# National Institute of Technology, Raipur


## Term Project
## Computer Vision
## CSE 6th Semester

**Roll number : 19115071**

**Name : Sahil Silare**

**Branch : Computer Science and Engineering**

**Semester : 6th**

**Submitted to: Dr. Sonal Yadav**

**Subject : Computer Vision**

**Date :07/04/2022**

**Project Link and Instructions:**
**https://github.com/sahil9001/Smoothing-Project-OpenCV**

# Abstract

Upon thinking about the topic regarding computer vision, my thoughts stumbled upon the topic of image processing, and through that I chose the topic “Image smoothing using OpenCV”. This topic caught my interest as I think that image smoothing can be applied in various real-world applications. Some of the examples include medical technology, image filters in apps, portrait mode images, noise reduction, etc. My term paper focuses on 4 different types of image smoothing algorithms provided in the OpenCV library. It aims to upload a sharp image and returns a smoothened image as response.

# Setup Instructions

Firstly, make sure that you have docker installed on your machine, else you can get it from https://docs.docker.com/engine/install/.

After installing docker on your system, the next step is to clone/download the repository and extract it on your local machine, once you’ve done this, we can move to the next step.

Next step is to open the extracted folder in VS Code with your preferred terminal open in the IDE. Then in the terminal, type **“docker-compose build”** to build the application.

```
gigawhiz@gigawhiz-laptop:~/Desktop/projects/smoothing-final-project$ docker-compose build
```

This step will build the frontend and backend in the docker image, the next step after the completion of the above step is to run the container in which the frontend and backend was built. You can do this by running the command **“docker-compose up”**

```
gigawhiz@gigawhiz-laptop:~/Desktop/projects/smoothing-final-project$ docker-compose up
```

**Application should be up and running, you can open your browser and go to “localhost:3000” to check it. Website demo below,**

# Welcome to Smoothen.ai

Your one-stop solution to get noise-free images! No more pixels which stand out. Only effortless smooth pictures.

Try For Free Learn More

## How it works?

Working with smooth.ai is easy and straightforward. Get started with these three simple steps!

**Upload image to smoothen**
Upload the image file from your system. Allowed formats (png,jpeg,jpg)

**Image gets processed**
Our software is at work! Sit back and relax. Your image will be ready in no time!

**Returns smoothened results**
Download your fresh & smooth image, without noise!

## Explore the solutions

Convert your noisy images with the help of this tool for free! Bring out the beauty of your pictures with Smooth.ai.

### Powerful suite of tools

Our simple and intuitive platform uses state-of-the-art technology to bring out the best in your pictures! We are constantly aiming towards providing a fast, efficient and hassle-free experience.

**Averaging Filter**

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element.

**Gaussian Blurring**

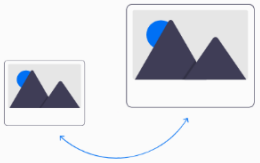
Gaussian smoothing is the result of blurring an image by a Gaussian function. Gaussian blurring is highly effective in removing Gaussian noise from an image.

**Median Blurring**

Median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image.

**Bilateral Filtering**

Highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters.



# Introduction

Image smoothing is an operation that's used to remove noise, sharpness and clutter in the image to give you a much smoother and blended effect. With the power of opencv and python, you can achieve several smoothing effects with a few lines of code.

Basically the way it works is first selecting a kernel (3x3, 5x6 etc) and then convolving with an image. The intensity of the centre pixel is determined by the pixel intensities of the neighbourhood pixel. Depending upon kernel values and the type of aggregation you get several smoothing effects

### Average Smoothing:

Here the kernel has uniform weights as shown below. Convolving with this filter simply yields average of pixel intensities in the neighbourhood

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

*5x5 averaging kernel*

We apply this filter on following image to see the averaging filter results,

*Before Average Smoothing*

Here is the output after applying average smoothing. As you can see noise( design details on building, trees) are removed giving you a much more smoother image

*After Average smoothing*

This can be achieved with just 1 line of code in opencv,

**blurred = cv2.blur(image,(5,5))**

*where 5x5 is kernel size*

**Gaussian Blur:**

In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, cv.GaussianBlur(). We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from

the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.

If you want, you can create a Gaussian kernel with the function, cv.getGaussianKernel().

The above code can be modified for Gaussian blurring:

```
blur = cv.GaussianBlur(img,(5,5),0)
```

| | | | | |
|---|---|---|---|---|
| w2 | w2 | w2 | w2 | w2 |
| w2 | w1 | wt1 | w1 | w2 |
| w2 | w1 | 1 | w1 | w2 |
| w2 | w1 | w1 | w1 | w2 |
| w2 | w2 | w2 | w2 | w2 |

*Gaussian Kernel and its effects*

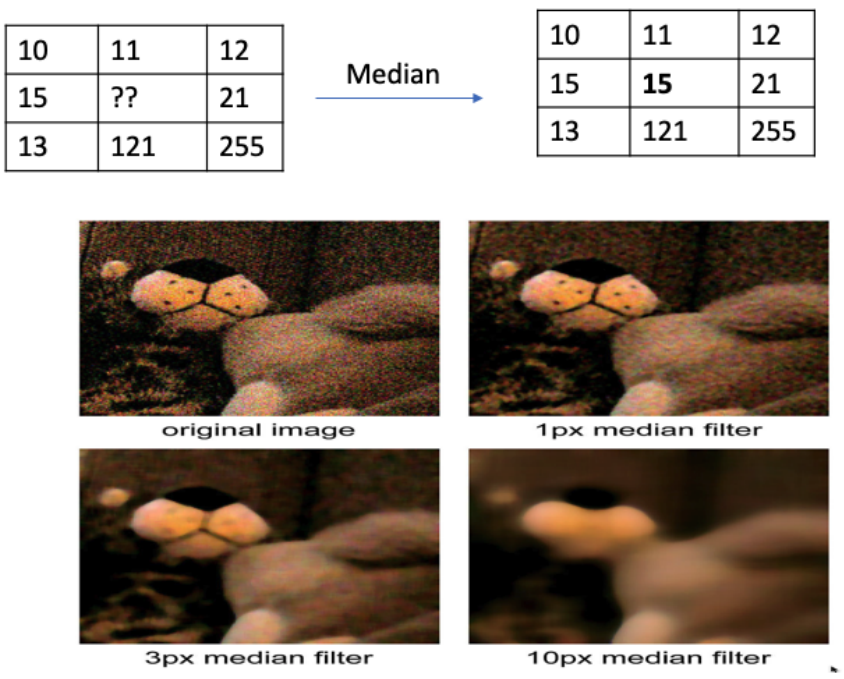
Gaussian blur usually appears more natural and good to start with first.

**Median Blur:**

Here, the function **cv.medianBlur()** takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

In this demo, I added a 50% noise to our original image and applied median blurring. Check the result:

```
median = cv.medianBlur(img,5)
```

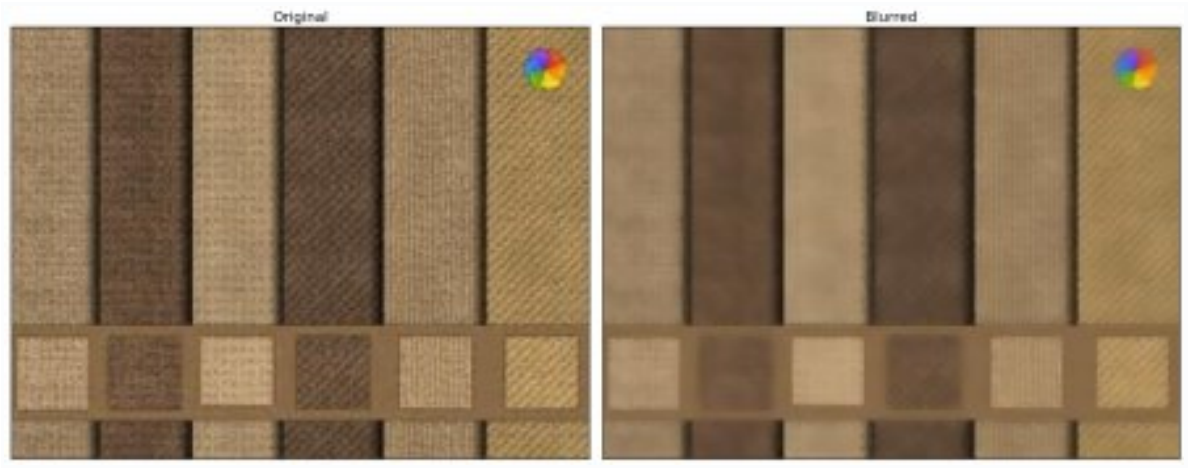



*Median Blur*

**Bilateral Blur:**

Bilateral filtering also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

```
blur = cv.bilateralFilter(img,9,75,75)
```




See, the texture on the surface is gone, but the edges are still preserved.