VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031
Department of Computer Science & Engineering

INNOVATION IN TEACHING

Course Name: Image Processing **Faculty Name:** Dr. Nagaratna P Hegde

Topic: Smoothing filter's **Year/Sem:** VI Sem

Teaching Aid / Tool Used: OpenCV

Description of the Tool:

OpenCV provides tools for image processing using C++ and Python. The goal is to provide image processing tool to the students and faculty to work on processing images which includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms etc.

Tool usage in Teaching:

Image Smoothing: Image Smoothing is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image. So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.

1. **Average 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. This is done by the function **cv.blur()** or **cv.boxFilter()**. Check the docs for more details about the kernel. We should specify the width and height of the kernel. A 3x3 normalized box filter would look like the below:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Check a sample demo below with a kernel of 5X5 size

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('opencv-logo-white.png')

blur = cv.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Result:

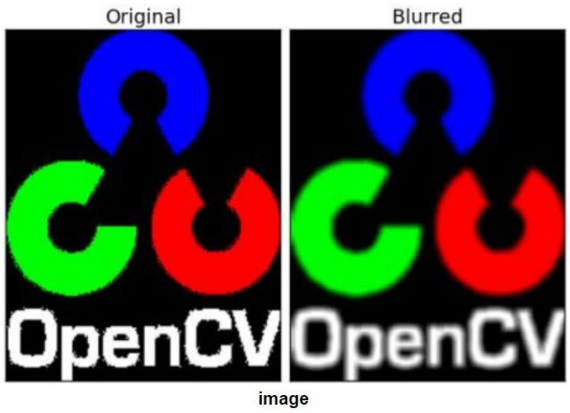

Figure1.(a) Original image (b) Blurred image after applying average filter

2. **Gaussian filter**: 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)
```

Result:

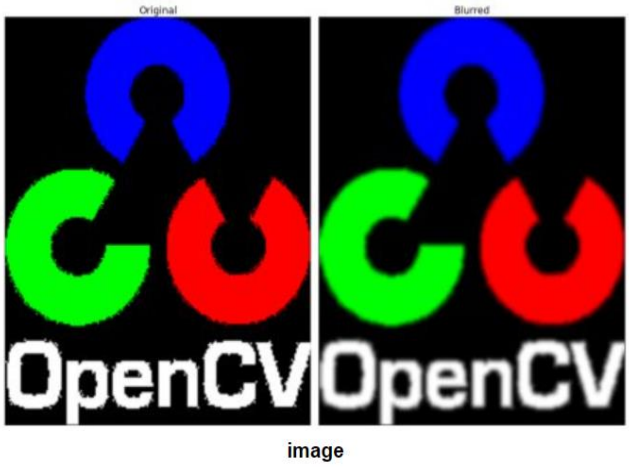

Figure 2.(a) Original image (b) Blurred image after applying Gaussian filter

3. **Median filter**: 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)
```

Result:

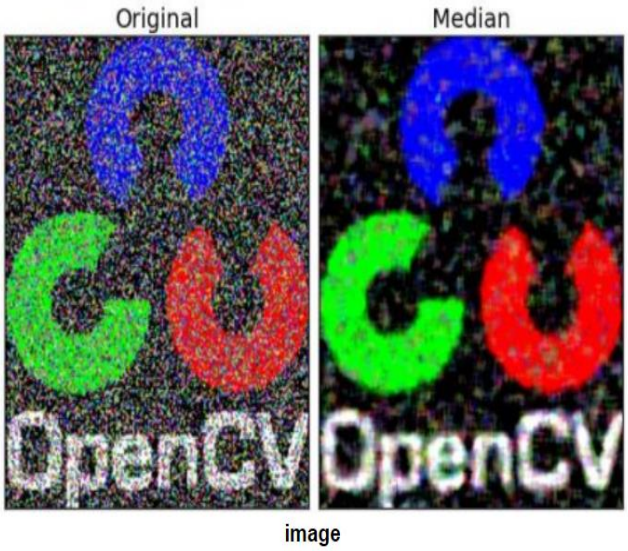

Figure 3.(a) Original image (b) Blurred image after applying Median filter

4. **Bilateral filter**: **cv.bilateralFilter()** is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It doesn't consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we don't want to do.

The below sample shows use of a bilateral filter

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

Result:

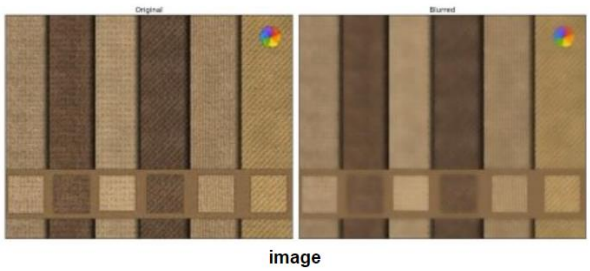

Figure 4.(a) Original image (b) Blurred image after applying bilateral filter