An Architectural Design For Gaussian Filter

Submitted By:

Aditya Raj Singh Gour(19116005)

Gopal Gupta(19116025)

Hardik Rathore(19116028)

Rudraksh Aggarwal(19116061)

Introduction

Gaussian Smoothing

The Gaussian Filter is one of the processing methods to smoothen images. Before detecting an object, this method is used as pre-processing step, for eliminating useless detail, and connecting unnatural parts. Gaussian filter is a low pass linear filter as it reduces high frequency components of the image. Mathematically, applying a Gaussian filter to an image is the same as convolving the image with a Gaussian kernel.

Gaussian Kernel

Because the Gaussian function has infinite support (meaning it is non-zero everywhere), the approximation would require an infinite convolution kernel. So, we need to limit the kernel size. For Gaussian, 99.3% of the distribution falls within 3 standard deviations after which the values are effectively close to zero. So, we limit the kernel size to contain only values within 3σ from the mean. This approximation generally yields a result sufficiently close to that obtained by the entire Gaussian distribution.

The Gaussian kernel weights(1-D) can be obtained quickly using the Pascal's Triangle. For instance the third row corresponds to the 3×3 filter as demonstrated below.

```
                  1
               1     1
            1     2     1
         1     3     3     1
      1     4     6     4     1
   1     5    10    10     5     1
1     6    15    20    15     6     1
```

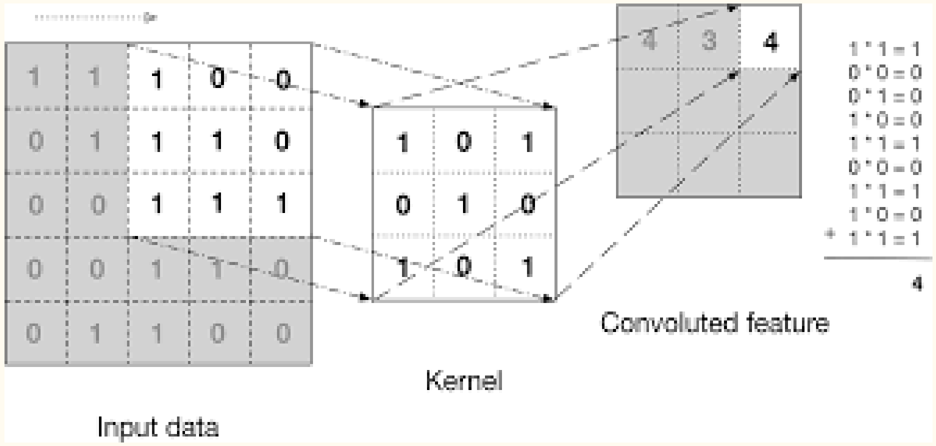
Pascal Triangle

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

Using the third row of Pascal Triangle to make a 3x3 Kernel

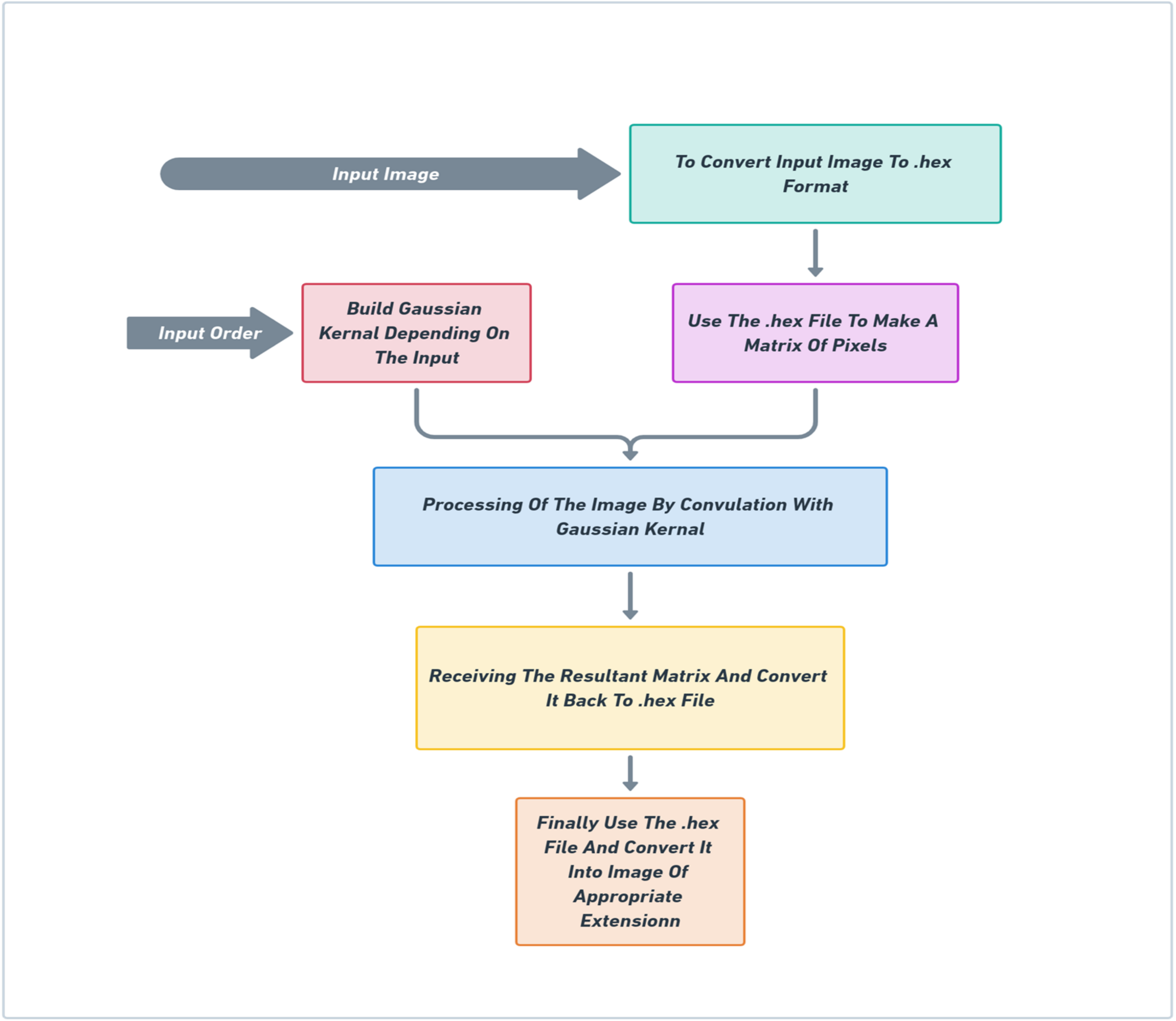
Convolution

In the process of Gaussian convolution one of the pixels is taken as centre the corresponding adjacent elements of the image are multiplied with the elements of the kernel and the the sum is calculated. After that the concerned pixel of the image is replaced by the sum calculated before. In this process the concerned pixel of the image is given more weight than distant pixels as it happens in a Gaussian function.



Demonstration of Convolution Process

Flow Chart


Input Image
To Convert Input Image To .hex Format
Input Order
Build Gaussian Kernal Depending On The Input
Use The .hex File To Make A Matrix Of Pixels
Processing Of The Image By Convulation With Gaussian Kernal
Receiving The Resultant Matrix And Convert It Back To .hex File
Finally Use The .hex File And Convert It Into Image Of Appropriate Extensionn

Architecture of the Design: The Complete Project

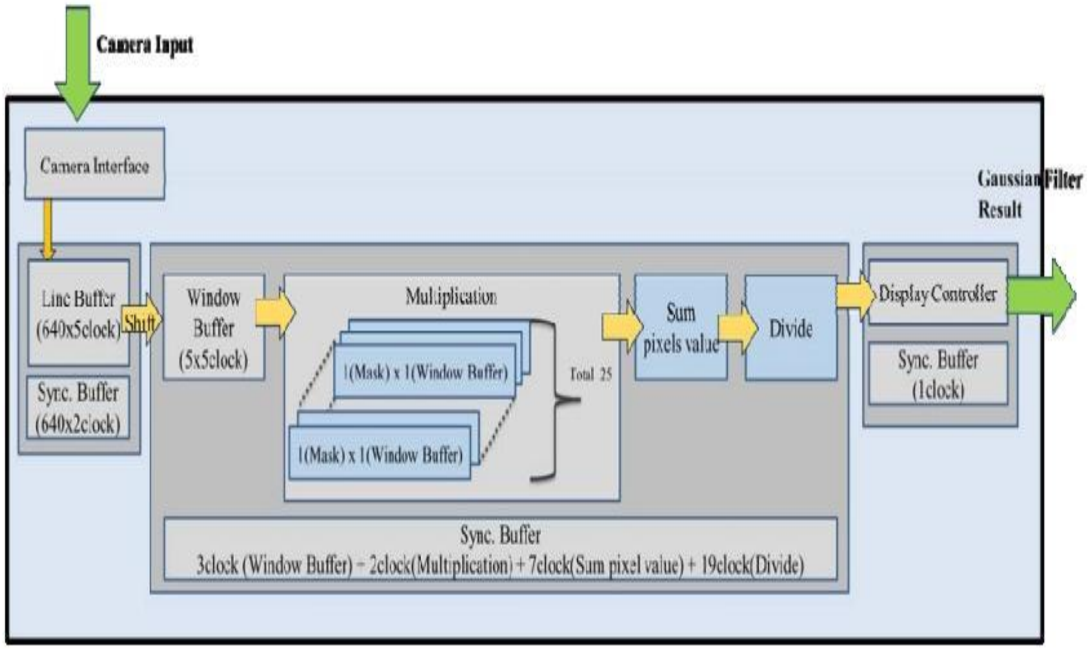


Figure 4. Proposed hardware architecture.

Initially we were planning to make a hardware design as shown above, but due to **unavailability** of resources, we were **not able** to work on any **hardware** related stuff. Hence, we limited ourselves to just implement the **Software Component(Verilog Code)** of the Overall project.

Architecture of the Design: The Software part

Following is the detailed procedure that take place for applying Gaussian filter on the image.

First of all, we have made **two different testbenches** for this project one for applying **Gaussian filter to Colored images (testbench_color.v)** and other for applying filter on **Grayscale images (testbench.v).**

1. We need to give the modules of our test bench input parameters - the input image , the kernel matrix order an integer in the range of {3,5,7}, rows of image, columns of image and size of image.
2. Now this input image is required to be in a .hex file. So we need to convert the image file to .hex because Verilog is not able to read a .jpg/.bmp file directly. For this we have written two codes one in **MATLAB** for converting .bmp file to .hex and other in **PYTHON** for converting .jpg file to .hex.(**NOTE: Make sure that image is needed to be in Grayscale for using testbench.v**)

Architecture of the Design: The Software part

3) Now we will discuss about the structure of the module called by our testbench:

   a) *First a function(**Verilog $readmemh() property**) is called that converts the .hex file to a Rectangular Matrix of order* M X N *(passed as a row and column parameter by our testbench)*

   b) *Then from the integer input we received ,by using one of our self made function it designs a Kernel(using **Verilog functions and loops)**.*

   c) *Once our **Kernel** and **Matrix** are made the module performs the processing of the image matrix (**backbone of the project)** i.e. **CONVOLUTION** using another of our function, which finally gives an **output matrix** of **order*** **M X N***.*

   d) *Finally this output matrix is converted to a .hex file for further use and marking **the end of our verilog code.***

Architecture of the Design: The Software part

4. Now, from this .hex file we generate the new output image Using **PYTHON.**
5. Also, to test our result we directly apply gaussian filter on the same image using one of the function in **PYTHON**.

The **new image** is the same image as input but with the **Gaussian Filter applied** on it i.e. we get a much **smoother image** than the input which is **all set to be used for further processing**.

Major Challenges faced

We had faced a few problems while working on this project, some are because of lack of familiarity with few concepts of verilog and others due to the queries that we were unable to get answer of from the internet:

- Processing of Edge pixels, there were majorly three different approaches for applying gaussian filter on edge pixels while convolution each having its own demerit.We were unable to decide which method were we supposed to choose, So we went for the one we felt was the best.

- On our first few stimulation test program was hanging and after some debugging we realized the code went into infinite loop and fixed the same.

Some more General problems

1. Using modules in verilog- We initially got confused between difference between modules and functions and thought modules could return values but soon realized our mistake.Also we were unable to initiate our process module from testbench but then fixed it.

2. We wanted to take an input for the order of our Gaussian kernel at run time but was unable to find if its possible. Hence we decided on an alternative where the user could pass a parameter in the test bench called ksize while calling the module named process which will be the order of the kernel user want to apply.

3. Initially we wrote code for applying Gaussian filter only on Grayscale image as was finding working with colored images difficult .But finally worked out on another code and added another testbench for colored images.

Some more General problems

- Currently, we have to send the image converted to hex file to verilog code manually and then the generated hex file by verilog to python for conversion to output image manually too. We tried automating this process but failed to do so.

We also faced some problem due to the ongoing Covid-19 pandemic :

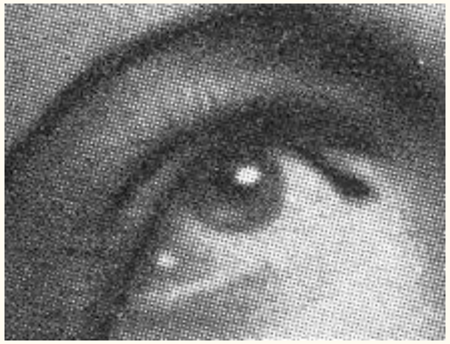
- Hardware implementation of the project was not possible.
- Lack of proper guidance in person also affected the project.Also faced problems while communicating with team members.

Some more General problems

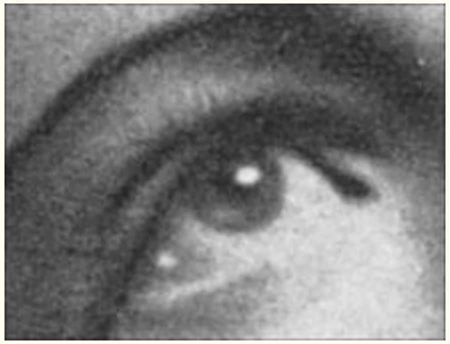
- Due to the lack of proper internet facilities we were unable to download MATLAB software due to its large size. So we had to use Python for conversion of image file to .hex file and vice versa .But after sometime came to know about online MATLAB, hence wrote a code for converting .bmp image to .hex in MATLAB to and will be attaching python as well as MATLAB code for the same.

Simulation Results

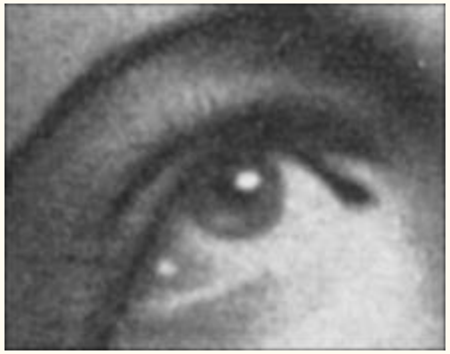
INPUT IMAGE



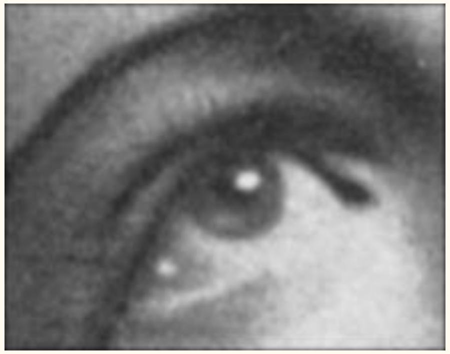
OUTPUT IMAGE



Order 3 Kernel



Order 5 Kernel

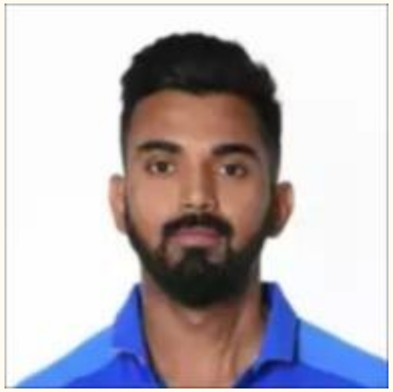


Order 7 Kernel

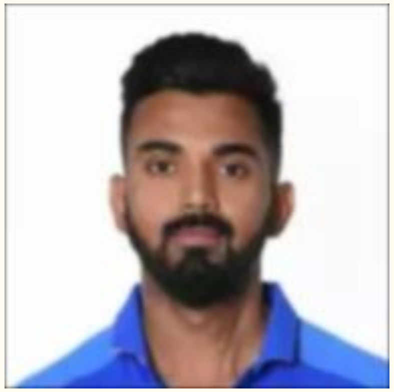
Simulation Results

INPUT IMAGE

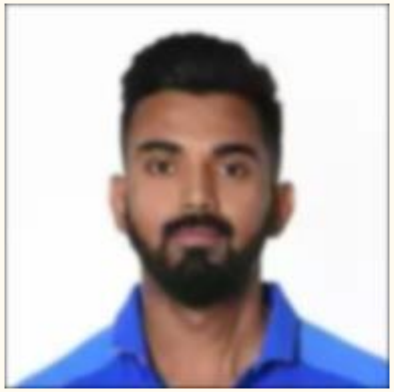
OUTPUT IMAGE



Order 3 Kernel



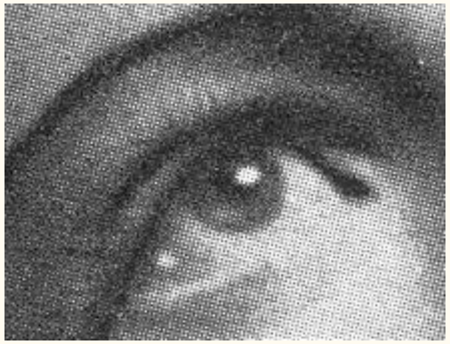
Order 5 Kernel



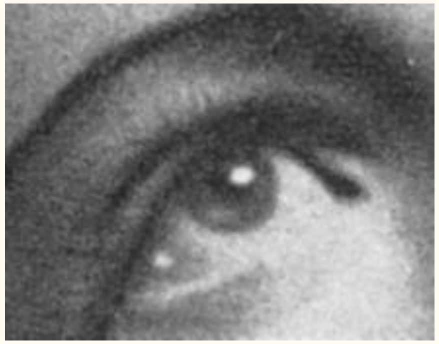
Order 7 Kernel

VERIFICATION OF RESULT

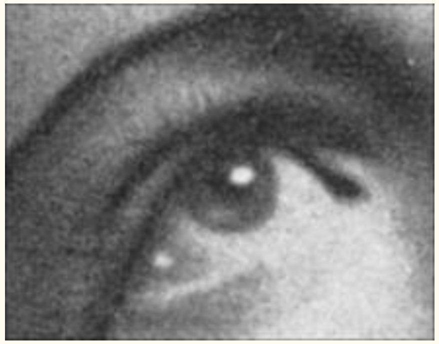
INPUT IMAGE



PYTHON FILTER RESULT

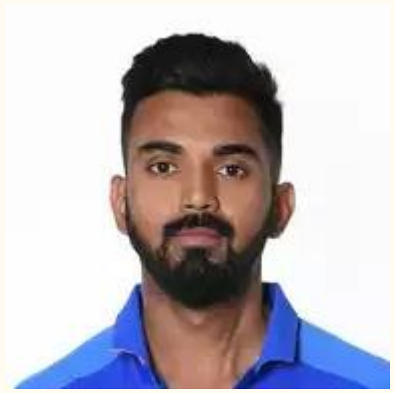


OUR OUTPUT RESULT

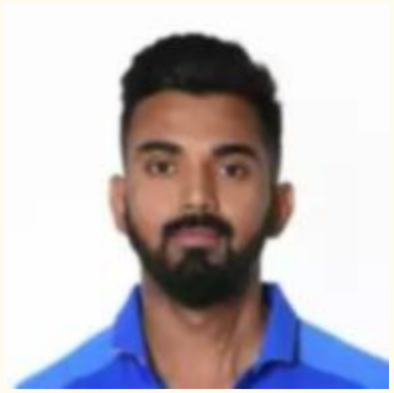


VERIFICATIO N OF RESULT

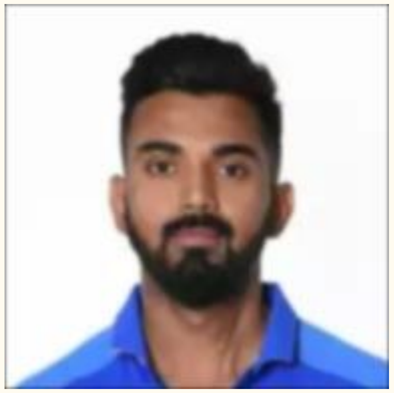
INPUT IMAGE



PYTHON FILTER RESULT



OUR OUTPUT RESULT



Summary

This project demonstrates how a Gaussian filter is used as the pre processing step for image smoothing. We used Verilog(for applying Gaussian convolution) and Python(for converting image to .hex file and vice versa), also MATLAB(if want to use for conversion of .bmp image to .hex file but not vice a versa).

The overall learning experience was great though was full of problem and brainstorming for finding their solutions.

We divided this entire project into different phases:

1. First we searched and read about what exactly is Gaussian filter. And also read some papers about how image processing is done.

Summary

2. Then we planned and made a flowchart of various steps that will be involved in image processing from the point of receiving the image to creating output image.
3. Then each one of us individually wrote codes for different parts and finally after individual testing merged all the code fragments together and made them compatible.
4. Then we stimulated the code and caught some error hence debugged them.

Summary

5. After all this we were finally able to get an output image, but next step was to check the credibility of our result, for that we used Python and applied Gaussian filter on the original image using that and compared the result that we got from our verilog code. Thankfully, it matched with our result.
6. Though we completed the project there are still some shortcomings that could not be dealt due to time constraint but we will be working on them even after the submission.

The image and the processed output by the Gaussian filter are shown in the previous slides as the output.