Convolution with OpenCV & Python

# Convolution:

*Convolution is simply an element-wise multiplication of 2 matrices followed by a sum.*

* Blurring, Smoothing, Edge-detection, Sharpening => Convolutions

Image -> Big matix

Kernel -> Small matrix

Essentially, this *tiny* kernel sits on top of the *big* image and slides from left-to-right and top-to-bottom, applying a mathematical operation (i.e., a *convolution*) at each *(x, y)*-coordinate of the original image.

It’s normal to hand-define kernels to obtain various image processing functions. In fact, you might already be familiar with blurring (average smoothing, Gaussian smoothing, median smoothing, etc.), edge detection (Laplacian, Sobel, Scharr, Prewitt, etc.), and sharpening — *all* of these operations are forms of hand-defined kernels that are *specifically designed*to perform a particular function.

* we are sliding the kernel from left-to-right and top-to-bottom along the original image.
* At each *(x, y)*-coordinate of the original image, we stop and examine the neighborhood of pixels located at the ***center***of the image kernel. We then take this neighborhood of pixels, *convolve* them with the kernel, and obtain a single output value. This output value is then stored in the output image at the same *(x, y)*-coordinates as the center of the kernel.

Convolution itself is actually very easy. All we need to do is:

1. Select an *(x, y)*-coordinate from the original image.
2. Place the **center** of the kernel at this *(x, y)*-coordinate.
3. Take the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. The sum of these multiplications is called the **kernel output**.
4. Use the same *(x, y)*-coordinates from **Step #1**, but this time, store the kernel output in the same *(x, y)*-location as the output image.

***Below you can find an example of convolving (denoted mathematically as the “\*” operator) a 3 x 3 region of an image with a 3 x 3 kernel used for blurring:***

Figure 4: Convolving a 3 x 3 input image region with a 3 x 3 kernel used for blurring. Source: PyImageSearch Gurus

**Figure 4:** Convolving a *3 x 3* input image region with a*3 x 3* kernel used for blurring. [Source: PyImageSearch Gurus](https://www.pyimagesearch.com/pyimagesearch-gurus/?src=post-convolutions)

Therefore,

Figure 5: The output of the convolution operation is stored in the output image. Source: PyImageSearch Gurus

**Figure 5:** The output of the convolution operation is stored in the output image. [Source: PyImageSearch Gurus](https://www.pyimagesearch.com/pyimagesearch-gurus/?src=post-convolutions)

# Quick review

## Read Image from file using OpenCV in Python

img = cv2.imread('colorImage.jpg',0)

* cv2.IMREAD\_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.
* cv2.IMREAD\_GRAYSCALE : Loads image in grayscale mode
* cv2.IMREAD\_UNCHANGED : Loads image as such including alpha channel

Instead of these three flags, you can simply pass integers 1, 0 or -1 respectively.

## Create a kernel, grab the shape of image and kernel

## Pad the borders of Image

## Cv2.copyMakeBorder

## Create memory for output

## Loop over Image and kernel to get ROI and Convolution

## Rescale intensity

## Display an Image:

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Write an Image:

cv2.imwrite(convolutedImage.png',img)