**TABLE OF CONTENTS:**

1. INTRODUCTION……………………………………………………………………….................2
2. AIM AND APPROACH……………………………………………………………………………….2-3
3. DATA PREPARATION……………………………….…….............................................4-

3.1. Resize Operation……………………………………………………………

3.2. Research question……………………………………………………………....……….......

3.3. Extracting Pixel Values……………………….

1. DATA AND CALCULATIONS……………………………………………………

4.1. Calculations Of Kernels Respect To The 2D Gaussasian Equation

4.2. Apply Convulution……………………………………………………………....……….......

4.2.1 Calculating 1’st Pixel Of Our Picture…………………………….

4.2.2 Calculating 1’st Pixel Of Our Picture…………………………….

4.2.3 Calculating The Other Pixels Of Our Picture…………………………

4.2.4 Creating A New Picture…………………………….

1. CONCLUSION ……………………………………………..….
2. LIMITATIONS AND IMPROVEMENT……………………………………………………………………

6.1 Importance of sigma() value

6.2 Bokeh Effect…………………………..

6.3 Background Noise Cancelation

6.4 Edge Dedection

1. BIBLIOGRAPHY…………………………………………………………………...…...
2. LIST OF EQUATIONS, TABLES AND FIGURES
3. APPENDIX
4. **INTRODUCTION:**

Mathematics is everywhere in our life. There is always math behind every step we take, in the growth of trees and the orderliness of life. Many things in life can be solved and can provide an explanation with mathematics. This has always intrigued me. As I gradually learned these mathematical calculations that exist in every step of our daily life, my interest in mathematics increased day by day. Over time, when I started to think of even ordinary simple daily events mathematically, my perspective on life also changed.

Nowadays, people love to take photos. With the introduction of mobile phones into our lives, we have always carried a camera with us. Then a selfie trend started all over the world and people loved to share their photos on social media. Then people started to use various photo correction techniques to look more beautiful. One of them is the blur feature. People usually want to get blurred the background behind their face. Most of phone has “Portrait” camera mode for this purpose.

As someone who tries to discover the mathematics behind the events over time, I wanted to investigate the mathematics behind these famous photo tricks. It is clear that technology, which has entered our lives at a great speed, will always be with us. Therefore, I thought it would be appropriate to examine the mathematics behind the blur technique.

In this study, my main aim is to understand the mathematics behind digital image processing known as Gaussian blur (also known as Gaussian smoothing or Gaussian filtering), one of the most commonly used image tricks in today's photographs, and I will go through various mathematical steps to use this feature to create a blur effect on a photograph. I will mostly use Gaussian formulas while carrying out this exploration.

*“The Gaussian filtering method is named after a scientist, Carl Friedrich Gauss. The purpose of using this method is to apply a different effect to the photo by reducing the image noise and details of the photo. The visual effect of this blurring technique is a smooth blur resembling that of viewing the* [*image*](https://en.wikipedia.org/wiki/Image) *through a translucent screen, distinctly different from the* [*bokeh*](https://en.wikipedia.org/wiki/Bokeh) *effect produced by an out-of-focus lens or the shadow of an object under usual illumination.[[1]](#footnote-1)* (Gaussian blur, n.d.)*”*

To use Gaussian filtering method and reach my main purpose I will do followigs;

firstly I took a photo as our data. I thought the photo of my younger brother was very suitable for this purpose. Secondly, I need to prepare this photo to do my calculations. Because of its size, shape and some features are not suitable for calculation. I need to convert the actual size of the image I use as data from 2448\*3264 pixels to 256\*256 pixels because otherwise, the calculations will take too long and my kernel will not fit in it. That is, I will first reduce the Pixel values ​​of my photo and then convert the color to black and white so that it can be suitable for calculations on the computer. Thirdly, I will convert this image into a mathematical array to be able to do my mathematical calculations, because I will need these values ​​to apply our concepts to the data. Then I need a kernel to perform mathematical operations. An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. I will calculate and define a kernel by using 2D Gaussian that will get my image blurred. Finally, I will apply the convolution process over my image after calculating the kernel.

As a result, after doing all these steps, I will have blurred my original image, and besides, I will have explored the mathematical topics of standard normal distribution function, Gauss function, 2D Gauss distribution, convolution, statistics, and Euler’s number and performed all these newly learned mathematical calculations with the help of a photo I chose.

1. **DATA PREPARATION:**

As I mentioned before, I selected my younger brother’s picture for this project. I like to use real data and find very interesting to implement new things on them. Here is my raw data, my younger brother Poyraz:



Figure 1 :Original Coloured Data

**2.1 Resize Operation:**

Pixels are the basic and smallest element of a photograph. Pictures, just like a wall, is made of pixels like bricks. Today's computer technology also uses these pixels by arranging them with squares used on a 2-dimensional plane. Each pixel is a tiny piece of an orginal image. In digital imaging and printing techniques, each picture actually consists of 3 basic colors. This system is called RGB which is form with R= red, G=green and B=blue. The computer system gives a mathematical value to all the pixels it squares. In fact, the images we describe as colors are just numbers for a computer. In order for mathematical calculations to be made, it only needs to have a number value for each pixel.

*“The term resolution is often used as a pixel count in digital imaging. When the pixel counts are referred to as resolution, the convention is to describe the pixel resolution with the set of two numbers. The first number is the number of pixel columns (width) and the second is the number of pixel rows (height), for example as 640 by 480. Another popular convention is to cite resolution as the total number of pixels in the image, typically given as number of megapixels, which can be calculated by multiplying pixel columns by pixel rows and dividing by one million. An image that is 2048 pixels in width and 1536 pixels in height has a total of 2048×1536 = 3,145,728 pixels or 3.1 megapixels.[[2]](#footnote-2)* (digital image, n.d.)*”*

I need to reduce the size of this picture, which I will use as data, because otherwise, my calculations will take too long and the kernel calculations will not fit here. This operation also known as lowering the resulution. Just like we watch videos online, if we lower the resulution, totol number of calculations is also reduces dramaticlly. For this reason, I will reduce my photo from 2448\*3264 pixels to 256\*256 pixels. In order to make this operation, I simply opened my photo in windows paint application (the default windows well-known application) and used the crop/resize feature. I converted the photo, which I used as data, from 2448\*3264 pixels to 256\*256 pixels.



Figure 2: Data After Resize operation

* 1. **Converting Grayscale**

The next step is changing the picture’s colourmap. In other word, I will change our coloured RGB picture into black and white, also known as grayscale. I need to convert this picture into grayscale because we are working on 2d arrays. The other reason of why i need to change the pictures’s colurmap is make the picture more easy to process. Since, colored pictures has 3D dimentions, it is harder to implement Gaussian Blur and it is clearly out of our purpose.

I used the python programming language for this operatian. Python is a very common freeware, open source and easy to use programming language. It also has some tools for digital image processing. “PIL (Python Image Library)” library and “.convert('L')” attribute. This simple code change our cloured photo,which name is “img” in the codeline, into grayscale, which has new name called “imgGray” in the codeline. The code is as follows:

from PIL import Image

img = Image.open('poyraz\_256.jpg')

imgGray = img.convert('L')

Now the picture is converted into grayscale. The picture is as follows:



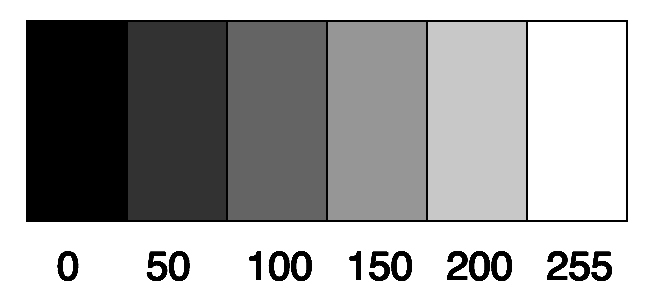
Figure 3: Grayscaled data

* 1. **Extracting Pixel Values**

Every image consists of pixels. In addition, each pixel has a value. Our eyes see the pixels as a colour. However, in the mathematical world we need to see them as a number to make calculations. From now on, we make calculations with these pixel values. Therefore, we need to extract pixel values from our picture. the computer system numbers grayscale images in a range of values ​​from 0 to 255. 0 represents black and 255 represents white. Each value between these numbers belongs to different shades of gray. Here are some examples of a pixel and color:



Mario with 16 bit pixels (coloured)



Some Grayscale Pixels And its Values

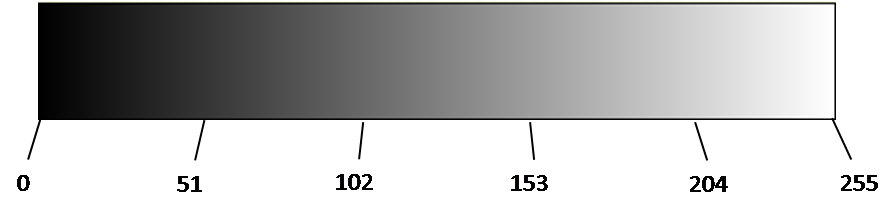
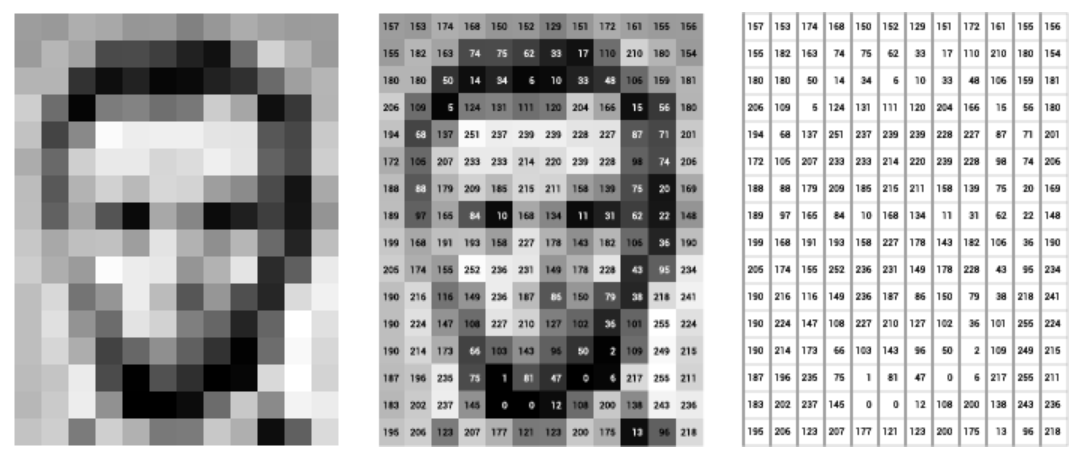


Figure 5 : Mario with 16 bit pixels / Grayscele colourmap

As you see in the pictures above, there is a mario character which is drawn with low resulution so that you can see pixels easily. The other picture gives us the information about grayscaled pixel values and the corresponding tones of gray.



Just Display of Pixels

Pixels with its values

Just Values of Pixels

Figure 6:Pixel visualisation and numbered pixel[[3]](#footnote-3)

As you see the in Figure …we can change the picture into numerical values. Every value has a colour meaning. “0” refers to black, “255” refers to white ,”100” refers to a tone of gray. Our eyes understand only colour of pixels just like first picture above. On the other hand, Each pixel has a value and we need them in order to make calculations.Since, computers only see the same picture as numbers, as in the third photo above.

As I explained in detail, we need to convert the pixel values ​​of our photograph into numbers in order to convey our mathematical calculations. Therefore, I will convert the picture that I use as data into numeric values. In the values that I will use in my photo between 0 and 255, “0” will mean black, “255” white, “1” to “254” refers to different gray tones.

Since we have a grayscale image, we created a 2D array of our pixels. I used the Python programming language to extract pixel values from our picture. I simply used “.getdata()” attribute to do this action. It is a basic code that convert photo into numerical vaues. Here is my code;

pixel\_values = list(imgGray.getdata())

pixel\_values = np.array(pixel\_values)

pixel\_values = pixel\_values.reshape(256,256)

Our photo had 256 rows and 256 columns and total of 65536 cells (256 times 256). Pixel values for the first 16 rows and 16 columns at the top left of our image cells are as follows:

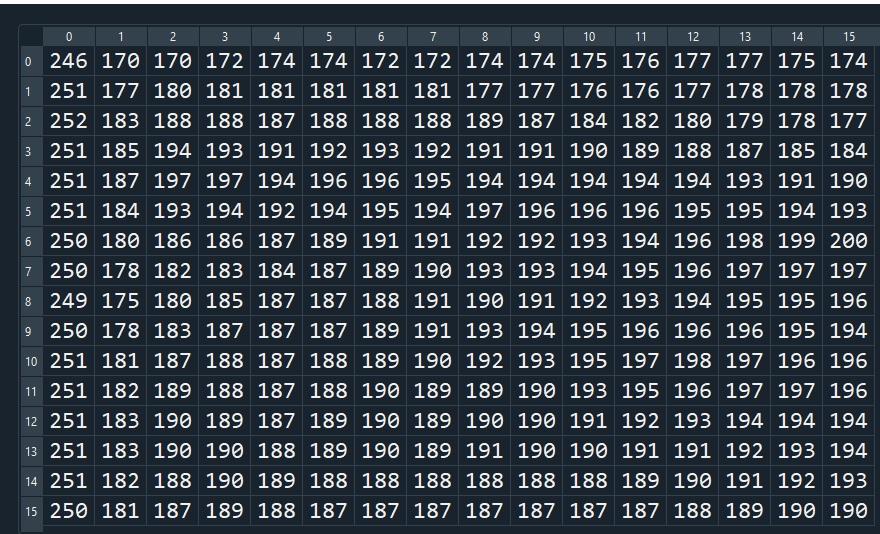


Figure 7: First 16 Rows and 16 Columns of Data

Besides, in Figure 7?8 is the visualization of the pixels above. I used Python image show function to visualize numbers above;

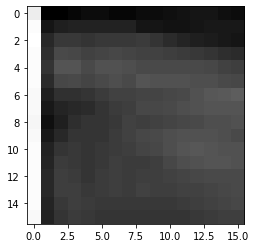


Figure 8: Visualized data of first 16 rows and 16 columns / Location of first 16 rows and 16 columns on grayscaled image

This is the first 16 rows and 16 columns of our picture’s pixel values. Our picture consists of 256 rows and 256 columns. Therefore, the left picture above is the part 1 out of 256 (16 times 16). There are 255 more parts in the picture which could not be shown above. As mentioned before, “0” refers to the colour black and “255” refers to white. The values between 0 and 255 belong to other shades of gray. Because the upper right part of our picture consists mostly of clouds, the values are around “190” and it is close to the white, which is “255”. In the end, we could convert our pixels in to numerical values.

1. **DATA AND CALCULATIONS:**
   1. **Calculations Of Kernels Respect To The 2D Gaussian Equation**

Now, my picture is ready for digital image processing. To blur our picture I need a kernel. So that, the next step is creating a kernel. I will use some formulas to calculate our kernel. But before creating kernel I need to mention some basic consepts like standart deviation, Gauss function, 2-Dimentional Gauss function and what a kernel is.

The Standard deviation of the Gaussian function plays an important role in its behaviour. The values located between +/- σ account for 68% of the set, while two standard deviations from the mean (blue and brown) account for 95%, and three standard deviations (blue, brown and green) account for 99.7%. This is very important when designing a Gaussian kernel of fixed length.

Therefore increasing the standard deviation increases the effect of the filter on the image.[[4]](#footnote-4) (Gaussian Filtering, n.d.)

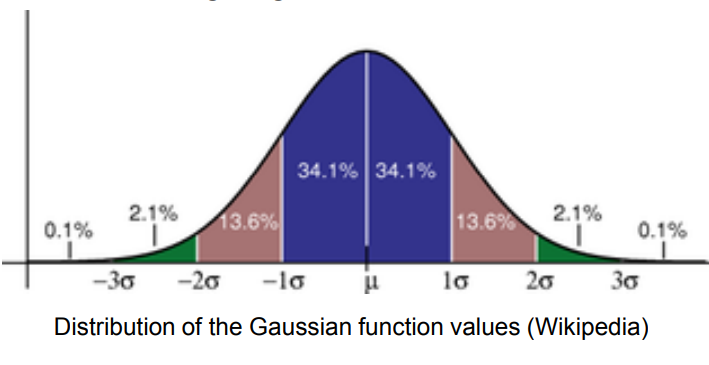


Figure 9:Distribution of the Gaussian function values

“The Gaussian distribution shown is normalized so that the sum over all values of x gives a probability of 1. The nature of the gaussian gives a probability of 0.683 of being within one standard deviation of the mean. The mean value is a=np where n is the number of events and p the probability of any integer value of x (this expression carries over from the binomial distribution ). The standard deviation expression used is also that of the binomial distribution.

The Gaussian distribution is also commonly called the "normal distribution" and is often described as a "bell-shaped curve".[[5]](#footnote-5) (Gaussian Distribution Function, n.d.)”

Equation :Gaussian Function

Where *x* is the distance from the origin in the horizontal axis, and *σ* is the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation" \o "Standard deviation) of the Gaussian distribution, is the pi value, is the Euler number.

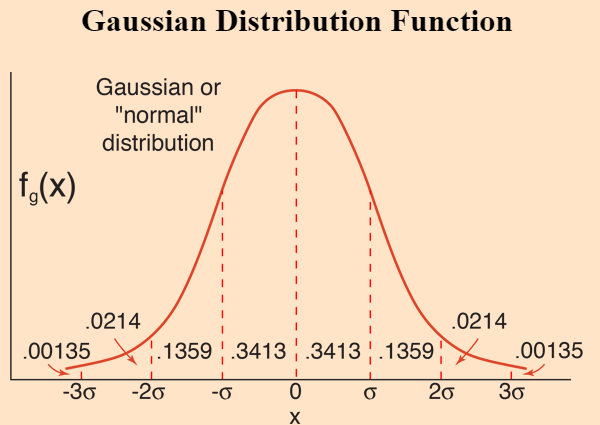


Figure 10:Gaussian Distribution Function

*When working with images we need to use the two dimensional Gaussian function. This is simply the product of two 1D Gaussian functions (one for each direction) and is given by:*

# 

Equation 2: 2D Gaussian Equation

Where *x* is the distance from the origin in the horizontal axis, *y* is the distance from the origin in the vertical axis, and *σ* is the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation" \o "Standard deviation) of the Gaussian distribution, is the pi value, is the Euler number.

*A graphical representation of the 2D Gaussian distribution with mean (0,0) and σ = 1 is shown belowto the bellow.*

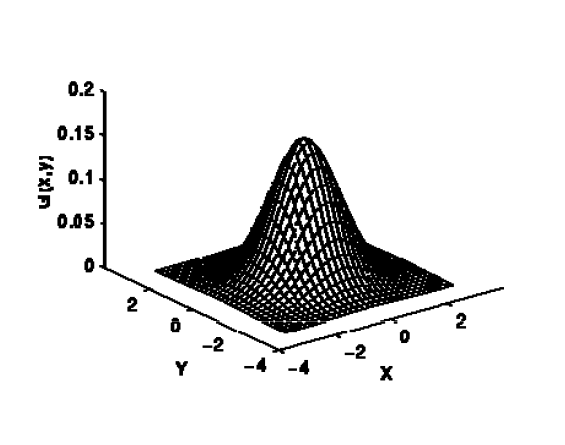
**

Figure 11: 2D Gaussian Distribution Function graph

*“The Gaussian filter works by using the 2D distribution as a point-spread function. This is achieved by convolving the 2D Gaussian distribution function with the image. We need to produce a discrete approximation to the Gaussian function. This theoretically requires an infinitely large convolution (Kernel, as the Gaussian distribution is non-zero everywhere. Fortunately, the distribution has approached very close to zero at about three standard deviations from the mean. 99% of the distribution falls within 3 standard deviations.*

*This means we can normally limit the kernel size to contain only values within three standard deviations of the mean.*

*Gaussian kernel coefficients are sampled from the 2D Gaussian function. Where σ is the standard deviation of the distribution.[[6]](#footnote-6)* (Gaussian Filtering, n.d.)*”*

*“An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image.[[7]](#footnote-7)* (Image Kernels, n.d.)”

Here is an example of a kernel which weights 2 in the center and 0 for the rest.

Table 1:Simple Kernel with only has center value

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

I will calculate a kernel using 2D Gaussian function and use this kernel to blur our picture. Kernel is can be named called mask or box.

Before going on I need to decide the size of my kernel. It should be big enough to blur our picture and also it shouldn’t be very big to make our calculations very complicated. I decided to use 5x5 kernel. Therefore I need to calculate these 25 values of my karnel.

I will calculate these values using the 2D Gaussian kernel function below. It is exactly same equation with Equation 2: 2D Gaussian Equation. I just changed its name because of its purpose. When we calculate our kernel the results in table 3 will be obtained. I assumed standart deviation () is “1”. I selected this value just for simplification. I will mention it in the limitation section.

# 

Equation 3: 2D Gaussian Kernel Function

Where *x* is the distance from the origin in the horizontal axis, *y* is the distance from the origin in the vertical axis, and *σ* is the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation" \o "Standard deviation) of the Gaussian distribution, is the pi value, is the Euler number.

Table 2:Calculation of Different values over 2D Gaussian Equation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| f(-2,-2)=0.00292  f(-2,-1)=0.01306  f(-2,0)=0.02154  f(-2,1)=0.01306  f(-2,2)=0.00292 | f(-1,-2)=0.01306  f(-1,-1)=0.05855  f(-1,0)=0.09653  f(-1,1)=0.05855  f(-1,2)=0.01306 | f(0,-2)=0.02154  f(0,-1)=0.09653  f(0,0)=0.15915  f(0,1)=0.09653  f(0,2)=0.02154 | f(1,-2)=0.01306  f(1,-1)=0.05855  f(1,0)=0.09653  f(1,1)=0.05855  f(1,2)=0.01306 | f(2,-2)=0.00292  f(2,-1)=0.01306  f(2,0)=0.02154  f(2,1)=0.01306  f(2,2)=0.00292 |

When we put the values into our kernel matrix will be like this:

Table 3: Kernel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.00292 | 0.01306 | 0.02154 | 0.01306 | 0.00292 |
| 0.01306 | 0.05855 | 0.09653 | 0.05855 | 0.01306 |
| 0.02154 | 0.09653 | 0.15915 | 0.09653 | 0.02154 |
| 0.01306 | 0.05855 | 0.09653 | 0.05855 | 0.01306 |
| 0.00292 | 0.01306 | 0.02154 | 0.01306 | 0.00292 |

This is our kernel. When we look at the table we can see that the center values, for instance f(0,0)=0.15915, is the highest value. It is expected because the gaussian function has the greatest value in the center. When ***x*** and ***y*** become zero, the 2D Gaussian equation becomes highest. In addition to this, the corner values,like f(-2,-2)=0.00292, are the minimum as expected again. When ***x*** and ***y*** become “-2,-2”, 2D Gaussian equation becomes lower than before.

The sum of our karnel values is 0.98179. Ideally our sum of kernel should be “1” instead of “0.981”. Since we round results and use 5x5 kernel instead of infinite kernel, sum of our kernel value is “0.981” instead of “1”. Therefore in the end we need to divide our result to this number. We will do this process because otherwise our results would be out of scale.

* 1. **Convolution Process**

So far we perapared our picture and calculated the kernel. The next step is going to apply convolution this two objects.

*“In mathematics, convolution is an operation performed on two functions (f and g) to produce a third function. Convolution is one of the most important operations in signal and image processing. It could operate in 1D (e.g. speech processing), 2D (e.g. image processing) or 3D (video processing).*

In image processing, convolution is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image. The kernel is a matrix of values whose size and values determine the transformation effect of the convolution process.

The Convolution Process involves these steps.

*(1) It places the Kernel Matrix over each pixel of the image (ensuring that the full Kernel is within the image), multiplies each value of the Kernel with the corresponding pixel it is over.*

*(2) Then, sums the resulting multiplied values and returns the resulting value as the new value of the center pixel.*

*(3) This process is repeated across the entire image.*

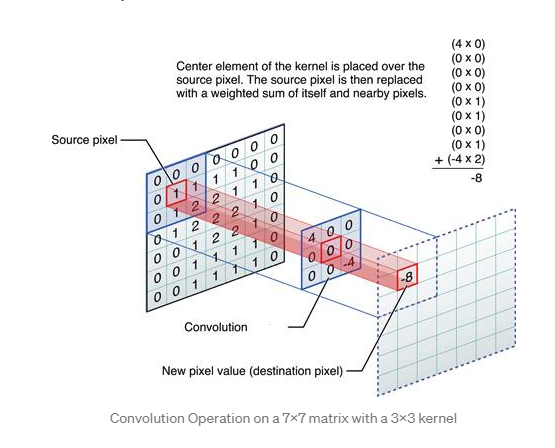
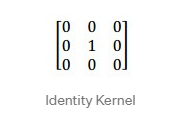


Figure 12:Convolution Operation on a 7x7 matrix with a 3x3 kernel

As we see in theFigure 12, a 3x3 kernel is convoluted over a 7x7 source image. Center Element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and surrounding pixels. The output is placed in the destination pixel value. In this example, at the first position, we have 0 in source pixel and 4 in the kernel. 4x0 is 0, then moving to the next pixel we have 0 and 0 in both places. 0x0 is 0. Then again, 0x0 is 0. Next at the center, there is 1 in the source image and 0 in the corresponding position of kernel. 0x1 is 0. Then again, 0x1 is 0. Then 0x0 is 0 and 0x1 is 0 and at the last position it is -4x2 which is -8. Now summing up all these results, we get -8 as the answer so the output of this convolution operation is -8. This result is updated in the Destination image.

The output of the convolution process changes with the changing kernel values. For example, an Identity Kernel shown below, when applied to an image through convolution, will have no effect on the resulting image. Every pixel will retain its original value as shown in the following figure.[[8]](#footnote-8) (Madhushree, 2019)”

Table 4:Identity Kernel



* + 1. **Calculating 1’st Pixel Of Our Picture:**

I will aply convolution and calculete first pixel of our new image. We need to use top left 25 pixels of our existing data to calculate this pixel. These are the first 5 rows and 5 columns (0 through 4) of our data. The pixels are shown below and circled in red. Center pixel is marked to blue for future using.

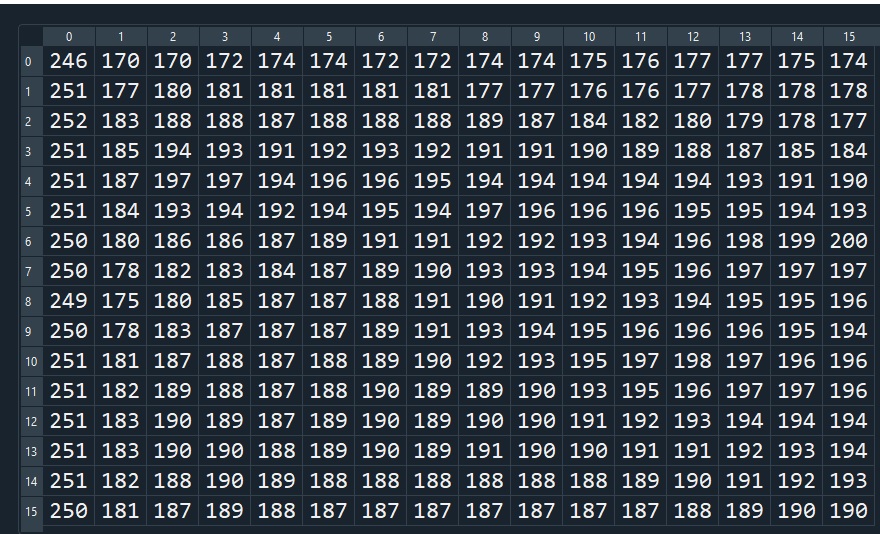


Figure 13: Data and area of which we make calculations

As mentioned in convolution process step 1, I basically place our kernel, which is calculated before, over the picture’s pixels above. Then, multiply each kernel values with the corresponding pixel over. This is the figure that how I calculate convolution values:

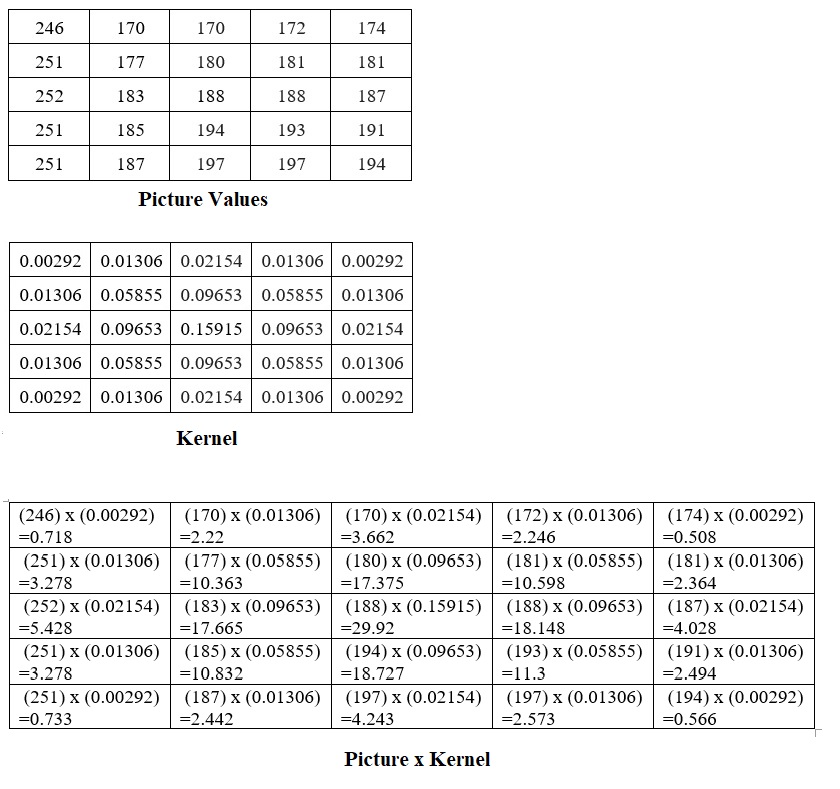


Figure 14: Convolution Process for 1'st Pixel ( To avoid confusion only 3 of the 25 arrows are shown)

In other words, I multiply each picture pixel with the related kernel value. This process can also be expressed as below:

Table 5:Results for the 1'st Step of Convolution Process

|  |  |
| --- | --- |
| picture(0,0) x kernel(0,0) = 0.718  picture(0,1) x kernel(0,1) = 2.22  picture(0,2) x kernel(0,2) = 3.662  picture(0,3) x kernel(0,3) = 2.246  picture(0,4) x kernel(0,4) = 0.508  picture(1,0) x kernel(1,0) = 3.278  picture(1,1) x kernel(1,1) = 10.363  picture(1,2) x kernel(1,2) = 17.375  picture(1,3) x kernel(1,3) = 10.598  picture(1,4) x kernel(1,4) = 2.364  picture(2,0) x kernel(2,0) = 5.428  picture(2,1) x kernel(2,1) = 17.665  picture(2,2) x kernel(2,2) = 29.92 | picture(2,3) x kernel(2,3) = 18.148  picture(2,4) x kernel(2,4) = 4.028  picture(3,0) x kernel(3,0) = 3.278  picture(3,1) x kernel(3,1) = 10.832  picture(3,2) x kernel(3,2) = 18.727  picture(3,3) x kernel(3,3) = 11.3  picture(3,4) x kernel(3,4) = 2.494  picture(4,0) x kernel(4,0) = 0.733  picture(4,1) x kernel(4,1) = 2.442  picture(4,2) x kernel(4,2) = 4.243  picture(4,3) x kernel(4,3) = 2.573  picture(4,4) x kernel(4,4) = 0.566 |

I am following convolution process steps. Now we are on 2’nd step of convolution process, which is sum of all the results. I add all values eachother like 0.718+2.22+3.66+2.24+… Sum of the all convulution values above is “185.71”.

Now, we need to divide this value to the sum of our kernel which is 0,981. We did this process because otherwise our results would be out of scale. First we multiply picture values with our kernel which sum of coefficients is 0,981, then divide the result with this number. Therefore the coefficient of our result is “1”. I explained this process detailed in “kernel calculation” . Finally, our new image’s first pixel O(0,0) is will be like this:

## 

Equation 4:Calculation formula of O(0,0)

## =189,155

Equation 5: Mathematical Operation

“189” (simply round 189,155 to 189) is the first pixel’s value of our new filtered picture. Our original value of pixel was “188”, which is the center and marked as blue in Figure 13 . İt is very close to our center value as expected. But it is a little distorted by the peculiar weight of other neighboring cells. Remember that “255” is the maximum value of a pixel and which means white. Therefore, our first pixel is now closer to white boundary (closer to 255) so our first pixel became lighter than the original. This is because neighboring cells have higher values than center value. If we calculated this value below “188”, which is our original pixel value, this means that our new picture’s first pixel would be darker than the original. Because the lower the pixel value, the darker it will be.

I just calculated the first pixel value of our new picture which is “189”. I will write it as a fist pixel of our new filtered image. This is how our new image look like:

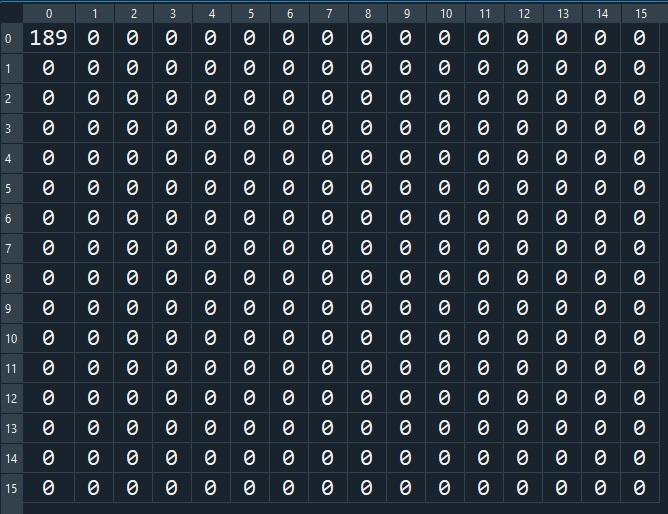


Figure 15: First Pixel of New image

Here is the visualization of our new picture. Remember that we just had 1 pixel value. Therefore the other pixel is shown as a black (0 value).

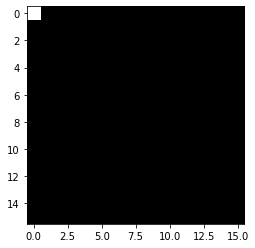


Figure 16: Visualization of New İmage’s first pixel

* + 1. **Calculating 2’nd Pixel Of Our Picture:**

In order to calculate the next pixel we need to shift our kernel to the right for 1 pixel. Then we will do exactly the same things above with our new pixel values. So that I will not write all the steps. Our new sort of interest pixels are like this:

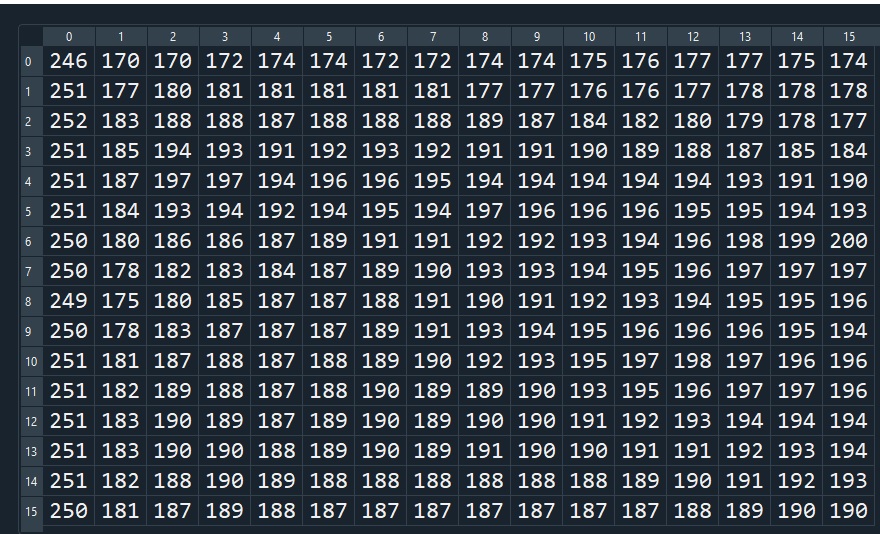


Figure 17: Data and Area of which we make new calculations

When calculating the first pixel, I started to calculate from picture’s (0,0) pixel. It was the first pixel of the picture. Now, I start from the picture’s (0,1) pixel because our kernel is shifted 1 pixel to the right.

When we multiply our kernel with our new sort of interest pixels, the results will be as follows:

|  |  |
| --- | --- |
| picture(0,1) x kernel(0,0)=0.496  picture(0,2) x kernel(0,1)=2.22  picture(0,3) x kernel(0,2)=3.705  picture(0,4) x kernel(0,3)=2.272  picture(0,5) x kernel(0,4)=0.508  picture(1,1) x kernel(1,0)=2.312  picture(1,2) x kernel(1,1)=10.539  picture(1,3) x kernel(1,2)=17.472  picture(1,4) x kernel(1,3)=10.598  picture(1,5) x kernel(1,4)=2.364  picture(2,1) x kernel(2,0)=3.942  picture(2,2) x kernel(2,1)=18.148  picture(2,3) x kernel(2,2)=29.92 | picture(2,4) x kernel(2,3)=18.051  picture(2,5) x kernel(2,4)=4.05  picture(3,1) x kernel(3,0)=2.416  picture(3,2) x kernel(3,1)=11.359  picture(3,3) x kernel(3,2)=18.63  picture(3,4) x kernel(3,3)=11.183  picture(3,5) x kernel(3,4)=2.508  picture(4,1) x kernel(4,0)=0.546  picture(4,2) x kernel(4,1)=2.573  picture(4,3) x kernel(4,2)=4.243  picture(4,4) x kernel(4,3)=2.534  picture(4,5) x kernel(4,4)=0.572 |

Now I need to add all values eachother. Sum of the all multiplication values above is “183.16 “. Then i need to divide this value with sum of kernel values as i explained before.

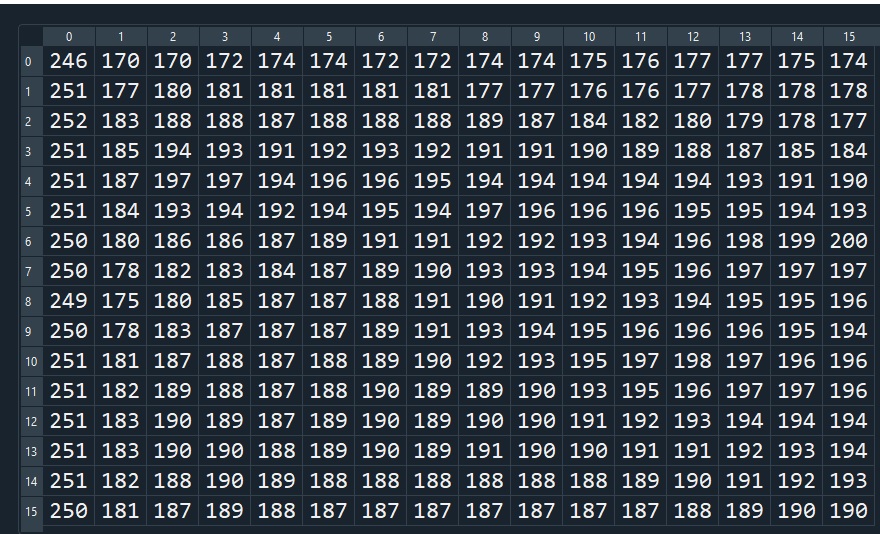
## 

Equation 6: Calculation formula of Second pixel ( O(0,1) )

## 

Equation 7: Mathematical Operation to Calculte Second Pixel

I just calculated the second pixel value of our new picture which is “187”. İt is again very close to our center value, which is “188”. But now it is a little darker. İt is darker because in this calculation we use darker pixels. In the first pixel calculation, I used the first five rows of the first column which are “246,251,252,251,251”. In the second pixel calculation, these values are not in our sort of interest pixels. Instead of these values we use new values of the first 5 rows of 6’th columns which are “174,181,188,192,196”. Therefore our final value became a little smaller than before.



Exclude

Include

Center Value

Kernel Box

Figure 18: Explanation of differances Between pixels

This is our new image:

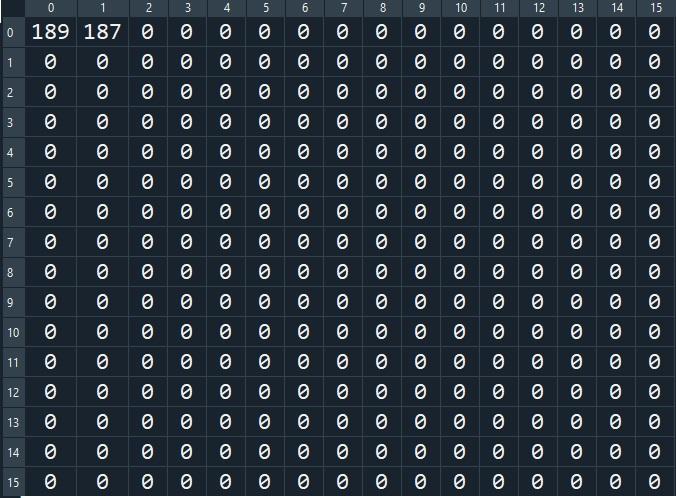


Figure 19: First and Second Pixels of New image

Here is the visualization of our new picture with new values. Now we had 2 pixel values;

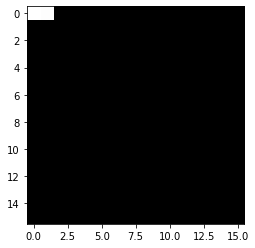


Figure 20: Visualization of New İmage’s First and Second Pixels

* + 1. **Calculations The Other Pixels of Our Picture:**

Calculating the other pixels is the same as above. The thing is I just need to shift the kernel box pixel by pixel. There are 65536 pixels in my picture. So far I calculated 2 pixel. There are 65534 more. Therefore I need use computer to finish calculations. Python Programming Language to calculate the rest of the values. The code that I used is as follows;

new\_picture=np.zeros((252,252))

for row in range (252):

for column in range (252):

sum=0

for i in range(5):

for j in range(5):

conv=kernel[i,j]\*pix\_val[i+row,j+column]

sum+=conv

new\_picture[row,column]=sum/(0.98179)

This code does the same thing with we did in previus section. I simply wrote 3 “For loop” and calculate results. As a result, computer calculated all pixels for me.

This is the first 16 rows and columns of our result, And now it is our new filtered image;

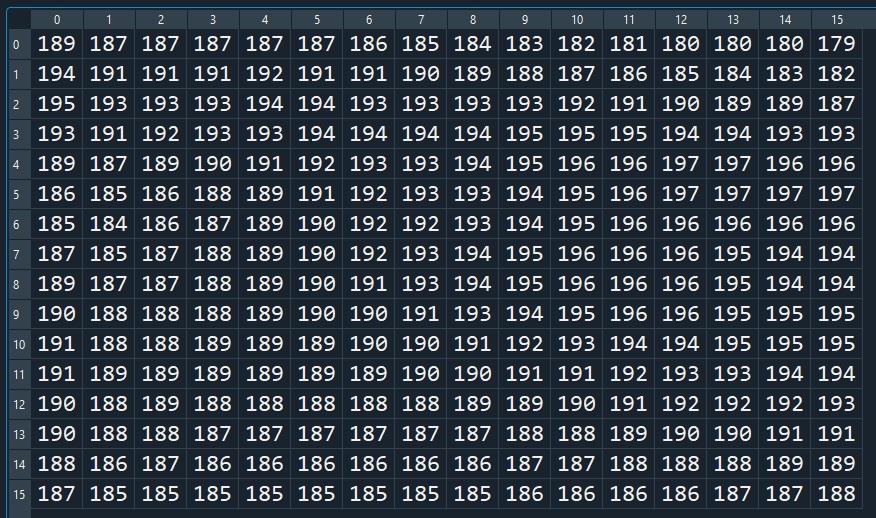
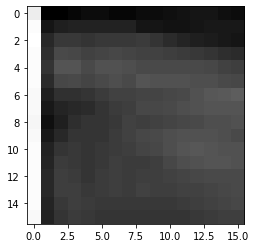
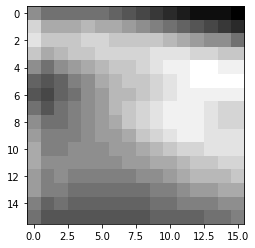


Figure 21: First 16 rows and 16 columns of Blurred(filtered) New Data

* + 1. **Creating A New Picture**

We finished convolution process and calculated our filtered image pixel values. Now we need to create new image from new pixels. In other words we need to visiulize the new calculated pixel previus section. I used Python programming language same as before for this operation. Here is the visualization of our new picture with new values and original picture. Now we had 65536 pixel values and first 256 is like this.

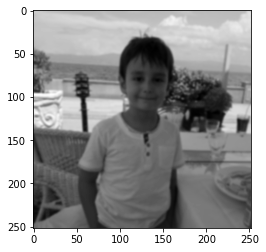


Orginal Image

Filtered Image

Figure 22: First 16 rows and 16 colums of Orginal and Filtered Image

Here is the filtered image and orginal image with all 65536 pixels;



Orginal Image

Filtered Image

Figure 23: Orginal Image and Filtired Image(result)

1. **CONCLUSION:**

Digital image processing is a very common concept to make calculations and analyze images. Gaussian blur is one of the key techniques of digital image processing. In this assessment, we tried to understand this technique step by step and to discover mathematics behind it.

We investigated each pixel in the picture one by one, and we changed it by looking at the values ​​in the neighboring cells. Since we used 5x5 Kernel, we take into account 24 neighboring cells (1 is the original pixel) while calculating new pixel. Then we used Gaussian equation and several methots to make this calculation. There were some important operations while we blur. I will explain them in several ways. First, we use Gauss distrubution function to calculate kernel. Sigma) value in *Equation 3* 2D Gaussian Kernel Function was the significant constant for these calculations. We assumed sigma) value as “1”. Therefore, we obtain that a fair blurness on the final picture. I will give detailed information to this subject inlimitation and improvement part.

As a result, as we see in *Figure 22* Orginal Image and Filtired Image(result), we achived to blur our picture using the Gaussian blur technique. A painter mixes the colours with his brush in order to blur his paint. We understand that, like a painter, in digital image processing we shuffle the numbers in order to blur the picture. Shuffling methods can be done in many different ways but Gauissan Blur technique is the one of the best option and the interesting one.

Overall, we can easily say that gaussian blurring is a very easy and useful technique to blur images. It has various options to make different blurring over the given image and it is used in a vast area.

1. **LIMITATION AND IMPROVEMENT:**
   * 1. **Importance Of Sigma() Value:**

Sigma value is an important parameter of calculations. Sigma value changes the shape of gaussian distribution. This causes kernel values are more same like each other. As a result, the sigma value defines how much blur will be on the image. The higher sigma number, the more blur there is on the image.

As Imentioned before, we took sigma as “1” and the result is above. if we took it as “2”, it is obvious that the picture would be more blurred.

* + 1. **Bokeh Effect:**

In this exploration I blurred all parts of the image, not any specific part. In bokeh effect, that our phone has it as a camera mode (portrait mode), picture is blurred around the face. In other words, the face is not blurred but the rest of the image blurred. Unfortunately I couldn’t do it. Because face detection is a very hard process and out of our subject topic. We basically blur all the image and just think of mathematics behind.

* + 1. **Programming Language and Calculations:**

I need to use some programming language to make calculations. However, I have limited knowledge background of computer programming. Therefore, I can do only simple calculations and a very limited image process.

* + 1. **Background Noise Cancelation:**

Gaussian filtering is widely used for background noise cancellation and smoothing the images. This is commonly used for determine objects more clearly. I can use different kernels to make images more clear.

* + 1. **Edge Detection:**

Gaussian filtering is the first step to edge detection. Edge detection is commonly used to understand and analyze the unknown areas. We can use gaussian equations to find edges and use them for different projects.

1. **BIBLIOGRAPHY:**

# Bibliography

(n.d.). Retrieved from Gaussian blur: https://en.wikipedia.org/wiki/Gaussian\_blur#:~:text=In%20image%20processing%2C%20a%20Gaussian,image%20noise%20and%20reduce%20detail.

(n.d.). Retrieved from Gaussian Distribution Function: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/gaufcn.html#:~:text=Gaussian%20Distribution%20Function&text=The%20nature%20of%20the%20gaussian,over%20from%20the%20binomial%20distribution%20).

(n.d.). Retrieved from digital image: https://northcoastphoto.com/digital-explained/#:~:text=Q%3A%20What%20is%20a%20Pixel,accurate%20representations%20of%20the%20original.

*Gaussian Filtering*. (n.d.). Retrieved from University of Auckland: https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering\_1up.pdf

*Image Kernels*. (n.d.). Retrieved from https://setosa.io/ev/image-kernels/

Madhushree, B. (2019, 03 30). Retrieved from 6 basic things to know about Convolution: https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

1. **LIST OF EQUATIONS, TABLES AND FIGURES**

[Equation 1:Gaussian Function 10](#_Toc77101266)

[Equation 2: 2D Gaussian Equation 10](#_Toc77101267)

[Equation 3: 2D Gaussian Kernel Function 12](#_Toc77101268)

[Equation 4:Calculation formula of O(0,0) 17](#_Toc77101269)

[Equation 5: Mathematical Operation 17](#_Toc77101270)

[Equation 6: Calculation formula of Second pixel ( O(0,1) ) 20](#_Toc77101271)

[Equation 7: Mathematical Operation to Calculte Second Pixel 20](#_Toc77101272)

[Table 1:Simple Kernel with only has center value 11](#_Toc77101274)

[Table 2:Calculation of Different values over 2D Gaussian Equation 12](#_Toc77101275)

[Table 3: Kernel 12](#_Toc77101276)

[Table 4:Identity Kernel 14](#_Toc77101277)

[Table 5:Results for the 1'st Step of Convolution Process 16](#_Toc77101278)

[Figure 1: Gaussian Blur 3](#_Toc77101281)

[Figure 2 :Original Coloured Data 4](#_Toc77101282)

[Figure 3: Data After Resize operation 5](#_Toc77101283)

[Figure 4: Grayscaled data 6](#_Toc77101284)

[Figure 5 : Mario with 16 bit pixels / Grayscele colourmap 6](#_Toc77101285)

[Figure 6:Pixel visualisation and numbered pixel 7](#_Toc77101286)

[Figure 7: First 16 Rows and 16 Columns of Data 8](#_Toc77101287)

[Figure 8: Visualized data of first 16 rows and 16 columns / Location of first 16 rows and 16 columns on grayscaled image 8](#_Toc77101288)

[Figure 9:Distribution of the Gaussian function values 9](#_Toc77101289)

[Figure 10:Gaussian Distribution Function 10](#_Toc77101290)

[Figure 11: 2D Gaussian Distribution Function graph 11](#_Toc77101291)

[Figure 12:Convolution Operation on a 7x7 matrix with a 3x3 kernel 13](#_Toc77101292)

[Figure 13: Data and area of which we make calculations 15](#_Toc77101293)

[Figure 14: Convolution Process for 1'st Pixel ( To avoid confusion only 3 of the 25 arrows are shown) 16](#_Toc77101294)

[Figure 15: First Pixel of New image 18](#_Toc77101295)

[Figure 16: Visualization of New İmage’s first pixel 18](#_Toc77101296)

[Figure 17: Data and Area of which we make new calculations 19](#_Toc77101297)

[Figure 18: Explanation of differances Between pixels 20](#_Toc77101298)

[Figure 19: First and Second Pixels of New image 21](#_Toc77101299)

[Figure 20: Visualization of New İmage’s First and Second Pixels 21](#_Toc77101300)

[Figure 21: First 16 rows and 16 columns of Blurred(filtered) New Data 22](#_Toc77101301)

[Figure 22: First 16 rows and 16 colums of Orginal and Filtered Image 23](#_Toc77101302)

[Figure 23: Orginal Image and Filtired Image(result) 23](#_Toc77101303)

1. **APPENDIX**

This is the Phyton Program codes to calculate kernel

This is the Phyton Program codes to apply convolution

1. (Gaussian blur, n.d.) https://en.wikipedia.org/wiki/Gaussian\_blur#:~:text=In%20image%20processing%2C%20a%20Gaussian,image%20noise%20and%20reduce%20detail. [↑](#footnote-ref-1)
2. https://northcoastphoto.com/digital-explained/#:~:text=Q%3A%20What%20is%20a%20Pixel,accurate%20representations%20of%20the%20original. (digital image, n.d.) [↑](#footnote-ref-2)
3. https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html [↑](#footnote-ref-3)
4. https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering\_1up.pdf (Gaussian Filtering, n.d.) [↑](#footnote-ref-4)
5. http://hyperphysics.phy-astr.gsu.edu/hbase/Math/gaufcn.html#:~:text=Gaussian%20Distribution%20Function&text=The%20nature%20of%20the%20gaussian,over%20from%20the%20binomial%20distribution%20). (Gaussian Distribution Function, n.d.) [↑](#footnote-ref-5)
6. https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering\_1up.pdf (Gaussian Filtering, n.d.) [↑](#footnote-ref-6)
7. <https://setosa.io/ev/image-kernels/> (Image Kernels, n.d.) [↑](#footnote-ref-7)
8. https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411 (Madhushree, 2019) [↑](#footnote-ref-8)