OpenCV Package

opencv Package

- **Open source computer vision package is a good source of image processing and computer vision algorithms.**
- **Several algorithms can be used to manipulate image and extract features that can be used later for machine learning approaches.**
- **Originally developed by Intel.**
- **The library is cross-platform and free for use under the open-source BSD license.**
- **OpenCV is written in C++ and its primary interface is in C++**
- **There are bindings in Python, Java, and MATLAB/OCTAVE**
- **Install opencv: pip install opencv-python**

opencv Package

- **Loading and saving images**

import opencv

Read an image and convert it into a gray level image

Plot the image with gray color map

Don't show axes

Use plt.show() to show the image

```
In [1]: %matplotlib notebook
        import matplotlib.pyplot as plt
        import numpy as np
        import cv2
        img = cv2.imread('noise_lung.png',cv2.IMREAD_GRAYSCALE)
        plt.imshow(img,cmap='gray')
        plt.axis('off')
        plt.show()
```

Figure 1

get image dimensions

```
In [2]: img.shape
Out[2]: (290, 400)
```

Save image with tif format

```
In [4]: cv2.imwrite('noise_lung.tif',img)
Out[4]: True
```

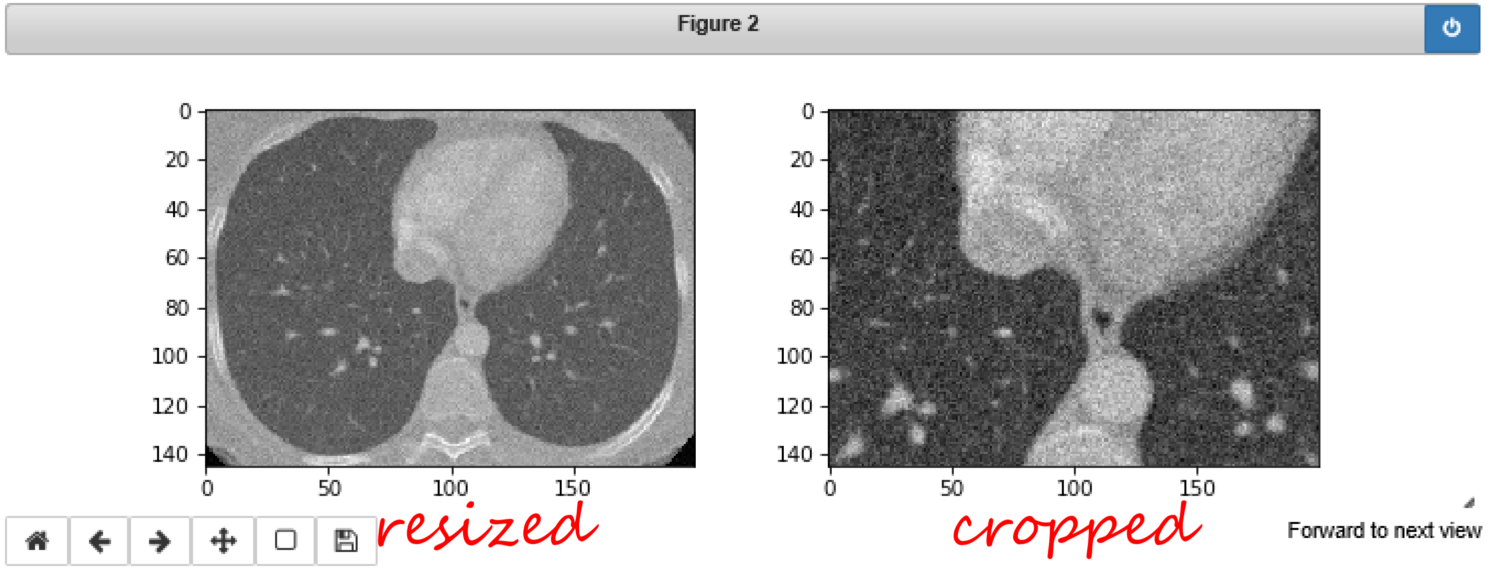
opencv Package

- **Resizing and cropping images**

Resize an image: provide the new dimensions as (width, height). You may specify the interpolation type as cubic interpolation

Use slicing to determine the region of the image you want to crop

In [13]:
```
plt.figure(2)
plt.subplot(1,2,1)
im_2 = cv2.resize(img,(int(img.shape[1]*0.5),int(img.shape[0]*0.5)), interpolation=cv2.INTER_CUBIC)
plt.imshow(im_2,cmap='gray')
im_3 = img[73:218,100:300]
plt.subplot(1,2,2)
plt.imshow(im_3,cmap='gray')
plt.show()
```



resized *cropped*

Dimensions of the resized image (height, width)

In [9]:
```
im_2.shape
```
Out[9]: (145, 200)

Dimensions of the cropped image

In [12]:
```
im_3.shape
```
Out[12]: (145, 200)

What is image filtering?

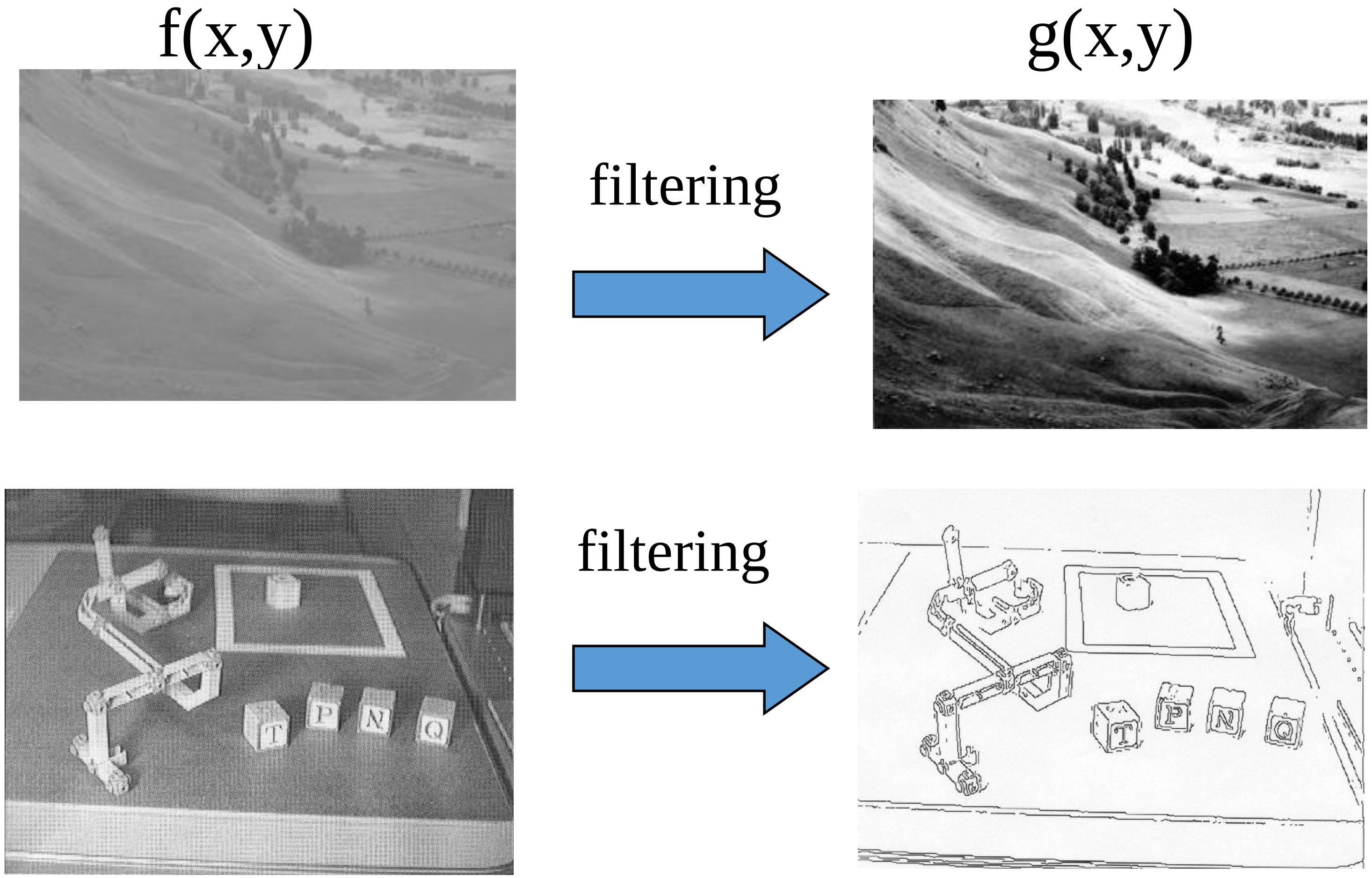
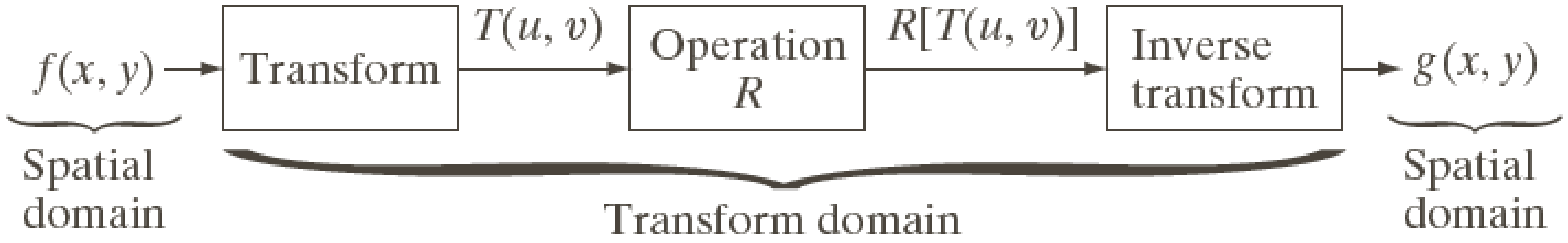


Image Filtering Methods

- Spatial Domain

- Frequency Domain (i.e., uses Fourier Transform)

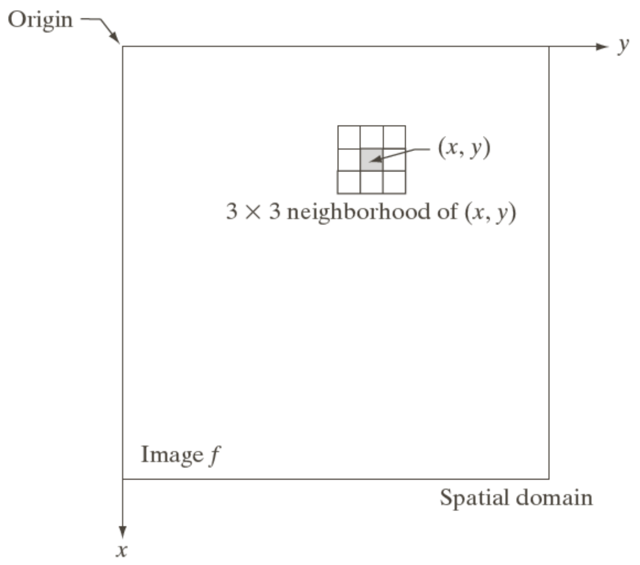


Area Shape and Size of the mask/kernel/filter

- Area shape is typically defined using a rectangular mask.

- Area size is determined by mask size.
  e.g., 3x3 or 5x5

- Mask size is an important parameter!



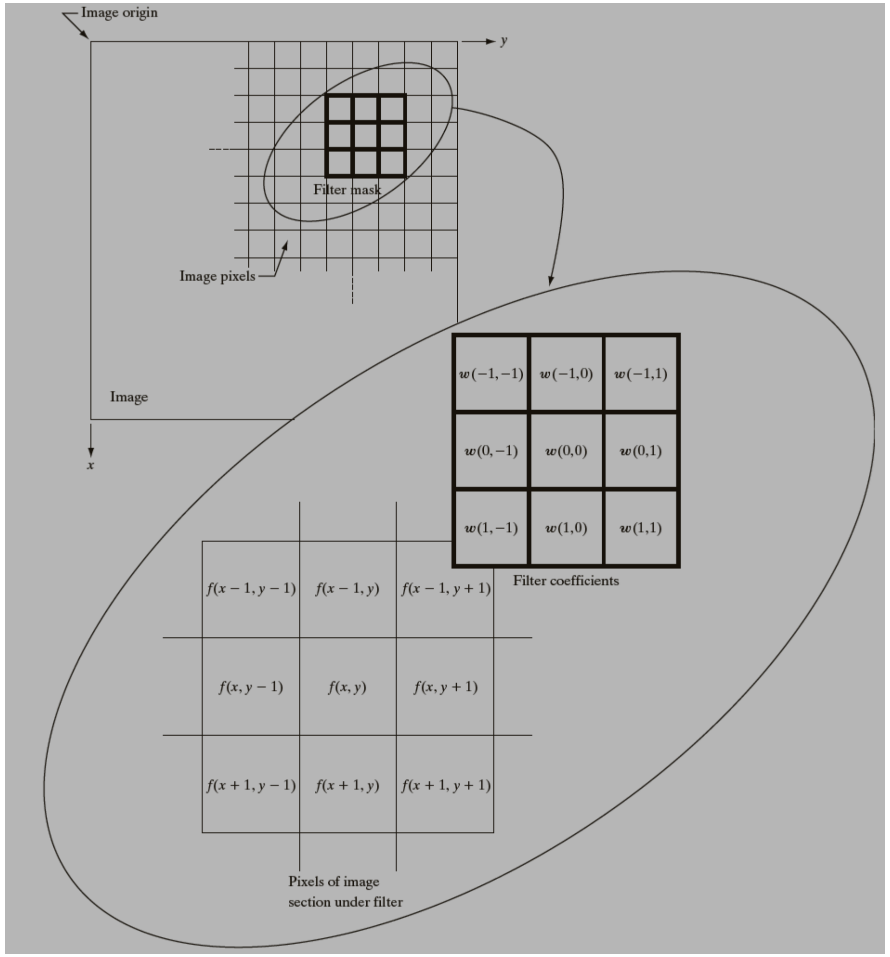
Operation

- Typically, linear combinations of pixel values.
  - e.g., weight pixel values and add them together.

- Different results can be obtained using different weights.
  - (e.g., smoothing, sharpening, edge detection).

mask

w1	w2	w3
w4	w5	w6
w7	w8	w9



Example: Convolution Operation

1	-1	-1
-1	1	-1
-1	-1	1

Filter

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

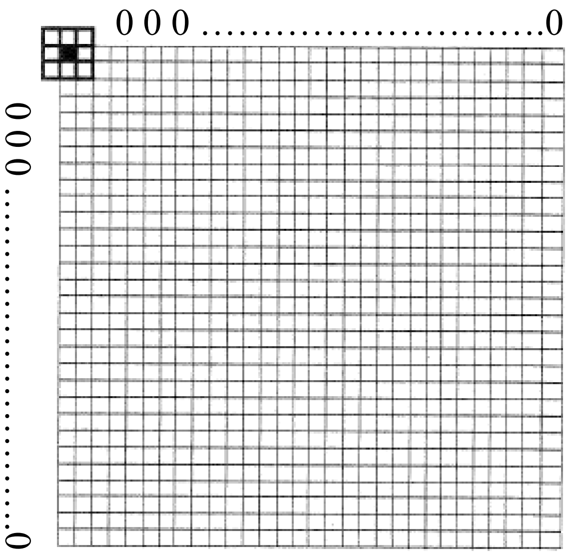
6 x 6 image

Dot product →

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

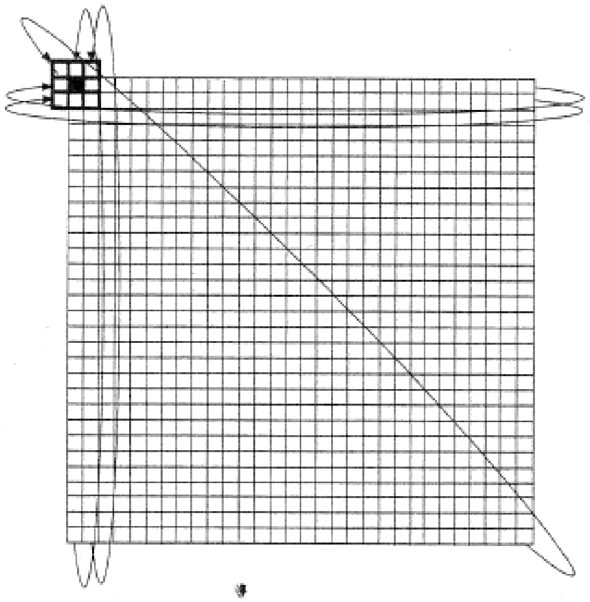
Handling Pixels Close to Boundaries

pad with zeroes



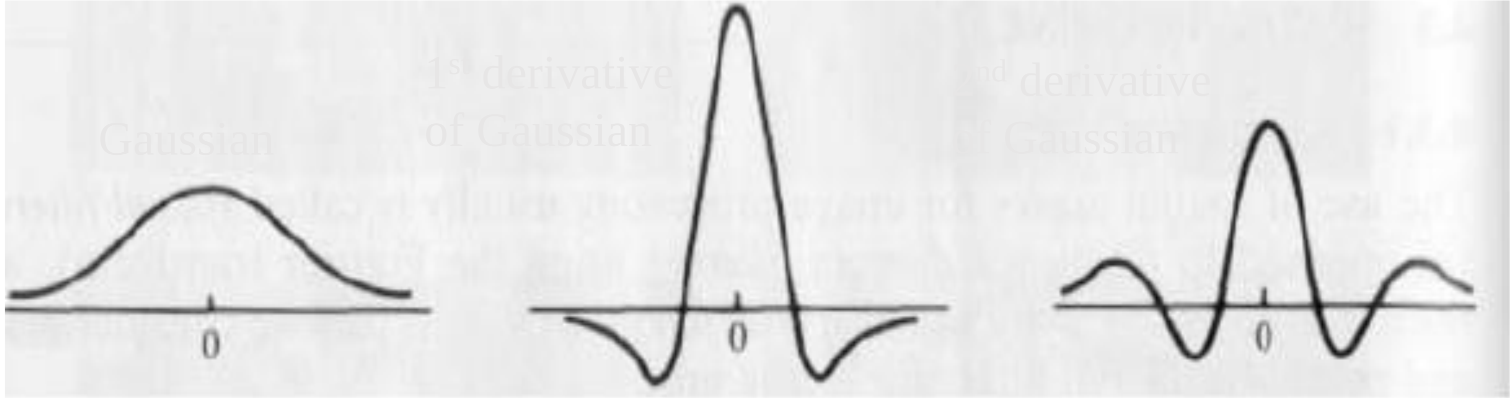
or

wrap around



How do we choose the mask weights?

- Depends on the application.
- Usually by sampling certain functions and their derivatives.

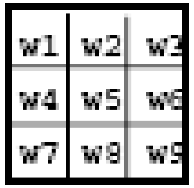


Good for
image smoothing

Good for
image sharpening

Normalization of Mask Weights

- Sum of weights affects overall intensity of output image.
- Positive weights
  - Normalize them such that they sum to **one**.
- Both positive and negative weights
  - Should sum to **zero** (but not always)



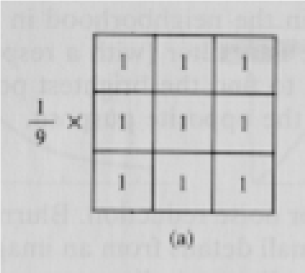
$$\frac{1}{9}\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \qquad \frac{1}{16}\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

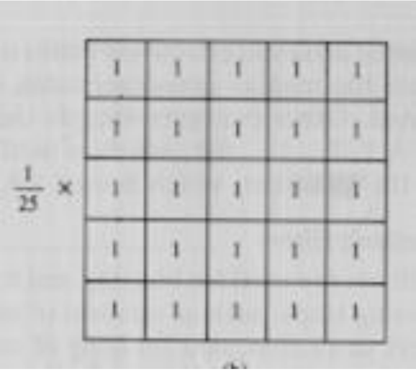
Smoothing Using Averaging

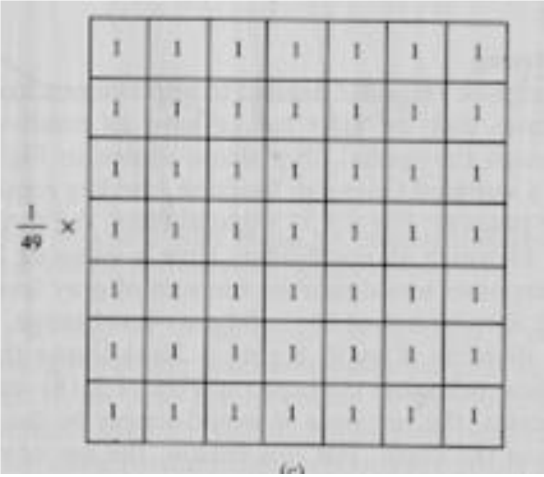
- **Idea:** replace each pixel by the average of its neighbors.

- Useful for reducing noise and unimportant details.

- The size of the mask controls the amount of smoothing.



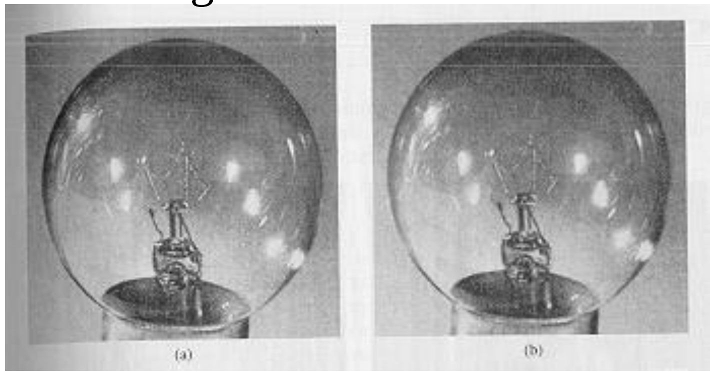




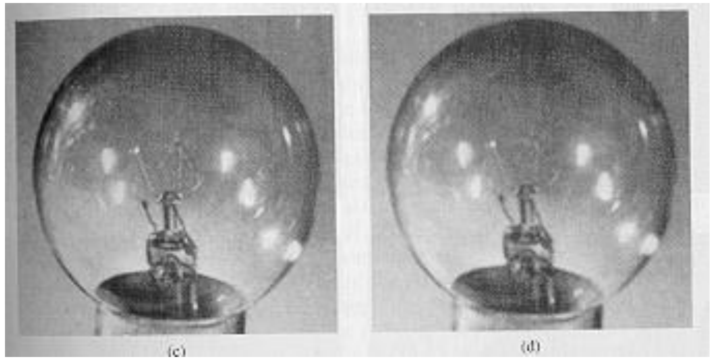
Smoothing Using Averaging (cont’d)

- **Trade-off:** noise vs blurring and loss of detail.

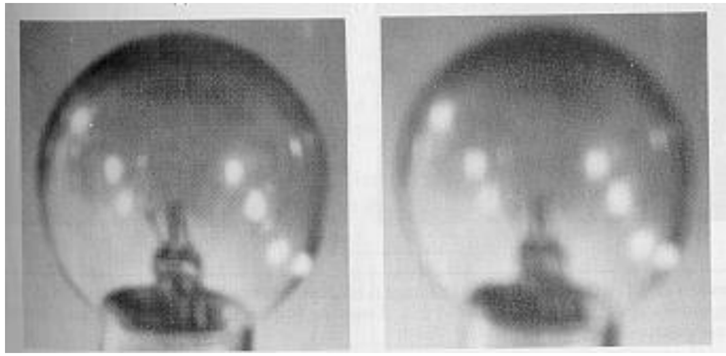
original 3x3



5x5 7x7



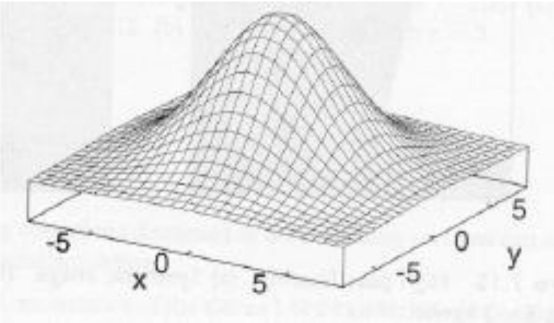
15x15 25x25



Gaussian Smoothing

- **Idea:** replace each pixel by a weighted average of its neighbors
- Mask weights are computed by sampling a Gaussian function

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}}$$

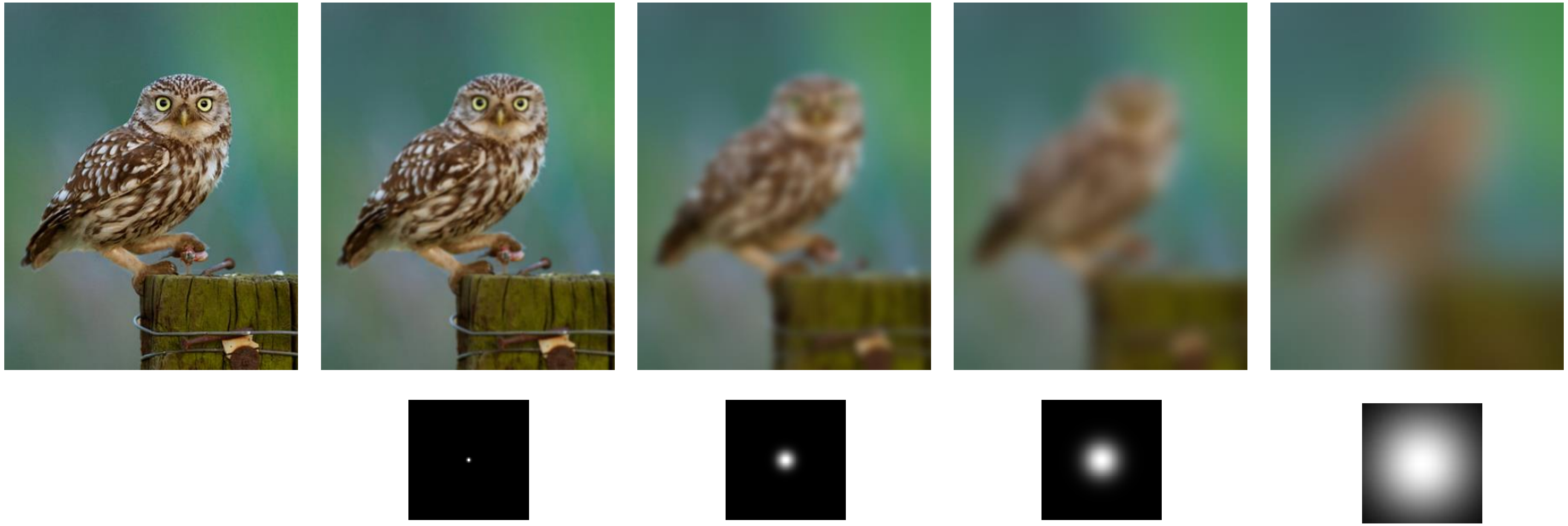


7×7 Gaussian mask

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

Note: weight values decrease with distance from mask center!

Gaussian Smoothing - Example



σ = 1 pixel σ = 5 pixels σ = 10 pixels σ = 30 pixels

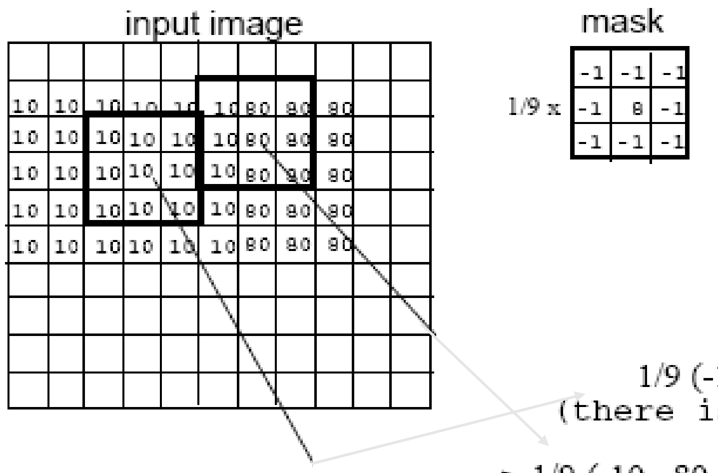
Averaging vs Gaussian Smoothing

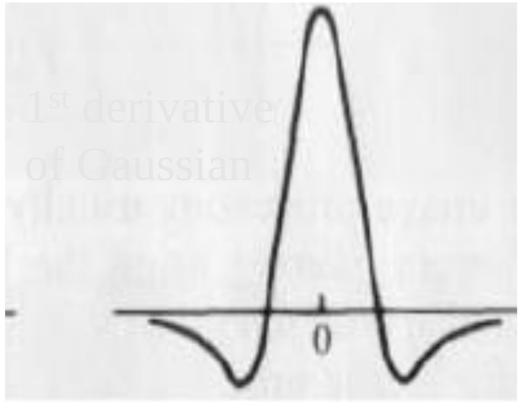
Averaging

Gaussian

Image Sharpening

- <u>Idea:</u> compute intensity differences in local image regions.
- Useful for emphasizing transitions in intensity (e.g., in edge detection).

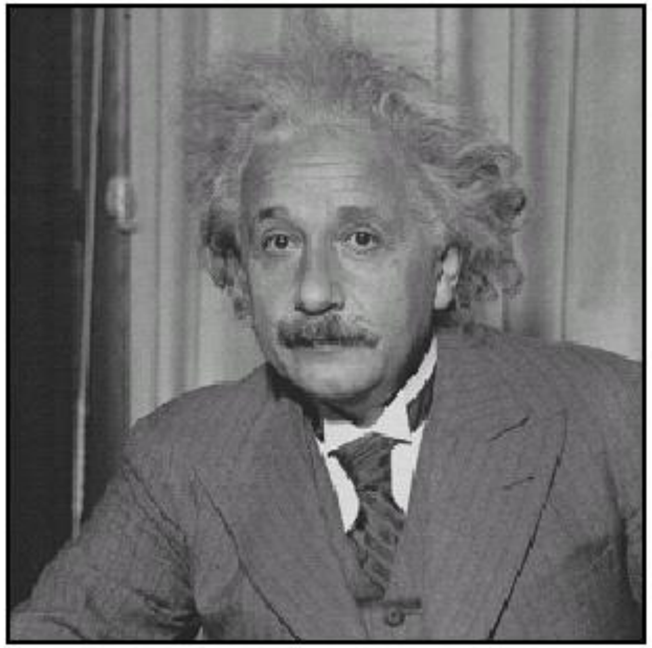




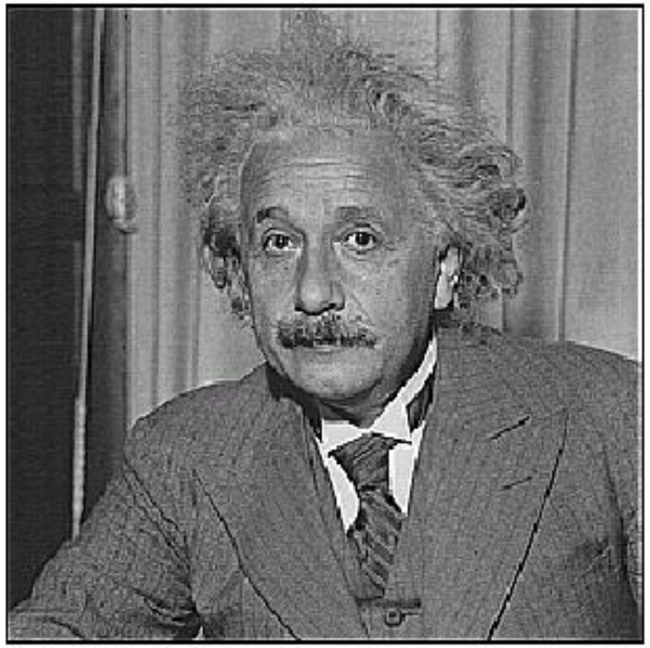
1/9 (-10 - 10 - 10 -10 + 80 -10 -10 -10 -10) = 0
(there is no variation in the gray-levels)

1/9 (-10 - 80 - 80 -10 + 640 -80 -10 -80 -80) = 210/9 > 0
(there is variation in the gray-levels)

Example



before



after

opencv Package

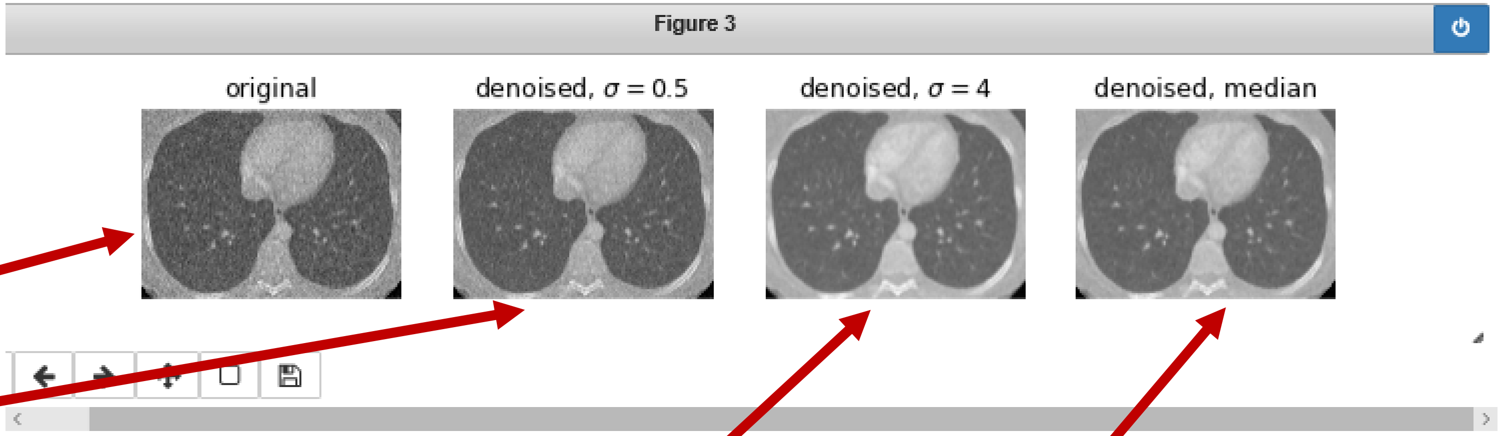
- **Image filtering: Denoising/Smoothing**

Use 5x5 Gaussian blurring/smoothing with σ = 0.5

Use 5x5 Gaussian blurring/smoothing with σ = 4

Use 5x5 median filter

```
In [11]: im_denoise_1 = cv2.GaussianBlur(img,(5, 5), 0.5)
im_denoise_2 = cv2.GaussianBlur(img,(5, 5), 4)
im_denoise_3 = cv2.medianBlur(img, 5)
plt.figure(3)
plt.subplot(1,4,1)
plt.imshow(img,cmap='gray')
plt.title('original')
plt.axis('off')
plt.subplot(1,4,2)
plt.imshow(im_denoise_1,cmap='gray')
plt.title('denoised, '+'$\sigma = 0.5$')
plt.axis('off')
plt.subplot(1,4,3)
plt.imshow(im_denoise_2,cmap='gray')
plt.title('denoised, '+'$\sigma = 4$')
plt.axis('off')
plt.subplot(1,4,4)
plt.imshow(im_denoise_3,cmap='gray')
plt.title('denoised, median')
plt.axis('off')
plt.show()
```



Original

Gaussian Smoothing

Smoothing more as increasing σ

Median filtering

opencv Package

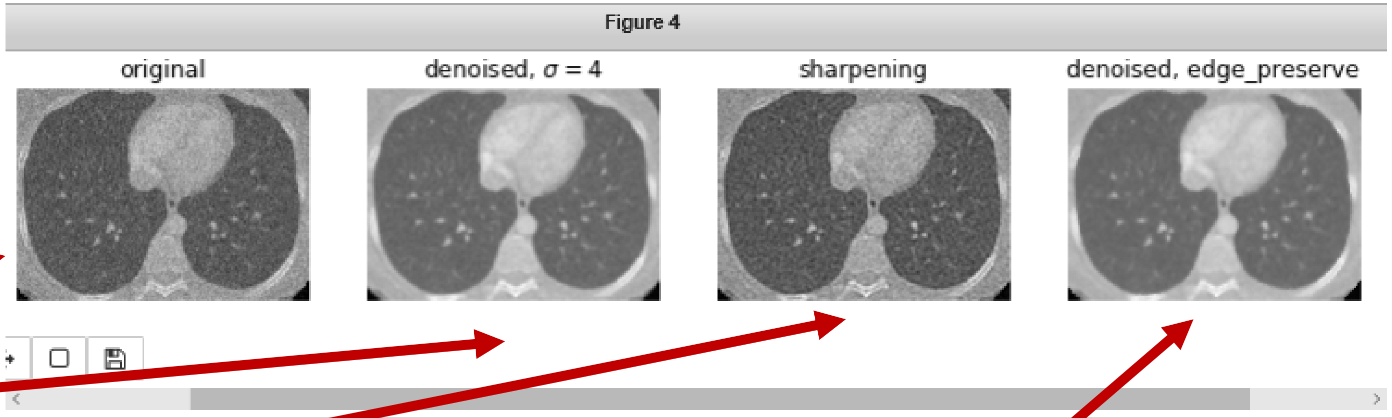
- **Image filtering: sharpening, and edge preserve filters**

Define 3x3 sharpening filter

Apply the filter

Apply bilateral filter to denoise and preserve edges from blurring

In [19]:
```
kernel = np.array([[-1,-1,-1],
                   [-1, 9,-1],
                   [-1,-1,-1]])
im_sharp = cv2.filter2D(im_denoise_2,-1, kernel)
plt.figure(4)
plt.subplot(1,4,1)
plt.imshow(img,cmap='gray')
plt.title('original')
plt.axis('off')
plt.subplot(1,4,2)
plt.imshow(im_denoise_2,cmap='gray')
plt.title('denoised, '+'$\sigma = 4$')
plt.axis('off')
plt.subplot(1,4,3)
plt.imshow(im_sharp,cmap='gray')
plt.title('sharpening')
plt.axis('off')
plt.subplot(1,4,4)
im_edge_preserve = cv2.bilateralFilter(img,9,31,31)
plt.imshow(im_edge_preserve,cmap='gray')
plt.title('denoised, edge_preserve')
plt.axis('off')
plt.show()
```



Original

Gaussian Smoothing

Sharpening after smoothing

Smoothing with edge preserve filtering

opencv Package

- **Image binarization using adaptive thresholding**

the output pixel either zero or max_gray

Sliding window size e.g. 301x301

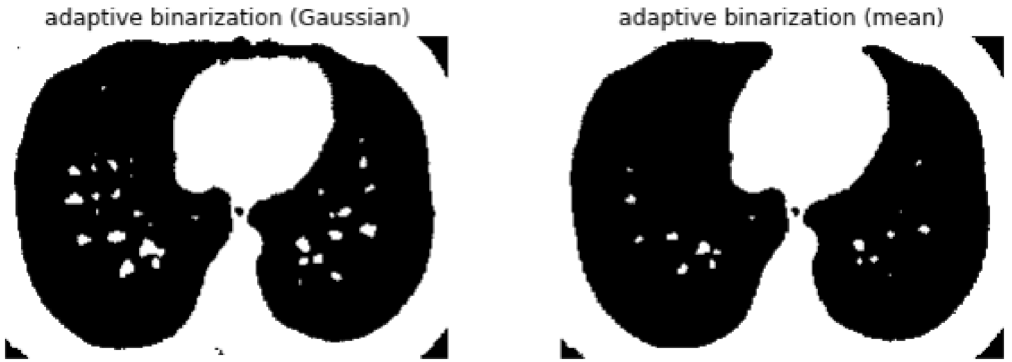
Shift threshold by this value to fine tune the output

Input image

Thresholding approach

```python
max_gray = 255
local_window_size = 301
delta_threshold = 0
im_binary_1 = cv2.adaptiveThreshold(im_edge_preserve, max_gray, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                    cv2.THRESH_BINARY, local_window_size, delta_threshold)
im_binary_2 = cv2.adaptiveThreshold(im_edge_preserve, max_gray, cv2.ADAPTIVE_THRESH_MEAN_C,
                                    cv2.THRESH_BINARY, local_window_size, delta_threshold)
plt.figure(5)
plt.subplot(1,2,1)
plt.imshow(im_binary_1,cmap='gray')
plt.title('adaptive binarization (Gaussian)')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(im_binary_2,cmap='gray')
plt.title('adaptive binarization (mean)')
plt.axis('off')
plt.show()
```

Figure 5



Gaussian

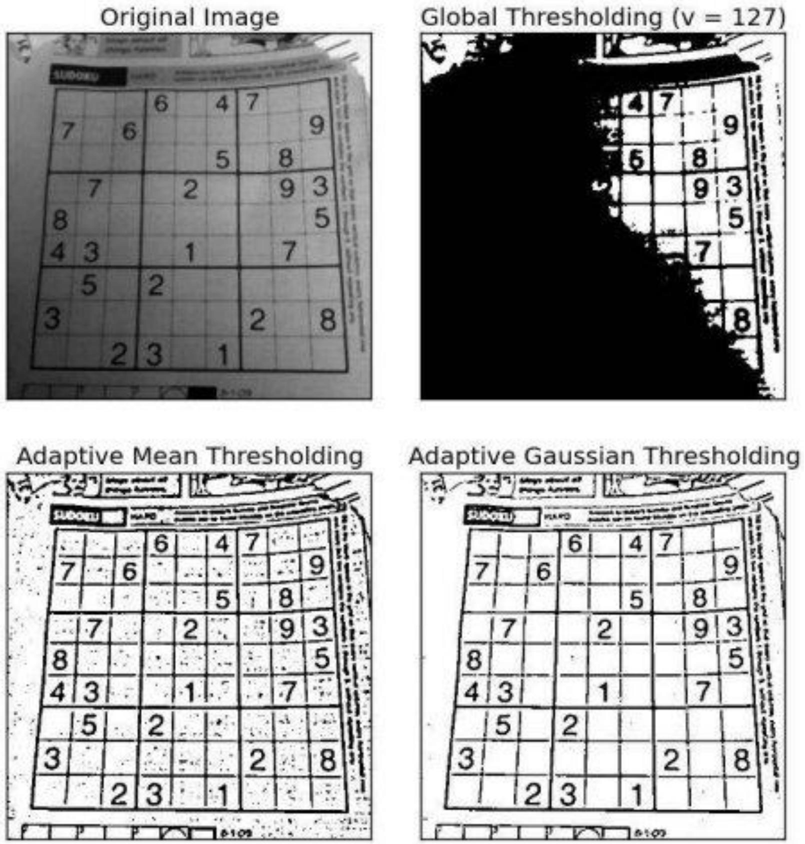
mean

- There are two options: cv2.THRESH_BINARY or cv2.THRESH_BINARY_INV
  You choose a binary output or inverted binary
- The thresholding approach is either Gaussian (weighted average) or mean.

cv2.ADAPTIVE_THRESH_MEAN_C: The threshold value is the mean of the neighbourhood area minus the constant **C**.
cv2.ADAPTIVE_THRESH_GAUSSIAN_C: The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant **C**

opencv Package

- **Image binarization using adaptive thresholding**



opencv Package

- **Edge Detection: There are several algorithms such as:**
  - **Sobel,**
  - **Laplacian,**
  - **Canny**

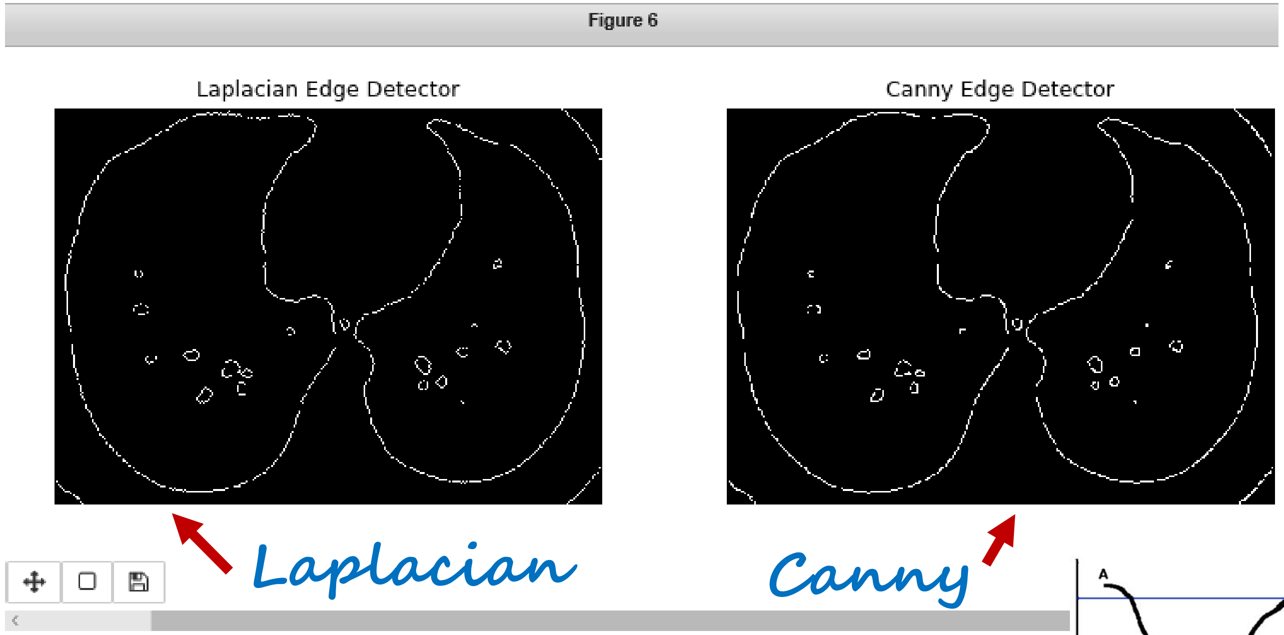
Input image (binary image)

Output image bit depth

Laplacian edge detector

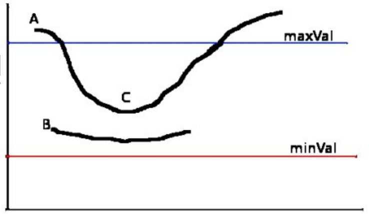
Canny edge detector,
Lower threshold = 20
Upper threshold = 30
Stronger edges are above the upper and weaker edges are greater than a lower threshold and less than the upper one.

In [49]:
```
plt.figure(6)
plt.subplot(1,2,1)
im_edge_1 = cv2.Laplacian(im_binary_2,cv2.CV_8U)
plt.imshow(im_edge_1,cmap='gray')
plt.title('Laplacian Edge Detector')
plt.axis('off')
plt.subplot(1,2,2)
im_edge_2 = cv2.Canny(im_binary_2,20,30)
plt.imshow(im_edge_2,cmap='gray')
plt.title('Canny Edge Detector')
plt.axis('off')
plt.show()
```



Laplacian

Canny



opencv Package

- **Corner Detection: There are several algorithms such as Harris, minimum eigenvalue and FAST**

Input image

Window size to calculate the cornerness

Free parameter k used in calculating the cornerness

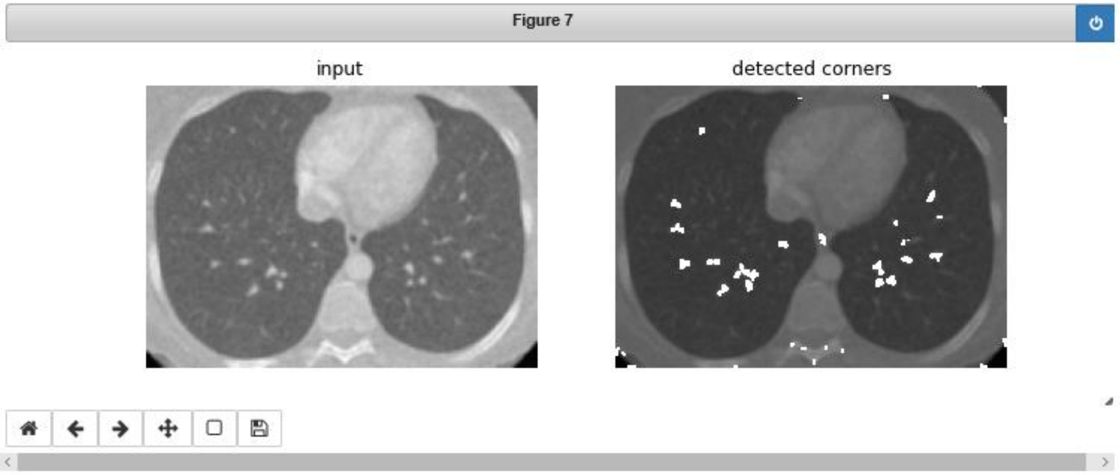
Find the cornerness measure

It is not needed for the algorithm. It is just to replicate corners to look in a good size to visualize

Filter size to calculate the gradient

Threshold to find the strongest corners. Then overlay these corners (white pixels) on top of the original image

```
cornerness = cv2.cornerHarris(np.float32(im_denoise_2), 3, 3, 0.04)
cornerness = cv2.dilate(cornerness, None)
overlay_corners = np.copy(im_denoise_2)
overlay_corners[cornerness > 0.01 * cornerness.max()] = 255
plt.figure(7)
plt.subplot(1,2,1)
plt.imshow(im_denoise_2, cmap='gray')
plt.title('input')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(overlay_corners, cmap='gray')
plt.title('detected corners')
plt.axis('off')
```



More details: https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html

opencv Package

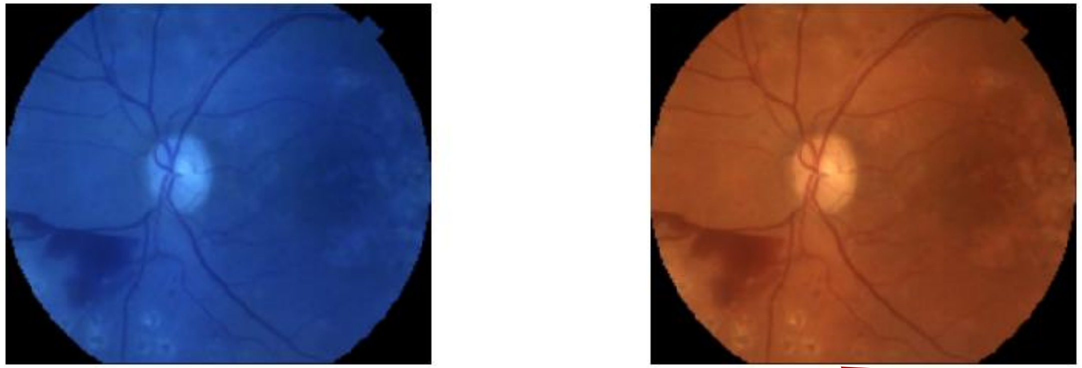
- **Processing Color Images**

Read a color image

Convert image from BGR to the regular RGB. (opencv uses BGR order by default)

In [1]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import cv2
img = cv2.imread('fundus_1.png',cv2.IMREAD_COLOR)
plt.figure(1)
plt.subplot(1,2,1)
plt.axis('off')
plt.imshow(img)
img1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.subplot(1,2,2)
plt.axis('off')
plt.imshow(img1)
plt.show()
```

Figure 1



Matplotlib plots the image as RGB while opencv read it as BGR. that is why the R and B are swapped in the plot

Correct RGB order

Reset original view

opencv Package

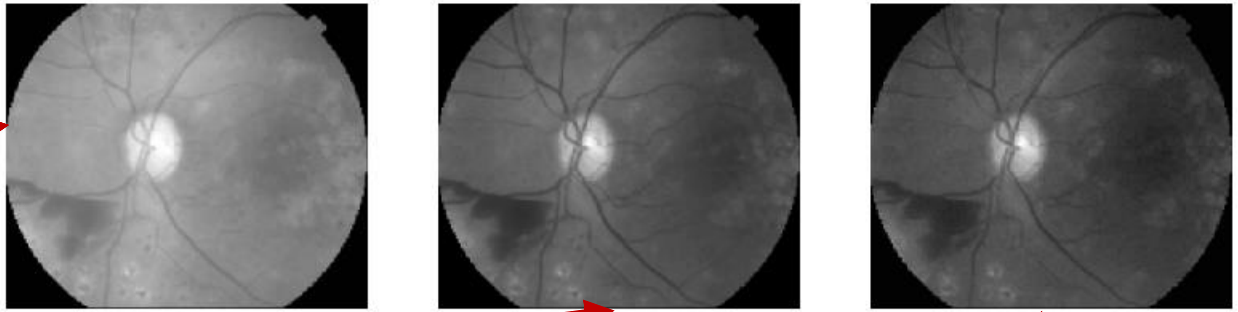
- **Color Image Components**

Color image three components

```
In [2]: r = img1[:,:,0]
g = img1[:,:,1]
b = img1[:,:,2]
plt.figure(2)
plt.subplot(1,3,1), plt.axis('off')
plt.imshow(r,cmap='gray')
plt.subplot(1,3,2), plt.axis('off')
plt.imshow(g,cmap='gray')
plt.subplot(1,3,3), plt.axis('off')
plt.imshow(b,cmap='gray')
plt.show()
```

Plot the three components separately

Figure 2



Red

Green

Download plot

Blue

opencv Package

- **Image Histogram**

Use calcHist() to find the histogram of red with 256 bins

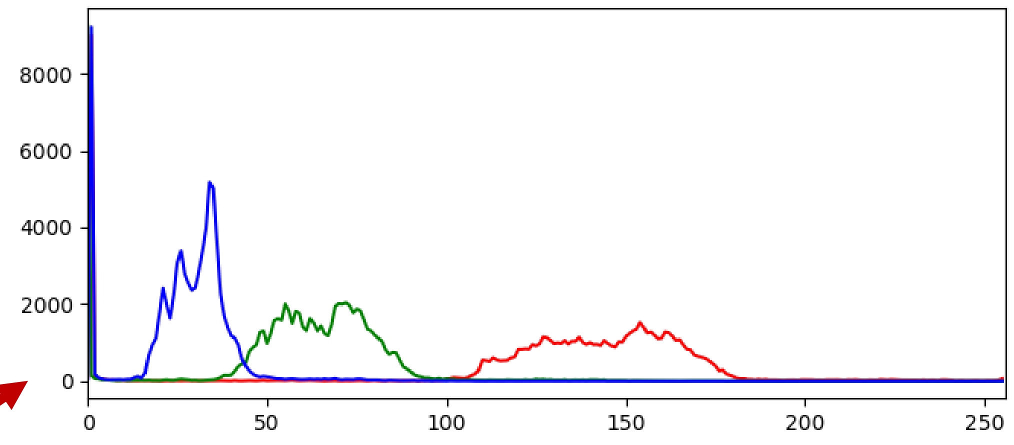
Histogram of Green

Histogram of Blue

Plot of R, G, and B histograms

```
In [3]: plt.figure(3)
hist_r = cv2.calcHist([img1],[0],None,[256],[0,256])
hist_g = cv2.calcHist([img1],[1],None,[256],[0,256])
hist_b = cv2.calcHist([img1],[2],None,[256],[0,256])
plt.plot(hist_r,'r'),plt.plot(hist_g,'g'),plt.plot(hist_b,'b')
plt.xlim([0,256])
plt.show()
```

Figure 3



opencv Package

- **Histogram Equalization**

Convert image to YUV color space, where Y is the intensity and U and V are chroma components.

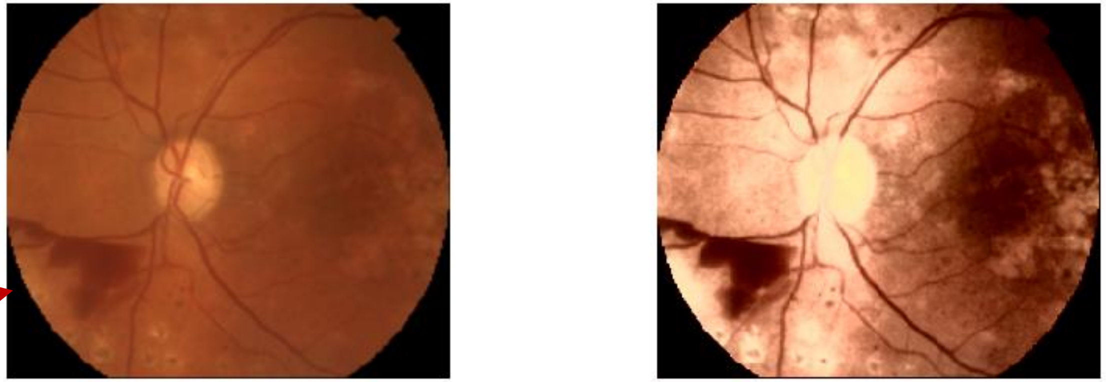
Histogram of Y before equalization

Apply equalization to Y

In [4]:
```
img_yuv = cv2.cvtColor(img1,cv2.COLOR_RGB2YUV)
hist_y_before = cv2.calcHist([img_yuv],[0],None,[256],[0,256])
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
hist_y_after = cv2.calcHist([img_yuv],[0],None,[256],[0,256])
img_rgb = cv2.cvtColor(img_yuv,cv2.COLOR_YUV2RGB)
plt.figure(4)
plt.subplot(1,2,1)
plt.imshow(img1), plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(img_rgb), plt.axis('off')
plt.show()
```

Histogram of Y After equalization

Figure 4



input

equalized

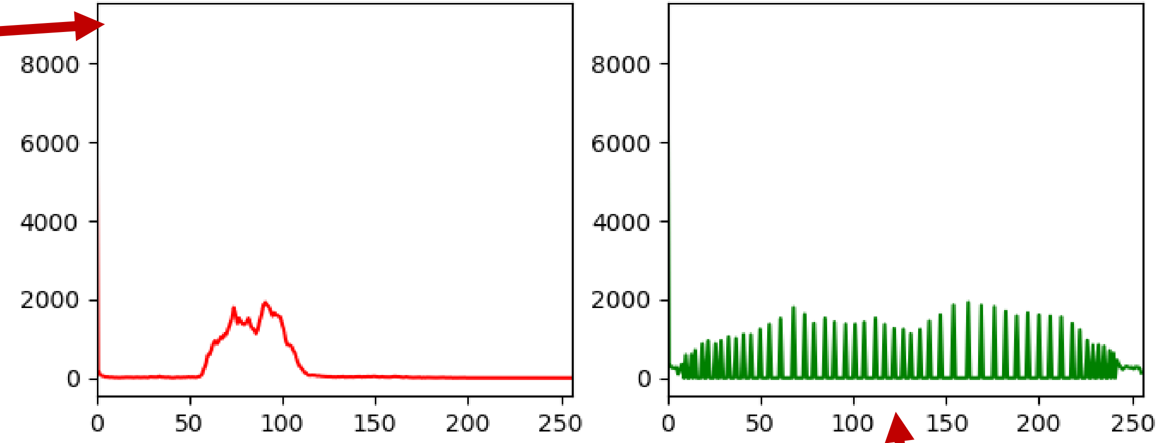
opencv Package

- **Histogram Equalization**

Plot histograms

In [5]:
```
plt.figure(5)
plt.subplot(1,2,1)
plt.plot(hist_y_before,'r'),
plt.xlim([0,256])
plt.subplot(1,2,2)
plt.plot(hist_y_after,'g')
plt.xlim([0,256])
plt.show()
```

Y histogram before equalization (levels are not uniformly distributed)



You need to convert RGB to BGR to write the image in the standard RGB order

In [7]:
```
cv2.imwrite('img_before_eq.png', cv2.cvtColor(img1,cv2.COLOR_RGB2BGR))
cv2.imwrite('img_after_eq.png', cv2.cvtColor(img_rgb,cv2.COLOR_RGB2BGR))
```

Out[7]: True

Y histogram after equalization equalized, more uniform, more contrast