Slides and example codes are available:
https://github.com/mint-lab/cv_tutorial

Image Processing

Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

(Digital) Image Processing

- **Digital image processing** is a process to analyze, modify, and synthesize digital images using a digital computer through an algorithm.
  - A subfield of digital signal processing

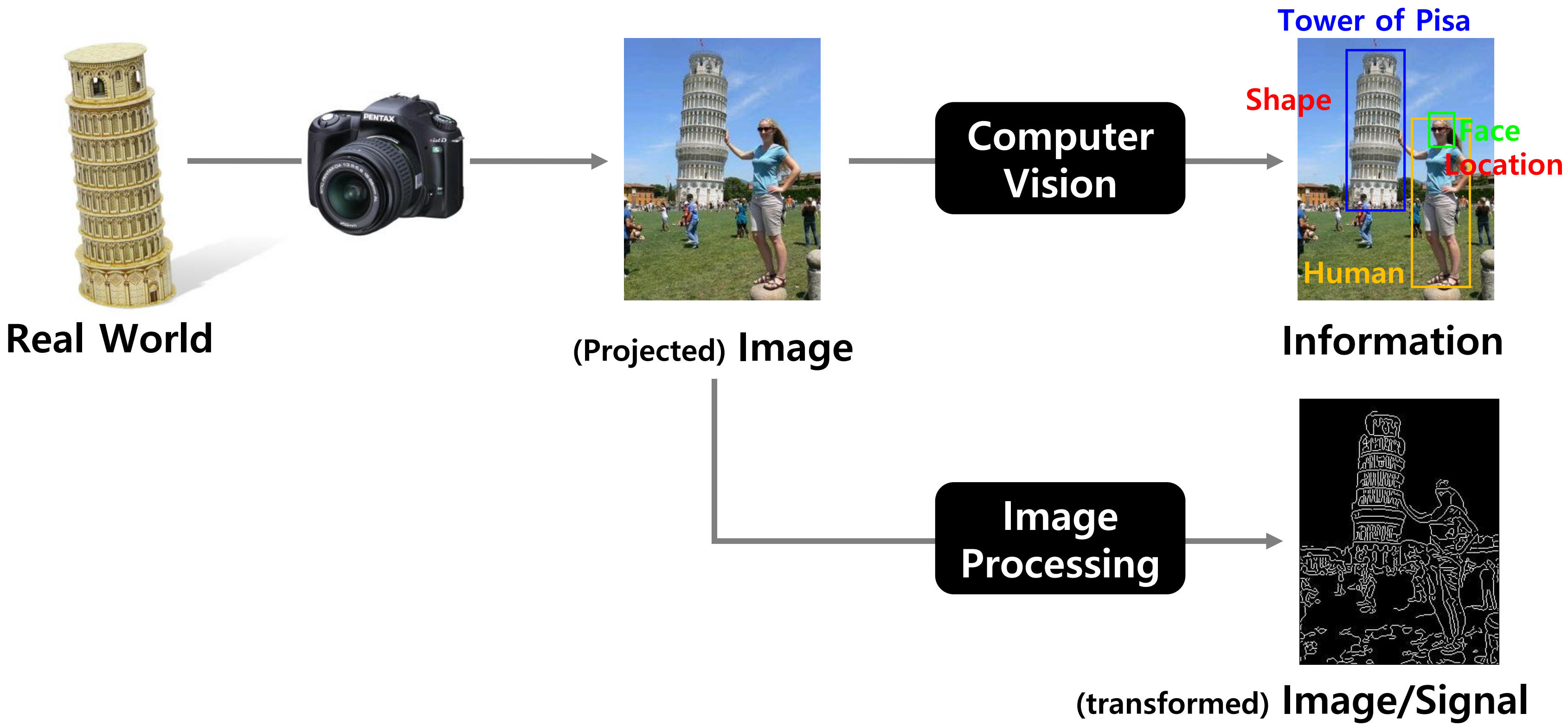
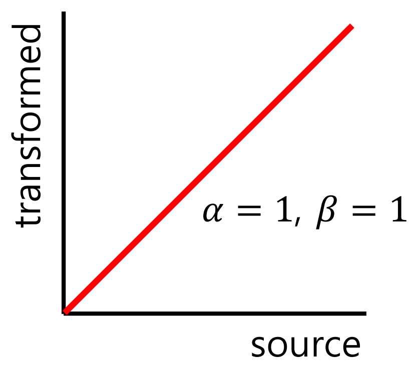
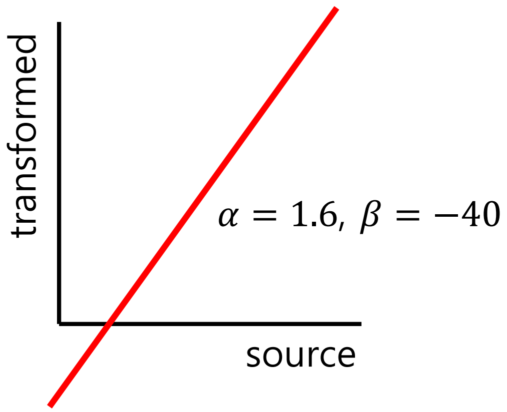


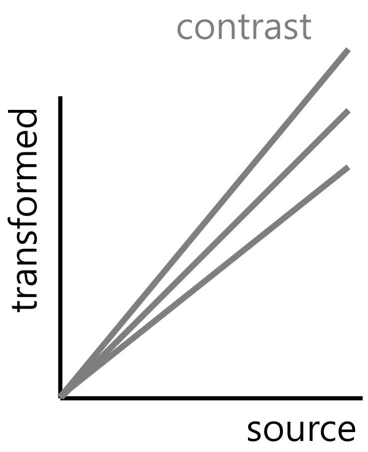
Table of Contents

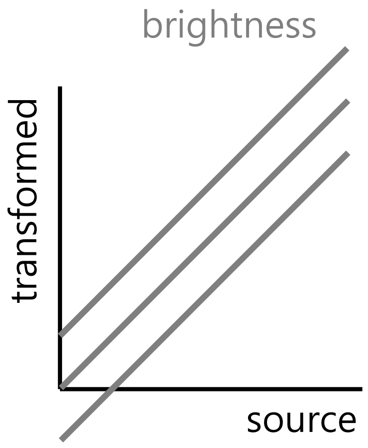
Review) Image Editing: Intensity Transformation

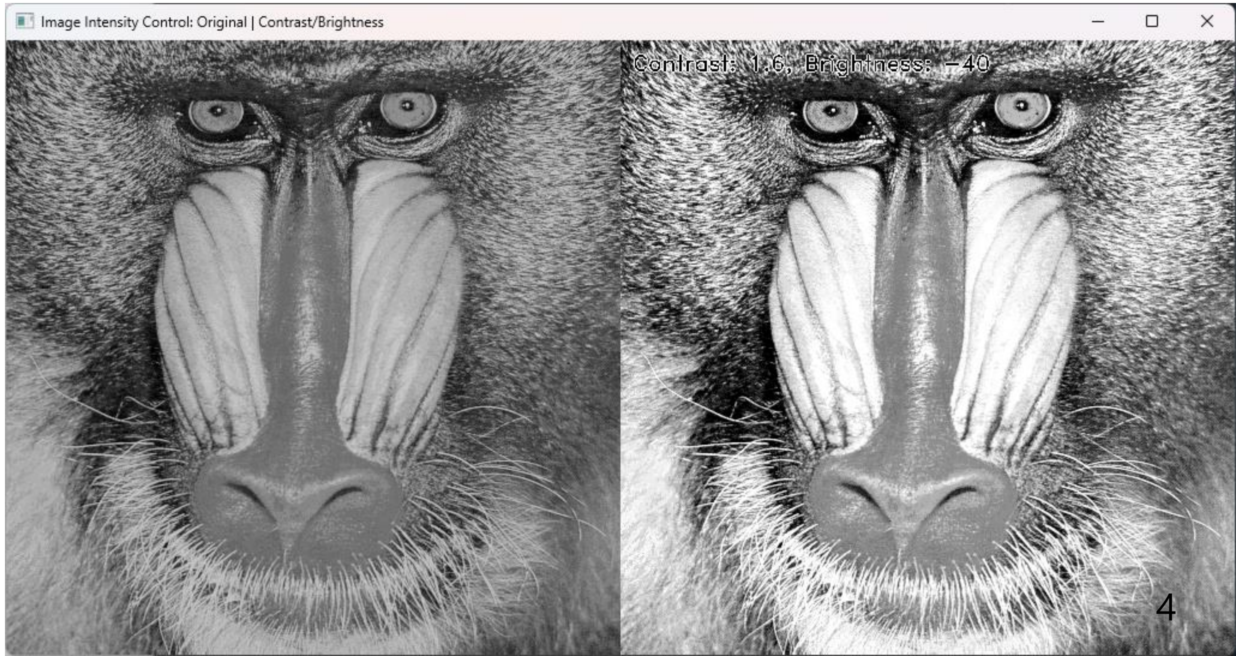
- Example) Intensity Transformation with contrast and brightness
  - *Contrast* is the property that makes an object (or its representation in an image or display) distinguishable.
  - *Brightness* is the strength of overall luminance.
  - A simple formulation: $I' = \alpha I + \beta$
    - α: contrast (slope)
    - β: brightness (Y intercept)











Histogram

- A **histogram** is an approximate graphical representation of the distribution of numerical data.
  - ~ Probability distribution
  - Note) The bin width (and data range) is important. → The number of bins
- Example) 500 items (1.27, 0.50, 0.12, 3.29, -1.18, …)

Frequency table

Bin/Interval	Count/Frequency
-3.5 to -2.51	9
-2.5 to -1.51	32
-1.5 to -0.51	109
-0.5 to 0.49	180
0.5 to 1.49	132
1.5 to 2.49	34
2.5 to 3.49	4

Histogram

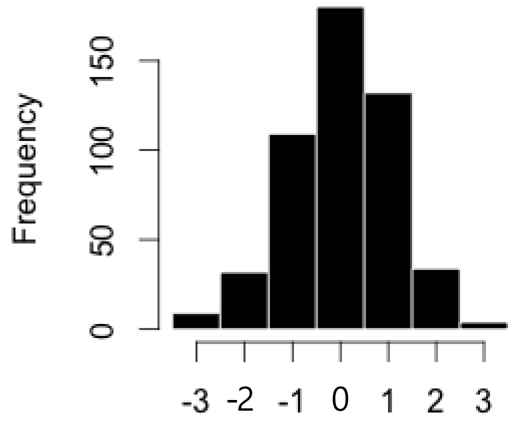
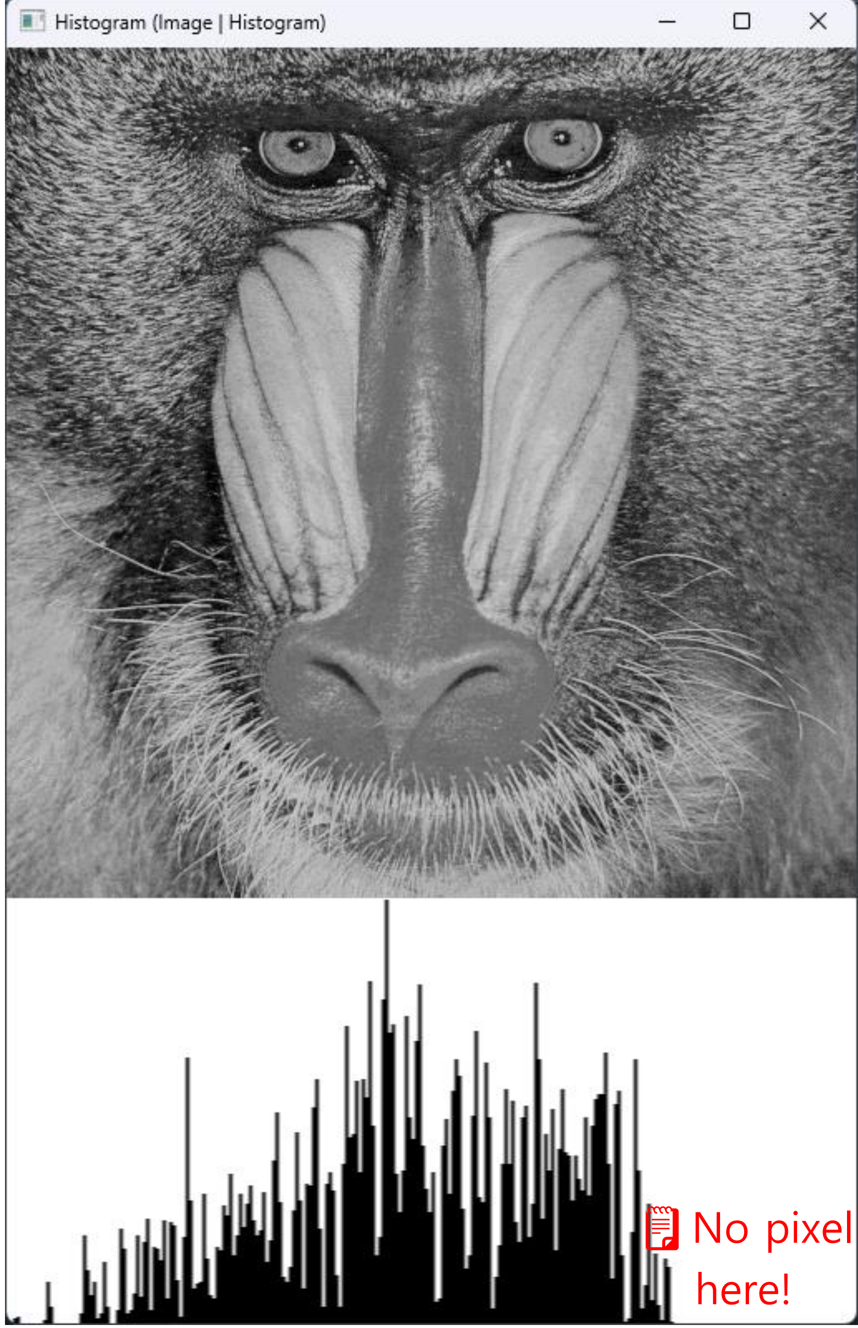


Image: Wikipedia

Histogram

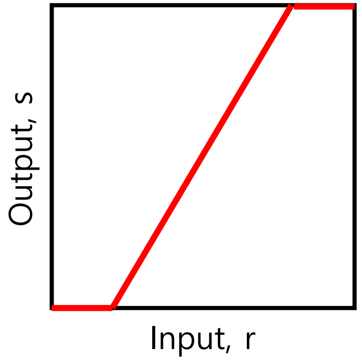
- Example) Image histogram

```python
def get_histogram(gray_img): # Alternative) cv.calcHist()
    # Assume a gray input image
    # Fix the bin range [0, 256) and bin size 256
    hist = np.zeros((256), dtype=np.uint32)
    for val in range(0, 256):
        hist[val] = sum(sum(gray_img == val)) # Count the occurence in 2D
    return hist

def conv_hist2img(hist):
    img = np.full((256, 256), 255, dtype=np.uint8)
    max_freq = max(hist)
    for val in range(len(hist)):
        normalized_freq = int(hist[val] / max_freq * 255)
        img[0:normalized_freq, val] = 0 # Mark as black
    return img[::-1,:]

if __name__ == '__main__':
    # Read the given image as gray scale
    img = cv.imread('data/baboon.tif', cv.IMREAD_GRAYSCALE)

    # Get its histogram
    hist = get_histogram(img)

    # Show the image and its histogram
    img_hist = conv_hist2img(hist)
    img_hist = cv.resize(img_hist, (len(img[0]), len(img_hist))) # Note) Be careful at (width, height)
```

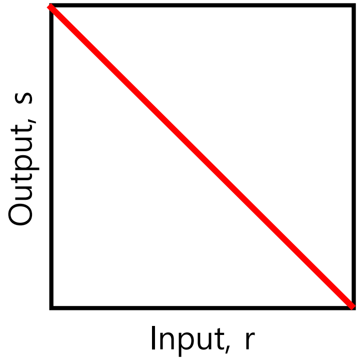


Intensity Transformation: Contrast Stretching

- **Contrast stretching** is a process to change the range of pixel intensity values.
- Note) Linear stretching API in OpenCV Intensity Transformation module,
  - `cv.intensity_transform.contrastStretching(input, output, r1, s1, r2, s2) → None`
    - `(r1, s1)` and `(r2, s2)`: Two control points
    - e.g. Min-max stretching: `s1=0` and `s2=255`



    - e.g. Negative imaging: `(0, 255), (255, 0)`



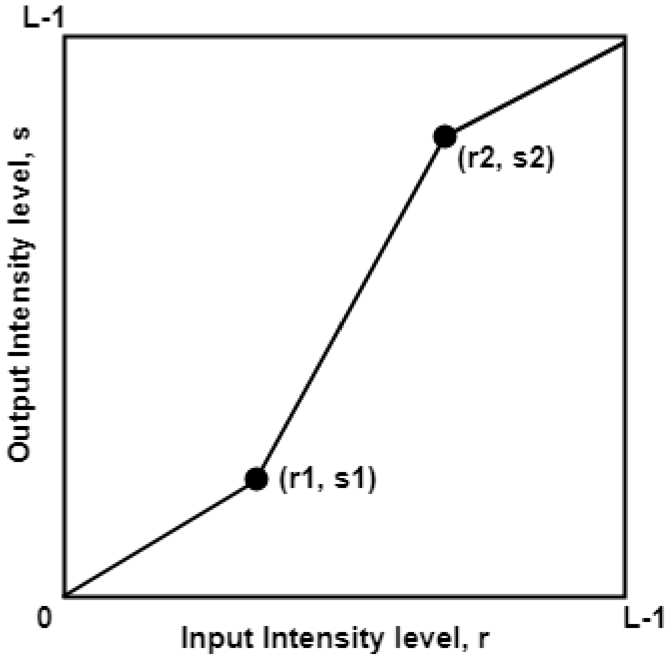
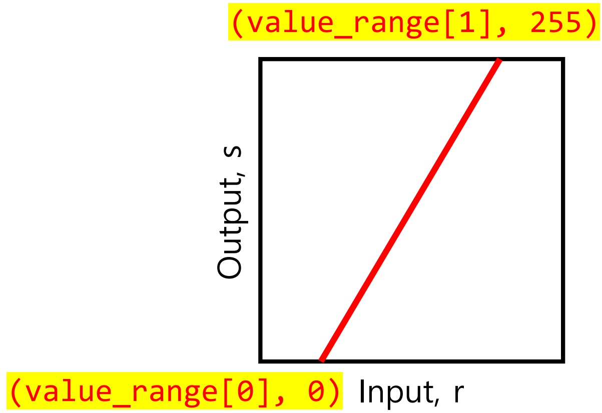


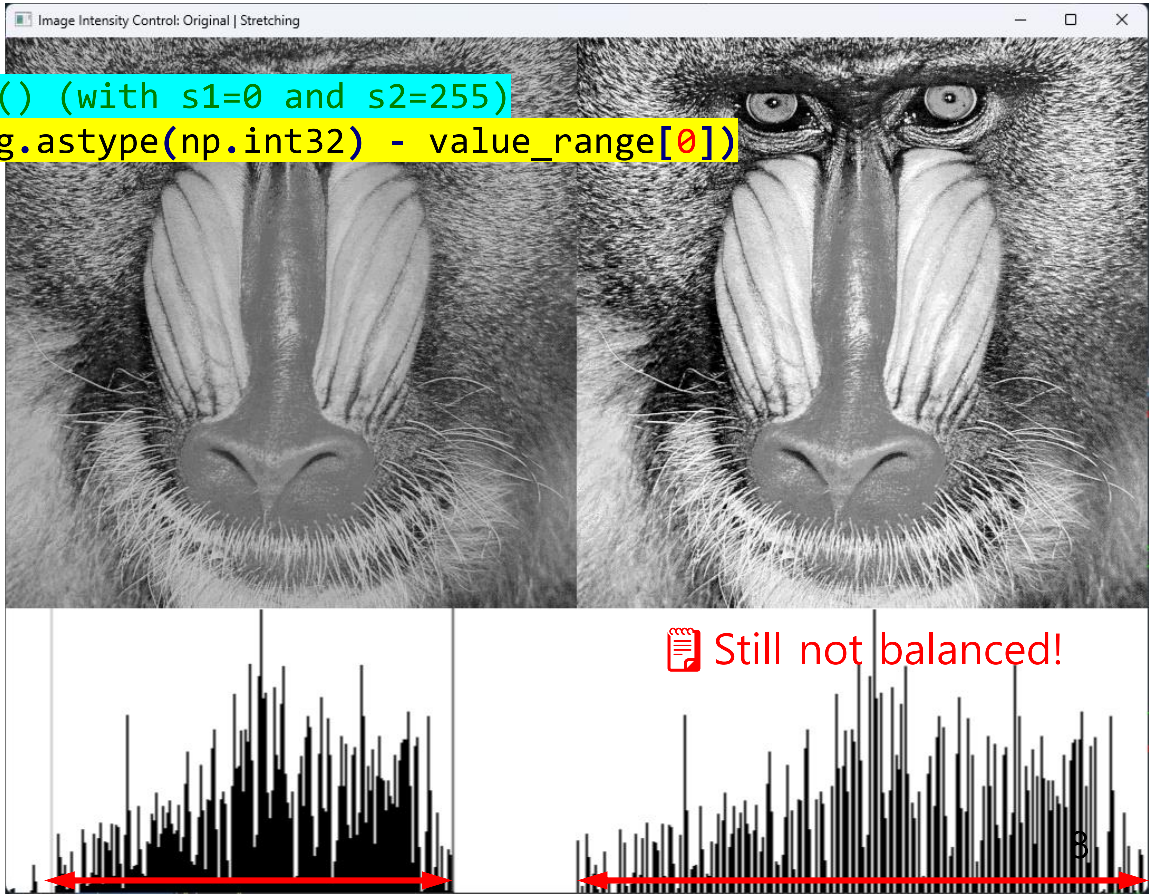
Image: TheAILearner

Intensity Transformation: Contrast Stretching



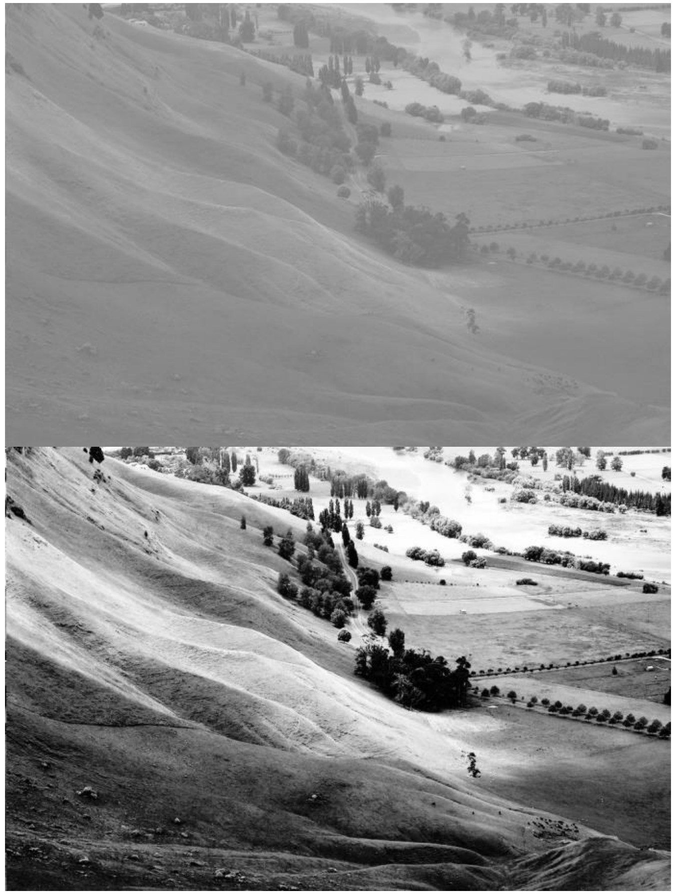
- Example) Contrast stretching with min-max stretching

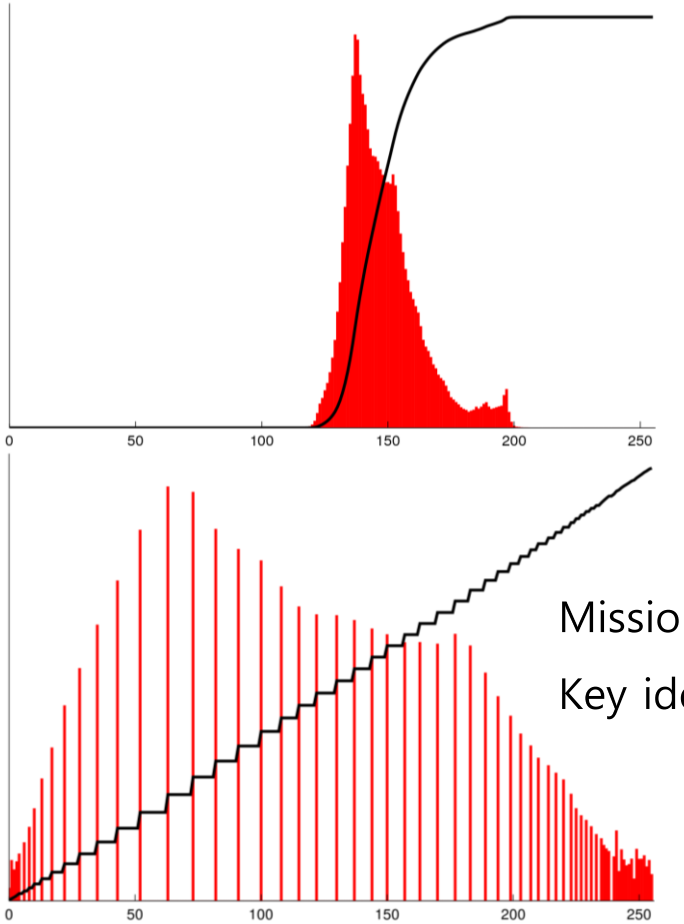
```python
# Read the given image as gray scale
img = cv.imread('data/baboon.tif', cv.IMREAD_GRAYSCALE)

# Initialize control parameters
value_range = [20, 200] # [lower limit, upper limit]

while True:
    # Apply contrast and brightness
    # Alternative) cv.intensity_transform.contrastStretching() (with s1=0 and s2=255)
    img_tran = 255 / (value_range[1] - value_range[0]) * (img.astype(np.int32) - value_range[0])
    img_tran = img_tran.astype(np.uint8) # Apply saturation

    # Get image histograms
    hist = conv_hist2img(get_histogram(img))
    hist_tran = conv_hist2img(get_histogram(img_tran))

    # Mark the intensity range, 'value_range'
    if value_range[0] >= 0 and value_range[0] <= 255:
        mark = hist[:, value_range[0]] == 255
        hist[mark, value_range[0]] = 200
    if value_range[1] >= 0 and value_range[1] <= 255:
        mark = hist[:, value_range[1]] == 255
        hist[mark, value_range[1]] = 100
    ...
```

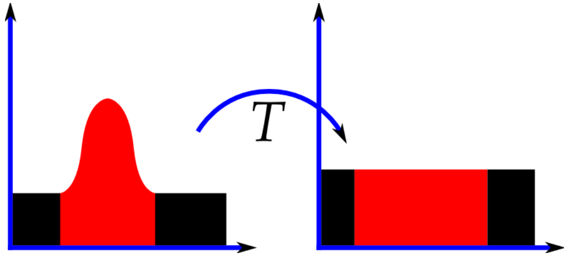


Intensity Transformation: Histogram Equalization

- **[Histogram equalization]** is a contrast adjustment method to make intensity values distributed uniformly.
- Note) Histogram equalization API in OpenCV
  - `cv.equalizeHist(src[, dst]) → dst`
- Example) Histogram and **cumulative histogram (cdf)**







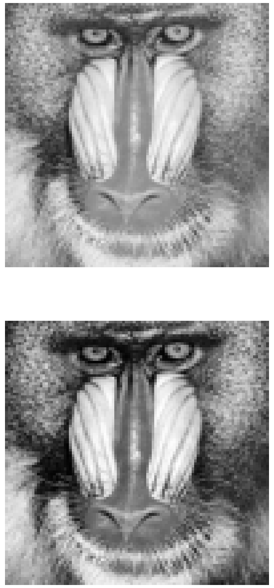
Mission) Make the **cdf** linear.

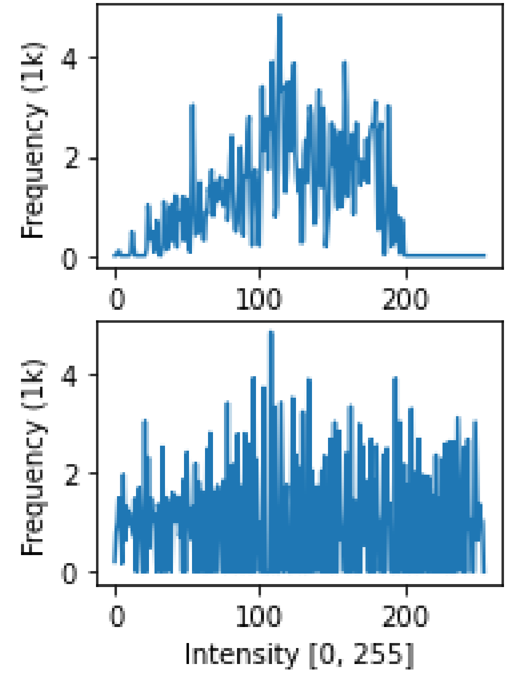
Key idea) Use the **cdf** as the transformation function.

Intensity Transformation: Histogram Equalization

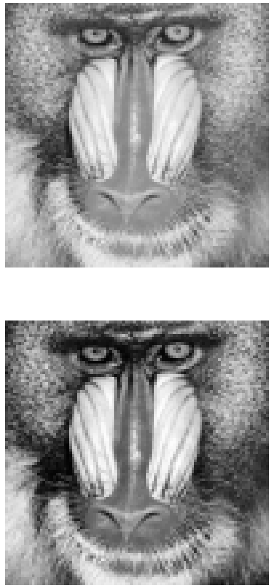
- Example) Histogram equalization

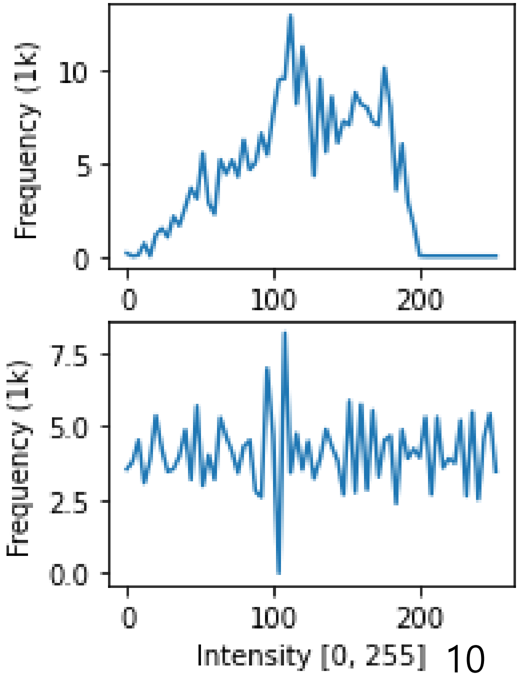
```
import matplotlib.pyplot as plt
import cv2 as cv

# Read the given image as gray scale
img = cv.imread('data/baboon.tif', cv.IMREAD_GRAYSCALE)

# Apply histogram equalization
img_tran = cv.equalizeHist(img)

# Derieve the histogram
bin_width = 4 # Note) The value shold be the power of 2.
bin_num = int(256 / bin_width)
hist = cv.calcHist([img], [0], None, [bin_num], [0, 255])
hist_tran = cv.calcHist([img_tran], [0], None, [bin_num], [0, 255])

# Show all images and their histograms
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.subplot(2, 2, 2)
plt.plot(range(0, 256, bin_width), hist / 1000)
plt.xlabel('Intensity [0, 255]')
plt.ylabel('Frequency (1k)')
plt.subplot(2, 2, 3)
plt.imshow(img_tran, cmap='gray')
plt.axis('off')
```





bin_width = 1



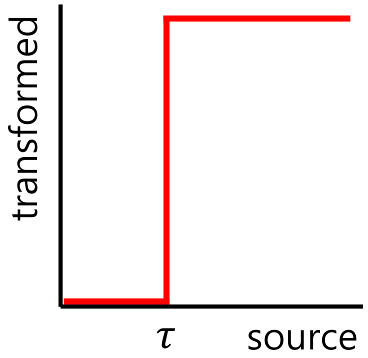


bin_width = 4

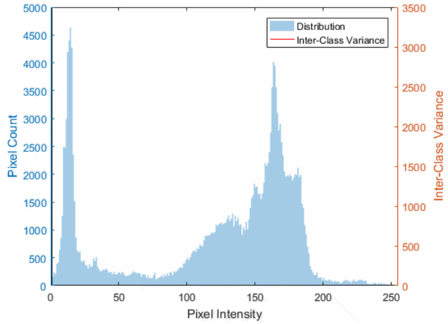
Lager bin_width reveals uniform distribution.

Thresholding

- **Thresholding**: The simplest method of segmenting images (with the given threshold value)
  - e.g. $I'(x,y) = \begin{cases} 1 & \text{if } I(x,y) > \tau \\ 0 & \text{otherwise} \end{cases}$ will generate a binary image I'.



  - Application: Specifying a region-of-interest (ROI)
  - Issue) How to select the threshold value?
    - User-defined vs. automatic (e.g. Otsu's method)
    - Global thresholding vs. local thresholding (a.k.a. adaptive thresholding)
      - Global thresholding with a <u>fixed</u> threshold
      - Local thresholding with a locally <u>adjustable</u> threshold w.r.t. the local image region

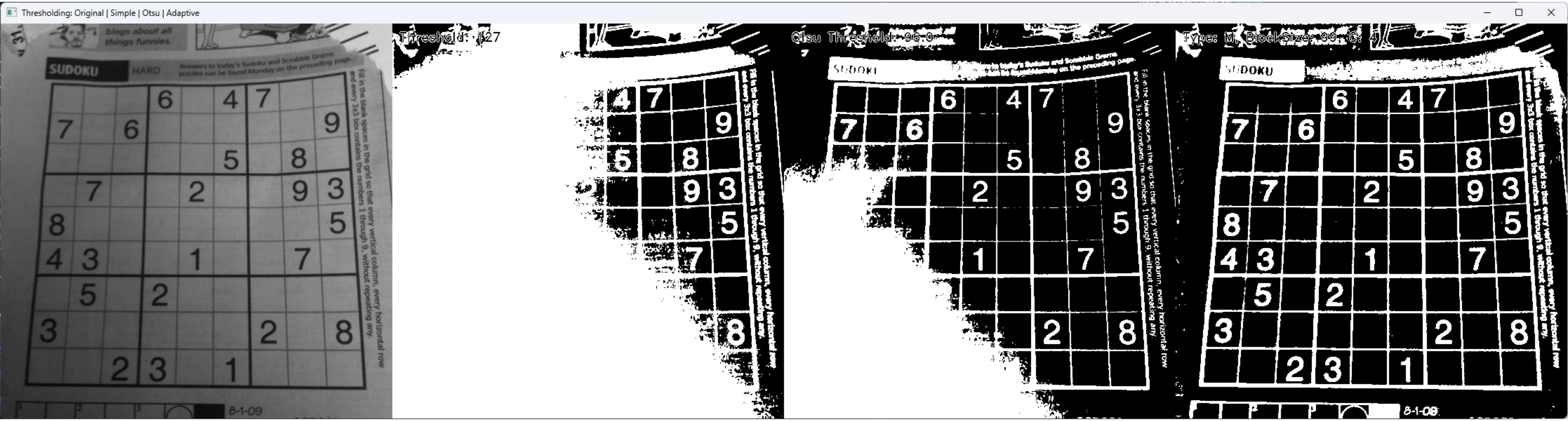


Otsu's method
minimizing intra-class variance
~ maximizing inter-class variance

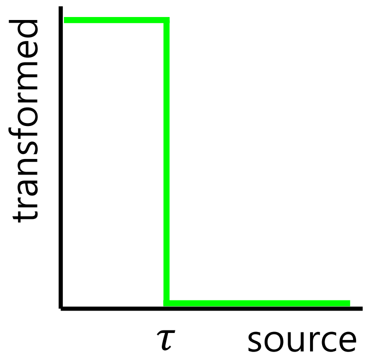
Image: Wikipedia (Thresholding, Otsu's method)

Thresholding

- Example) Thresholding: **Original** | **User threshold** | **Otsu's method** | **Adaptive**



Thresholding



- Example) Thresholding

```
# Read the given image as gray scale
img = cv.imread('data/sudoku.png', cv.IMREAD_GRAYSCALE)
img_threshold_type = cv.THRESH_BINARY_INV # Type: Detect pixels close to 'black' (inverse)

# Initialize control parameters
threshold = 127
adaptive_type = cv.ADAPTIVE_THRESH_MEAN_C
adaptive_blocksize = 99
adaptive_C = 4

while True:
    # Apply thresholding to the image
    _, binary_user = cv.threshold(img, threshold, 255, img_threshold_type)
    threshold_otsu, binary_otsu = cv.threshold(img, threshold, 255, img_threshold_type | cv.THRESH_OTSU)
    binary_adaptive = cv.adaptiveThreshold(img, 255, adaptive_type, img_threshold_type, adaptive_blocksize, adaptive

    # Show the image and its thresholded result
    ...
    merge = np.vstack((np.hstack((img, binary_user)),
                       np.hstack((binary_otsu, binary_adaptive))))
    cv.imshow('Thresholding: Original | Simple | Otsu | Adaptive', merge)

    # Process the key event
    key = cv.waitKey()
    if key == 27: # ESC
        break
```

Image Filtering

- **Image filtering** is a process to modify or enhance image properties and/or to extract information such as edges, corners, and blobs.
  - Its process usually performed using **2D convolution** with a specific **kernel** (a.k.a. mask, operator)
  - Design factors
    - Kernel coefficients
    - Kernel size
    - Boundary handling (e.g. mirror, zero padding, …)
  - Note) Some 2D convolutions can be replaced by two 1D convolutions (so-called a separable filter).
    - Their time complexity is reduced from $O(MNmn)$ to $O(MN(m+n))$.

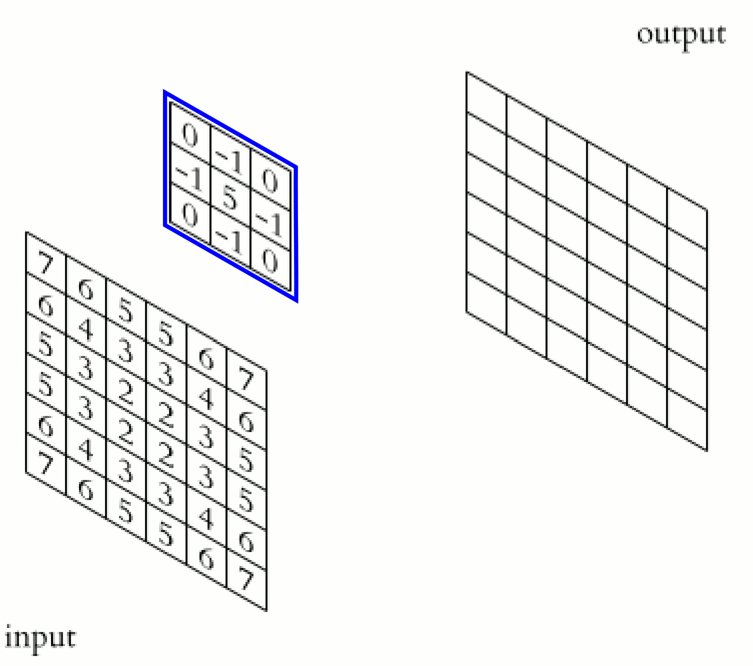


Image: Wikipedia

Image Filtering: Smoothing → For Noise/Detail Reduction

- **Box blur** (a.k.a. averaging smoothing)
  - Each pixel has the (equally weighted) average value of its neighbor pixels.
    - Note) Due to the central limit theorem, repeated box blur will result the same effect of Gaussian blur.
  - e.g. 3-by-3 box kernel (~~radius: 1,~~ kernel size: 3)
    - $G = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix}\begin{bmatrix} 1 & \mathbf{1} & 1 \end{bmatrix}$
  - Note) The sum of all coefficients is **one**.

- **Gaussian blur** (a.k.a. Gaussian smoothing)
  - Each pixel has the Gaussian-weighted average value of its neighbor pixels.
  - e.g. 3-by-3 Gaussian kernel (kernel size: 3, sigma: 0.85)
    - $G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{4} & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{16}\begin{bmatrix} 1 \\ \mathbf{2} \\ 1 \end{bmatrix}\begin{bmatrix} 1 & \mathbf{2} & 1 \end{bmatrix}$
  - Note) Gaussian kernel and blur in OpenCV API
    - `cv.getGaussianKernel(ksize, sigma[, ktype]) → kernel1D`
    - `cv.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst`

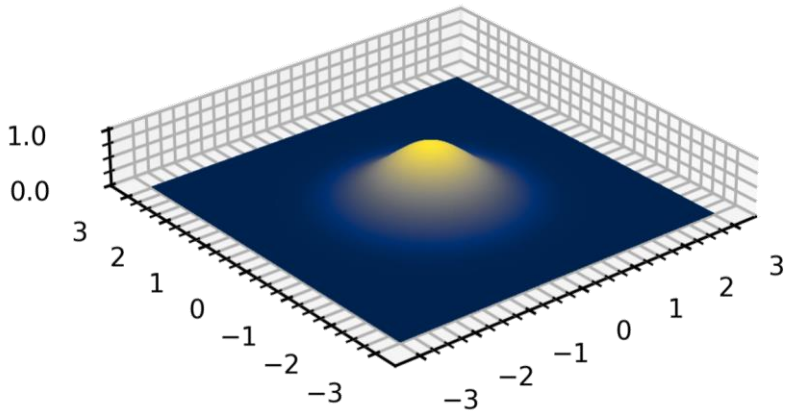


Image Filtering: Smoothing

- Example) Image filtering

```python
# Define kernels
kernel_table = [
    {'name': 'Box 3x3',          'kernel': np.ones((3, 3)) / 9},    # Alternative: cv.boxFilter(), cv.blur()
    {'name': 'Gaussian 3x3',     'kernel': np.array([[1, 2, 1],     # Alternative: cv.GaussianBlur()
                                                     [2, 4, 2],
                                                     [1, 2, 1]]) / 16},
    {'name': 'Box 5x5',          'kernel': np.ones((5, 5)) / 25},
    {'name': 'Gaussian 5x5',     'kernel': np.array([[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]]) / 256},

    ...
]
```

Image Filtering: Smoothing

- Example) Image filtering

```
img_list = ['data/lena.tif', ...]

# Initialize control parameters
kernel_select = 0
img_select = 0

while True:
    # Read the given image as gray scale
    img = cv.imread(img_list[img_select], cv.IMREAD_GRAYSCALE)

    # Apply convolution to the image with the given 'kernel'
    name, kernel = kernel_table[kernel_select].values() # Make (short) alias
    result = cv.filter2D(img, -1, kernel)               # Note) dtype: np.uint8 (range: [0, 255])

    # Show the image and its filtered result
    merge = np.hstack((img, result))
    cv.imshow('Image Filtering: Original | Filtered', merge)

    # Process the key event
    key = cv.waitKey()
    if key == 27: # ESC
        break
    elif key == ord('+') or key == ord('='):
        kernel_select = (kernel_select + 1) % len(kernel_table)
    elif key == ord('-') or key == ord('_'):
        kernel_select = (kernel_select - 1) % len(kernel_table)
    elif key == ord('\t'):
        img_select = (img_select + 1) % len(img_list)
```

Image Filtering: Smoothing

- Example) Image filtering: **Original | Box Blur (5x5) | Gaussian Blur (5x5)**

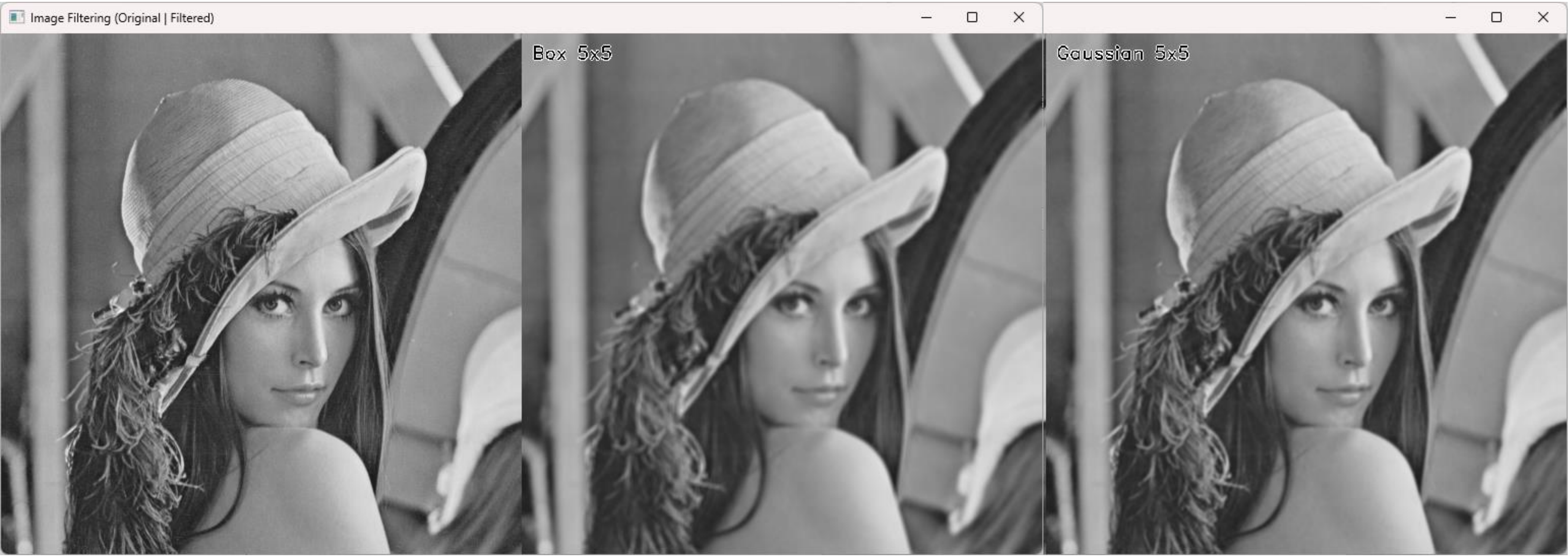


Image Filtering: **Smoothing**

- **Median filter**: A non-linear noise reduction filter using median instead of average
  - Process) Similar to 2D convolution process,
    - ~~Each pixel has the average value of its neighbor pixels in the kernel.~~
    - Each pixel has the median value of its neighbor pixels in the kernel
  - Advantages
    - Effective to remove impulse noise (e.g. salt-and-pepper noise)
      - The median is less sensitive to outliers than average.
        - e.g. 1, 3, 5, 7**0**, 9 – Average: 17.6 / Median: 5
    - Image blurring, but preserving edges more than averaging
  - Disadvantage
    - Slower than averaging ← the median operation
      - Of course, the median operation is not separable.
  - Note) Median filter in OpenCV API
    - `cv.medianBlur(src, ksize[, dst]) → dst`

Image Filtering: Smoothing

- Example) Median filter

```python
img_list = [..., 'data/black_circle.png', ...]

# Initialize control parameters
kernel_size = 5
img_select = -1

while True:
    # Read the given image
    img = cv.imread(img_list[img_select])

    # Apply the median filter
    result = cv.medianBlur(img, kernel_size)

    # Show all images
    merge = np.hstack((img, result))
    cv.imshow('Medial Filter: Original | Result', merge)

    # Process the key event
    key = cv.waitKey()
    if key == 27: # ESC
        break
    elif key == ord('+') or key == ord('='):
        kernel_size = kernel_size + 2
    elif key == ord('-') or key == ord('_'):
        kernel_size = max(kernel_size - 2, 3)
    elif key == ord('\t'):
```

Image Filtering: Image Gradient and Robert Cross Kernels

- **Image gradient kernels**: X- and Y-directional changes (1st derivative) in an image
  - X-directional change: $\mathrm{I}(x+1, y) - \mathrm{I}(x, y)$
    - $D_X = \begin{bmatrix} \mathbf{-1} & 1 \end{bmatrix}$
  - Y-directional change: $\mathrm{I}(x, y+1) - \mathrm{I}(x, y)$
    - $D_Y = \begin{bmatrix} \mathbf{-1} \\ 1 \end{bmatrix}$
  - Note) The sum of all coefficients is **zero**.

- **Robert cross kernels**: Diagonal directional changes in an image
  - The 1st diagonal directional changes: $\mathrm{I}(x+1, y+1) - \mathrm{I}(x, y)$
    - $D_D = \begin{bmatrix} \mathbf{-1} & 0 \\ 0 & 1 \end{bmatrix}$
  - The 2nd diagonal directional changes: $\mathrm{I}(x+1, y) - \mathrm{I}(x, y+1)$
    - $D_U = \begin{bmatrix} \mathbf{0} & 1 \\ -1 & 0 \end{bmatrix}$
  - Note) The sum of all coefficients is **zero**.

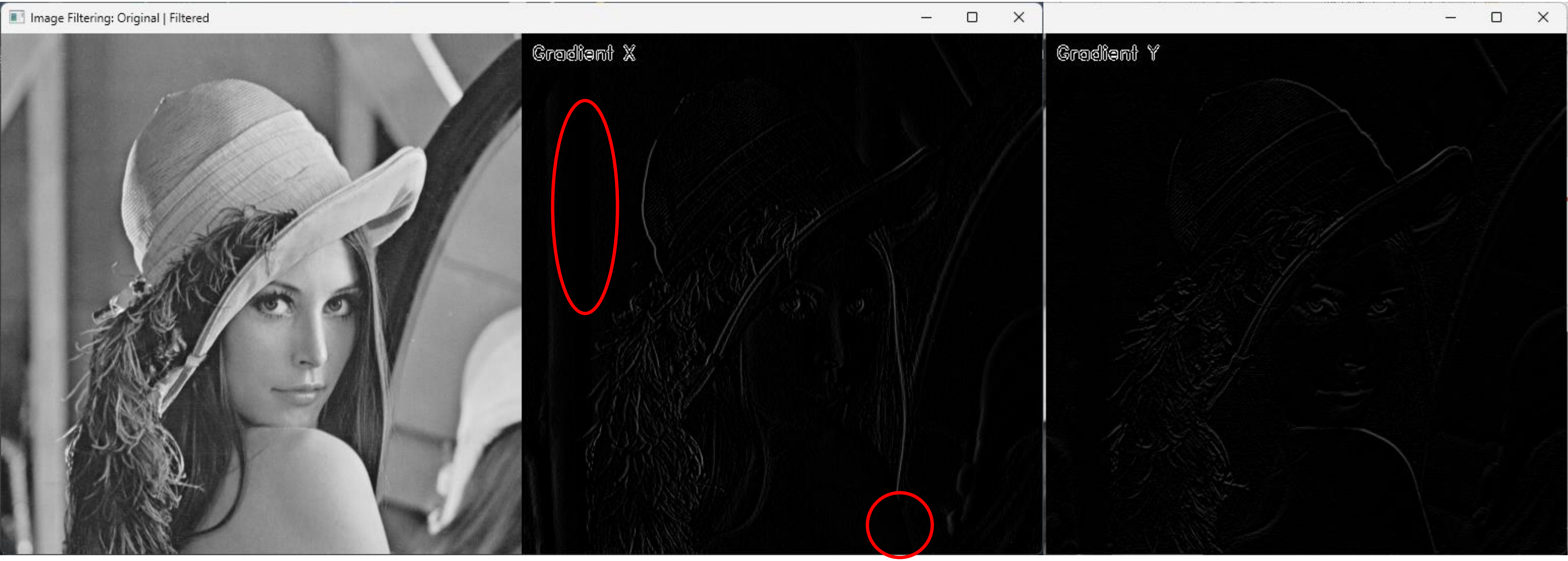
Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Image filtering (with `'data/lena.tif'`)

```
# Define kernels
kernel_table = [
    {'name': 'Gradient X',       'kernel': np.array([[-1,  1]])},
    {'name': 'Robert DownRight','kernel': np.array([[-1,  0],
                                                     [ 0,  1]])},
    {'name': 'Gradient Y',       'kernel': np.array([[-1], [1]])},
    {'name': 'Robert UpRight',  'kernel': np.array([[ 0,  1],
                                                     [-1,  0]])},
    ...
]

img_list = ['data/lena.tif', ...]

# Initialize control parameters
kernel_select = 0
img_select = 0

while True:
    # Read the given image as gray scale
    img = cv.imread(img_list[img_select], cv.IMREAD_GRAYSCALE)

    # Apply convolution to the image with the given 'kernel'
    name, kernel = kernel_table[kernel_select].values() # Make (short) alias
    result = cv.filter2D(img, -1, kernel)               # Note) dtype: np.uint8 (range: [0, 255])

    # Show the image and its filtered result
```

Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Gradients (with `'data/lena.tif'`)



Where is gradient?

Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Gradients (with `'data/lena.tif'`)

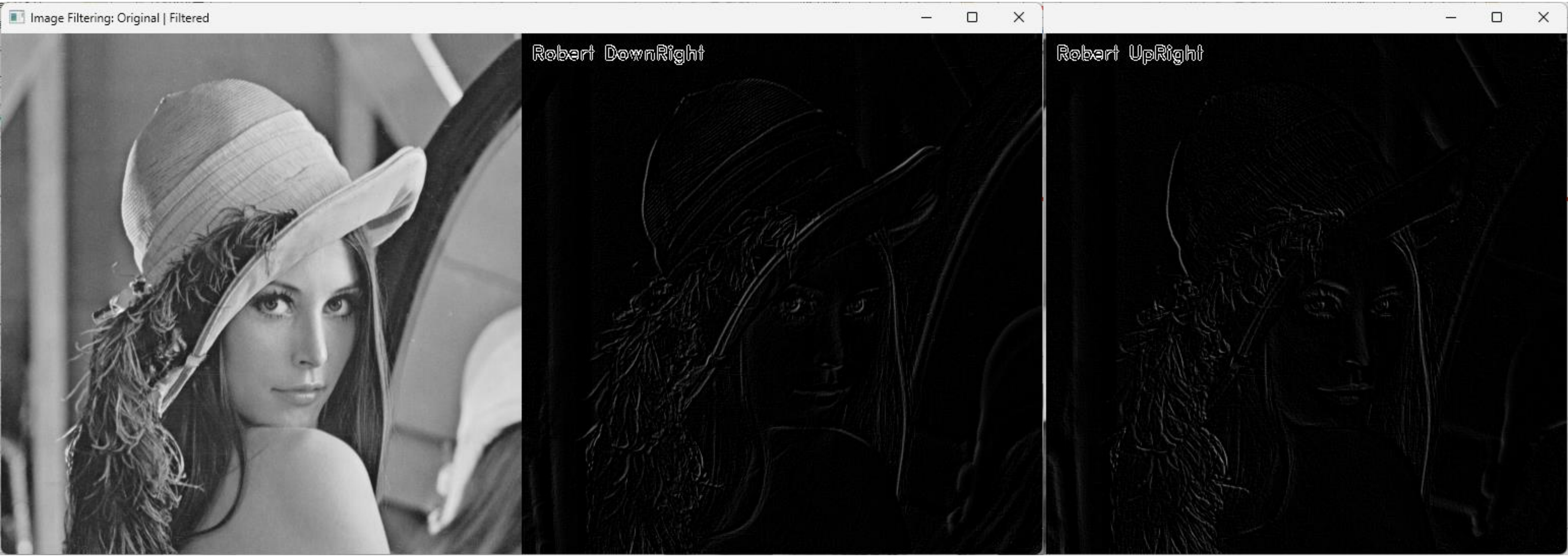


Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Gradients (with `'data/black_circle.png'`)

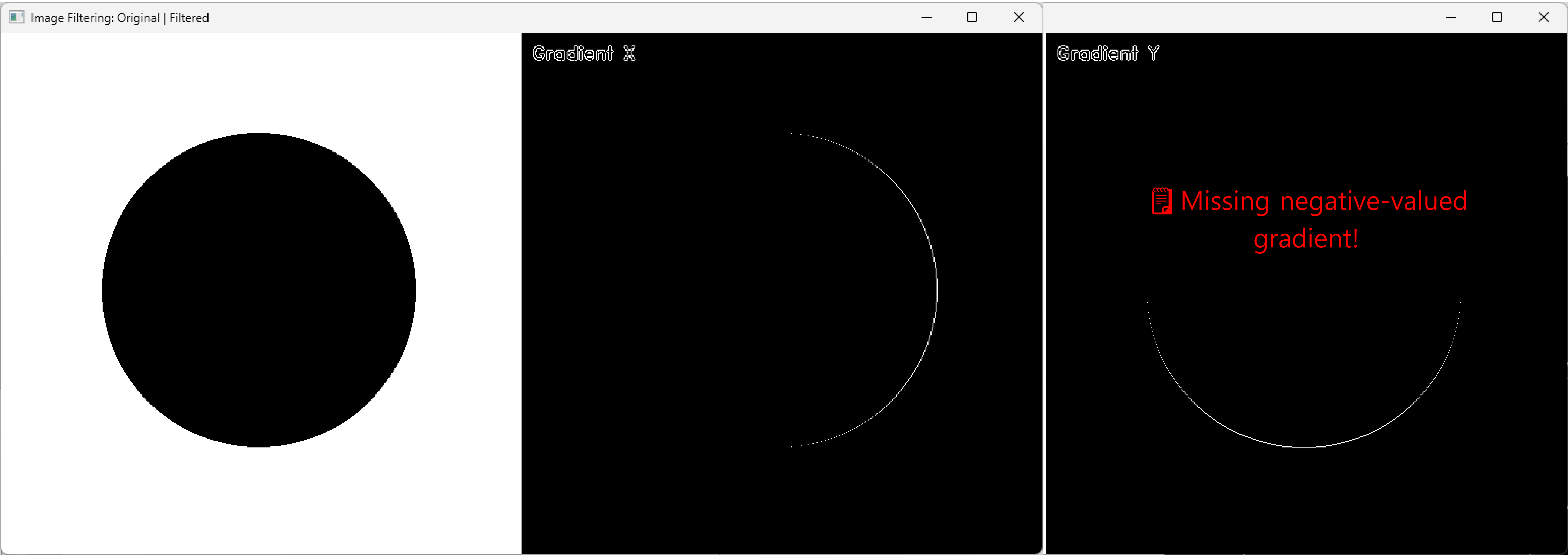


Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Gradients (with `'data/black_circle.png'`)

```
# Define kernels
kernel_table = [
    {'name': 'Gradient X',       'kernel': np.array([[-1,  1]])},
    {'name': 'Robert DownRight','kernel': np.array([[-1,  0],
                                                    [ 0,  1]])},
    ...
]

img_list = ['data/lena.tif', ...]

# Initialize control parameters
kernel_select = 0
img_select = 0

while True:
    # Read the given image as gray scale
    img = cv.imread(img_list[img_select], cv.IMREAD_GRAYSCALE)

    # Apply convolution to the image with the given 'kernel'
    name, kernel = kernel_table[kernel_select].values() # Make (short) alias
    result = cv.filter2D(img, cv.CV_64F, kernel) # Note) dtype: np.float64
    result = cv.convertScaleAbs(result)          # Convert 'np.float64' to 'np.uint8' with saturation

    # Show the image and its filtered result
    merge = np.hstack((img, result))
    cv.imshow('Image Filtering: Original | Filtered', merge)
```

Be careful when your kernel coefficients derive values out of [0, 255].

Image Filtering: Image Gradient and Robert Cross Kernels

- Example) Gradients (with `'data/black_circle.png'`)

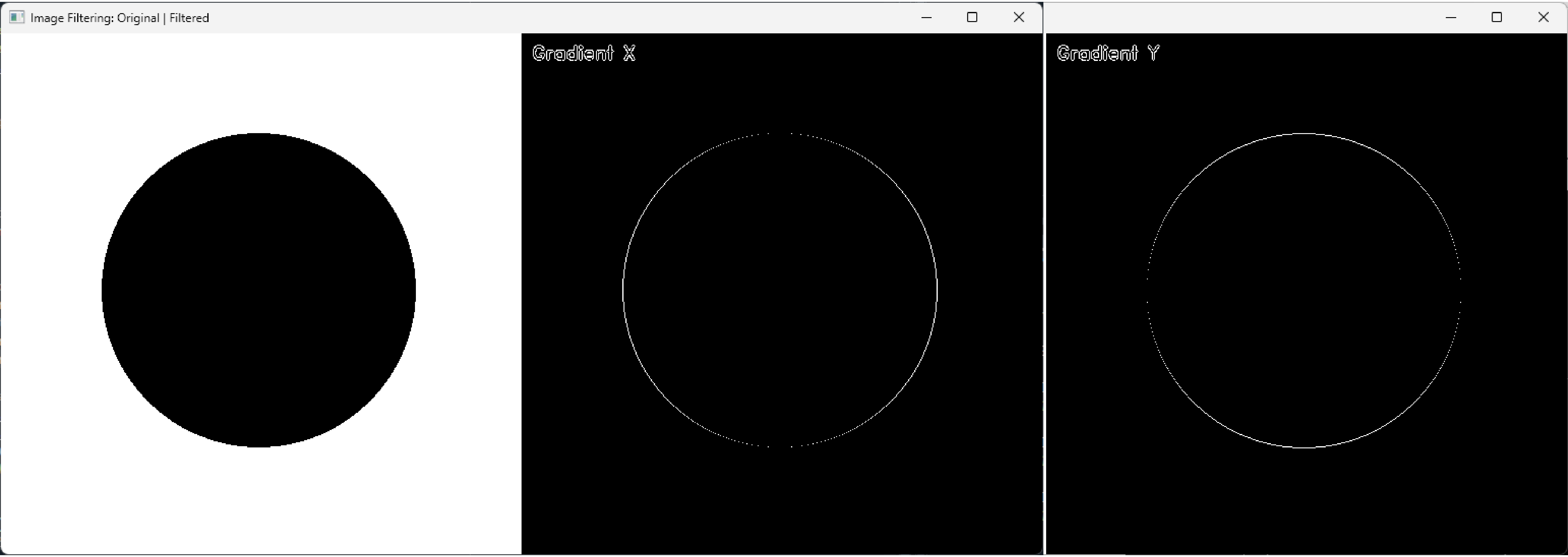


Image Filtering: Image Gradient and Robert Cross Kernels

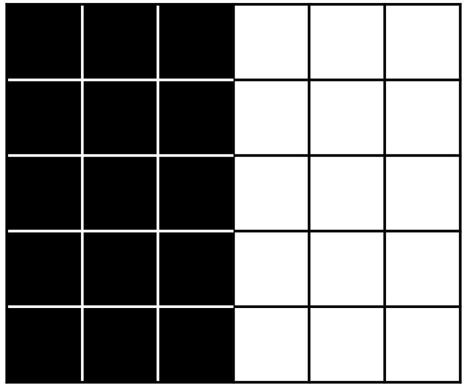
- Example) Gradients (with `'data/lena.png'`)

Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- **Prewitt kernels**: X- and Y-directional edges in an image
  - X-directional edges
    - $G_X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix} \begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix}$
  - Y-directional edges
    - $G_Y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{1} & 1 \end{bmatrix}$
  - Note) The sum of all coefficients is **zero**.

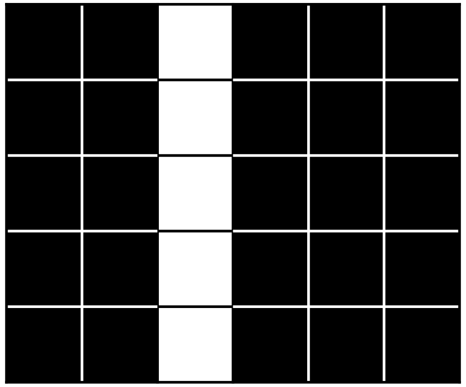
  - Discussion) Why 3-by-3 kernel for edges? (A 1-by-2 gradient kernels also can extract edges.)

Input image



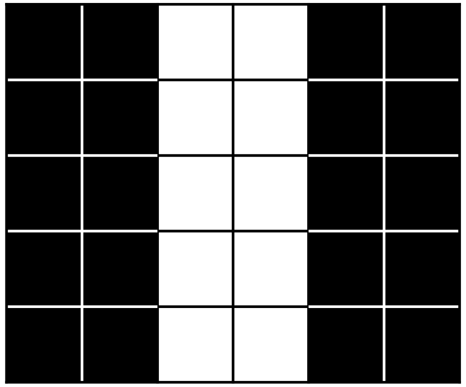
→

Gradient kernel



vs.

Prewitt kernel



Central difference operator

$\sim \dfrac{\mathrm{I}(x+1, y) - \mathrm{I}(x-1, y)}{2}$

Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- **Sobel kernels**: X- and Y-directional edges in an image with weighted smoothing
  - X-directional edges
    - $G_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{2} \\ 1 \end{bmatrix} \begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix}$
  - Y-directional edges
    - $G_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{2} & 1 \end{bmatrix}$

- **Scharr kernels**: Sobel kernels with more optimized coefficients (in the view of rotational symmetry)
  - X-directional edges
    - $G_X = \begin{bmatrix} -3 & 0 & 3 \\ -10 & \mathbf{0} & 10 \\ -3 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 3 \\ \mathbf{10} \\ 3 \end{bmatrix} \begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix}$
  - Y-directional edges
    - $G_Y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & \mathbf{0} & 0 \\ 3 & 10 & 3 \end{bmatrix} = \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix} \begin{bmatrix} 3 & \mathbf{10} & 3 \end{bmatrix}$

Note) 5-by-5 Scharr kernel (X-direction)

$$G_X = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & \mathbf{0} & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Example) X-directional edge detection: **Prewitt** | **Sobel** | **Scharr**

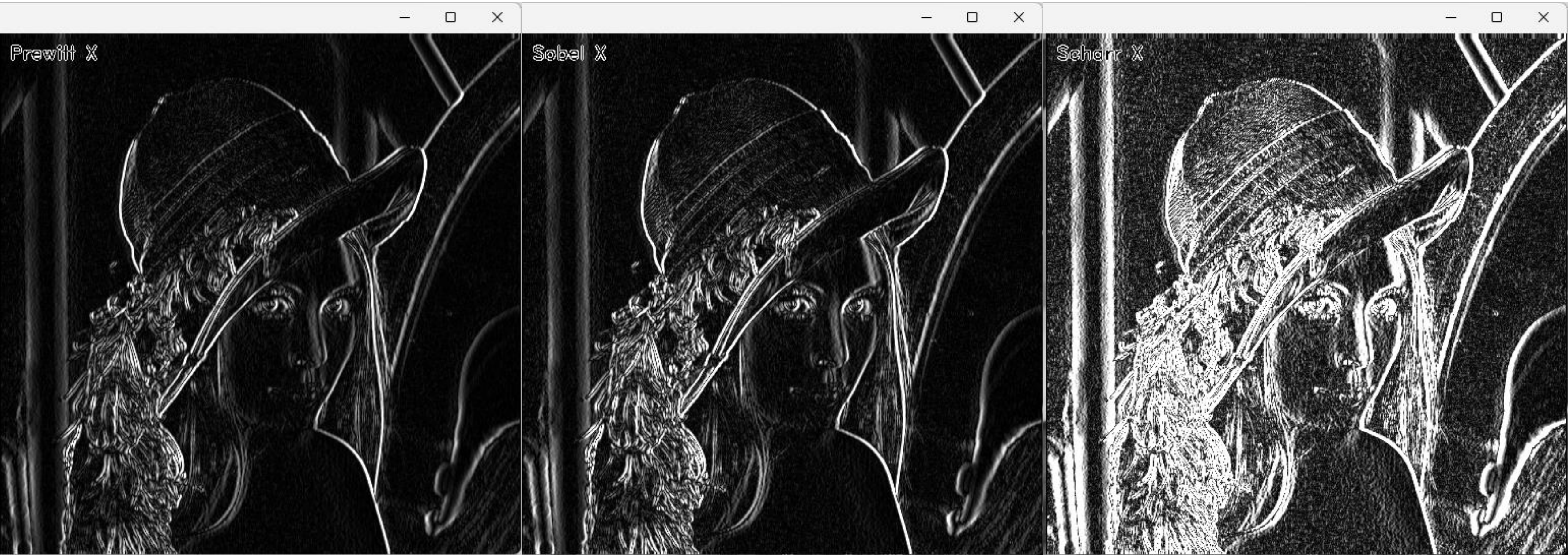
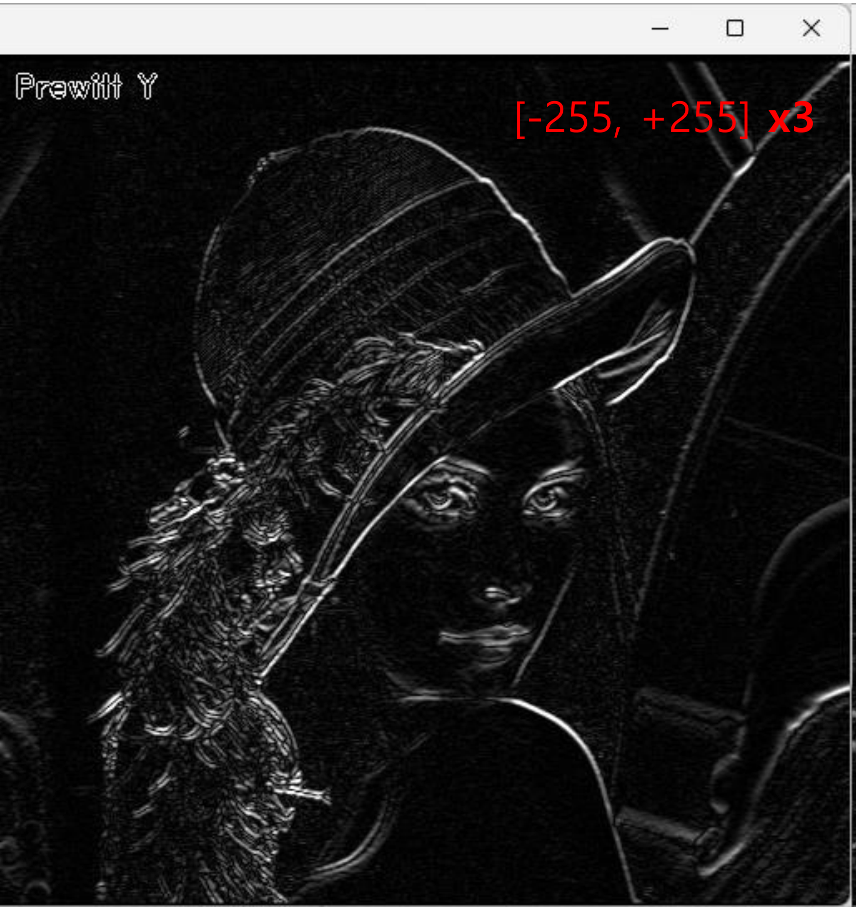
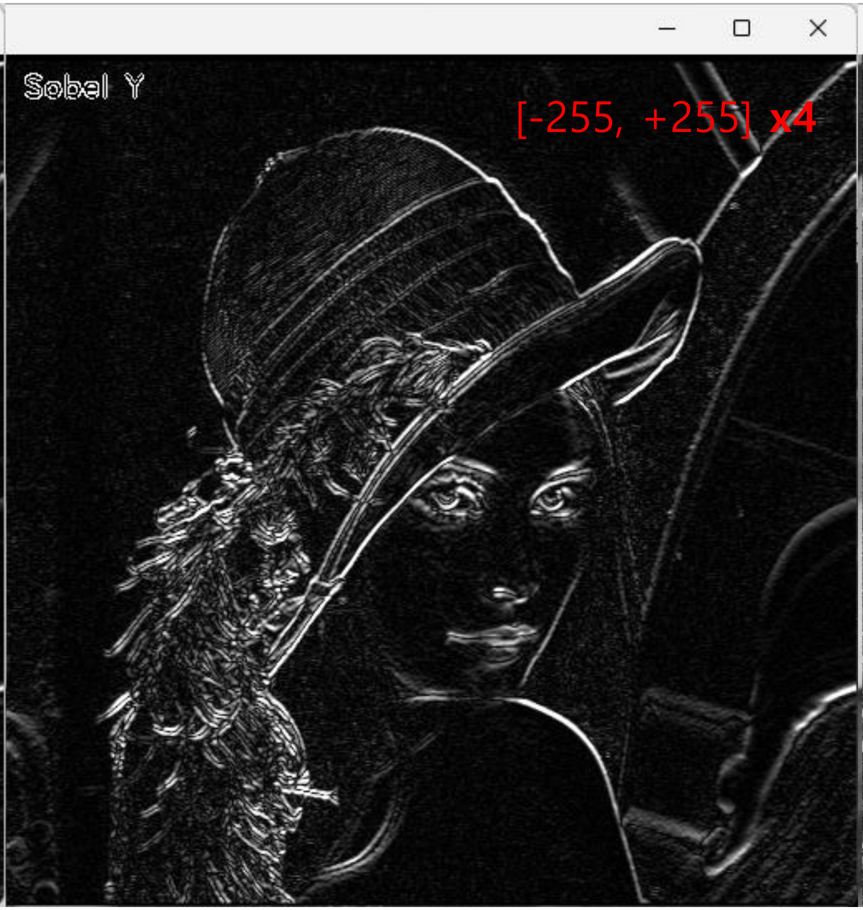


Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Example) Y-directional edge detection: **Prewitt** | **Sobel** | **Scharr**





Why stronger edge response?
Bigger kernel coefficients

Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Example) Sobel edge detection

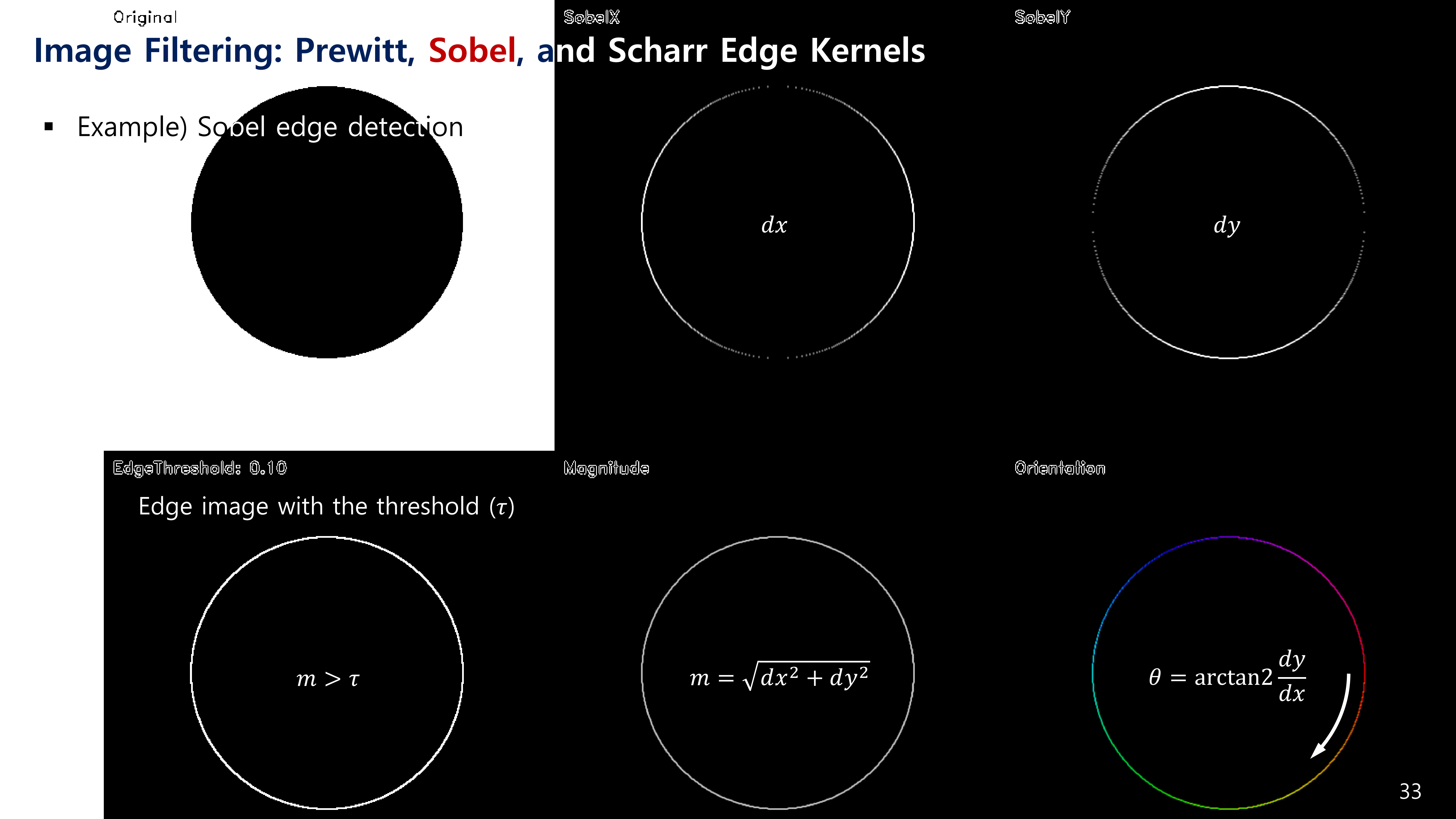


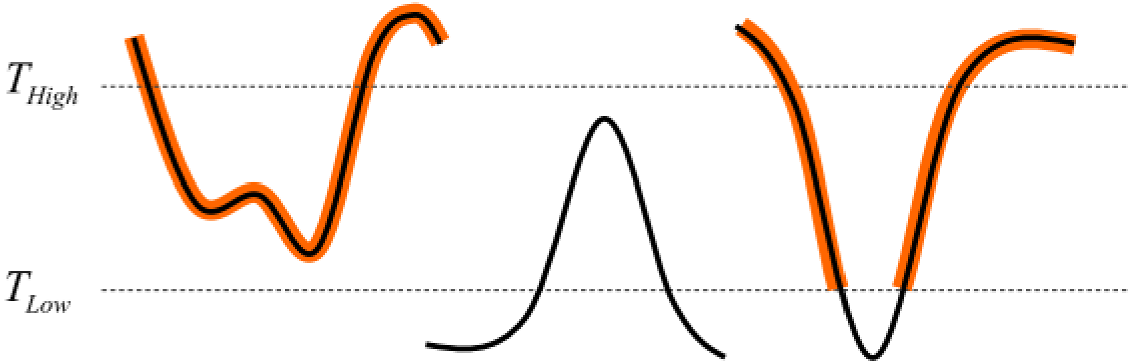
Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Example) Sobel edge detection

```
img_list = [..., 'data/black_circle.png', ...]

# Initialize control parameters
edge_threshold = 0.1
img_select = 3

while True:
    # Read the given image as gray scale
    img = cv.imread(img_list[img_select], cv.IMREAD_GRAYSCALE)

    # Extract edges using two-directional Sobel responses
    #  and normalize their values within [0, 1] (Note: 1020 derived from 255 * (1+2+1))
    dx   = cv.Sobel(img, cv.CV_64F, 1, 0) / 1020 # Sobel x-directional response
    dy   = cv.Sobel(img, cv.CV_64F, 0, 1) / 1020 # Sobel y-directional response
    mag  = np.sqrt(dx*dx + dy*dy) / np.sqrt(2)   # Sobel magnitude
    ori  = np.arctan2(dy, dx)                    # Sobel orientation
    edge = mag > edge_threshold # Alternative) cv.threshold(), cv.adaptiveThreshold()

    # Prepare the orientation image as the BGR color
    ori[ori < 0] = ori[ori < 0] + 2*np.pi        # Convert [-np.pi, np.pi) to [0, 2*np.pi)
    ori_hsv = np.dstack((ori / (2*np.pi) * 180,  # HSV color - Hue channel
                         np.full_like(ori, 255), # HSV color - Saturation channel
                         mag * 255))             # HSV color - Value channel
    ori_bgr = cv.cvtColor(ori_hsv.astype(np.uint8), cv.COLOR_HSV2BGR)

    # Prepare the original, Sobel X/Y, magnitude, and edge images as the BGR color
```

Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Canny edge detector: A popular multi-stage edge detector
  - Process
    1. Gaussian smoothing (for noise reduction)
    2. Sobel kernel (for edge detection)
    3. Sobel magnitude thresholding (for edge candidate selection)
       - Double thresholding (strong/weak edge candidates)
    4. Edge tracking by hysteresis (Note: Useful for abnormal toggling)



  - Advantages: Accurate edges and suitable to various images
  - Disadvantages: Heavy computation and difficult to adjust parameters

Image: Jinsol Kim's Blog

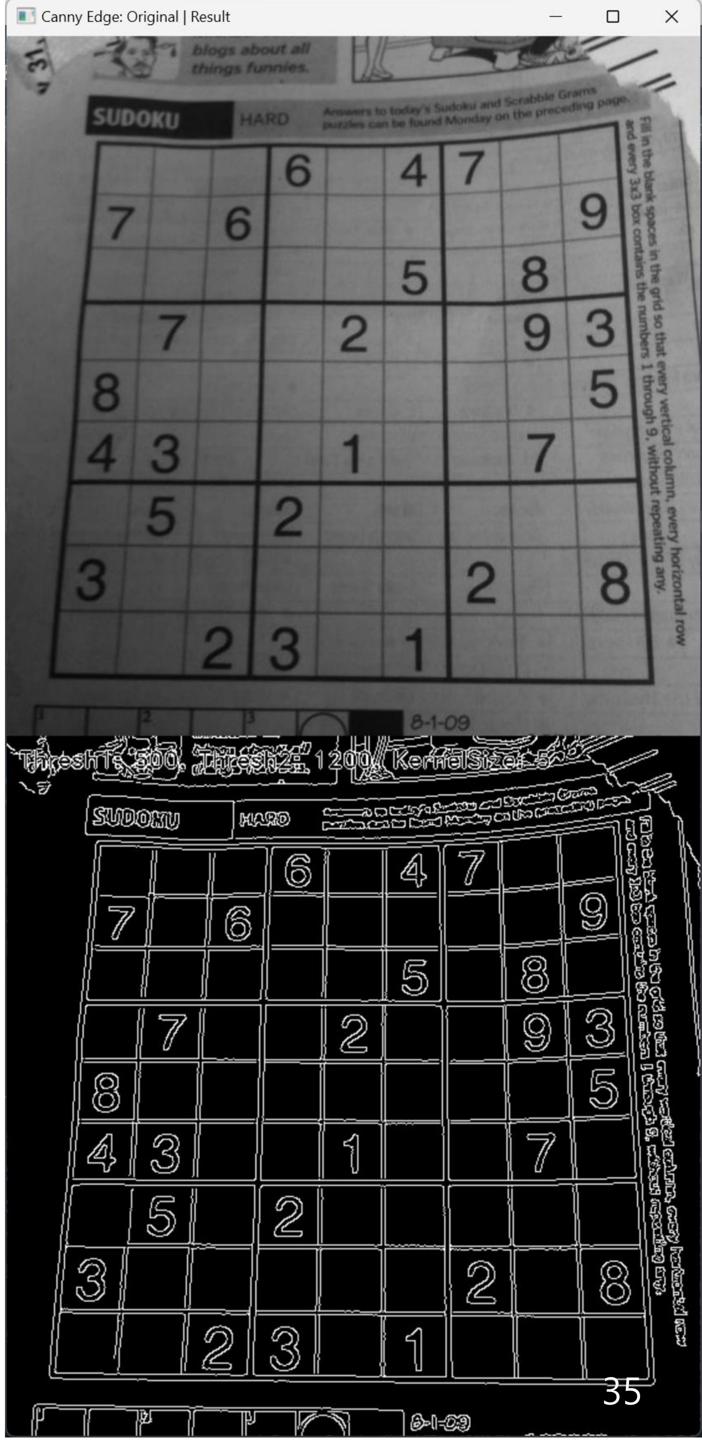


Image Filtering: Prewitt, Sobel, and Scharr Edge Kernels

- Example) Canny edge detection

```python
import cv2 as cv
import numpy as np

img_list = [..., 'data/sudoku.png']

# Initialize control parameters
threshold1 = 500
threshold2 = 1200
aperture_size = 5
img_select = -1

while True:
    # Read the given image
    img = cv.imread(img_list[img_select], cv.IMREAD_GRAYSCALE)
    assert img is not None, 'Cannot read the given image, ' + img_list[img_select]

    # Get the Canny edge image
    edge = cv.Canny(img, threshold1, threshold2, apertureSize=aperture_size)

    # Show all images
    ...
    merge = np.hstack((img, edge))
    cv.imshow('Canny Edge: Original | Result', merge)

    # Process the key event
    key = cv.waitKey()
    if key == 27: # ESC
```

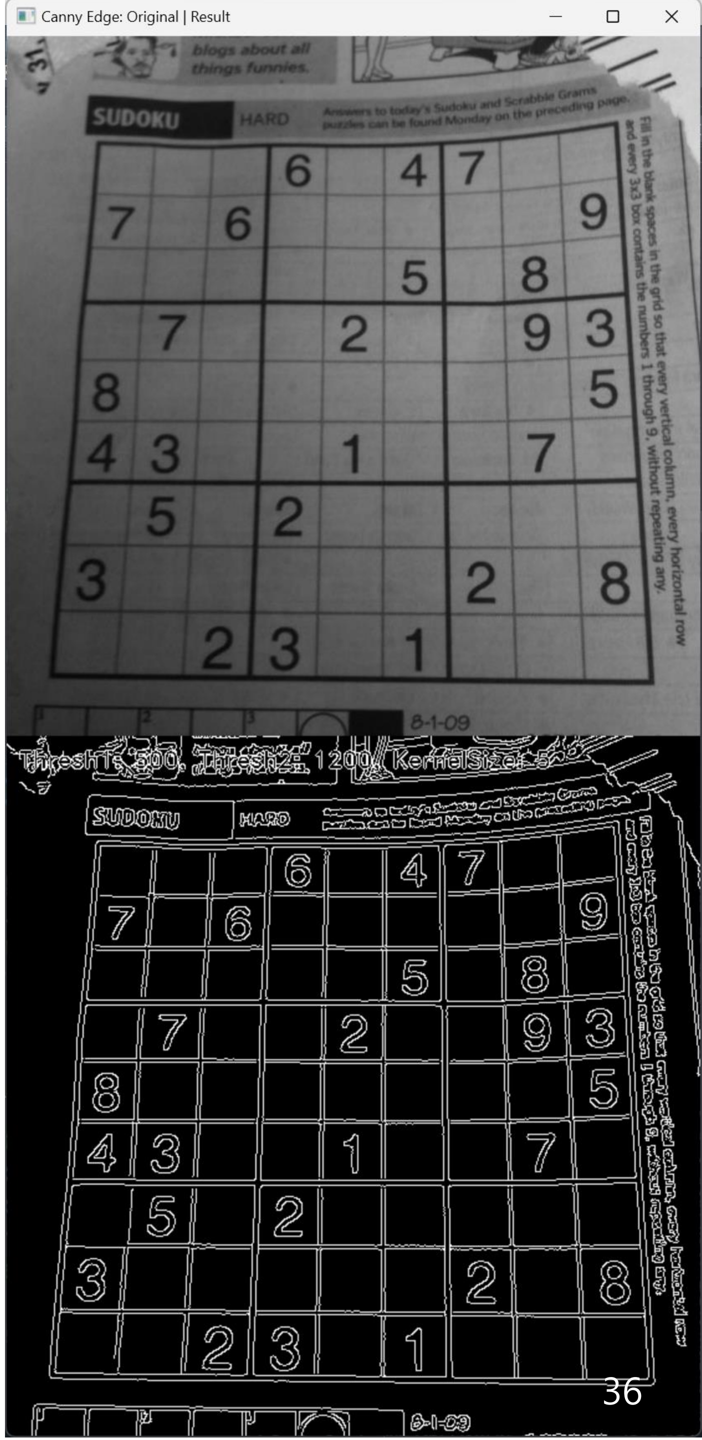


Image Filtering: Laplacian and Sharpening

- **Laplacian operator**: The sum of the 2nd-order derivatives (more mathematically divergence of gradient)
  - Note) 1D example: 1st-order derivative $D_X = [-1 \quad 1] \rightarrow$ 2nd-order derivative $D_{XX} = [-1 \quad \mathbf{2} \quad -1]$
  - 2D kernel: Not separable
    - $D_{XY} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & \mathbf{4} & -1 \\ 0 & -1 & 0 \end{bmatrix}$
    - Effect: Finding edge boundaries
  - Note) The sum of all coefficients is **zero**.

- **Sharpening kernel**: Highlighting edge boundaries on an image
  - 2D kernel
    - $G = \begin{bmatrix} 0 & -1 & 0 \\ -1 & \mathbf{5} & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ -1 & \mathbf{4} & -1 \\ 0 & -1 & 0 \end{bmatrix}$
  - Note) The sum of all coefficients is **one**.

Original function

1st derivative
(edge)

2nd derivative
(edge boundary)

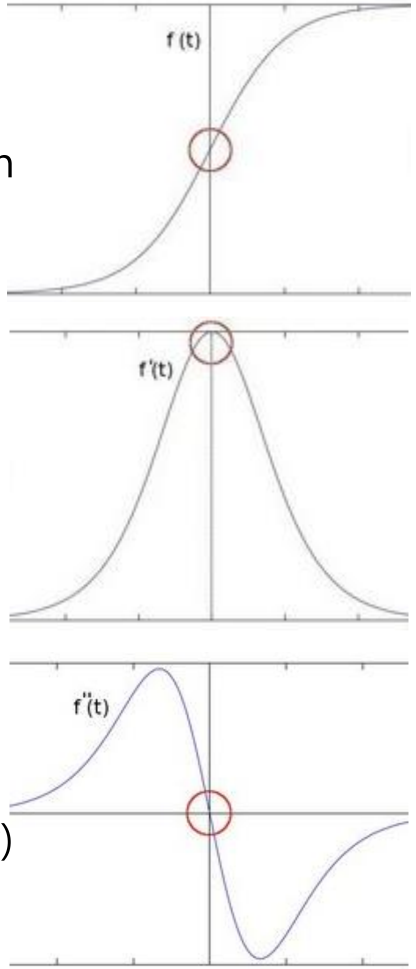


Image: OpenCV Tutorial

Image Filtering: Laplacian and Sharpening

- Example) Image sharpening: **Original** | **Laplacian** | **Sharpening**

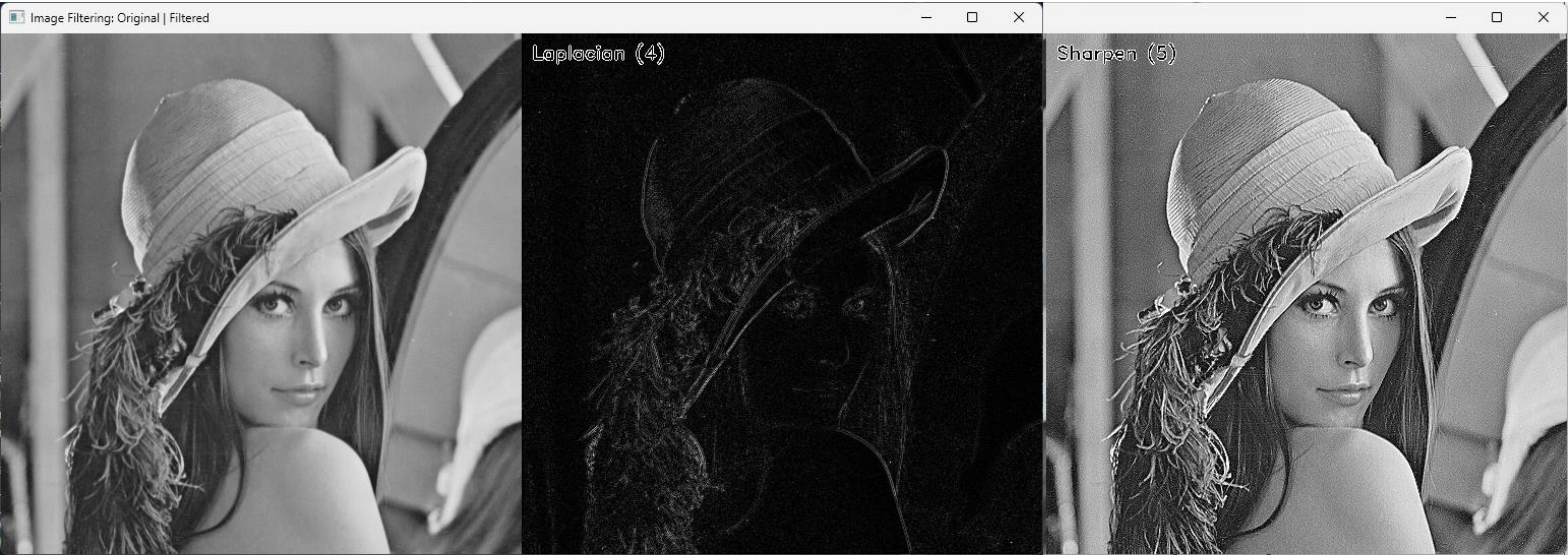
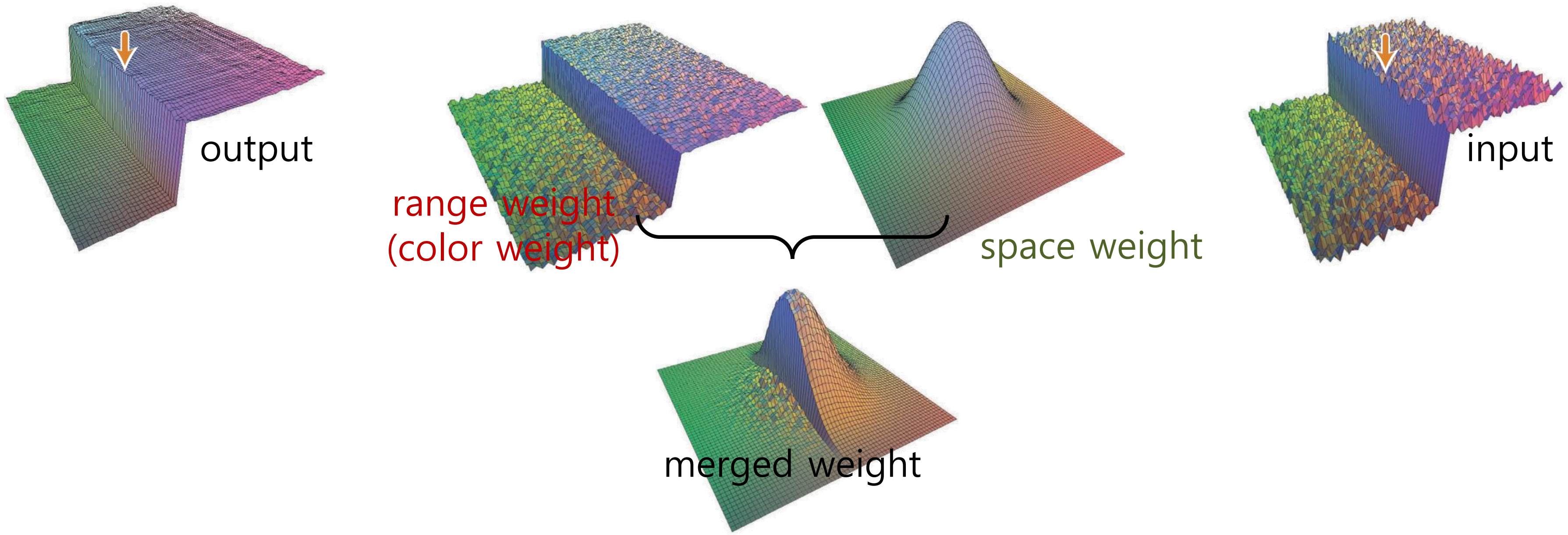


Image Filtering: Bilateral Filter

- **Bilateral filter**: A non-linear, edge-preserving, and smoothing filter for images
  - Process)

$$I^{BF}(\mathbf{x}) = \frac{1}{W_p} \sum_{\mathbf{x}_i \in \Omega} f_r(\|I(\mathbf{x}_i) - I(\mathbf{x})\|) \, g_s(\|\mathbf{x}_i - \mathbf{x}\|) \, I(\mathbf{x}_i)$$



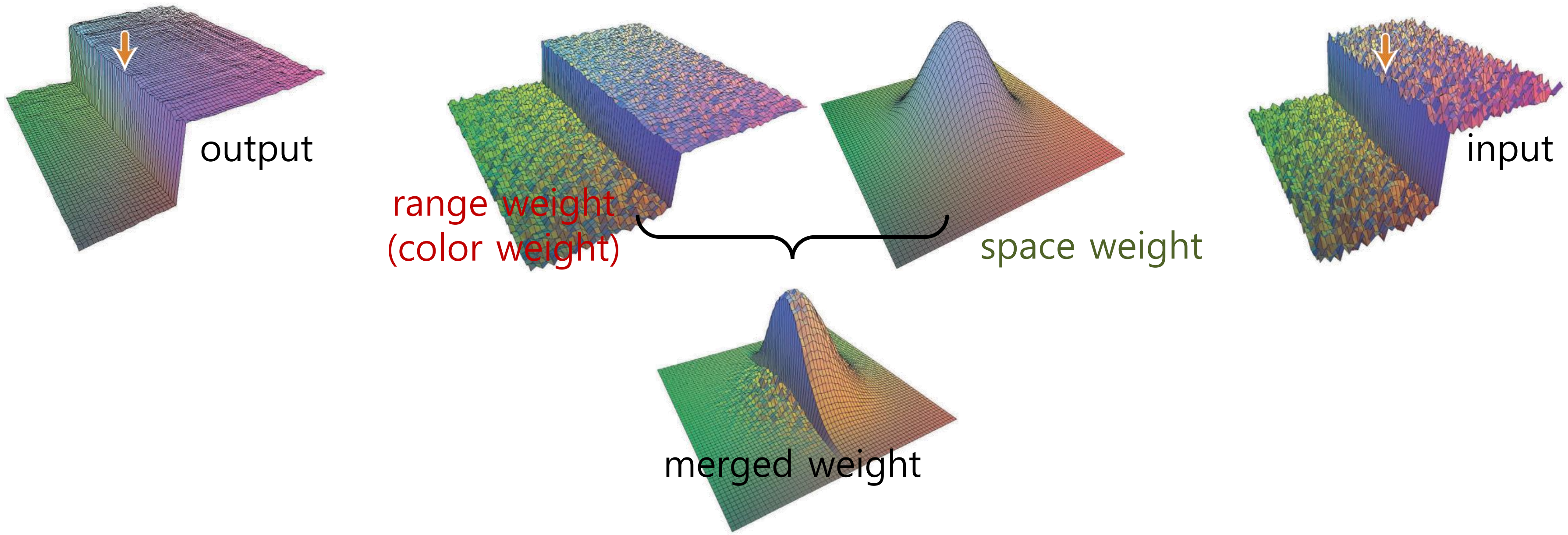
  - Note) Bilateral filter in OpenCV API
    - `cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) → dst`
      - d: Diameter of kernel (kernel size) / If it is non-positive, it is computed from `sigmaSpace`.

Image: Paris et al., "Bilateral Filtering: Theory and Applications", Foundations and Trends in CG and CV, Vol. 4, No. 1, 2008

Image Filtering: Bilateral Filter

- **Bilateral filter**: A non-linear, edge-preserving, and smoothing filter for images
  - Process) When f_r and g_s are Gaussian,

$$I^{BF}(i,j) = \frac{\sum_{k,l} w(i,j,k,l)\, I(k,l)}{\sum_{k,l} w(i,j,k,l)} \quad where \quad w(i,j,k,l) = \exp\left(-\frac{\|I(i,j) - I(k,l)\|^2}{2\sigma_c^2} - \frac{(i-k)^2 + (j-l)^2}{2\sigma_s^2}\right)$$



  - Note) Bilateral filter in OpenCV API
    - `cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) → dst`
      - d: Diameter of kernel (kernel size) / If it is non-positive, it is computed from `sigmaSpace`.

Image: Paris et al., "Bilateral Filtering: Theory and Applications", Foundations and Trends in CG and CV, Vol. 4, No. 1, 2008

Image Filtering: Bilateral Filter

- Example) Bilateral filter

```python
img_list = ['data/lena.tif', ...]

# Initialize control parameters
kernel_size = 9
sigma_color = 150
sigma_space = 2.4
n_iterations = 1
img_select = 0

while True:
    # Read the given image
    img = cv.imread(img_list[img_select])
    assert img is not None, 'Cannot read the given image, ' + img_list[img_select]

    # Apply the bilateral filter iteratively
    result = img.copy()
    for itr in range(n_iterations):
        result = cv.bilateralFilter(result, kernel_size, sigma_color, sigma_space)

    # Show all images
    merge = np.hstack((img, result))
    cv.imshow('Bilateral Filter: Original | Result', merge)

    # Process the key event
    key = cv.waitKey()
    if key == 27: # ESC
```

Morphological Operations

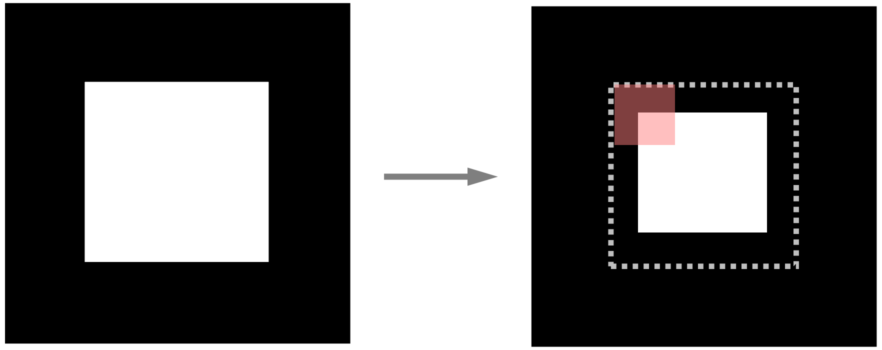
- **Morphological operations** are image processing techniques to manipulate the shape or structure of objects in an image, usually a binary image (e.g. 1: object, 0: otherwise).
  - It is based on mathematical morphology developed by Georges Matheron and Jean Serra in 1964.
  - It has two fundamental operations (erosion and dilation) and their combinations makes various operations.
  - Its operations works similar to *image filtering* with a kernel without coefficients (a.k.a. sliding window).

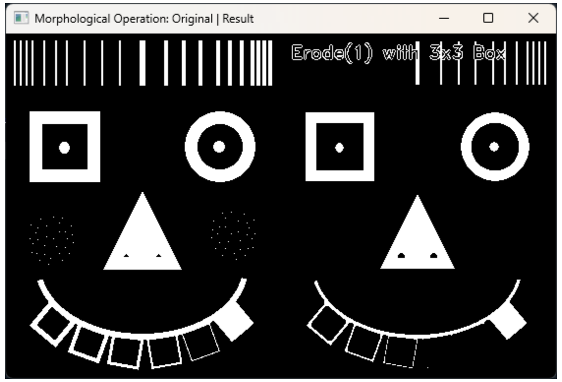
- **Erosion** (침식 in Korean): An operation for reducing the shape
  - $I'(x,y) = \begin{cases} 1 & \text{if } \textbf{all} \text{ pixels in the window is } 1 \\ 0 & \text{otherwise} \end{cases}$ ~ logical conjunction (intersection)

- **Dilation** (팽창 in Korean): An operation for expanding the shape
  - $I'(x,y) = \begin{cases} 1 & \text{if } \textbf{any} \text{ pixels in the window is } 1 \\ 0 & \text{otherwise} \end{cases}$ ~ logical disjunction (union)

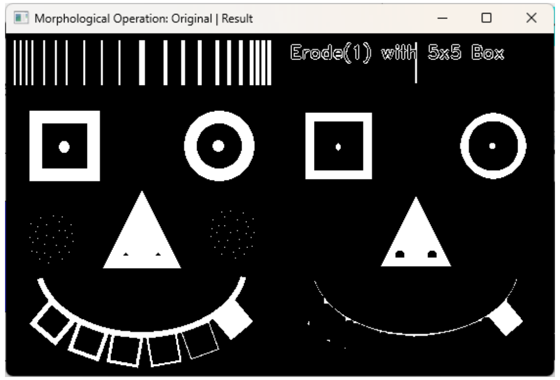
Morphological Operations

- **Erosion** (침식 in Korean): An operation for reducing the shape
  - $I'(x,y) = \begin{cases} 1 & \text{if } \textbf{all} \text{ pixels in the window is 1} \\ 0 & \text{otherwise} \end{cases}$ ~ logical conjunction (intersection)



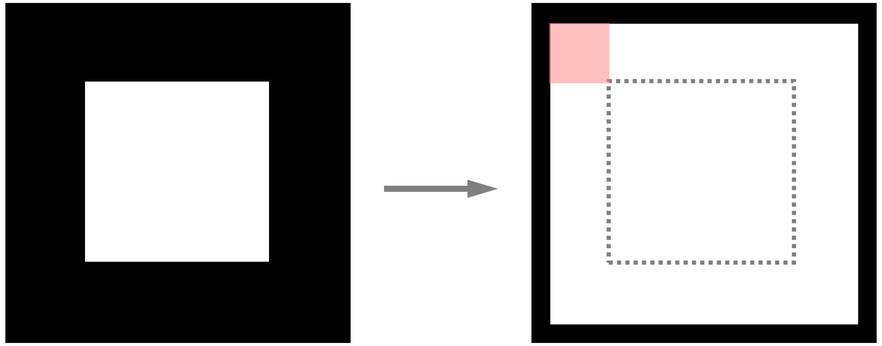


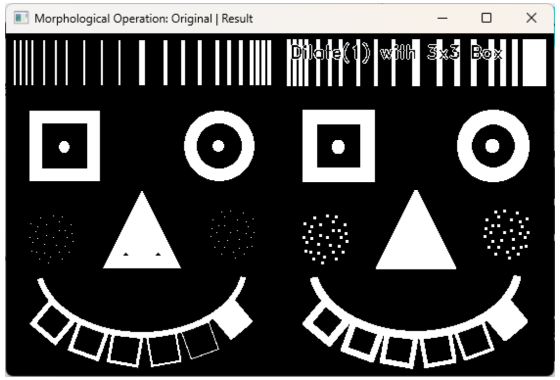
3x3 Box



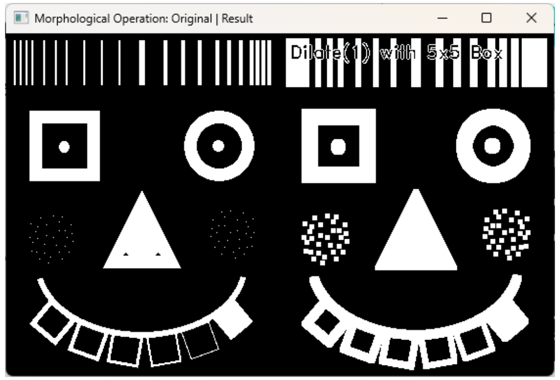
5x5 Box

- **Dilation** (팽창 in Korean): An operation for expanding the shape
  - $I'(x,y) = \begin{cases} 1 & \text{if } \textbf{any} \text{ pixels in the window is 1} \\ 0 & \text{otherwise} \end{cases}$ ~ logical disjunction (union)





3x3 Box



5x5 Box

Morphological Operations

- Erosion (`cv.MORPH_ERODE`) : `erode(src)`
  - Applications: Thinning lines, removing small dots
- Dilation (`cv.MORPH_DILATE`) : `dilate(src)`
  - Applications: Thickening lines
- Opening (`cv.MORPH_OPEN`) : `dilate(erode(src))`
  - Applications: Growing holes (while keeping object size)
- Closing (`cv.MORPH_CLOSE`) : `erode(dilate(src))`
  - Applications: Shrinking or blocking holes

- Gradient (`cv.MORPH_GRADIENT`) : `dilate(src) - erode(src)`
- Tophat (`cv.MORPH_TOPHAT`) : `src - open(src)`
- Blackhat (`cv.MORPH_BLACKHAT`) : `close(src) - src`
- …

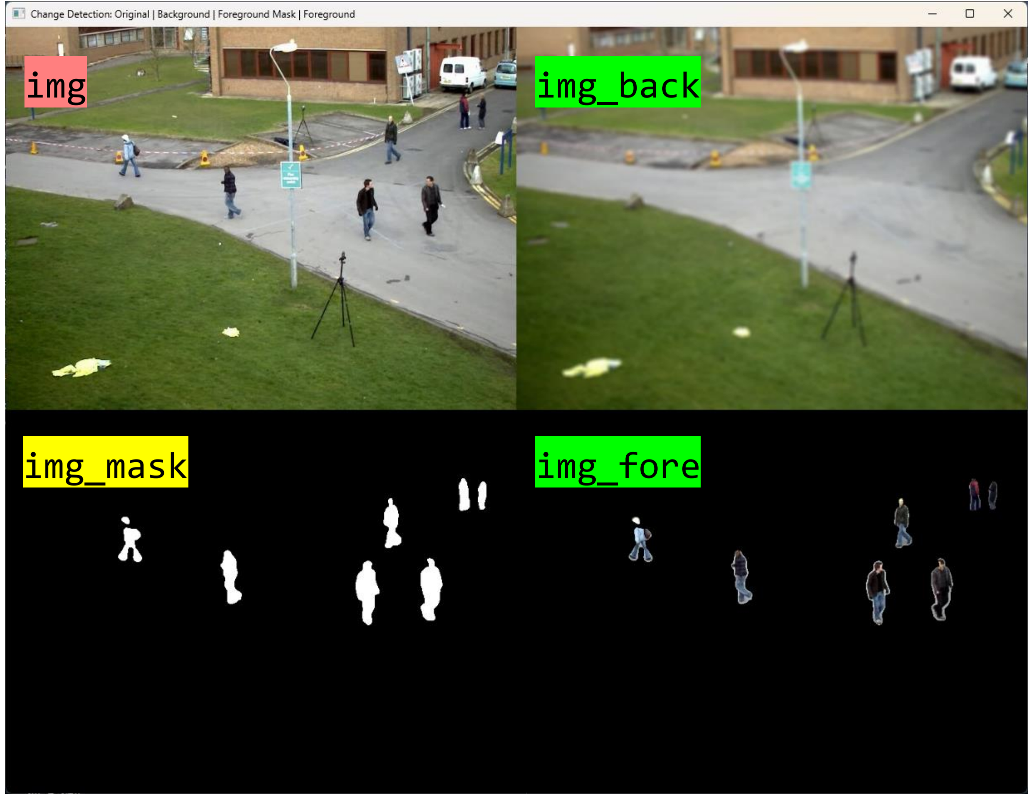
Morphological Operations

- Example) Morphological operations

```python
# Define morphological operations and kernels
morph_operations = [
    {'name': 'Erode',     'operation': cv.MORPH_ERODE},  # Alternative) cv.erode()
    {'name': 'Dilate',    'operation': cv.MORPH_DILATE}, # Alternative) cv.dilate()
    ...]

kernel_tables = [
    {'name': '3x3 Box',   'kenerl': np.ones((3, 3), dtype=np.uint8)},
    {'name': '5x5 Box',   'kenerl': np.ones((5, 5), dtype=np.uint8)},
    ...]

# Read the given image as gray scale
img = cv.imread('data/face.png', cv.IMREAD_GRAYSCALE)

# Initialize a control parameter
morph_select = 0
kernel_select = 0
n_iterations = 1

while True:
    # Apply morphological operation to the image with the given 'kernel'
    m_name, operation = morph_operations[morph_select].values() # Make alias
    k_name, kernel = kernel_tables[kernel_select].values()      # Make alias
    result = cv.morphologyEx(img, operation, kernel, iterations=n_iterations)

    # Show the image and its filtered result
```

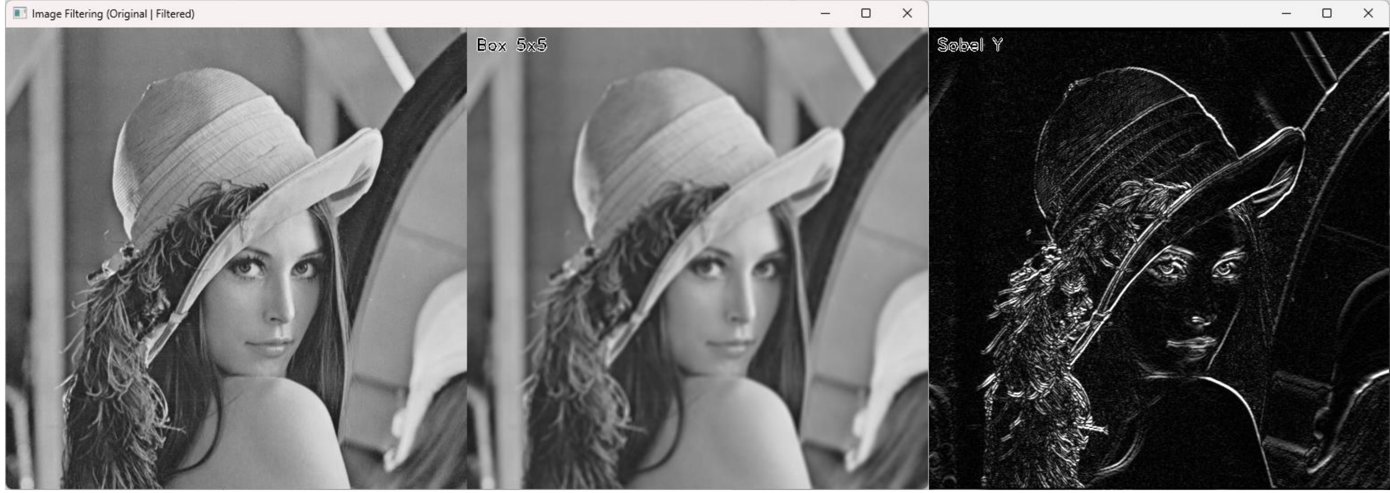
Morphological Operations

- Application) Change detection (foreground extraction)




```
while True:
    ...
    # Get the difference between the current image and background
    img_blur = cv.GaussianBlur(img, blur_ksize, blur_sigma)
    img_diff = img_blur - img_back

    # Apply thresholding
    img_norm = np.linalg.norm(img_diff, axis=2)
    img_bin = np.zeros_like(img_norm, dtype=np.uint8)
    img_bin[img_norm > diff_threshold] = 255

    # Apply morphological operations
    img_mask = img_bin.copy()
    img_mask = cv.erode(img_mask, box(3))                  # Suppress small noise
    img_mask = cv.dilate(img_mask, box(5))                 # Connect broken parts
    img_mask = cv.dilate(img_mask, box(3))                 # Connect broken parts
    fg = img_mask == 255                                   # Keep the (thick) foreground mask
    img_mask = cv.erode(img_mask, box(3), iterations=2) # Restore the thick mask thin

    # Update the background
    # Alternative) cv.createBackgroundSubtractorMOG2(), cv.bgsegm
    bg = ~fg
    img_back[bg] = (bg_update_rate * img_blur[bg] + (1 - bg_update_rate) * img_back[bg]) # With the higher weight
    img_back[fg] = (fg_update_rate * img_blur[fg] + (1 - fg_update_rate) * img_back[fg]) # With the lower weight

    # Get the foreground image
    img_fore = np.zeros_like(img)
    img_fore[img_mask == 255] = img[img_mask == 255]
```

Summary



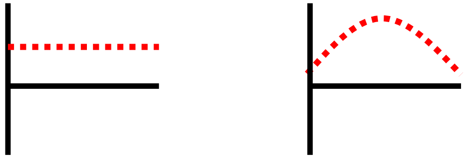
point-wise processing

- **Intensity Transformation**
  - Contrast stretching
  - Histogram equalization
    - How to eliminate user parameters?
- **Thresholding**
  - How to select the threshold?

region-wise processing

- **Image Filtering**
  - Smoothing filters
    - Advanced filters: Median Filter, bilateral filter
  - Edge detection (1st derivative): Prewitt, Sobel, Scharr
    - Popular edge detector: Canny edge detector
  - Laplacian operator (2nd derivative) → Sharpening
- **Morphological Operations**
  - Erosion, Dilation → Opening, Closing, …
  - Application) Change detection (foreground extraction)

low-pass filter



high-pass filter