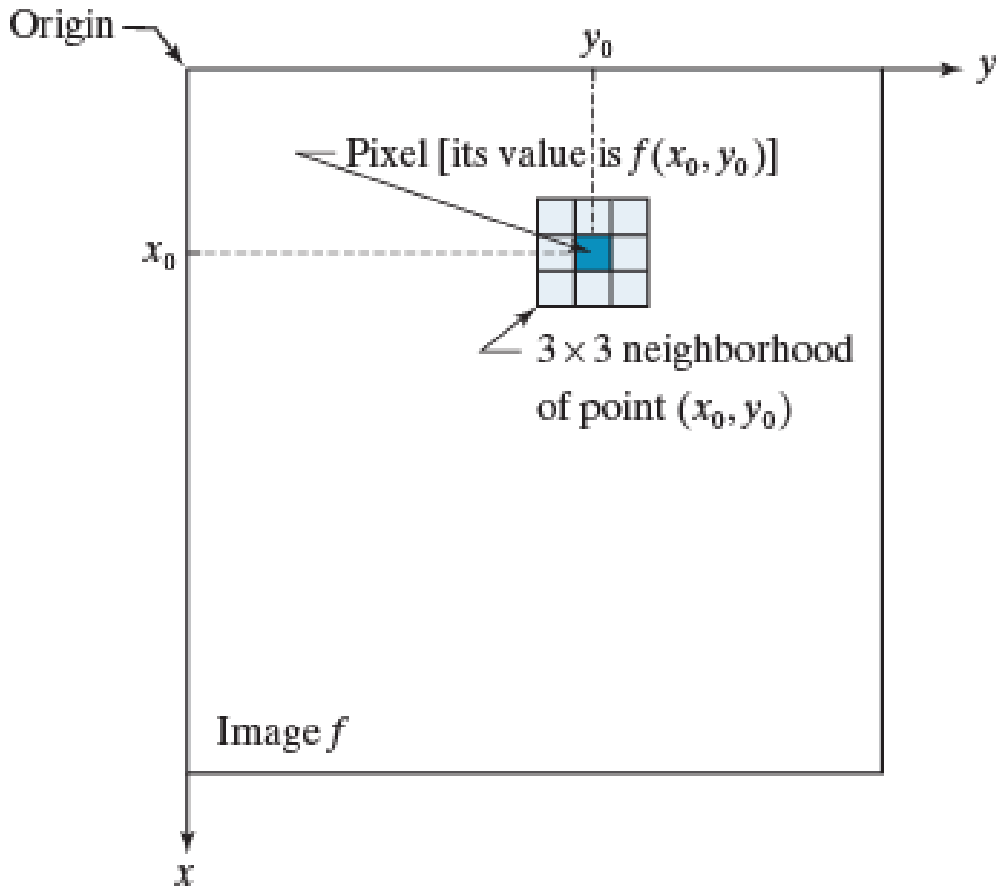
Spatial Filters

The outline

- Filtering
- Smoothing filters
- Order-statistic filters
- Sharpening (Derivative) filters

Neighborhood of a pixel

FIGURE 3.1
A 3×3 neighborhood about a point (x_0, y_0) in an image. The neighborhood is moved from pixel to pixel in the image to generate an output image. Recall from Chapter 2 that the value of a pixel at location (x_0, y_0) is $f(x_0, y_0)$, the value of the image at that location.



The **neighborhood** of a pixel $(x0, y0)$ is a small rectangular region, centered on $(x0, y0)$, and much smaller in size than the image

A Convolution

Filter / kernel

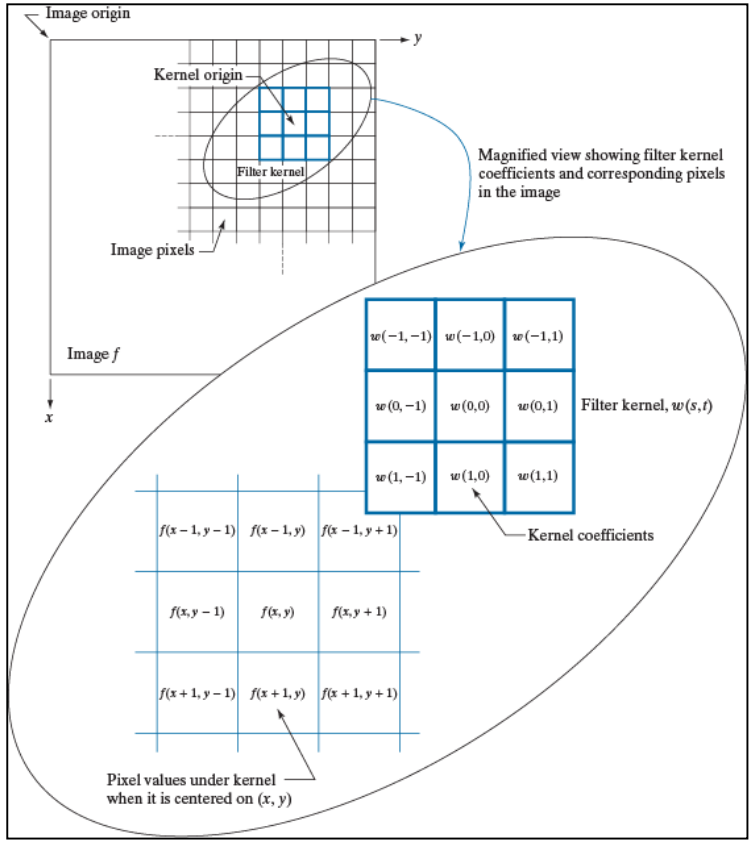
F_1	F_2	F_3
F_4	F_5	F_6
F_7	F_8	F_9

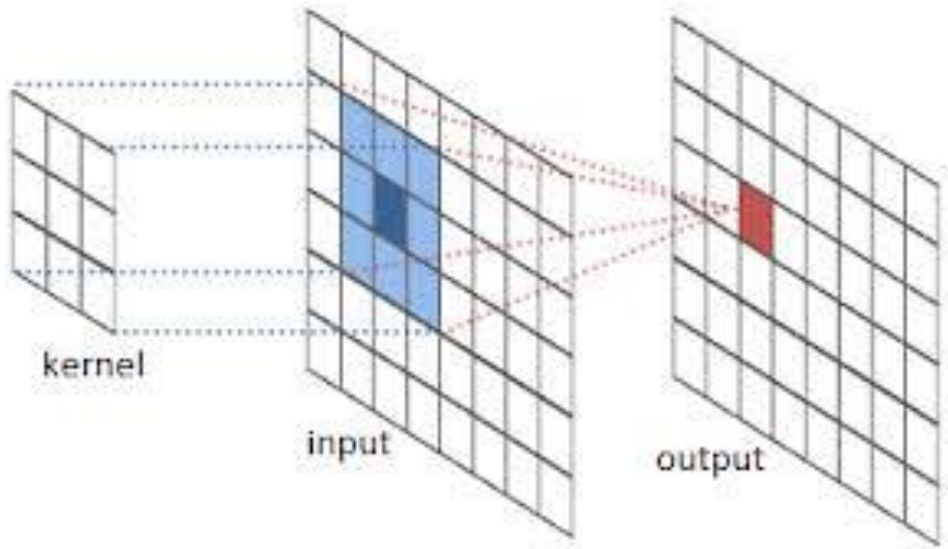
		$I(i-1,j-1)$	$I(i-1,j)$	$I(i-1,j+1)$	
		$I(i,j-1)$	$I(i,j)$	$I(i,j+1)$	
		$I(i+1,j-1)$	$I(i,j+1)$	$I(i+1,j+1)$	

$$I_{new}(i,j) = F_1 * I(i-1,j-1) + F_2 * I(i-1,j) + F_3 * I(i-1,j+1)$$
$$+ F_4 * I(i,j-1) + F_5 * I(i,j) + F_6 * I(i,j+1)$$
$$+ F_7 * I(i+1,j-1) + F_8 * I(i+1,j) + F_9 * I(i+1,j+1)$$

Spatial Convolution

FIGURE 3.28
The mechanics of linear spatial filtering using a 3×3 kernel. The pixels are shown as squares to simplify the graphics. Note that the origin of the image is at the top left, but the origin of the kernel is at its center. Placing the origin at the center of spatially symmetric kernels simplifies writing expressions for linear filtering.

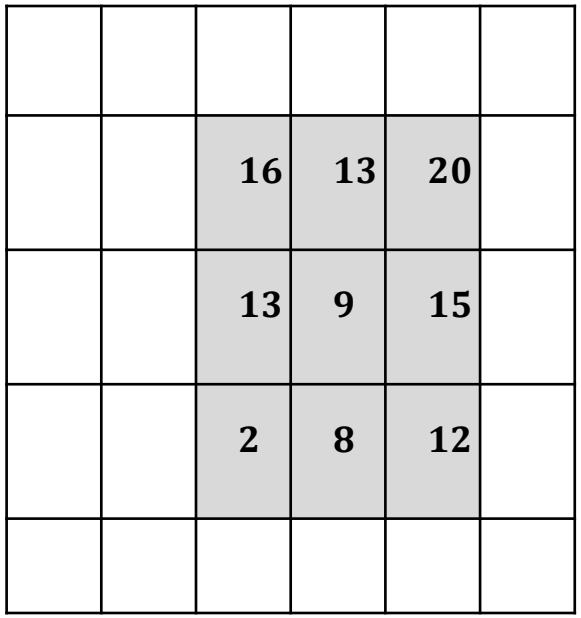




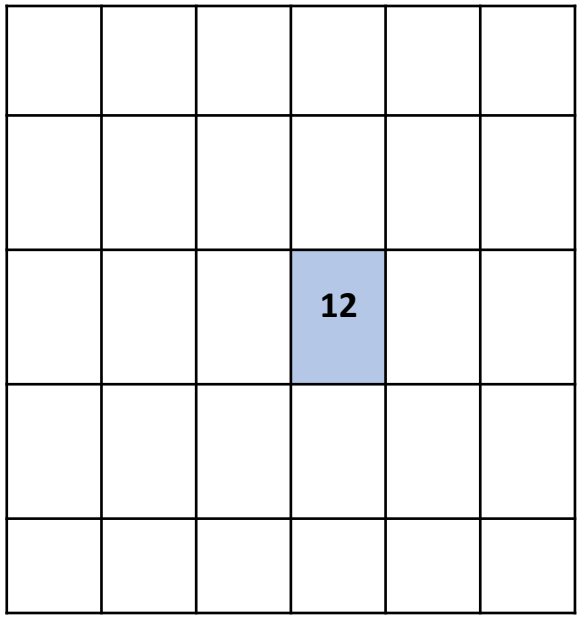
Filtering

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

kernel



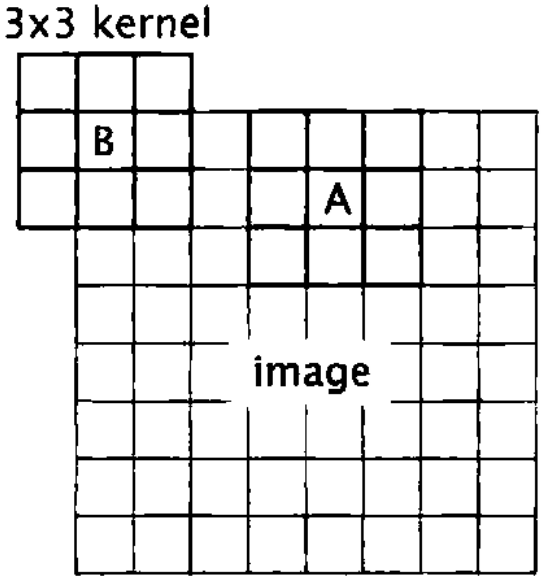
Input



output

1/9*16+1/9*13+1/9*20+1/9*13+1/9+1/9*15+1/9*2+1/*8+1/9*12=12

Boundary pixels



- Zero padding
  - All padded pixels are assigned a value of zero
- Constant padding
  - A constant value is used for all padded pixels
- Nearest neighbor
  - The values from the last row or column are used for padding
- Reflect
  - The values from the last row or column are reflected across the boundary of the image
- Wrap
  - the first row (or column) after the boundary takes the same values as the first row (or column) in the image and so on

Boundary pixels

0	2	5	7	3	10	9
11	1	4	6	8	2	0
0	12	10	9	7	4	5
1	9	7	8	13	11	0
5	10	14	6	2	1	1
7	6	11	3	13	8	4
3	9	6	12	7	10	5

(a) A 7-by-7 input image.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	2	5	7	3	10	9	0	0
0	0	11	1	4	6	8	2	0	0	0
0	0	0	12	10	9	7	4	5	0	0
0	0	1	9	7	8	13	11	0	0	0
0	0	5	10	14	6	2	1	1	0	0
0	0	7	6	11	3	13	8	4	0	0
0	0	3	9	6	12	7	10	5	0	0
0	0	0	0	0	0	0	0	0	0	0

(b) Padding with zeros.

5	5	5	5	5	5	5	5	5	5	5
5	5	0	2	5	7	3	10	9	5	5
5	5	11	1	4	6	8	2	0	5	5
5	5	0	12	10	9	7	4	5	5	5
5	5	1	9	7	8	13	11	0	5	5
5	5	5	10	14	6	2	1	1	5	5
5	5	7	6	11	3	13	8	4	5	5
5	5	3	9	6	12	7	10	5	5	5
5	5	5	5	5	5	5	5	5	5	5

(c) Padding with a constant.

Boundary pixels

0	2	5	7	3	10	9
11	1	4	6	8	2	0
0	12	10	9	7	4	5
1	9	7	8	13	11	0
5	10	14	6	2	1	1
7	6	11	3	13	8	4
3	9	6	12	7	10	5

(a) A 7-by-7 input image.

2	0	0	2	5	7	3	10	9	9	10
2	0	0	2	5	7	3	10	9	9	10
1	11	11	1	4	6	8	2	0	0	2
12	0	0	12	10	9	7	4	5	5	4
9	1	1	9	7	8	13	11	0	0	11
10	5	5	10	14	6	2	1	1	1	1
6	7	7	6	11	3	13	8	4	4	8
9	3	3	9	6	12	7	10	5	5	10
9	3	3	9	6	12	7	10	5	5	10

(e) Padding with reflect option.

5	10	3	9	6	12	7	10	5	3	9
10	9	0	2	5	7	3	10	9	0	2
2	0	11	1	4	6	8	2	0	11	1
4	5	0	12	10	9	7	4	5	0	12
11	0	1	9	7	8	13	11	0	1	9
1	1	5	10	14	6	2	1	1	5	10
8	4	7	6	11	3	13	8	4	7	6
10	5	3	9	6	12	7	10	5	3	9
9	10	0	2	5	7	3	10	9	0	2

(f) Padding with wrap option.

0	0	0	2	5	7	3	10	9	9	10
0	0	0	2	5	7	3	10	9	9	9
11	11	11	1	4	6	8	2	0	0	0
0	0	0	12	10	9	7	4	5	5	5
1	1	1	9	7	8	13	11	0	0	0
5	5	5	10	14	6	2	1	1	1	1
7	7	7	6	11	3	13	8	4	4	4
3	3	3	9	6	12	7	10	5	5	5
3	3	3	9	6	12	7	10	5	5	5

(d) Padding with nearest neighbor.

Smoothing filters

- Used to reduce sharp transitions in intensity
- Used to reduce irrelevant detail in an image
- Blur the image (degree of blurring depends on the size of the filter)
- Examples:
  - Mean filter (box kernel)
  - Gaussian filter

a b

FIGURE 3.31 Examples of smoothing kernels: (a) is a *box* kernel; (b) is a *Gaussian* kernel.

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

$$\frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

The python modules

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

src	input image.
dst	output image of the same size and the same number of channels as src.
ddepth	desired depth of the destination image, see **combinations**
kernel	convolution kernel (or rather a correlation 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 lie 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.
borderType	Specifies image boundaries while kernel is applied on image borders (cv2.BORDER_CONSTANT cv2.BORDER_REPLICATE cv2.BORDER_REFLECT cv.2BORDER_WRAP cv2.BORDER_REFLECT_101 cv2.BORDER_DEFAULT, …)

```
kernel = np.ones((5, 5), np.float32) / 25
```

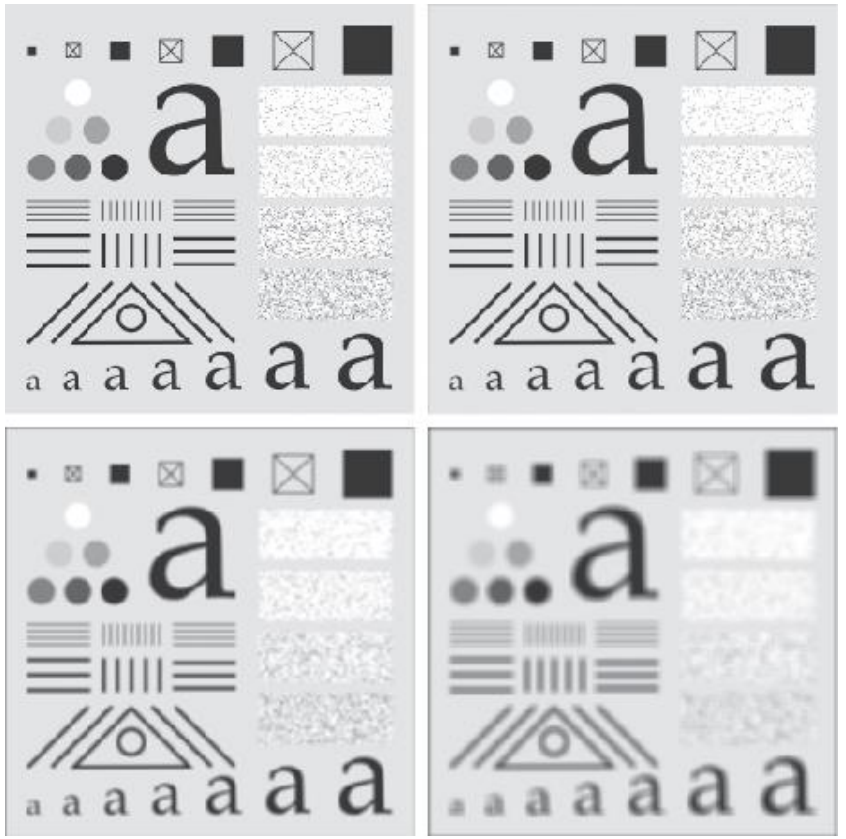
#when ddepth=-1, the output image will have the same depth as the source

```
dst = cv2.filter2D(img, -1, kernel)
```

Mean filter – kernels of different sizes

a b
c d

FIGURE 3.33
(a) Test pattern of size 1024×1024 pixels.
(b)-(d) Results of lowpass filtering with box kernels of sizes 3×3, 11×11, and 21×21, respectively.



Box Blur convenience function

dst = cv2.blur(src, ksize[, dst[, anchor[, borderType]]])

src	input image; it can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
dst	output image of the same size and type as src.
ksize	blurring kernel size.
anchor	anchor point; default value Point(-1,-1) means that the anchor is at the kernel center.
borderType	border mode used to extrapolate pixels outside of the image, see **BorderTypes**.

```
kernel = np.ones((5, 5), np.float32) / 25
dst = cv2.filter2D(img, -1, kernel)
```

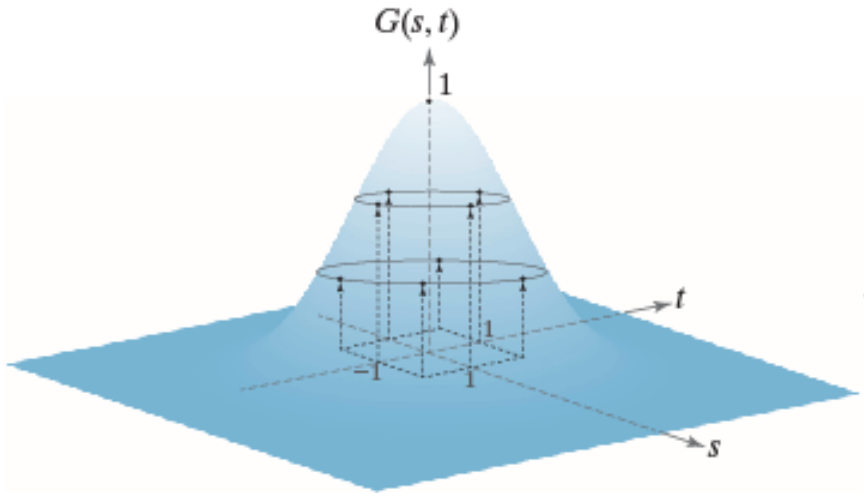
is equivalent to

```
dst = cv2.blur(img, (5, 5))
```

Gaussian filter

a b

FIGURE 3.35
(a) Sampling a Gaussian function to obtain a discrete Gaussian kernel. The values shown are for $K = 1$ and $\sigma = 1$. (b) Resulting 3×3 kernel [this is the same as Fig. 3.31(b)].



$$\frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

Gaussian filter

dst = cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType=BORDER_DEFAULT]]])

src	input image
dst	output image
ksize	Gaussian Kernel Size. [height width]. height and width should be odd and can have different values. If ksize is set to [0 0], then ksize is computed from sigma values.
sigmaX	Kernel standard deviation along X-axis (horizontal direction).
sigmaY	Kernel standard deviation along Y-axis (vertical direction). If sigmaY=0, then sigmaX value is taken for sigmaY. if both sigmas are zeros, they are computed from ksize.width and ksize.height, respectively Sigma=0.3∗((ksize−1)∗0.5−1)+0.8
borderType	Specifies image boundaries while kernel is applied on image borders (cv2.BORDER_CONSTANT cv2.BORDER_REPLICATE cv2.BORDER_REFLECT cv.2BORDER_WRAP cv2.BORDER_REFLECT_101 cv2.BORDER_DEFAULT, …)

Gaussian filter

original

$\sigma = 5$

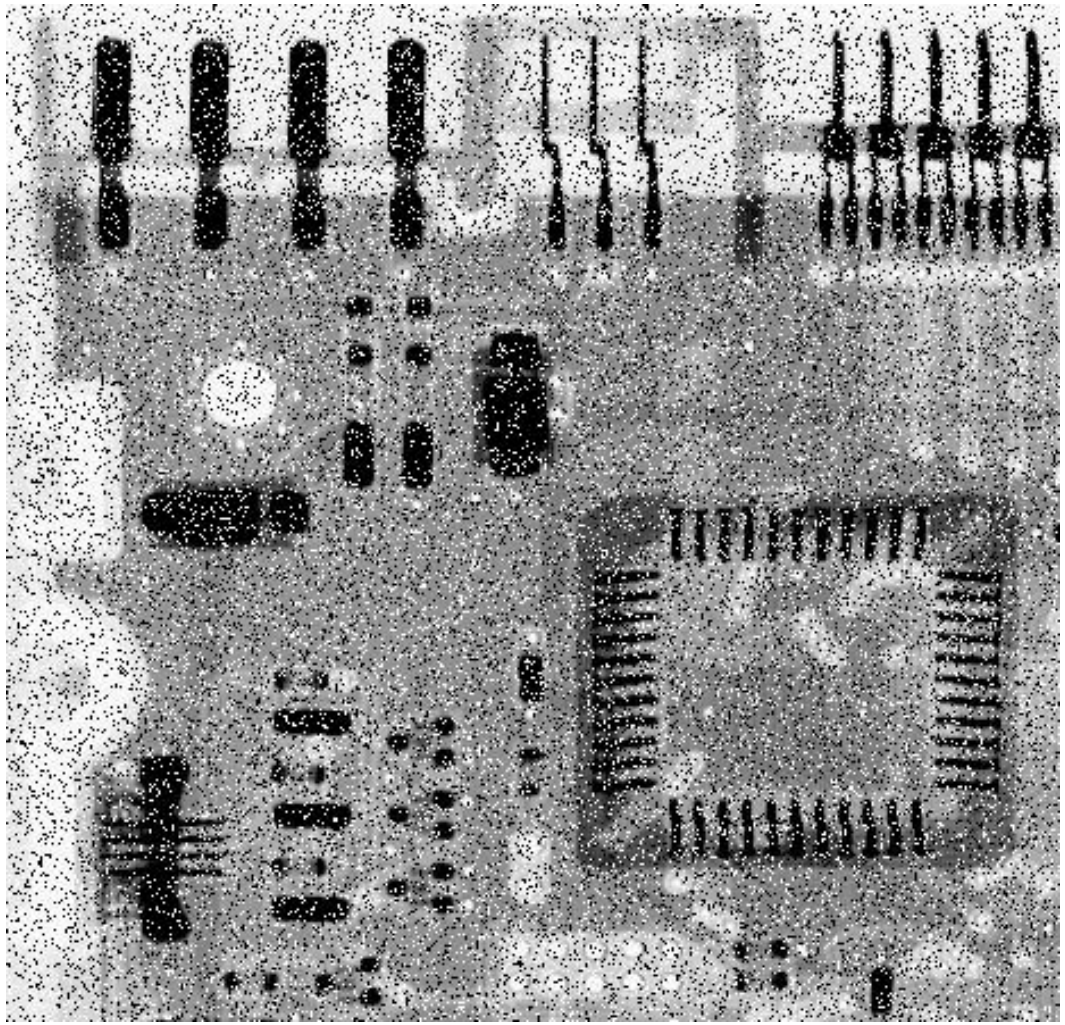
$\sigma = 7$

Order statistics (non-linear) filters

- Order statistics are nonlinear spatials filters whose response is based on ordering the pixels in the region encompassed by filter
- Examples:
  - Median filter (the best-known)
  - Minimum filter
  - Maximum filter
- A filter is called linear if f(x + y) = f(x) + f(y). Otherwise, f is non-linear.

Median filter

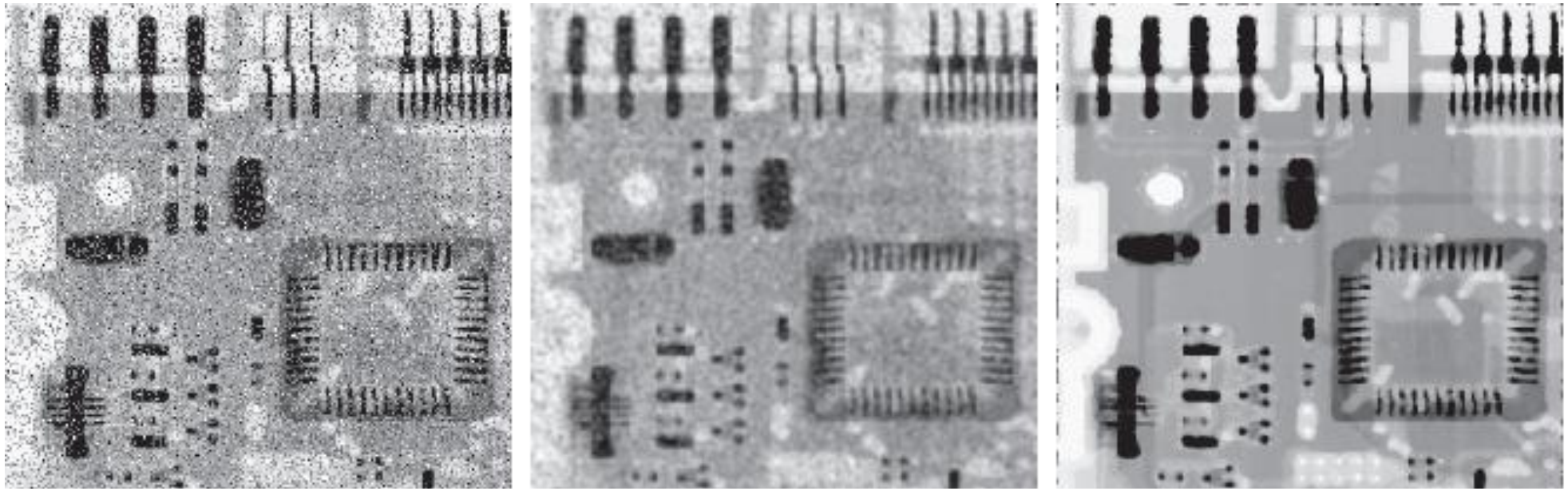
- Commonly used in removing salt-and-pepper noise and impulse noise.
- Salt-and-pepper noise is characterized by black and white spots randomly distribute in an image.



Median filter

dst=cv2.medianBlur(src, ksize)

- The median of a set of values is a value m in the set that half of values is less or equal to m, and half is greater or equal to m.
  - Example: the median of a set of values {5, 6, 7, 10, 13, 14, 15, 19, 23} is $m = 13$



a b c

FIGURE 3.43 (a) X-ray image of a circuit board, corrupted by salt-and-pepper noise. (b) Noise reduction using a 19×19 Gaussian lowpass filter kernel with $\sigma = 3$. (c) Noise reduction using a 7×7 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

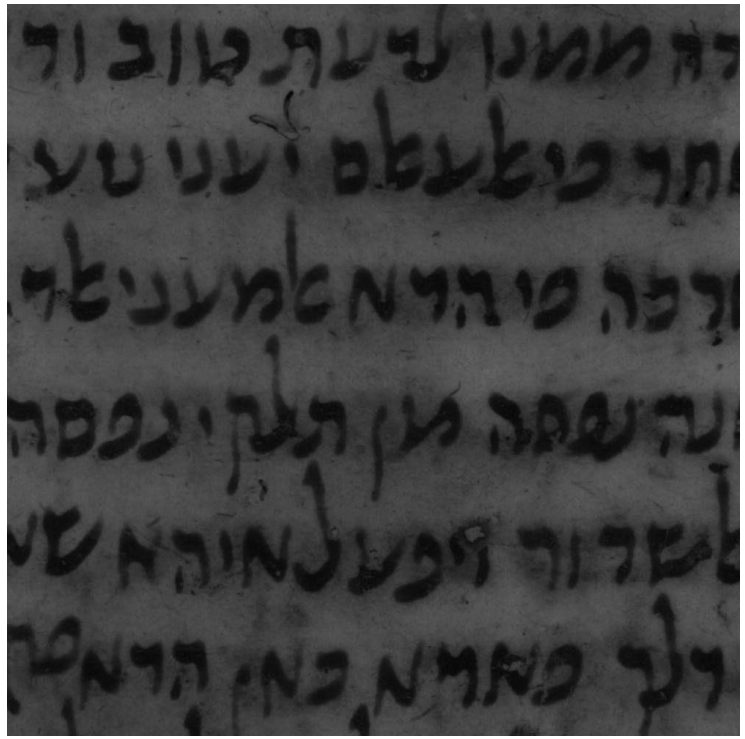
Max and Min filters

- Max filter
  - the maximum value in the sub-image replaces the value at (i, j)
  - enhances the bright points in an image.

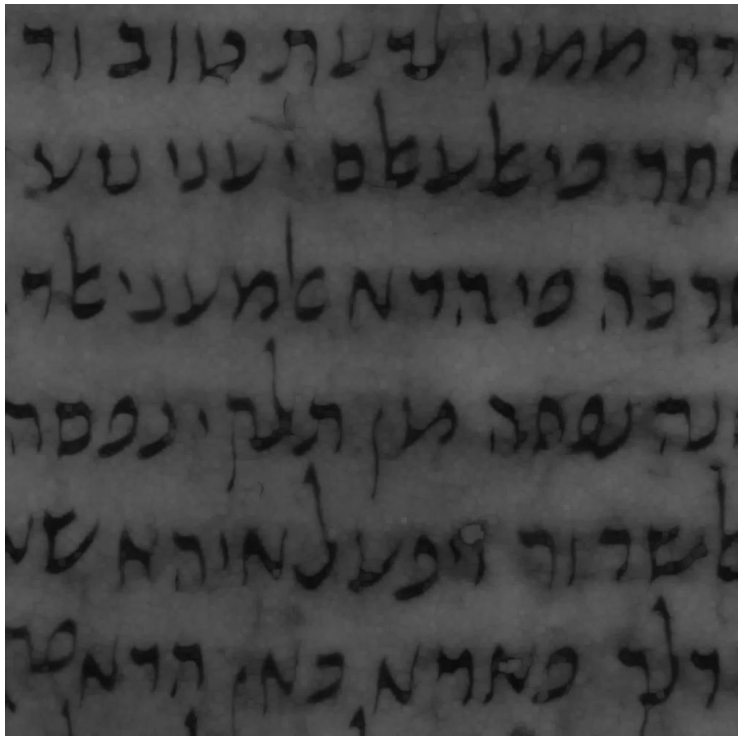
- Min filter
  - the minimum value in the sub-image replaces the value at (i, j)
  - enhances the darkest points in an image.

```
scipy.ndimage.filters.minimum_filter(im, size)
scipy.ndimage.filters.maximum_filter(im, size)
```

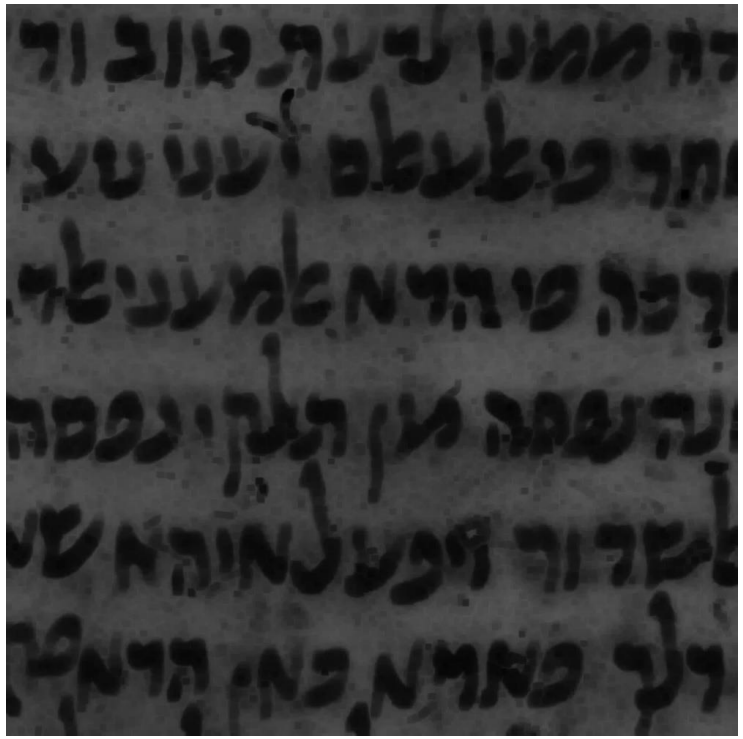
Max and Min filters



original



Max filter of size = 7



Min filter of size = 7

Sharpening filters

- Highlights transitions in intensity (edges)
- Based of derivatives
  - Examples:
    - Sobel
    - Prewitt
    - Canny
    - Laplacian

Images and derivative

An Image as a Function

- As we treat image as a function
  - It is possible to compute is derivative, but we need to exchange Δx by 1.
- It is possible to compute the first and the second derivative of an image.
- Derivative help in determining local minima, local maxima, and change in derivative direction

$$f'(x) = \lim_{\Delta x \to 0} \frac{f\ (x+\Delta x) - f(x)}{\Delta x}$$

$$I'(x) = \frac{I(x+1) - I(x)}{1} = I(x+1) - I(x)$$

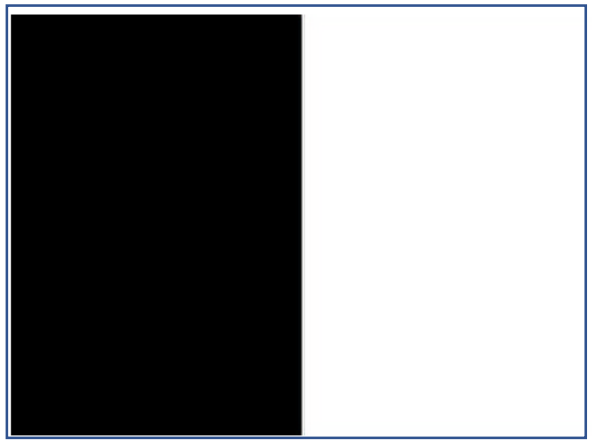
$$f''(x) = \lim_{\Delta x \to 0} \frac{f'(x+\Delta x) - f'(x)}{\Delta x}$$

$$I''(x) = \frac{I'(x+1) - I'(x)}{1}$$

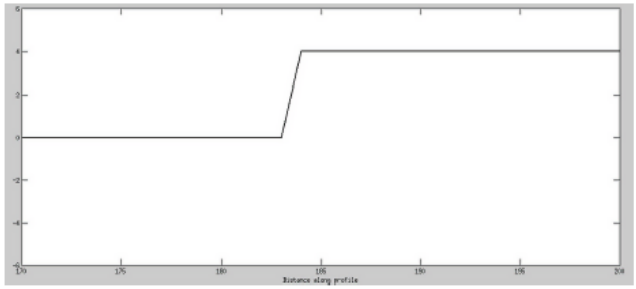
$$= I(x+2) - I(x+1) - I(x+1) - I(x)$$

$$= I(x+2) - 2I(x+1) - I(x)$$

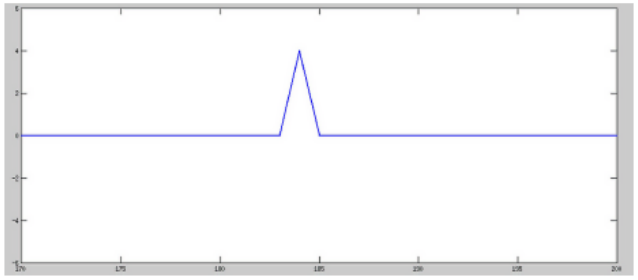
Images and derivative



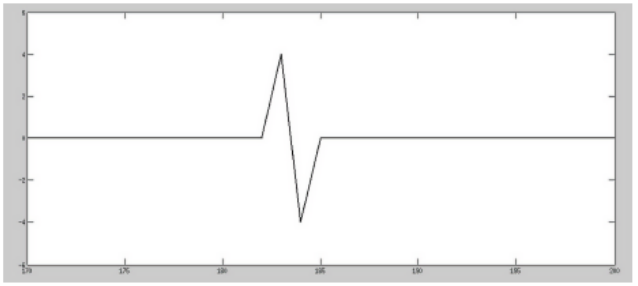
(a) Input image.



(b) Intensity profile.



(c) First derivative profile.



(d) Second derivative profile.

Images and derivative

If f(x, y) is a continuous function, then the gradient of f is a vector

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \qquad f_x = \frac{\partial f}{\partial x} \qquad f_y = \frac{\partial f}{\partial y}$$

The magnitude of the gradient is

$$|\nabla f| = [(f_x)^2 + (f_y)^2]^{\frac{1}{2}}$$

For computational purposes, we will use the simplified version of the gradient

$$|\nabla f| = |f_x| + |f_y|$$

$$\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$$

First derivative filters – Prewitt and Sobel

TABLE 4.6: Prewitt masks for horizontal and vertical edges.

−1	−1	−1
0	0	0
1	1	1

−1	0	1
−1	0	1
−1	0	1

TABLE 4.3: Sobel masks for horizontal and vertical edges.

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

The intuition

edge

		0	0	255	
		0	0	255	
		0	0	255	

Input

−1	0	1
−1	0	1
−1	0	1

kernel

			765		

output

constant intensity

		0	0	0	
		0	0	0	
		0	0	0	

Input

			0		

output

Sobel Filter

dst=cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

src	input image.
dst	output image of the same size and the same number of channels as src .
ddepth	output image depth, see **combinations**; in the case of 8-bit input images it will result in truncated derivatives.
dx	order of the derivative x.
dy	order of the derivative y.
ksize	size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
scale	optional scale factor for the computed derivative values; by default, no scaling is applied (see **getDerivKernels** for details).
delta	optional delta value that is added to the results prior to storing them in dst.
borderType	pixel extrapolation method, see **BorderTypes**. **BORDER_WRAP** is not supported.

Sobel Filtering Example

```
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1)

# The sobelX and sobelY images are now of the floating  point data type - we need convert back to an 8-bit unsigned integer

sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))

sobelCombined = cv2.bitwise_or(sobelX, sobelY)
```

Sobel Filtering Example

Input

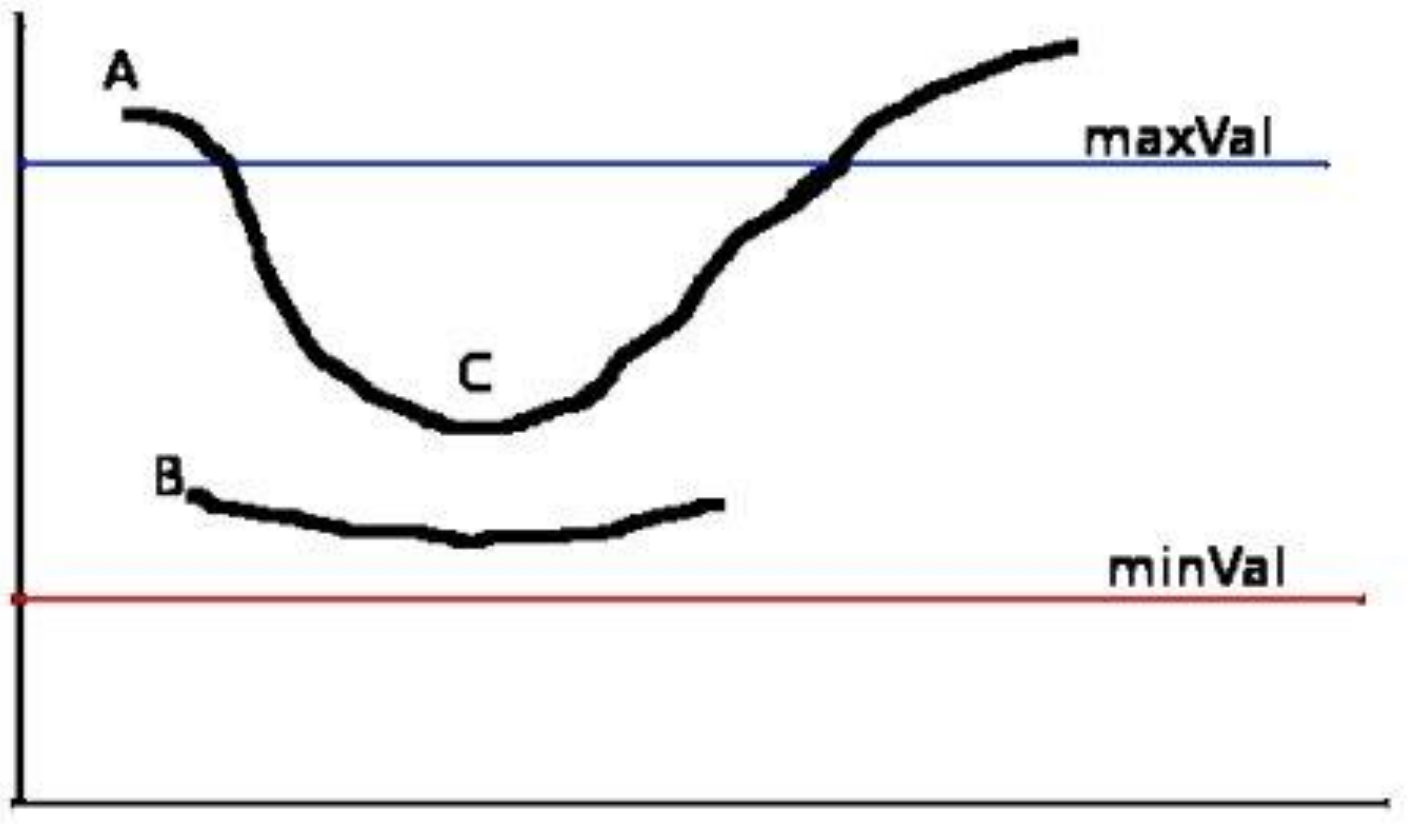
sobelX

sobelY

combined

Notice is that the edges are very “noisy”.

Canny Filter



The edge A is above the *maxVal*, so considered as “sure-edge”. Although edge C is below *maxVal*, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above *minVal* and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded.

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

Canny Filter

- A multi-stage algorithm
  - Smooth the images with Gaussian filter
  - Filter the smoothed image with a Sobel kernel in both horizontal and vertical directions and calculate a gradient value
  - Non-maximum Suppression
  - Classify edge pixels as strong and weak (need two threshold values)
  - Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded

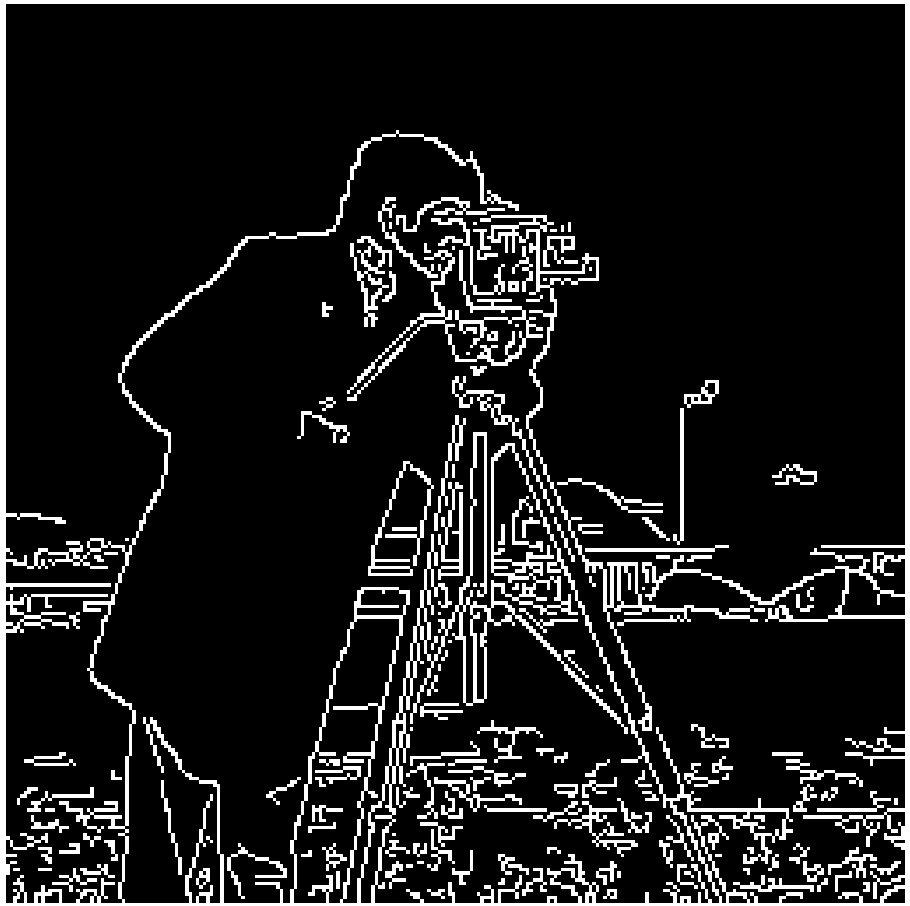
Canny

edges=cv2.Canny(image, threshold1, threshold2[, edges[, L2gradient]])

image	8-bit input image.
edges	output edge map; single channels 8-bit image, which has the same size as image .
threshold1	first threshold for the hysteresis procedure.
threshold2	second threshold for the hysteresis procedure.

Canny Filter

canny = cv2.Canny(img,100,200)



Laplacian Filter

- Based on a second derivative.
- Have several versions
- Most commonly used version

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

0	1	0
1	−4	1
0	1	0

dst=cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])

```
lap = cv2.Laplacian(img, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
```

Billateral filtering example

Smooth image while preserving the edges

Input

Blur, kernel = (5,5)

billateral

Bilateral Filter

- Smooth image while preserving the edges
- cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])

src	Source 8-bit or floating-point, 1-channel or 3-channel image.
dst	Destination image of the same size and type as src .
d	Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace.
sigmaColor	Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color.
sigmaSpace	Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.
borderType	border mode used to extrapolate pixels outside of the image, see **BorderTypes**

Photoshop-like filters - Pencil Sketch Filter

```
img = cv2.imread('FamilyTrip.jpg')
img_blur = cv2.GaussianBlur(img, (5,5), 0, 0)
sketch_grey, sketch_color = cv2.pencilSketch(img_blur)
```

Photoshop-like filters - Pencil Sketch Filter

- dst1, dst2 = cv2.pencilSketch(src[, dst1[, dst2[, sigma_s[, sigma_r[, shade_factor]]]]])
- dst1: Output 8-bit 1-channel image.
  dst2: Output image with the same size and type as src.

- Usually you use the default parameters

Photoshop-like filters - Stylization Filter

Photoshop-like filters - Stylization Filter

- dst = cv2.stylization(src[, dst[, sigma_s[, sigma_r]]])

  dst: output array of the same size and depth as src
- Usually you use the default parameters

Summary

- The mean and Gaussian filters smooth the image while blurring the edges in the image.
- The median filter is effective in removing salt-and-pepper noise
- The most widely used first derivative filters are Sobel and and Canny
- Laplacian is a popular second derivatives filter
- Photoshop-like filters