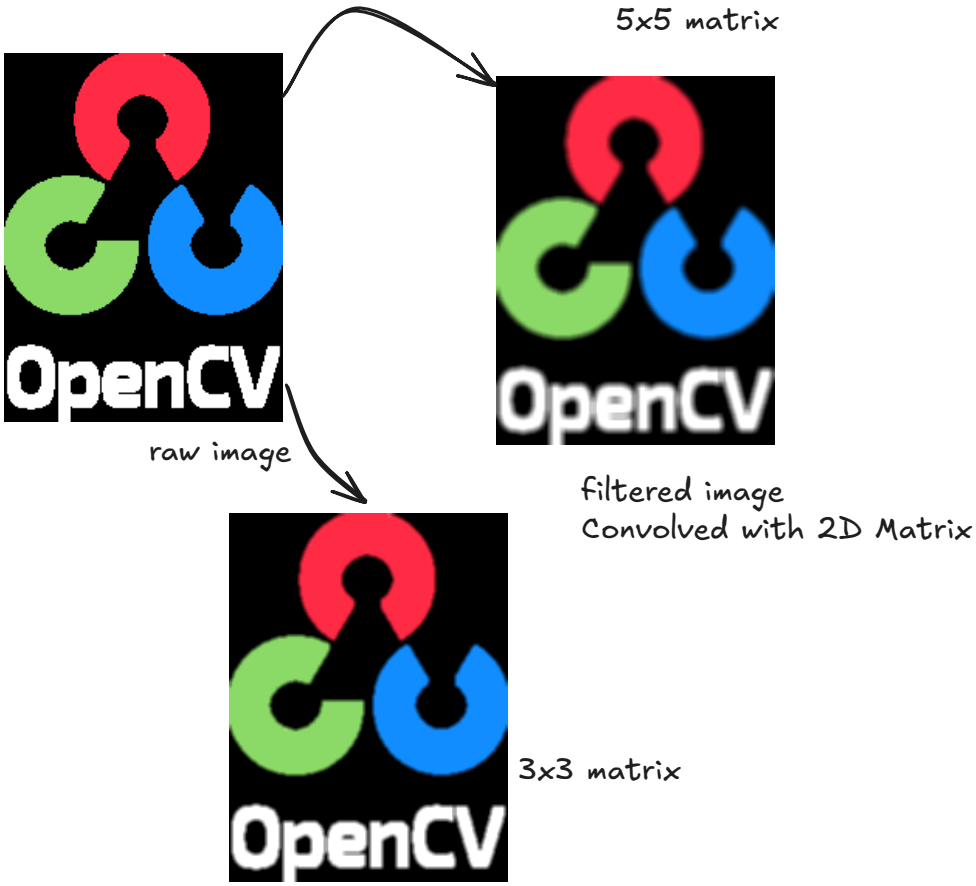
Image Processing - Filtering & Smoothing

Code Repository => https://github.com/Shiven-saini/OpenCV-Samples

Smoothing can be done to blur images. Generally, when we use Low pass Filters for blurring or smoothing the sample image. We use High pass filters for removing the noise from source image.

Image Filtering (2D Convolution)

OpenCV provides a function `cv.filter2D()` to convolve a kernel with an image. Convolution is basically mixing two input sources in right order and time to generate a mixed output data.



A 5x5 averaging kernel (kernel is nothing but a simple matrix, that is used in convolution operation) is just a 2-D array of 1's. It's really simple, what does it do exactly is that a 5x5 matrix will cover 25 pixels at once. The input pixel in question will be at the center, it's new value will

be an average/mean of all the 24 pixels surrounding it. Hence, the name **Averaging/Mean** Filter.

Source Code :-

```
img = cv.imread("samples/opencv-logo-white.png")
kernel = np.ones((3, 3), np.float32)/9
output = cv.filter2D(img, -1, kernel)

cv.imshow("Sample image", img)
cv.imshow("Processed image", output)
```

Code explanation :-

```
kernel = np.ones((3, 3), np.float32)/9
```

To create a 3x3 matrix, it will pick 9 surrounding pixels and take average from it.

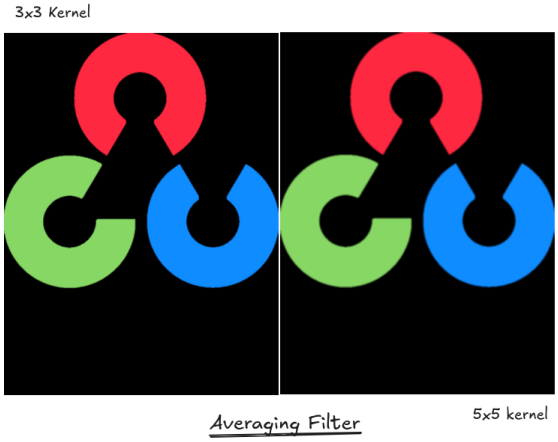
```
output = cv.filter2D(img, -1, kernel)
```

To process the final data, it will apply the kernel on the source, generated in the last step. -1(ddepth) shows we want to maintain the depth value same as we have in source image.

Different blurring techniques available in OpenCV

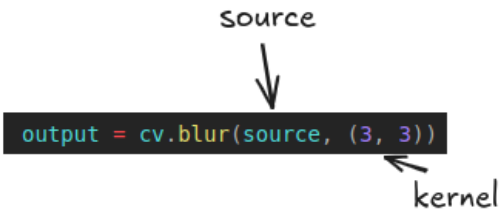
Same can be done using above mentioned `.filter2D()` method. However the following techniques generates and process kernel in one command itself.

1. Averaging



Method => `cv.blur()`

- First parameter : Image source
- Second parameter : dimension of 2-D kernel (tuples data type)



This will take a 2-D 1's matrix and take all the pixels surrounding the pixel in question, and set it's mean value to that pixel. Same as done above, though in this method we don't need to generate the kernel manually. The method will take care of it itself.

Source Code :-

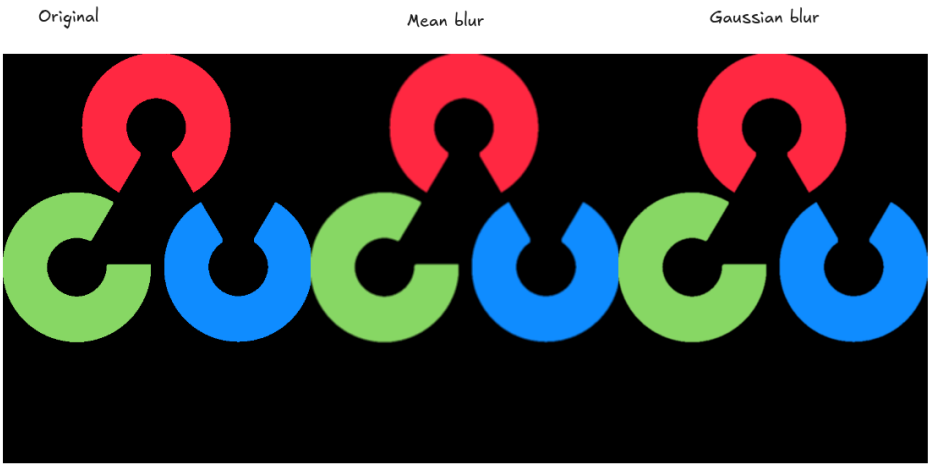
```
source = cv.imread("samples/opencv-logo.png")

output = cv.blur(source, (3, 3))
output2 = cv.blur(source, (5, 5))

cv.imshow("3x3 matrix", output)
cv.imshow("5x5 matrix", output2)
```

2. Gaussian Blur

Gaussian blur is when we take the weighted-sum of the surrounding pixels and assign it to pixel's intensity. Gaussian blur differs from simple averaging blur is due to the fact that it takes care of weight aka the distance from the pixel.



Method => `cv.GaussianBlur()`

- First parameter : source
- Second parameter : Size of 2-D kernel (tuple) (should be odd)
- Third parameter : sigmaX (weight variation along x-axis)
- Fourth parameter : sigmaY (weight variation along y-axis)
- If fourth parameter is not specified, same value of 3rd is used.

Source Code :-

```
source = cv.imread("samples/opencv-logo.png")

average_filter = cv.blur(source, (5, 5))
output = cv.GaussianBlur(source, (5, 5), 0)

cv.imshow("original image", source)
cv.imshow("Average filter", average_filter)
cv.imshow("Gaussian Blur", output)
```

3. Median Blurring

Denoising with Median BLur

Median blurring takes all the value of pixels under the kernel and take the median value. The concerned pixel is given this median value.
Median blurring is highly effective while dealing with *high noise images*.

Method => `cv.medianBlur()`

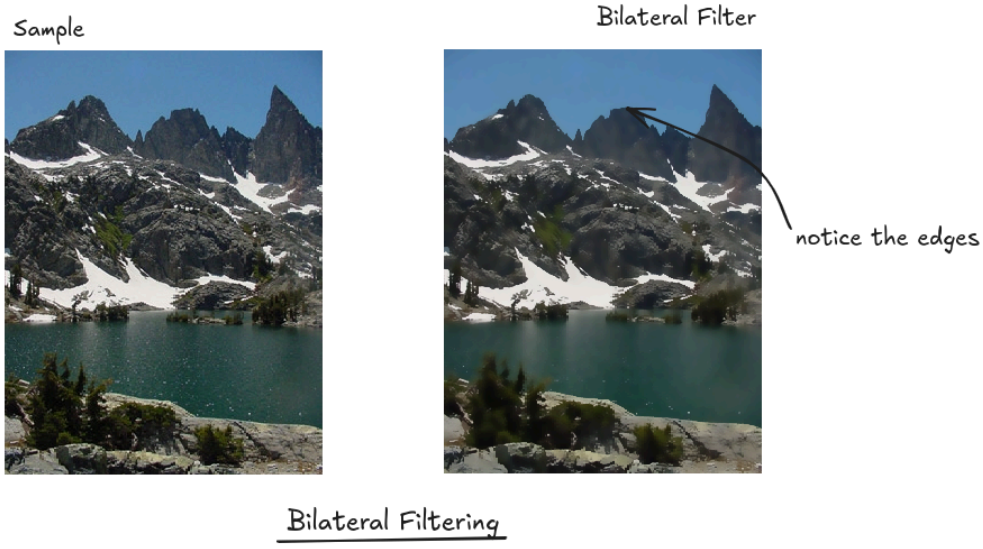
- First parameter : source
- Second parameter : kernel size (int odd)

Source Code :-

```
source = cv.imread("samples/noise.jpg")
filtered = cv.medianBlur(source, 7)

cv.imshow("Original image", source)
cv.imshow("Processed image", filtered)
```

4. Bilateral Filtering



Bilateral filtering is ideal for denoising images while also maintaining the *sharpness of edges*. It achieves so by using two different gaussian functions, one for weighted sum of nearby pixels, second for maintaining the pixel difference. Which will make the edges stay intact.

Method => `cv.bilateralFilter()`

- First parameter : Source
- Second parameter : diameter
- Third paramter : sigmaColor (how much color should be mixed)
- Fourth parameter : sigmaSpace (how much influence do farther pixels hold)

Source Code :-

```
source = cv.imread("samples/mountains.png")
output = cv.bilateralFilter(source, 9, 75, 75)
```

```
cv.imshow("original image", source)
cv.imshow("Processed image", output)
```