Lab Assignment 02

Name: Saurabh S. Ramteke
Roll No: 21-27-19
M.Tech: Data Science

Objective: To know how to use image filtering techniques

Importing Dependencies

In [1]:

```
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt

%matplotlib inline
```

LPF helps in removing noise, blurring images.
HPF filters help in finding edges in images.

Types of Kernels

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Identity kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpen kernel

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

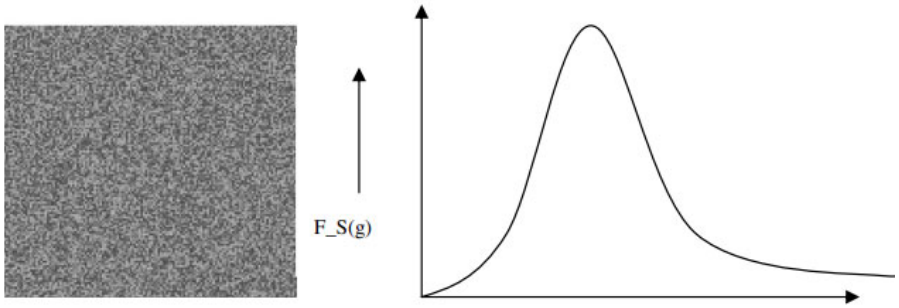
Box blur

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

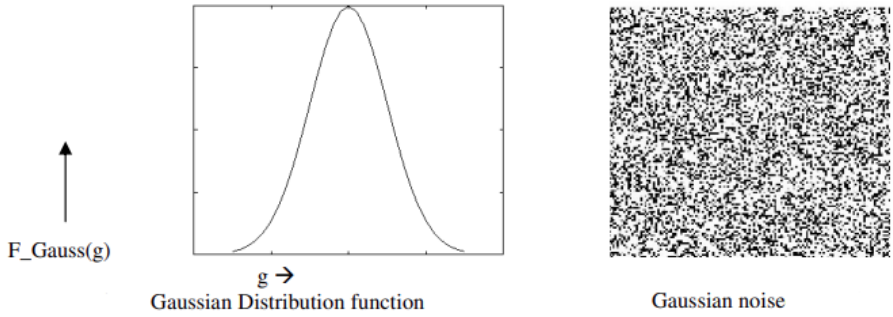
Gaussian blurr kernel

Noise Introduction

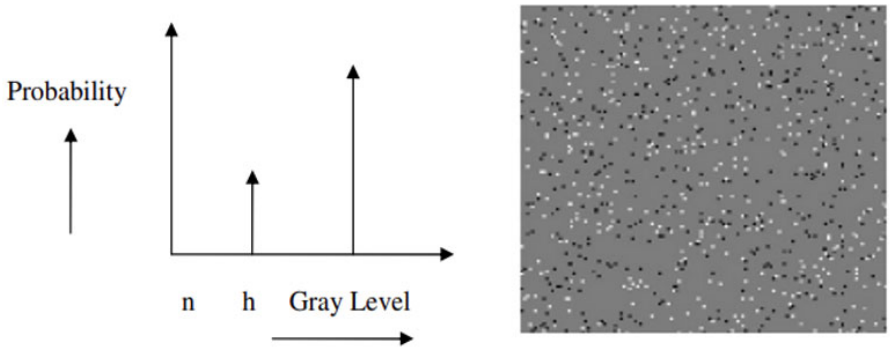
Speckle : a small spot or patch of colour.



Gauss : statistical noise having a probability density function (PDF) equal to that of the normal distribution, which is also known as the Gaussian distribution



Salt & Pepper : This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.



Poison : uncertainty associated with the measurement of light, inherent to the quantized nature of light and the independence of photon detections.

In [2]:

```
def copy_image():
    '''Copy the image, no parameter required'''
    image         = cv2.imread(r"Images//cat.jpg")
    image_RGB     = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image_RGB
```

In [3]:

```
def noisy(noise_typ, image):
    '''Function returns noisy image.
    Parameters
        Noise type = speckle, gauss, salt_pepper
        image = import image you want apply noise '''

    if noise_typ =="speckle":
        row,col,ch = image.shape
        print(row,col,ch)
        gauss = np.random.randn(row,col,ch)
        gauss = gauss.reshape(row,col,ch)
        noisy = image + gauss
        return noisy

    elif noise_typ == "gauss":
        row,col,ch= image.shape
        mean = 0
        var = 5
        sigma = var**0.5
        gauss = np.random.normal(mean,sigma,(row,col,ch))
        gauss = gauss.reshape(row,col,ch)
        noisy = image + gauss
        return noisy

    elif noise_typ == "salt_pepper":
        row,col,ch = image.shape
        noisy = np.copy(image)
        for i in range(row):
            y_cord = random.randint(0, row - 1)
            x_cord = random.randint(0, col - 1)
            noisy[y_cord][x_cord] = 255 # color pixel to white
        for i in range(col):
            y_cord = random.randint(0, row - 1)
            x_cord = random.randint(0, col - 1)
            noisy[y_cord][x_cord] = 0 # color pixel to black
        return noisy
```

Image Blurring (Smoothning)

Image blurring 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).

1. Averaging

It simply takes the average of all the pixels under the kernel area and replaces the central element.

2. Gaussian Blurring

In this method, instead of a box filter, a Gaussian kernel is used. 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.

Sigma controls amount of smoothning

3. Median Blur

It 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 [4]:

```
def smoothning_func(image, noise):
    '''Function will smoothen the noisy image and shows diff. o/p using diff
    Parameters:
        image = original image - to show it in the screen to compare.
        noise = noisy image you want to apply smoothening filter'''
    noise1 = np.copy(noise)
    noise2 = np.copy(noise)
    noise3 = np.copy(noise)
    noise4 = np.copy(noise)

    # 3*3 Kernel
    kernel = np.ones((3,3),np.float32)/9
    Filter_2D = cv2.filter2D(noise1,-1, kernel)

    # Averaging blur
    blur = cv2.blur(noise2, (3,3))

    # GaussianBlur
    Gaussian_Blur = cv2.GaussianBlur(noise3, (3,3),0)

    # medianBlur
    median = cv2.medianBlur(noise4.astype(np.float32), 3)

    f, axis = plt.subplots(2, 3, figsize = (20,10))

    axis[0,0].set_title('Original_Image')
    axis[0,1].set_title('Noise_image')
    axis[0,2].set_title('2D_convolution')
    axis[1,0].set_title('Averageing')
    axis[1,1].set_title('Gaussian')
    axis[1,2].set_title('Median')

    axis[0,0].imshow(image)
    axis[0,1].imshow((noise).astype(np.uint8))
    axis[0,2].imshow((Filter_2D).astype(np.uint8))
    axis[1,0].imshow((blur).astype(np.uint8))
    axis[1,1].imshow((Gaussian_Blur).astype(np.uint8))
    axis[1,2].imshow((median).astype(np.uint8))
```

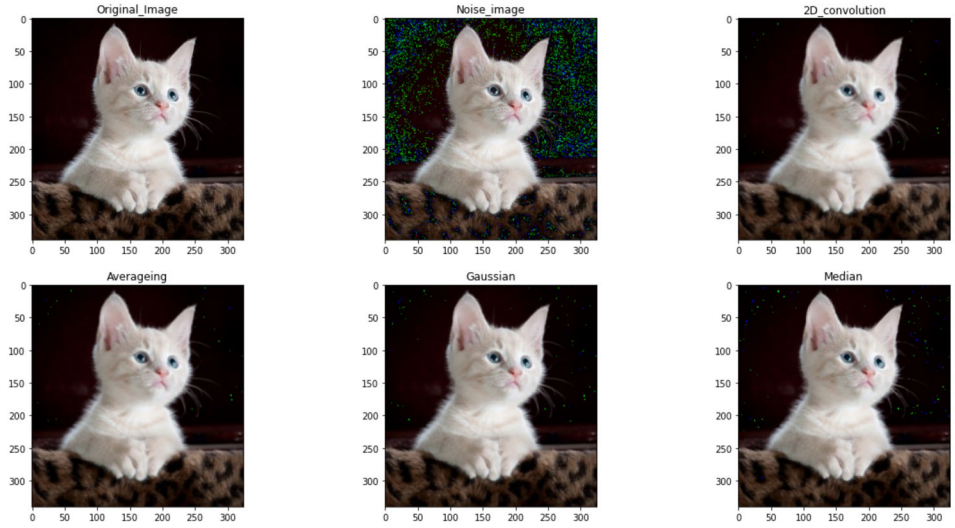
Noise = 'speckle'

In [5]:

```
image1 = copy_image()
noise1 = noisy('speckle', image1)

smoothning_func(image1,noise1)
```

```
340 325 3
```

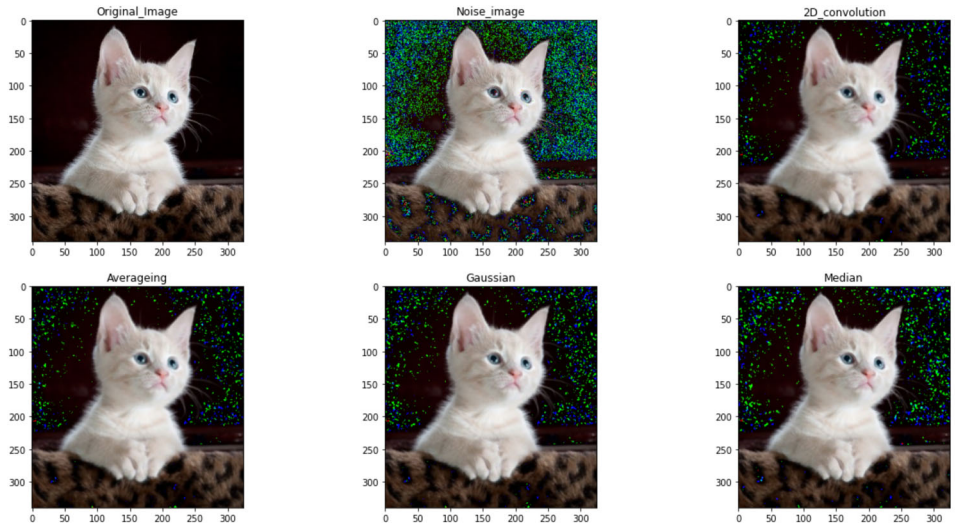


Noise = 'gauss'

In [6]:

```
image2 = copy_image()
noise2 = noisy('gauss', image2)

smoothning_func(image2,noise2)
```



Noise = 'salt_pepper'

In [7]:

```
image3 = copy_image()
noise3 = noisy('salt_pepper', image3)

smoothning_func(image3, noise3)
```