

```
In [5]: 1 from IPython.display import Image
        2 import numpy as np, cv2
```

```
In [3]: 1 Image('tutorial images/convolution and blurring.png')
```

Out[3]:

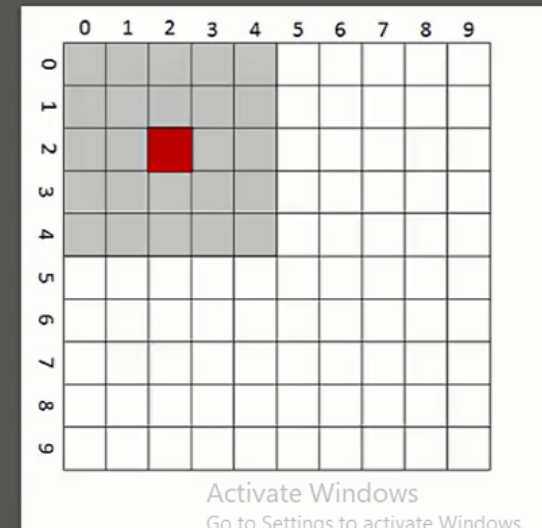


# Convolutions & Blurring

A **Convolution** is a mathematical operation performed on two functions producing a third function which is typically a modified version of one of the original functions.

$$\text{Output Image} = \text{Image} \otimes \text{Function}_{\text{Kernel Size}}$$

In Computer Vision we use **kernel's** to specify the size over which we run our manipulating function over our image.



5 x 5 Kernel over our image

**convolution operates one pixel at a time**

```
In [4]: 1 Image('tutorial images/Blurring.png')
```

```
Out[4]:
```



## Blurring

Blurring is an operation where we average the pixels within a region (kernel).

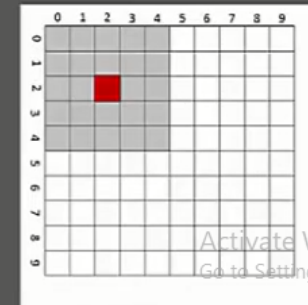
$$Kernel\_ = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



The above is a 5 x 5 kernel.

We multiply by 1/25 to normalize i.e. sum to 1, otherwise we'd be increasing intensity.

```
cv2.filter2D(image, -1, kernel)
```



5 x 5 Kernel over our image

## Convolution and Blurring

```
In [1]: 1 try:
2         import numpy as np, cv2
3
4         image = cv2.imread('my.JPG')
5         cv2.imshow('Original Image', image)
6         cv2.waitKey()
7
8         #Creating our 3 x 3 kernel
9         kernel_3x3 = np.ones((3, 3), np.float32) / 9
10
11        #We use the cv2.filter2D to convolve the kernel with an image
12        blurred = cv2.filter2D(image, -1, kernel_3x3)
13
14        cv2.imshow('3x3 Kernel Blurred', blurred)
15
16        cv2.waitKey()
17
18        #creating our 7 x 7 kernel
19        kernel_7x7 = np.ones((7, 7), np.float32) / 49
20
21        blurred2 = cv2.filter2D(image, -1, kernel_7x7)
22        cv2.imshow('7 x 7 kernel Blurring', blurred2)
23        cv2.waitKey()
24        cv2.destroyAllWindows()
25    except Exception as e:
26        print(f'{type(e).__name__}:-- {e}')
```

## Other commonly used blurring methods in OpenCV

```
In [3]: 1 import cv2, numpy as np
2 image = cv2.imread('my.JPG')
3
4 #Averaging done by convolving the image with a normalized box filter.
5 #This takes the pixels under the box and replaces the central element
6 #Box size needs to odd and positive
7
8 blur = cv2.blur(image, (3, 3))
9 cv2.imshow('Averaging', blur)
10 cv2.waitKey()
11
12 #Instead of box filter, gaussian kernel
13 Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
14 cv2.imshow('Gaussian Blurring', Gaussian)
15 cv2.waitKey()
16
17 #Takes median of all pixels under kernel area and central
18 #element is replaced with this median value
19 median = cv2.medianBlur(image, 5)
20 cv2.imshow('Median Blur', median)
21 cv2.waitKey()
22
23 #Bilateral is very effective in noise removal while keeping edges sharp
24 bilateral = cv2.bilateralFilter(image, 9, 75, 75)
25 cv2.imshow('Bilateral Blurring', bilateral)
26 cv2.waitKey()
27 cv2.destroyAllWindows()
```

## Other commonly used blurring methods used in opencv

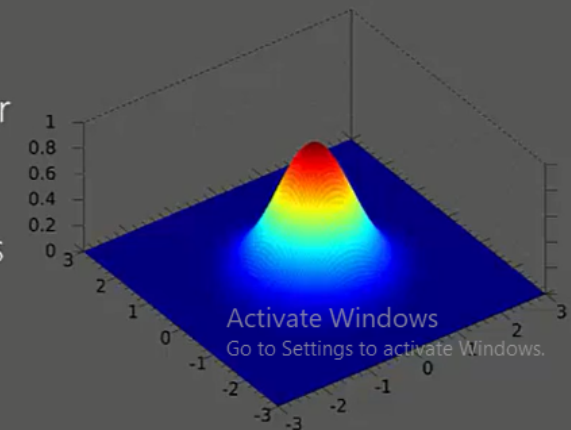
In [6]: 1 Image('tutorial images/other types of blurring.png')

Out[6]:

## Other Types of Blurring

- `cv2.blur` – Averages values over a specified window
- `cv2.GaussianBlur` – Similar, but uses a Gaussian window (more emphasis or weighting on points around the center)
- `cv2.medianBlur` – Uses median of all elements in the window
- `cv2.bilateralFilter` – Blur while keeping edges sharp (slower). It also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The pixel difference function makes sure only those pixels with similar intensity to central pixel is considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

	1	4	7	4	1
	4	16	26	16	4
$\frac{1}{273}$	7	26	41	26	7
	4	16	26	16	4
	1	4	7	4	1



^^^^^^This matrix and graph is for GaussianBlur^^^^^^

this is how gaussianBlur works it mainly focuses at center more

## Image De-nosing - Non-Local Means Denoising

```
▶ In [1]: 1 import numpy as np, cv2
2
3 image = cv2.imread('my.JPG')
4
5 cv2.imshow('Original image', image)
6 cv2.waitKey()
7
8 #parameters, after None are - the filter strength 'h' (5-10 is a good range)
9 #Next is h for ColorComponents, set as same value as h again
10 #
11 dst = cv2.fastNlMeansDenoisingColored(image, None, 6, 6, 7, 21)
12
13 cv2.imshow('Fast Means Denoising', dst)
14
15 cv2.waitKey()
16 cv2.destroyAllWindows()
```