the entire image.

Both filters attenuate high frequencies more than low frequencies, but the mean filter exhibits oscillations in its frequency response. The Gaussian on the other hand shows no oscillations. In fact, the shape of the frequency response curve is itself (half a) Gaussian. So, by choosing an appropriately sized Gaussian filter we can be confident about what range of spatial frequencies are still present in the image after filtering, which is not the case of the mean filter. This has consequences for some edge detection techniques, as mentioned in the section on zero crossings. The Gaussian filter also turns out to be very like the optimal smoothing filter for edge detection under the criteria used to derive the Canny edge detector.[6]

## IV. IMPLEMENTING AND RESULT

A. Kernel Creation:

1) Creating Kernel:

Using convolution process, two kernels are developed which are run over each pixel in the test

image. Kernel are matrix of the size of 3×3 and 5x5 that holds predetermined values. When a kernel is over a pixel it calculates values using neighboring pixels that are also under this kernel.

```
public static double[,] GaussianBlur3x3
{
    get
    {
        return new double [,]
        { { 1, 2, 1, },
          { 2, 4, 2, },
          { 1, 2, 1, }, };
    }
}

public static double[,] GaussianBlur5x5
{
    get
    {
        return new double[,]
        { { 2, 04, 05, 04, 2 },
          { 4, 09, 12, 09, 4 },
          { 5, 12, 15, 12, 5 },
          { 4, 09, 12, 09, 4 },
          { 2, 04, 05, 04, 2 }, };
    }
}
```

Fig-4.1: Gaussian 3x3 and 5x5 Kernel

2) Extracting RGB value from Test image:
RGB (Red, Green, Blue) value of each pixel from the test image is calculated and stored in the variables using the For loop.

```
for (int i = 0; i < data.GetLength(0); i++)
{
    for (int j = 0; j < data.GetLength(1); j++)
    {
        int r = (int)data[i, j, 0];
        int g = (int)data[i, j, 1];
        int b = (int)data[i, j, 2];
        bitmap.SetPixel(i, j, Color.FromArgb(255, r, g, b));
    }
}
```

Fig-4.2: RGB value from Test image

3) Running Kernel over the Test image:
After extracting the RGB value from the test image, the kernel is run over each pixel of the test image using the For loop.

```
        calcOffset = byteOffset + (filterX * 4) + (filterY * sourceData.Stride);

        blue += (double)(pixelBuffer[calcOffset]) * GaussianBlur5x5[filterY + filterOffset, filterX + filterOffset]/159;

        green += (double)(pixelBuffer[calcOffset + 1]) * GaussianBlur5x5[filterY + filterOffset, filterX + filterOffset]/159;

        red += (double)(pixelBuffer[calcOffset + 2]) * GaussianBlur5x5[filterY + filterOffset, filterX + filterOffset]/159;
    }
}

blue = factor * blue + bias;
green = factor * green + bias;
red = factor * red + bias;

blue = (blue > 255 ? 255 : (blue < 0 ? 0 : blue));
green = (green > 255 ? 255 : (green < 0 ? 0 : green));
red = (red > 255 ? 255 : (red < 0 ? 0 : red));
```

Fig-4.3: Gaussian kernel run over test image

Similar steps are followed for the Mean filter using the Mean filter kernel. For the combined result of both the filters over the test image, first the Gaussian Kernel is run over the test image and then the Mean Kernel is run over the output image of the Gaussian filter image.

B. Module Usage:

In this project there is a solution folder Gaussian And Mean Filter which contains class libraries and mstest Test project. The main algorithms are implemented in the class libraries and mstest is built to run the project and has been interconnected with the LearningApi. LearningApi is a learning foundation on top of .NET Core developed by Frankfurt University of Applied Sciences.

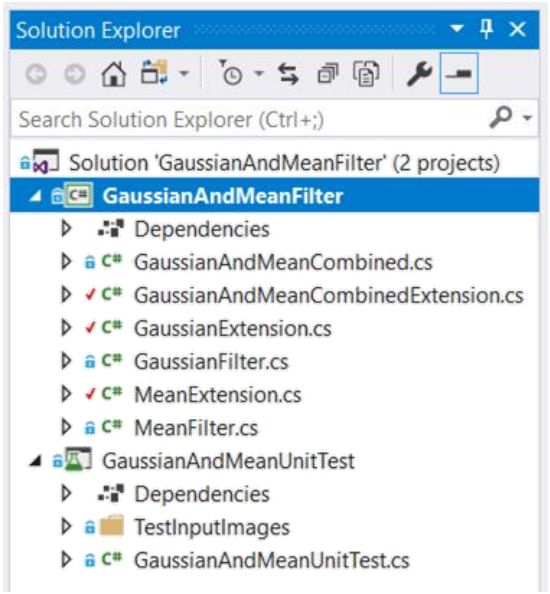



Fig-4.4: Application directory

There are five unit tests provided in this project and each unit test contains three unit tests each for Gaussian filter, Mean filter and Combined Gaussian and Mean filter. At first, the first unit test is run, it calls the test method of Gaussian Filter which executes the Gaussian algorithm which extracts the image width and height of the input image. The newly calculated width and height of image is fed to convolution filter which picks up each pixel and runs the kernel over it. It gives the Gaussian output image which is saved in the TestOutputImages folder. After that the test method of Mean Filter is called, it executes the same procedure as of Gaussian Filter and gives the Mean output image which is again saved in TestOutputImages folder. After that the final test method of Gaussian And Mean Combined is called, it calls the test method of Gaussian Filter first and the result of Gaussian Filter is fed into the method of Mean filter. The output image from Mean filter method gives the Output image of both combined filters which is again saved in folder TestOutputImages.

Similarly all remaining tests are executed and the output is saved in the TestOutputImages folder. The TestInputImages folder contains the images that needs to be processed and the TestOutputImages folder contains the processed Images.

C. Procedure to run the code:

We need to follow the below steps to get the output.

Step 1: Put the images in “TestInputImages” folder, which needs to be filtered.

Step 2: Go to images in the solution explorer which you recently added and click on properties.

Step 3: In the advanced column, under Copy to Output Directory select the option “Copy if Newer”.

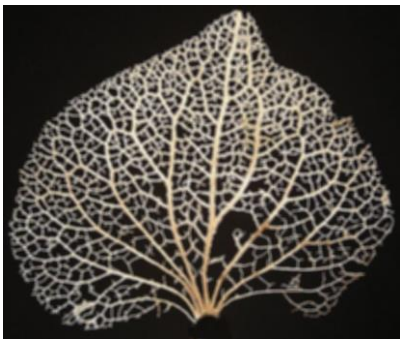
Step 4: Change the name of the image and its format as in the TestInputImages folder in Unit tests. Alternatively change the name of the output image as desired.

Step 5: Now run the tests. The output images will be stored inside the bin folder “TestOutputImages”.

**Test Images**

Input Image

Gaussian Filter



Mean Filter

Combined Gaussian And Mean Filter

## V. CONCLUSION

The fast Gaussian and Mean filtering technique presented here has successfully reduced the time requirements for smoothing operations with mean filters, especially for large images. This implementation reduces the number of additions to approximately 1/nth of the original number, where n×n is the neighbourhood size and completely eliminates the division operation by store and-fetch methods very efficiently. The gain in the performance and usefulness of this method has been demonstrated for different images.

The various applications of Gaussian and Mean filtering, as described in detail at the beginning of this paper can all benefit tremendously from this improved implementation. Noise removal by threshold averaging in remote-sensing applications is one such important example. The effect of this filtering technique will be tremendous in such an application. This method can easily be extended to higher bit-level grayscale images or colour images.

## VI. REFERENCES

1. https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf
2. https://www.markschulze.net/java/meanmed.html
3. http://www1.inf.tu-dresden.de/~ds24/lehre/bvme_ss_2013/ip_03_filter.pdf
4. http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm
5. Eero P. Simoncelli & Edward H. Adelson, “Noise removal via Bayesian Wavelet Coring”, 3rd IEEE Conference on Image Processing, Lausanne, Switzerland 1996
6. “Estimation and Removal of Gaussian Noise in Digital Images”, International Journal of Electronics and Communication Engineering. ISSN 0974-2166 Volume 5, Number 1 (2012), pp. 23-33
7. https://pdfs.semanticscholar.org/df57/2b11b245267686aba59a564fd56f472cf845.pdf
8. https://softwarebydefault.com/2013/06/08/calculating-gaussian-kernels/
9. Yeqiu Li et.al, “Removal of Gaussian Noise from Degraded Image in Wavelet Domain,” Translated from Denki Ronbunshi, vol.126-c, No.11, Nov.2006, pp.1351-1358.