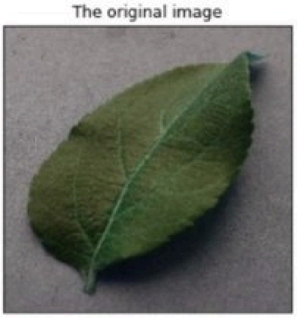
Image Filtering - Spatial Filtering Import required packages for image filtering.

In [ ]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

In [ ]:
```
ref = cv2.imread('/content/02d0257a-d21a-43a4-9348-472043df801e___RS_HL 6065.JPG')
plt.imshow(ref),plt.grid(False)

#while Learing how to perform spatial filtering,
#you can also note how to apply different python commands.
plt.title('The original image')
plt.xticks([])
plt.yticks([])
plt.show()
```

The original image



There are two key factors in applying a filter on an image in digital processing ;

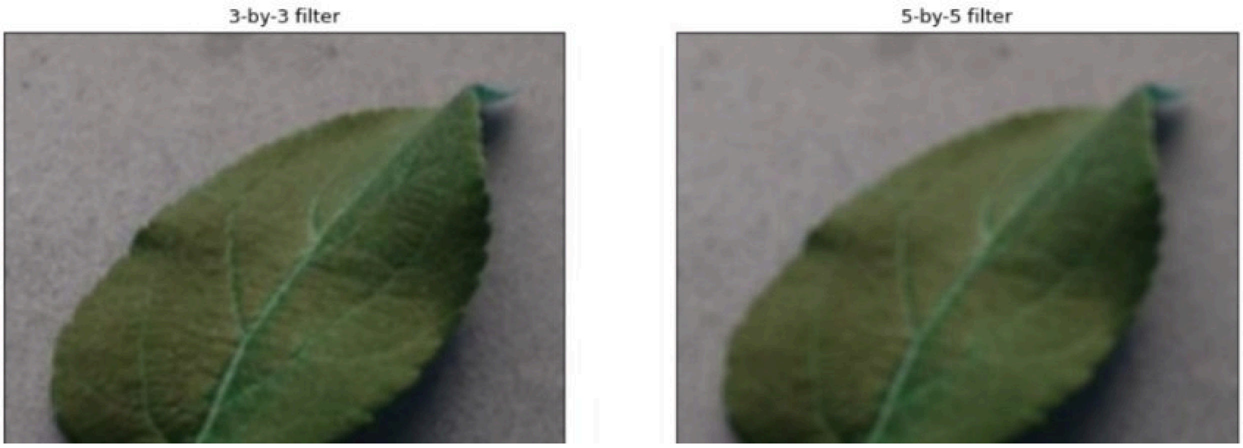
1. the kernal type

2)the padding method

1-Averagingfilter Below ypu see how to define a very simple averaging kernel and apply it on your images

In [ ]:

```
#Defining a kernel using numpy.
kernel_5 = np.ones((5,5),np.float32)/25
kernel_3 = np.ones((3,3),np.float32)/9

#Convolves an image with the kernel.
#-1 means that the center of the kernel is located on the center pixel.
#compare two kernel sizes.
filtered_5 = cv2.filter2D(ref,-1,kernel_5)
filtered_3 = cv2.filter2D(ref,-1,kernel_3)

#plot the results in two subplots.
fig=plt.figure(figsize=(14,14), dpi=80, facecolor='w', edgecolor='k')

plt.subplot(121), plt.imshow(filtered_3), plt.title('3-by-3 filter')
plt.grid(False)
plt.xticks([])
plt.yticks([])

plt.subplot(122), plt.imshow(filtered_5), plt.title('5-by-5 filter')
plt.grid(False)
plt.xticks([])
plt.yticks([])

plt.show()
```



The complete command for performing 2D spatial filter over images in OpenCV is cv2.filter2D with the followig list of parameters.Some of the parameters are not necessarily used.

cv2.filter2D(src,ddepth,kernel[dst[anchor[delta[borderType]]]])

src- input image.

ddepth-desired depth of the destination image;if it is negative,it will be the same as src.depth();the following combinations of src.depth() and ddepth are supported:

src.depth()=CV_8U,ddepth=-1/CV_16S/CV_32F/CV_64F src.depth()=CV_16U/CV_16S,ddepth=-1/CV_32F/CV_64F

kernel-convolution kernel,a single-channel floating point matrix;if you want to apply different kernels to different channels,split the image into separate color planes using split() and process them individually.

anchor- anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should be within the kernel;default value(-1,-1) means that the anchor is at the kernel center.

delta-optional value added to the filtered pixels before storing them in dst.

border type- pixel extrapolation method(see[borderInterpolate()]

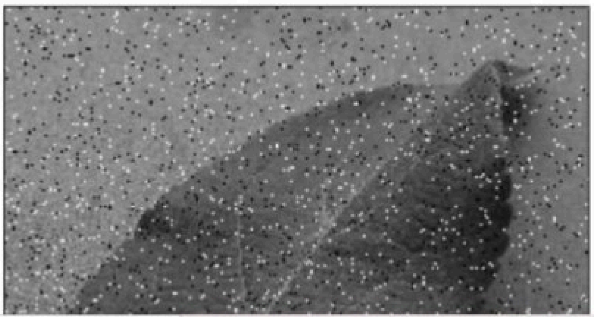
another simple way to apply a simple averaging filter is to use cv2.blur() function.The function can be applied as below:

In [ ]:

```
# you can check the docs for further information.
blurred = cv2.blur(ref,(5,5),-1)

plt.imshow(blurred), plt.grid(False), plt.xticks([]), plt.yticks([]), plt.show()
```

In [ ]:

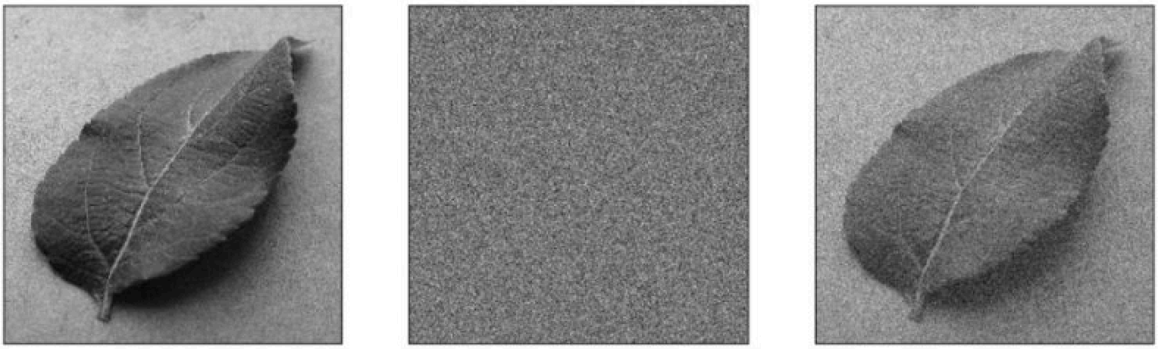
```
s_and_p = np.random.rand(ref.shape[0], ref.shape[1])

# if we consider 5% salt and pepper noise, we'd like to have
# 2.5% salt and 2.5% pepper. thus:
salt = s_and_p > .975
pepper = s_and_p < .025

# in order to add some noise, we should turn off black (pepper) locations and
# turn on white (white) locations.
channel_2 = np.atleast_1d(ref[:, :, 1])
noisy = np.zeros_like(channel_2)

for i in range(channel_2.shape[0]*channel_2.shape[1]):
  if salt.ravel()[i] == 1:
    noisy.ravel()[i] = 255
  elif pepper.ravel()[i] == 1:
    noisy.ravel()[i] = 0
  else:
    noisy.ravel()[i] = channel_2.ravel()[i]

# apply median filter with size 3
Med = cv2.medianBlur(noisy, 3)

# Display the results
fig=plt.figure(figsize=(14, 14), dpi= 80, facecolor='w', edgecolor='k')
plt.subplot(121), plt.xticks([]), plt.yticks([])
plt.imshow(noisy, cmap='gray'), plt.grid(False)
plt.subplot(122), plt.xticks([]), plt.yticks([])
plt.imshow(Med, cmap='gray'), plt.grid(False)
plt.show()
```



In [ ]:

```python
#Creating random normal (gaussian) noise with pre-defined mean and std.
# The noisy image should be the size of the reference image.
mean = 0
sigma = 20.0
gauss_noise = np.random.normal(mean, sigma, (ref.shape[0], ref.shape[1]))

# Convert RGB image to Grayscale image using cvtColor()
gray = cv2.cvtColor(ref, cv2.COLOR_BGR2GRAY)

# Add gaussian noise to the image
g_noisy = gray + gauss_noise # Gaussian noisy image

# Showing gray image, noise image, and noisy image
fig=plt.figure(figsize=(14, 14), dpi= 80, facecolor='w', edgecolor='k')
plt.subplot(131), plt.xticks([]), plt.yticks([])
plt.imshow(gray, cmap='gray'), plt.grid(False)
plt.subplot(132), plt.xticks([]), plt.yticks([])
plt.imshow(gauss_noise, cmap='gray'), plt.grid(False)
plt.subplot(133), plt.xticks([]), plt.yticks([])
plt.imshow(g_noisy, cmap='gray'), plt.grid(False)
```

Out[ ]: (, None)



Now, we use simple cv2.GaussianBlur()to reduce gaussian noise in g_noisy image created above.

cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) sigmaX – Gaussian kernel standard deviation in X direction.

sigmaY – Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height , respectively (see getGaussianKernel() for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX, and sigmaY.

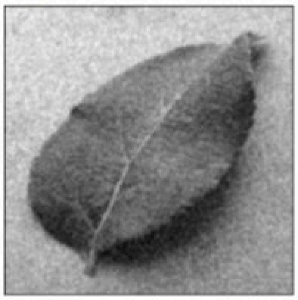
border Type-pixel extrapolation method

In [ ]:

```
g_filtered = cv2.GaussianBlur(g_noisy, (3, 3), 20, 20)

# Display the result
plt.imshow(g_filtered, cmap='gray'), plt.grid(False)
plt.xticks([]), plt.yticks([])
```

Out[ ]:

```
(([], ),
 ([], ))
```



In order to use cv2.sepFilter2D() function, we should create a gaussian kernel. This is done using cv2.getGaussianKernel(). It creates 1-dimensional gaussian coefficients.

cv2.getGaussianKernel(ksize, sigma[, ktype])

ksize – Aperture size. It should be odd and positive.

sigma – Gaussian standard deviation. If it is non-positive, it is computed from ksize as sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8.

ktype – Type of filter coefficients. It can be CV_32f or CV_64F .

In [ ]:

```python
#Create a single gaussian kernel
g_kernel = cv2.getGaussianKernel(3, 20)
print(g_kernel)

# Apply two separate kernels over the image.
g_filtered_2 = cv2.sepFilter2D(g_noisy, -1, g_kernel, g_kernel)

# Displaying the results.
fig=plt.figure(figsize=(14, 14), dpi= 80, facecolor='w', edgecolor='k')
plt.subplot(121), plt.xticks([]), plt.yticks([]), plt.title('first method')
plt.imshow(g_filtered, cmap='gray'), plt.grid(False)
plt.subplot(122), plt.xticks([]), plt.yticks([]), plt.title('second method')
plt.imshow(g_filtered_2, cmap='gray'), plt.grid(False)
```

```
[[0.33319442]
 [0.33361117]
 [0.33319442]]
```

Out[ ]: (, None)