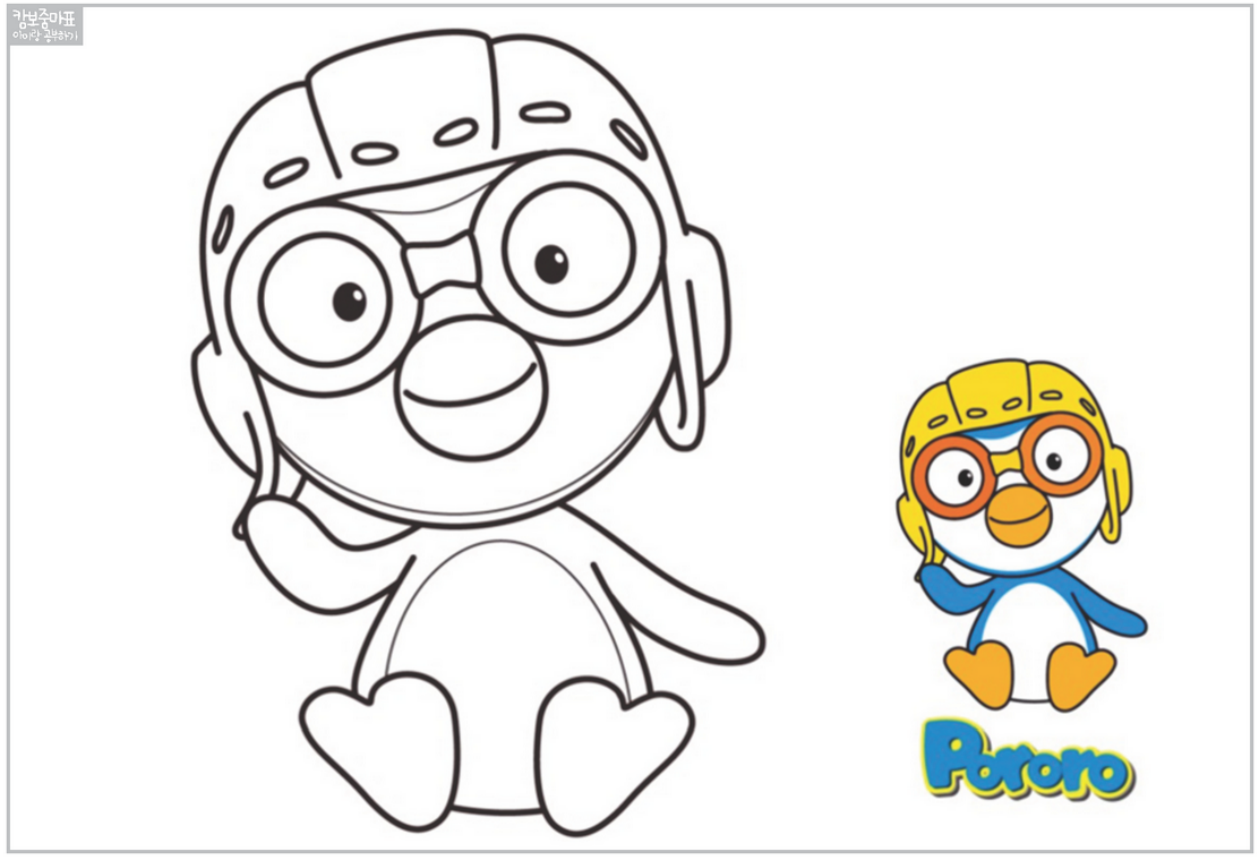
Image Gradients

김수환

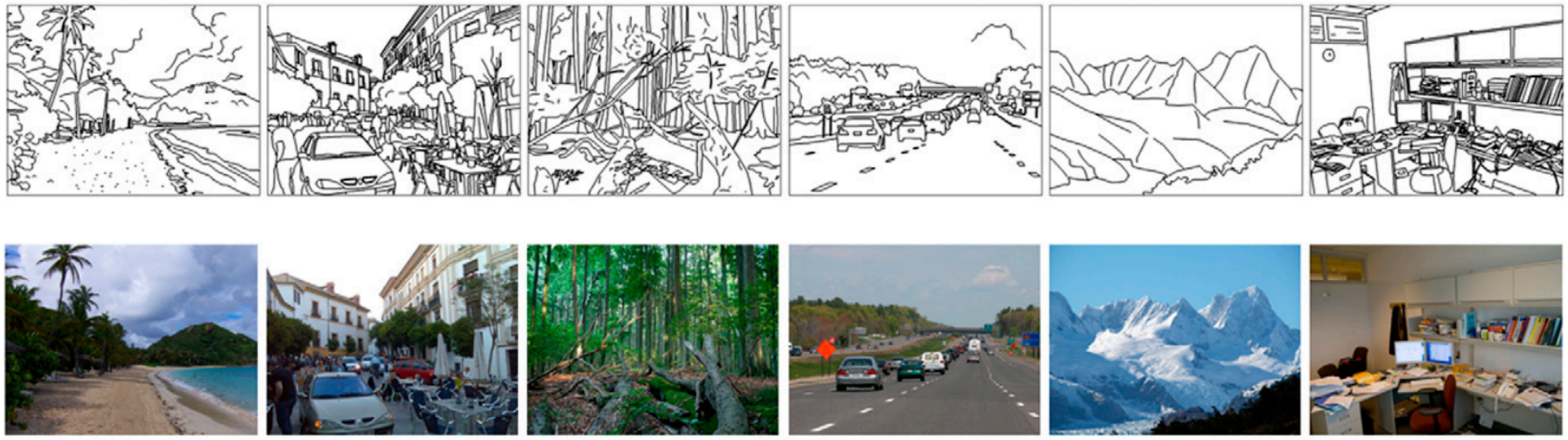
https://www.soohwan.kim

왜 Edge Detection이 중요한가?



https://lynn-kyu.tistory.com/entry/뽀로로-색칠공부-도안

왜 Edge Detection이 중요한가?



Much information is retained by the edges!

https://slideplayer.com/slide/13588937/


학습목표

학습목표

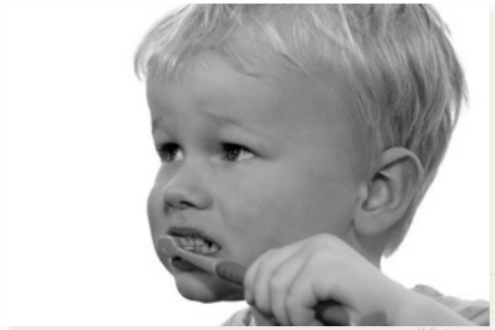
1. Image Processing
    1. Image Thresholding
    2. Image Blending
    3. Image Filtering
    4. Morphological Transformations
    5. Image Gradients

Image Gradients

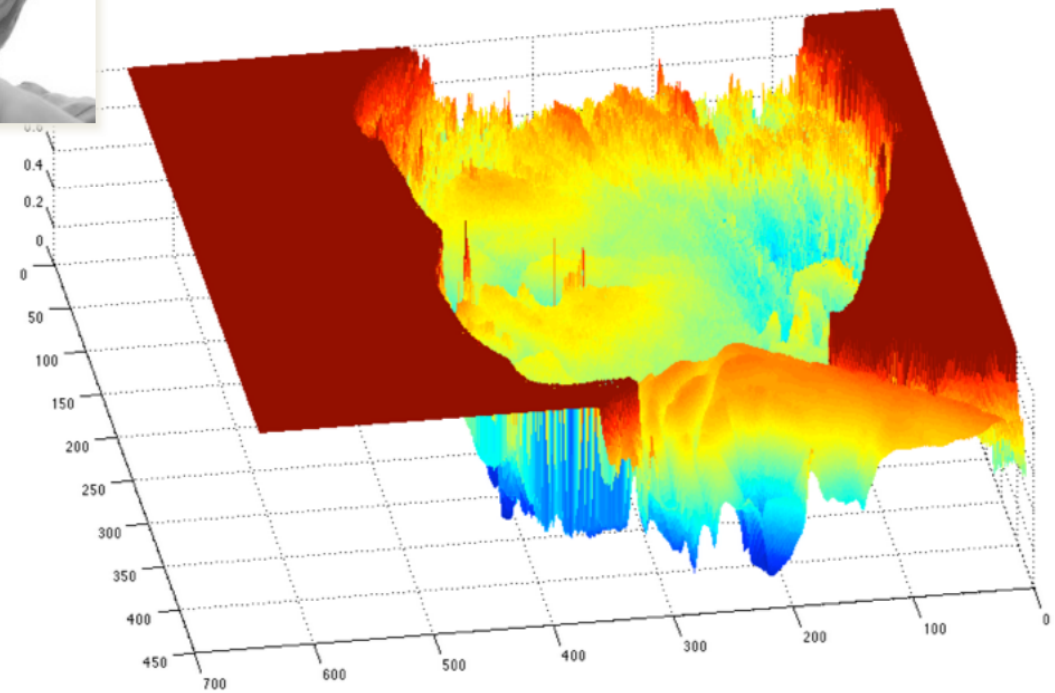
- Sobel Derivatives
- Scharr Derivatives
- Laplacian Derivatives

■ □ Sobel Filter

Grayscale Image as a 2D Function

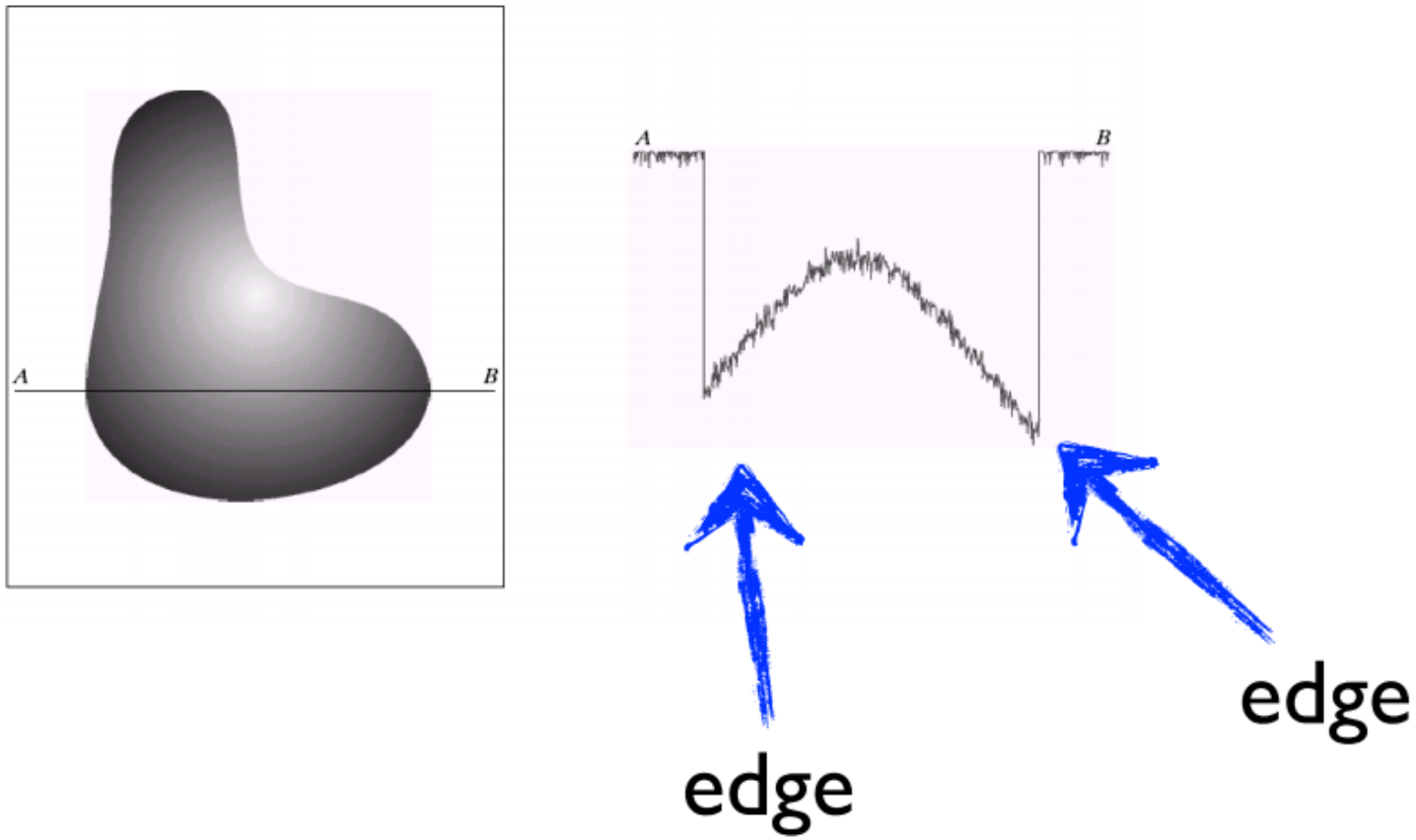


$f(\boldsymbol{x})$



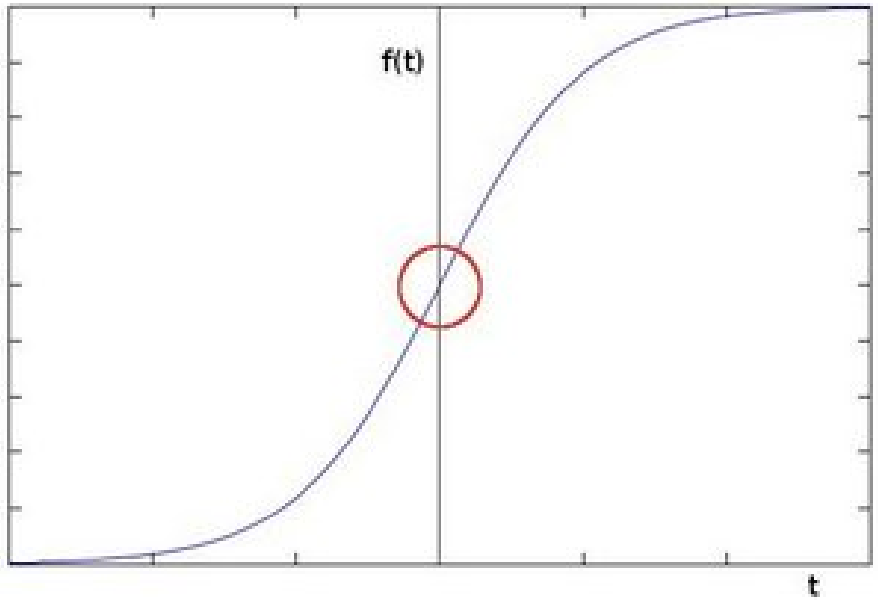
http://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf

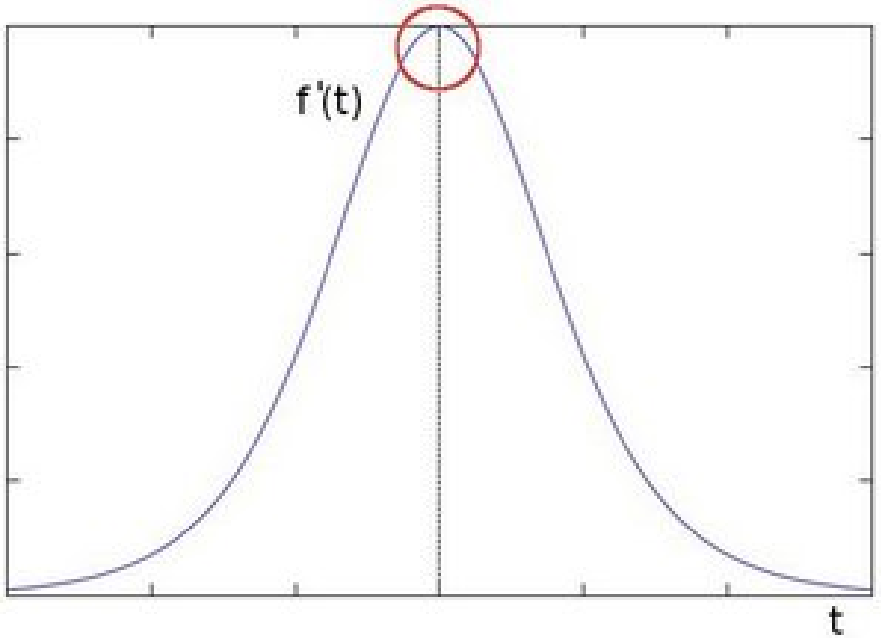
Grayscale Image as a 2D Function



http://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf

Intersection as a 1D Function





https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Derivatives for Continuous Functions

- 1D

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- 2D

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h) - f(x,y)}{h}$$

Discrete Approximation for Derivatives and Kernels

- 1D

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x) \;\Rightarrow\; \begin{bmatrix} -1 & 1 \end{bmatrix}$$

- 2D

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h} \approx f(x+1,y) - f(x,y) \;\Rightarrow\; \begin{bmatrix} -1 & 1 \end{bmatrix}$$

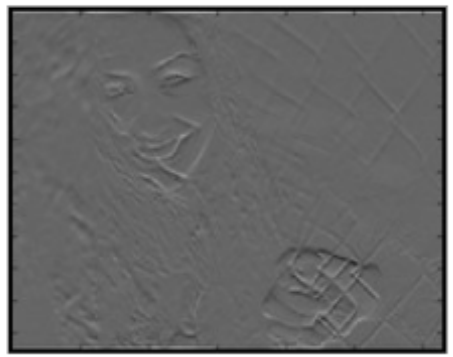
$$\frac{\partial f(x,y)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h) - f(x,y)}{h} \approx f(x,y+1) - f(x,y) \;\Rightarrow\; \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Note the convolution kernel is flipped!

Image Derivatives

Image I

$$I_x = I * \begin{bmatrix} -1 & 1 \end{bmatrix}$$



$$I_y = I * \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

https://slideplayer.com/slide/7592255/

Forward, Backward, and Central Difference

Simplest differentiating kernel is finite differences:

$$\begin{bmatrix} 1 & \underline{-1} \end{bmatrix} \qquad \begin{bmatrix} \underline{1} & -1 \end{bmatrix}$$

Forward **difference** ***Backward*** **difference**

Example:

$$\begin{bmatrix} 4 & 7 & 8 & 1 & 3 \end{bmatrix} \circledast \begin{bmatrix} 1 & \underline{-1} \end{bmatrix} = \begin{bmatrix} 3 & 1 & -7 & 2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 7 & 8 & 1 & 3 \end{bmatrix} \circledast \begin{bmatrix} \underline{1} & -1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 1 & -7 & 2 \end{bmatrix}$$

(Don't forget to flip kernel before convolving)

https://slideplayer.com/slide/7592255/

Forward, Backward, and Central Difference

To avoid undesirable shift, use central difference kernel:

$$\frac{1}{2}\begin{bmatrix}1 & \underline{0} & -1\end{bmatrix}$$

Central **difference**

Example:

$$\begin{bmatrix}4 & 7 & 8 & 1 & 3\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1 & \underline{0} & -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}3 & 4 & -6 & -5 & 2\end{bmatrix}$$

Note: Result has the same basic shape

https://slideplayer.com/slide/7592255/

Forward, Backward, and Central Difference

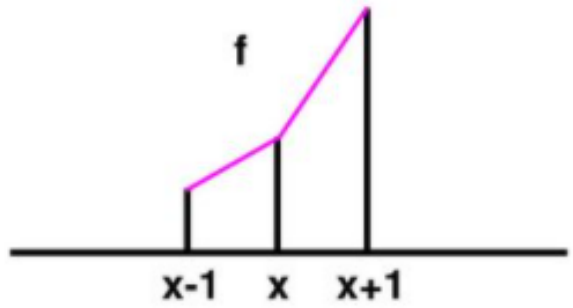
Central difference seems strange b/c it completely ignores the pixel:

$$\begin{bmatrix}1 & 3 & 5\end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix}1 & \underline{0} & -1\end{bmatrix} = 2$$

$$\begin{bmatrix}1 & 99 & 5\end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix}1 & \underline{0} & -1\end{bmatrix} = 2$$

same result!

But it averages the two slopes:



$$\frac{1}{2}\left(\frac{f(x)-f(x-1)}{1} + \frac{f(x+1)-f(x)}{1}\right) = \frac{f(x+1)-f(x-1)}{2}$$

(backward diff) *(forward diff)*

https://slideplayer.com/slide/7592255/

Discrete Approximation for Derivatives and Kernels

- 1D

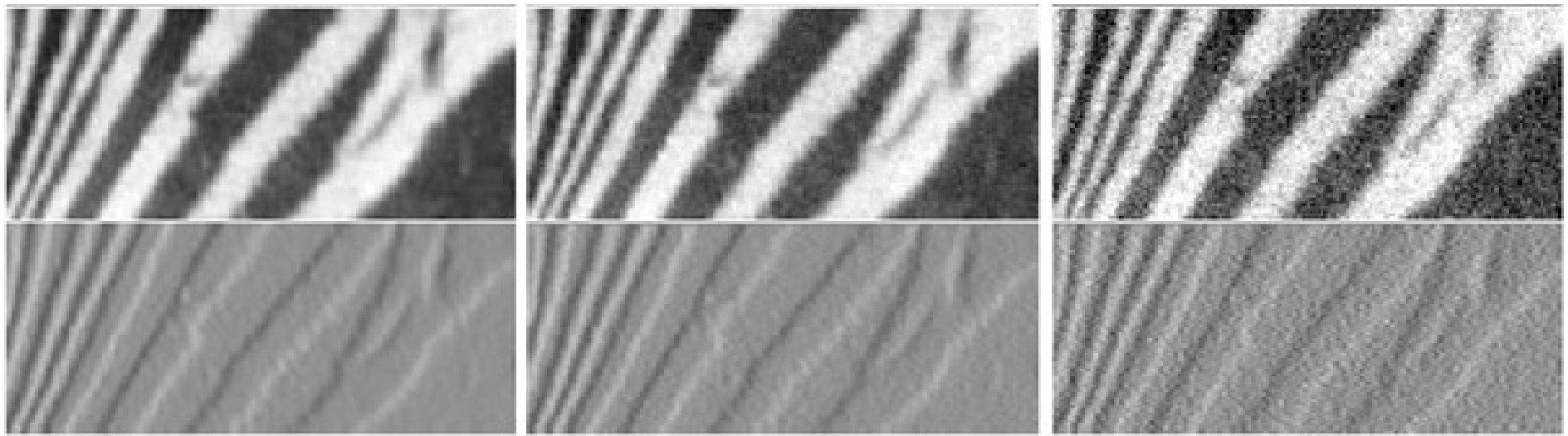
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+1) - f(x-1)}{2} \Rightarrow \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

- 2D

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h} \approx \frac{f(x+1,y) - f(x-1,y)}{2} \Rightarrow \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

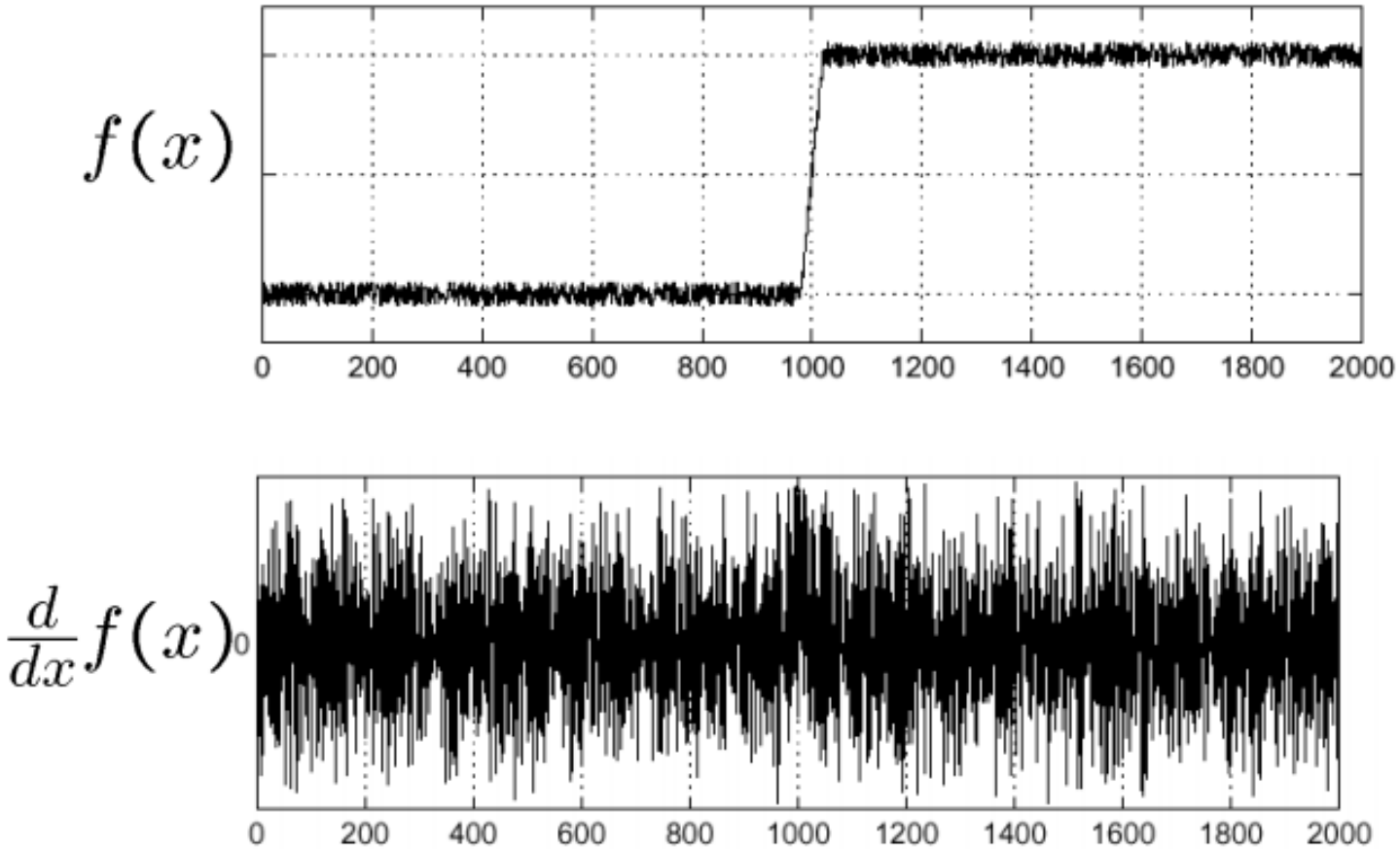
$$\frac{\partial f(x,y)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h) - f(x,y)}{h} \approx \frac{f(x,y+1) - f(x,y-1)}{2} \Rightarrow \frac{1}{2}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Image Derivatives are Sensitive to Noise



https://slideplayer.com/slide/7592255/

Derivatives are Sensitive to Noise



http://dept.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf

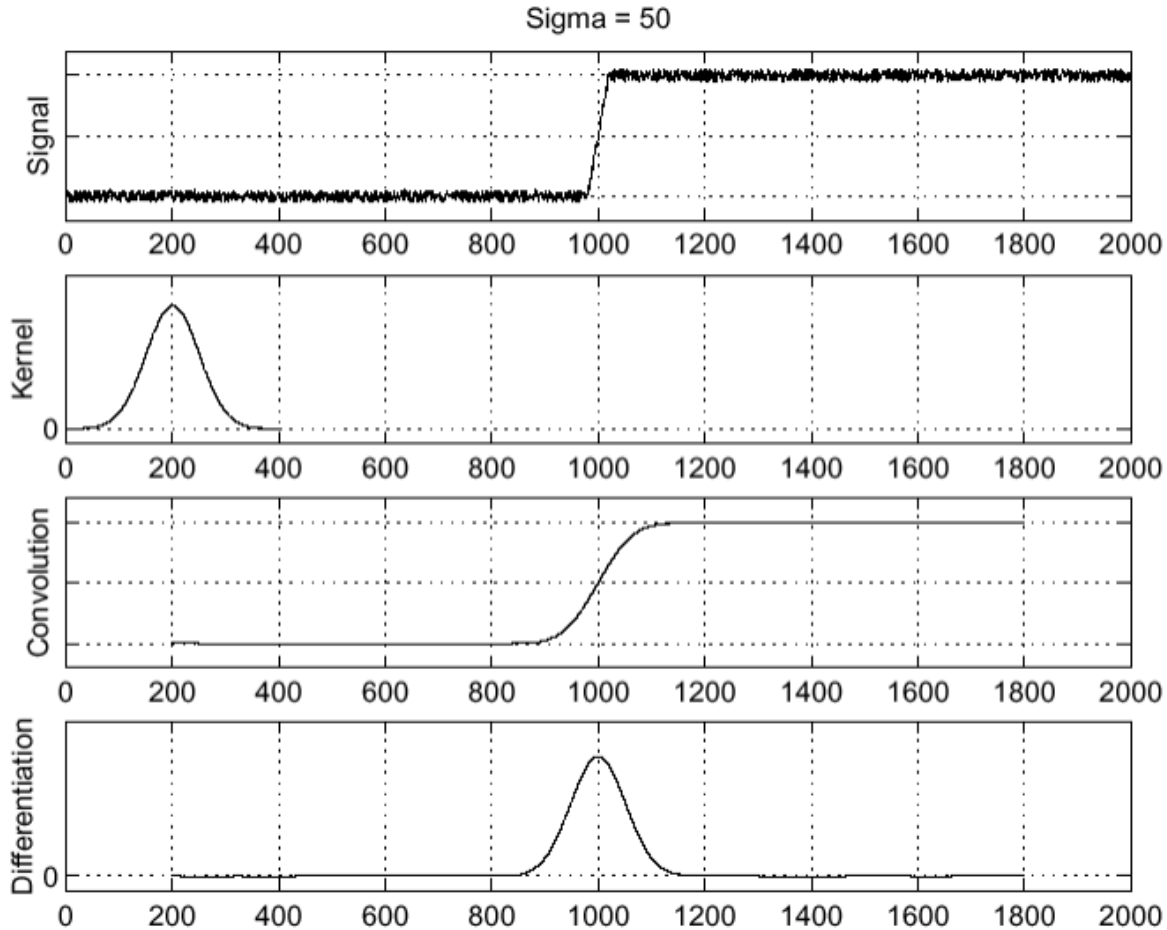
Smooth First, Then Take Derivatives

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



https://slideplayer.com/slide/7592255/

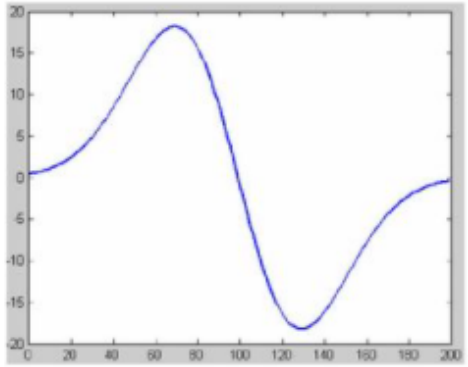
Derivative Theorem of Convolution

$$
\begin{aligned}
(f \otimes g)' &= \frac{d}{dt} \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \\
&= \int_{-\infty}^{\infty} f(\tau) \frac{d}{dt} g(t-\tau) d\tau \\
&= \int_{-\infty}^{\infty} f(\tau) g'(t-\tau) d\tau \\
&= f \otimes g'
\end{aligned}
$$

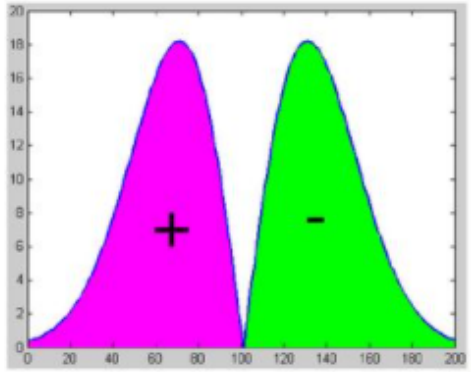
1D Gaussian Derivatives

$$\frac{\partial}{\partial x}\left(I(x) \circledast G(x)\right) = I(x) \circledast \left(\frac{\partial}{\partial x} G(x)\right)$$

$$\begin{aligned}
\frac{dG(x)}{dx} &= \frac{d}{dx}\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right)\right] \\
&= -\frac{x}{\sigma^2\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right) \\
&= -\frac{x}{\sigma^2} G(x)
\end{aligned}$$



Gaussian derivative



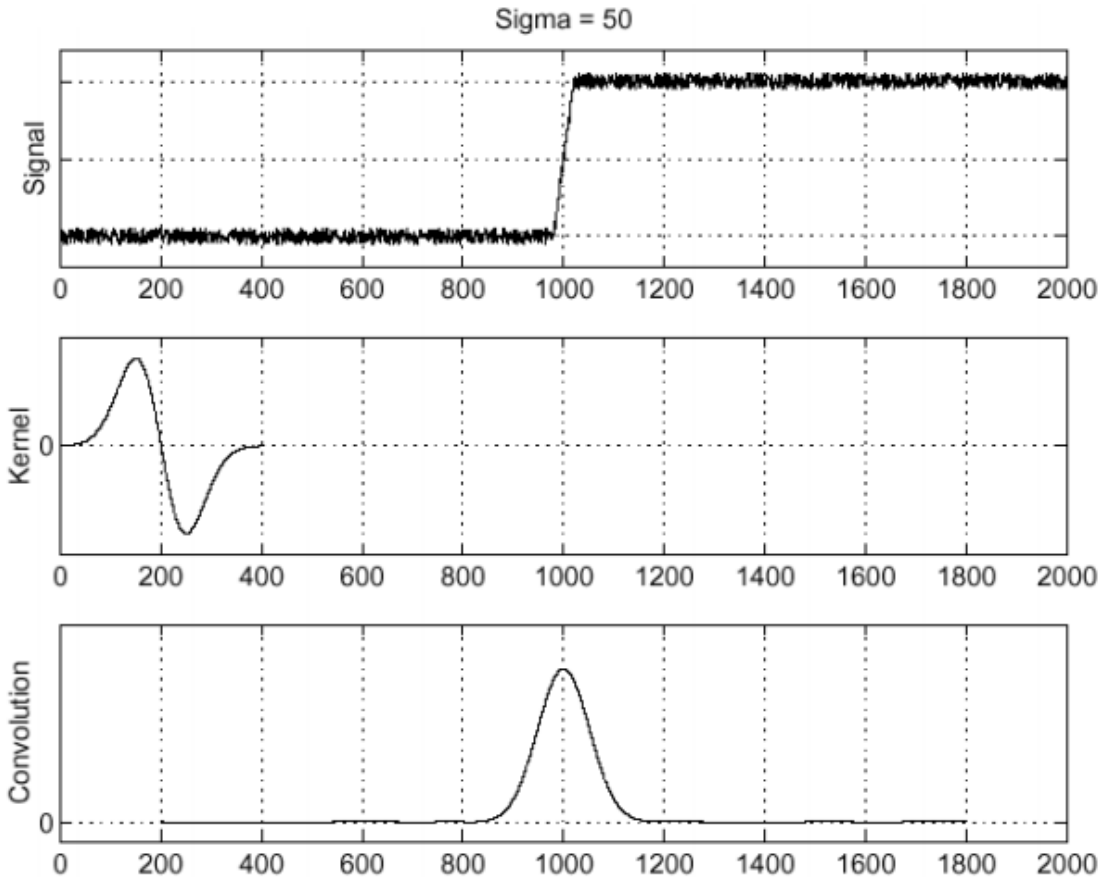
equivalent view

https://slideplayer.com/slide/13588937/

Derivative of Gaussian (DoG) Filter

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



https://inst.eecs.berkeley.edu/~cs194-26/fa17/Lectures/ConvEdgesTemplate.pdf

2D Gaussian Derivatives

- Gaussian

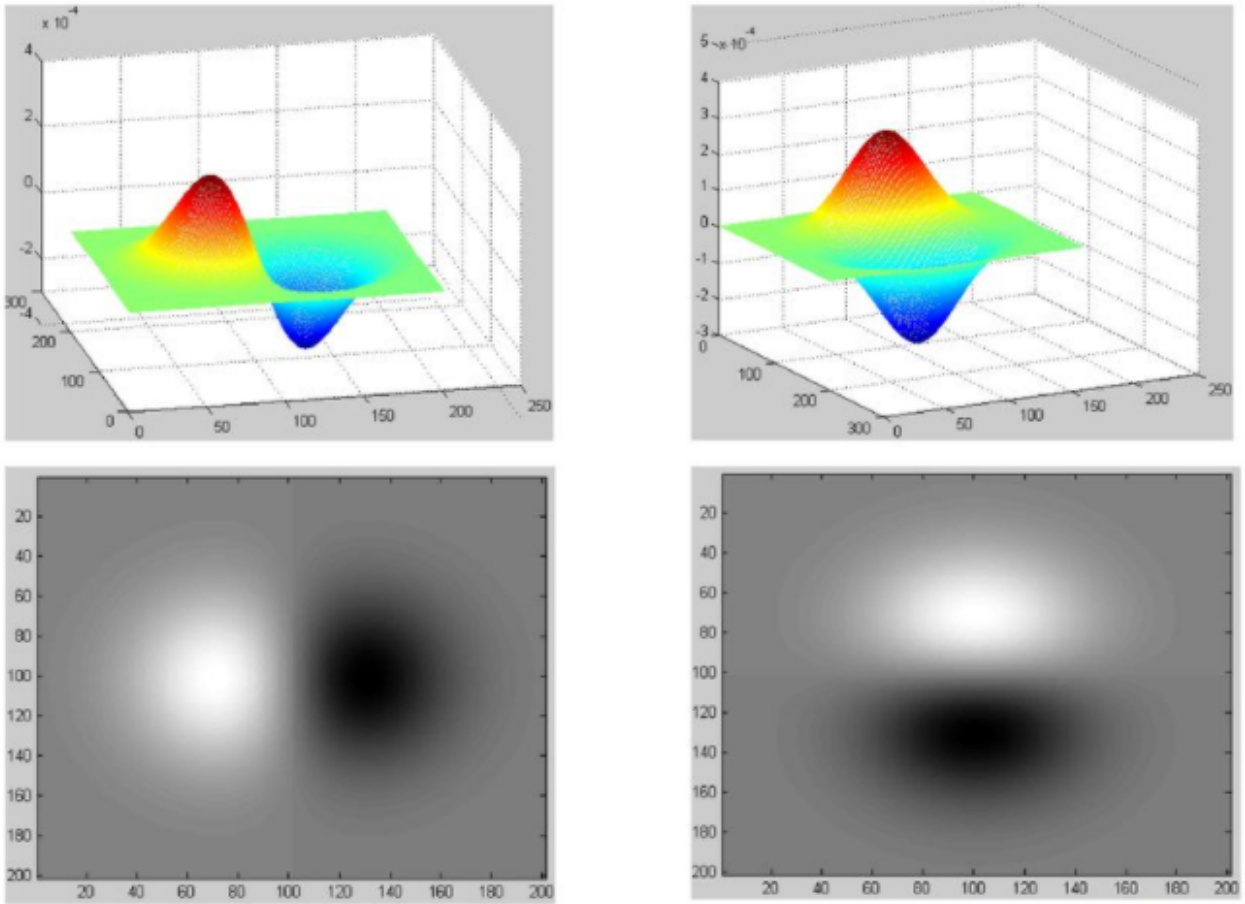
$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- Partial Derivatives

$$\frac{\partial G(x,y)}{\partial x} = \frac{-x}{4\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

$$\frac{\partial G(x,y)}{\partial y} = \frac{-y}{4\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

2D Gaussian Derivatives



https://slideplayer.com/slide/13588937/

Sobel Filters

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

2D Gaussian Derivative is Separable

- Gaussian

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$
$$= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y^2}{2\sigma^2}\right) = G_h(x)G_v(y)$$

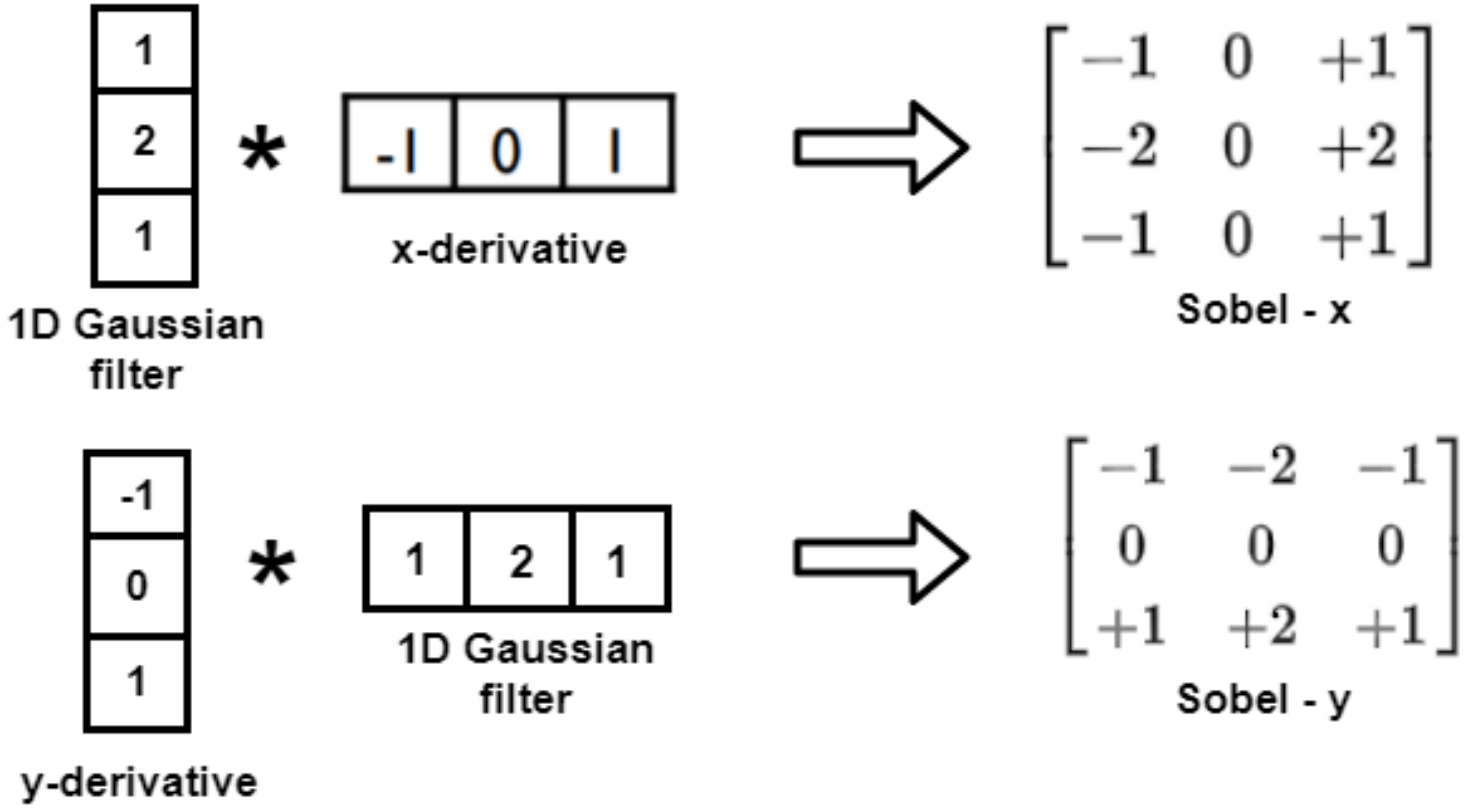
- Partial Derivatives

$$\frac{\partial G(x,y)}{\partial x} = \frac{-x}{\sigma^2}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y^2}{2\sigma^2}\right) = G'_h(x)G_v(y)$$

$$\frac{\partial G(x,y)}{\partial y} = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\frac{-y}{\sigma^2}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y^2}{2\sigma^2}\right) = G_h(x)G'_v(y)$$

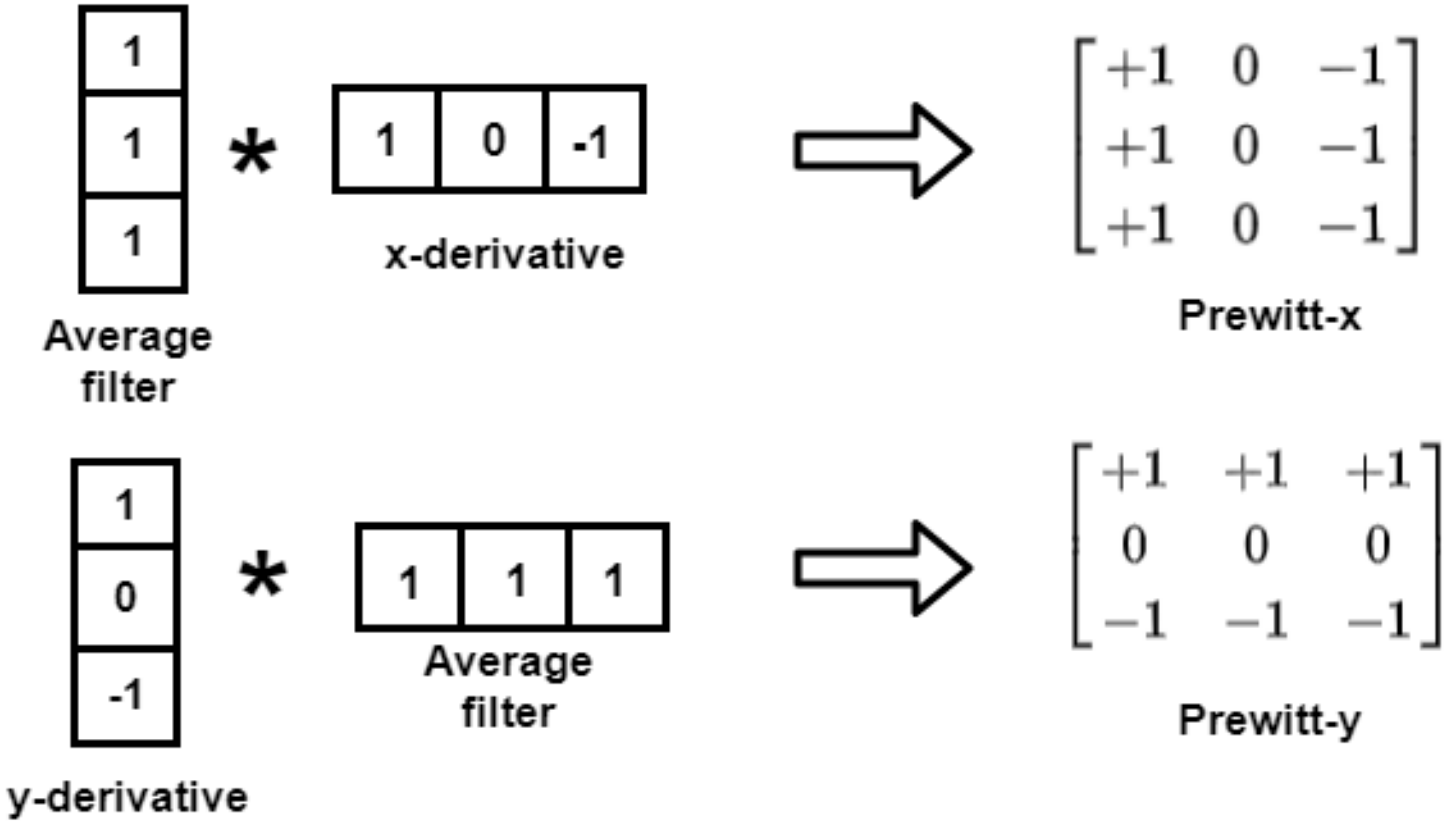
https://slideplayer.com/slide/13588937/

Sobel Operator/Filter



https://theailearner.com/2019/05/24/first-order-derivative-kernels-for-edge-detection/

Prewitt Operator/Filter



https://theailearner.com/2019/05/24/first-order-derivative-kernels-for-edge-detection/

Derivative of Gaussian Filters

Prewitt

$$Prewitt_x = \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{6}\begin{bmatrix}1 & 0 & -1\\1 & 0 & -1\\1 & 0 & -1\end{bmatrix}$$

$$Prewitt_y = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{6}\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix}$$

Sobel

$$Sobel_x = G_{0.5} \circledast (G'_{0.5})^T = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$$

$$Sobel_y = (G_{0.5})^T \circledast G'_{0.5} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$$

Scharr

$$Scharr_x = G_{0.375} \circledast (G'_{0.5})^T = \frac{1}{16}\begin{bmatrix}3\\10\\3\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 0 & -3\\10 & 0 & -10\\3 & 0 & -3\end{bmatrix}$$

$$Scharr_y = G_{0.375} \circledast (G'_{0.5})^T = \frac{1}{16}\begin{bmatrix}3 & 10 & 3\end{bmatrix} \circledast \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 10 & 3\\0 & 0 & 0\\-3 & -10 & -3\end{bmatrix}$$

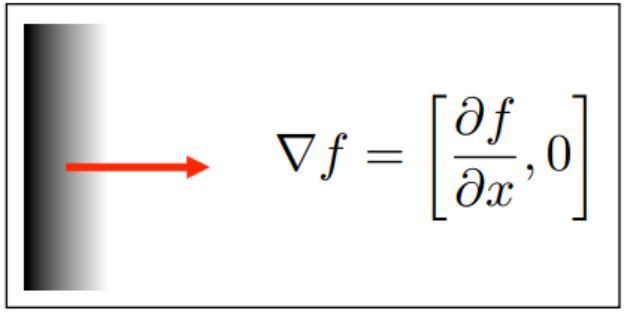
https://slideplayer.com/slide/13588937/

Sobel Operator

- Combination of Gaussian smoothing and differentiation
- Derivative of Gaussian filtering
- Output can be negative or positive

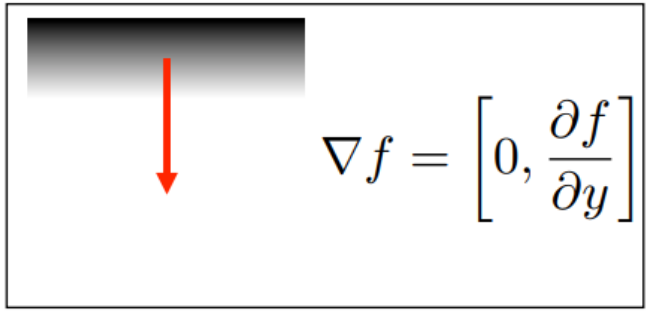
Image Gradient: Direction and Magnitude

Gradient in x only



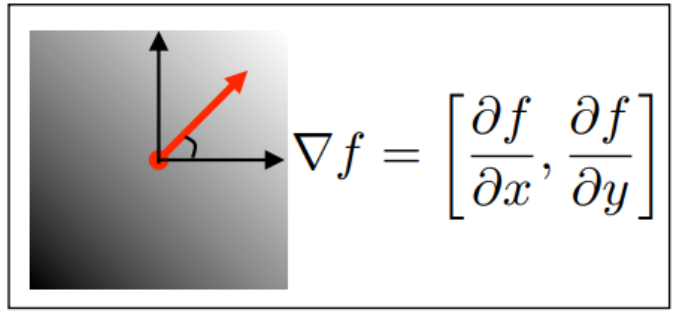
$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

Gradient in y only



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$

Gradient in both x and y



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Gradient direction

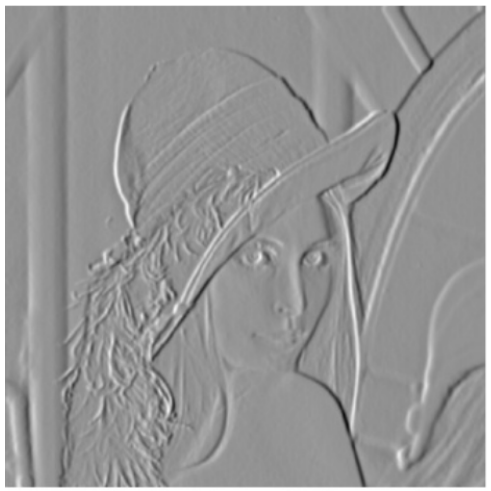
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Gradient magnitude

$$||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

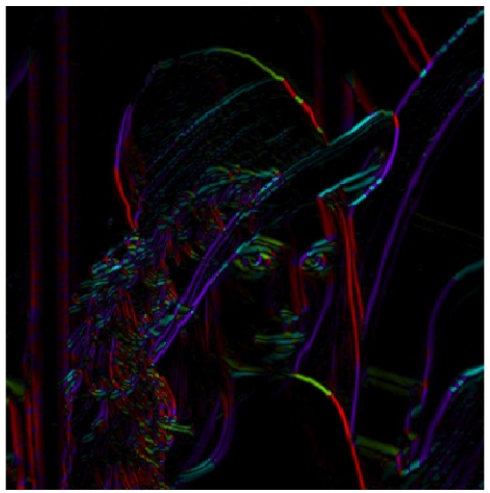
http://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf

Image Gradient: Direction and Magnitude


X-Derivative of Gaussian


Y-Derivative of Gaussian


Gradient Magnitude


Gradient Direction

https://www.slideshare.net/rushilanirudh/edges-and-lines

`cv.Sobel()`[1]

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

- Sobel operator를 이용하여 1차, 2차, 3차 image derivatives를 구한다.
  - `src`: input image
  - `dst`: output array of the same size and the same number of channels as `src`
  - `ddepth`: output image depth
  - `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
  - `delta`: optional delta value
  - `borderValue`: pixel extrapolation method

$$\texttt{dst}(x,y) = \frac{\partial^{xorder+yorder}\texttt{src}}{\partial x^{xorder}\partial y^{yorder}}$$

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d

`cv.Scharr()`[1]

```
dst = cv.Scharr(src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]]])
```

- Scharr operator를 이용하여 1차, 2차, 3차 image derivatives를 구한다.
  - `src`: input image
  - `dst`: output array of the same size and the same number of channels as `src`
  - `ddepth`: output image depth
  - `dx`: order of the derivative x
  - `dy`: order of the derivative y
  - `scale`: optional scale factor
  - `delta`: optional delta value
  - `borderValue`: pixel extrapolation method

$$\text{Scharr}(\text{src}, \text{dst}, \text{ddepth}, \text{dx}, \text{dy}, \text{scale}, \text{delta}, \text{borderType})$$

$$= \text{Sobel}(\text{src}, \text{dst}, \text{ddepth}, \text{dx}, \text{dy}, \text{FILTER\_SCHARR}, \text{scale}, \text{delta}, \text{borderType})$$

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9

`cv.convertScaleAbs()`[1]

```
dst = cv.convertScaleAbs(src[, dst[, alpha[, beta]]])
```

- 상수배를 한 뒤 절대값을 구해서 8비트 이미지로 만든다.
  - `src`: input image
  - `dst`: output array
  - `alpha`: optional scale factor
  - `beta`: optional delta added to the scaled values

$$\text{dst}(I) = \text{saturate\_cast}<\text{uchar}>(|\text{src}(I) * \text{alpha} + \text{beta}|)$$

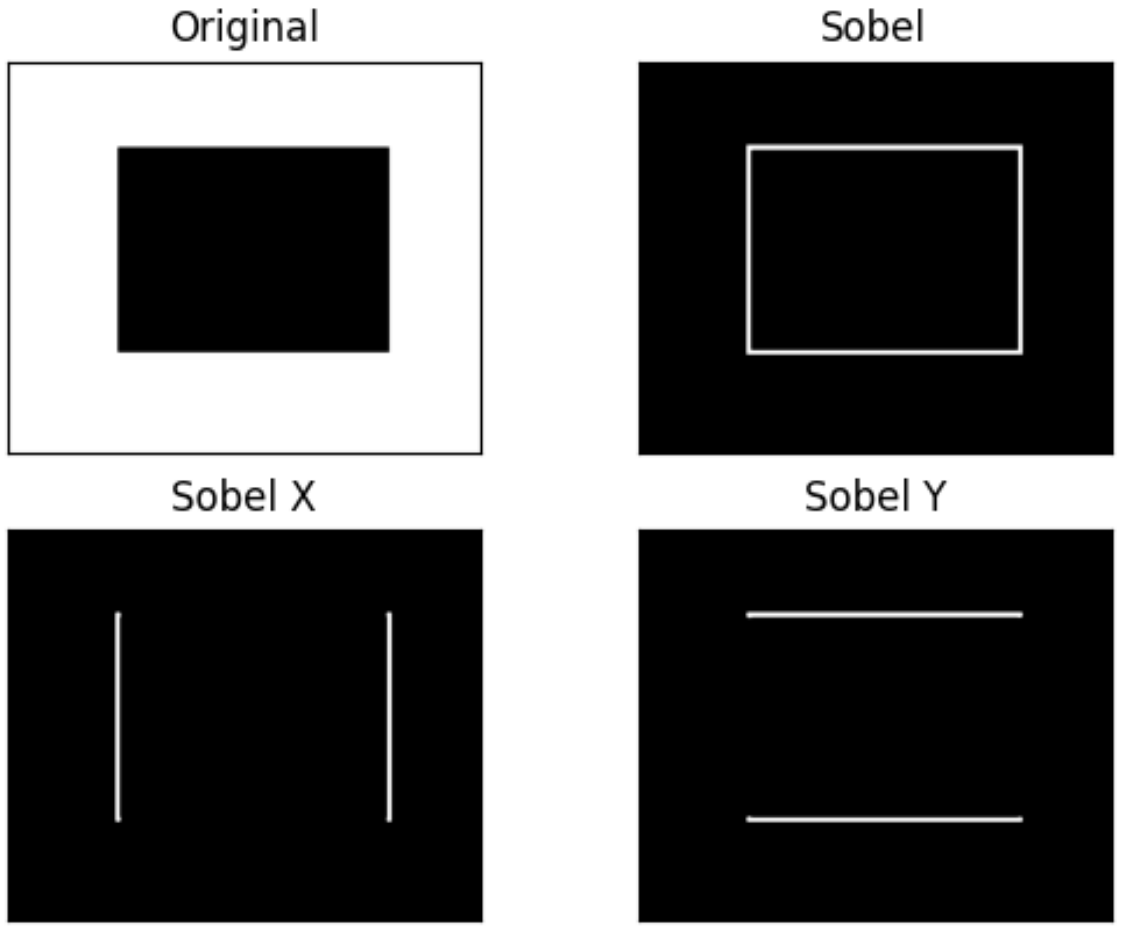
1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9

Sobel Filter: Code 1

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load a grayscale image
img_orig = cv2.imread('box.png', 0)

# Apply gradient filters
img_sobelX = cv2.Sobel(img_orig, cv2.CV_64F, 1, 0, ksize=5)
img_sobelY = cv2.Sobel(img_orig, cv2.CV_64F, 0, 1, ksize=5)

# Take absolute values and scale
img_sobelX = cv2.convertScaleAbs(img_sobelX)
img_sobelY = cv2.convertScaleAbs(img_sobelY)
img_sobel = cv2.addWeighted(img_sobelX, 1, img_sobelY, 1, 0)

# Display results
titles = ['Original', 'Sobel', 'Sobel X', 'Sobel Y']
images = [img_orig, img_sobel, img_sobelX, img_sobelY]

for i in range(4):
```

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

Sobel Filter: Result 1

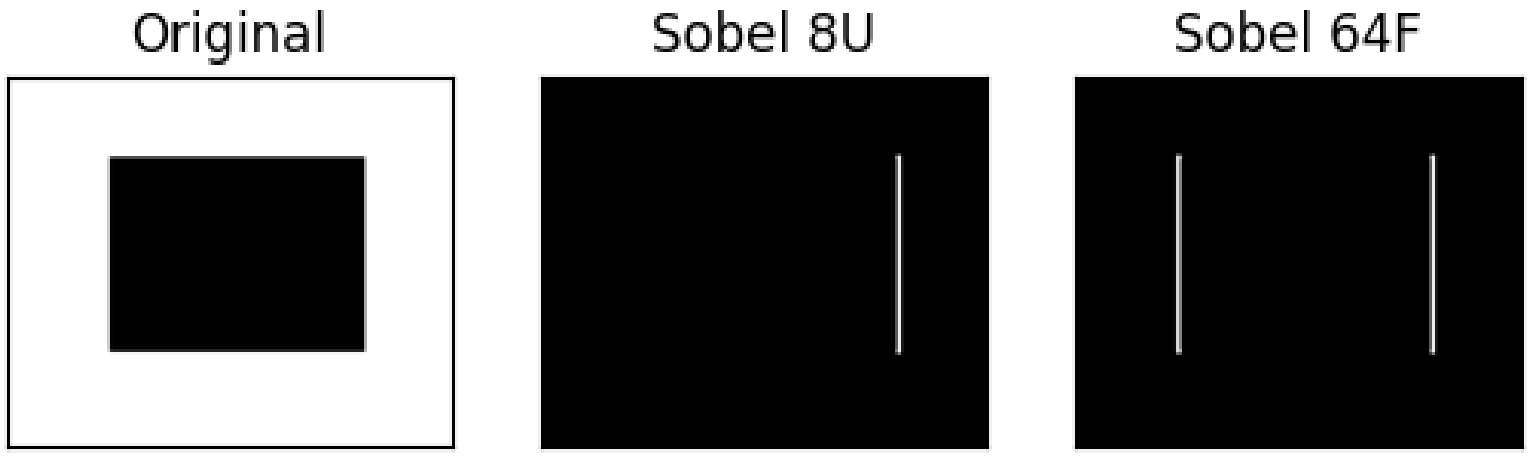


Sobel Filter: Code 2

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load a grayscale image
img_orig = cv2.imread('box.png', 0)

# Apply gradient filters
img_sobelX_8U = cv2.Sobel(img_orig, cv2.CV_8U, 1, 0, ksize=5)
img_sobelX_64F = cv2.Sobel(img_orig, cv2.CV_64F, 1, 0, ksize=5)

# Take absolute values and scale
img_sobelX_8U = cv2.convertScaleAbs(img_sobelX_8U)
img_sobelX_64F = cv2.convertScaleAbs(img_sobelX_64F)

# Display results
titles = ['Original', 'Sobel X', 'Sobel Y']
images = [img_orig, img_sobelX_8U, img_sobelX_64F]

for i in range(3):
    plt.subplot(1, 3, i+1)
```

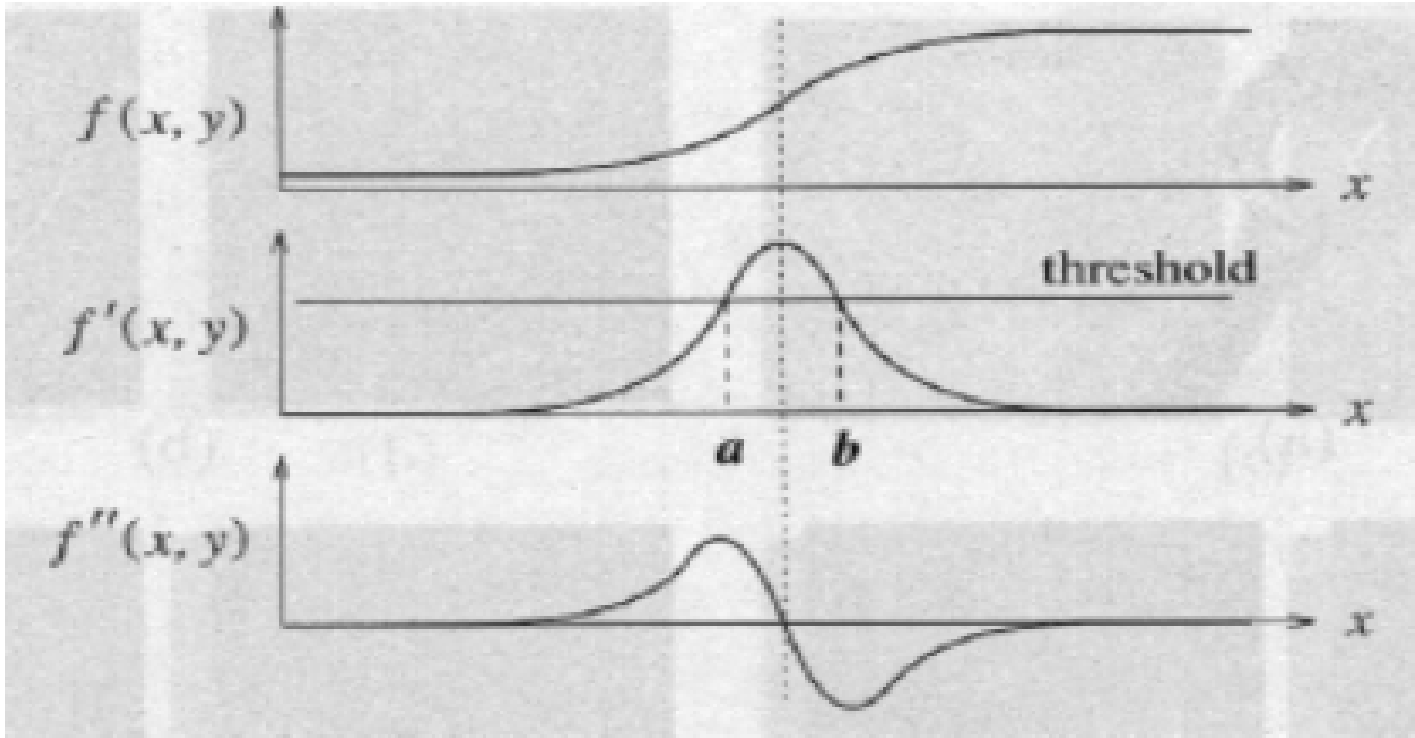
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_gradients/py_gradients.html

Sobel Filter: Result 2



Laplace Filter

Second Derivatives on Edges



http://dept.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf

Second Derivatives

- Second Derivatives

$$
\begin{aligned}
f''(x) &= \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} \\
&\approx f'(x+1) - f'(x) \approx f(x+2) - f(x+1) - (f(x+1) - f(x)) \\
&\approx f(x+2) - 2f(x+1) + f(x)
\end{aligned}
$$

- Centered Second Derivatives

$$
f''(x) \approx f(x+1) - 2f(x) + f(x-1)
$$

- Kernel

$$
\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}
$$

Laplacian

- Laplacian Operator

$$\Delta = \nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial x} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial x} \end{bmatrix} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Second Derivatives

$$\frac{\partial^2 f}{\partial x^2} \approx f(x+1, y) - 2f(x, y) + f(x-1, y)$$

$$\frac{\partial^2 f}{\partial y^2} \approx f(x, y+1) - 2f(x, y) + f(x, y-1)$$

Laplacian

- Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \approx f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

http://dept.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf

Properties of Laplacian

- Cheaper to implement than the gradient (i.e., one mask only)
- No information about the edge direction
- Isotropic operator (equal weights in all directions)
- More sensitive to noise (i.e., differentiates twice)

1D Gaussian Derivatives

- Gaussian

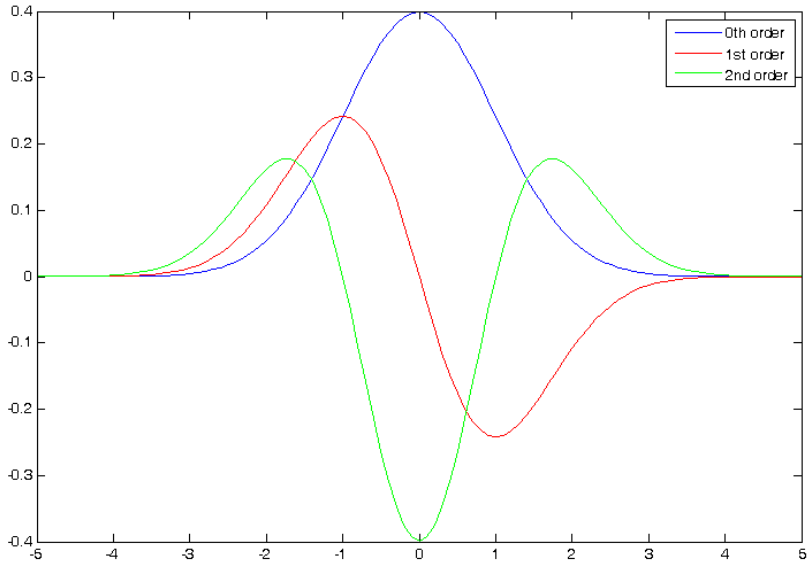
$$G(x) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2}{2\sigma^2}\right)$$

- First Derivative of Gaussian

$$G'(x) = -\frac{x}{\sigma^2}G(x)$$

- Second Derivative of Gaussian

$$G''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right)G(x)$$



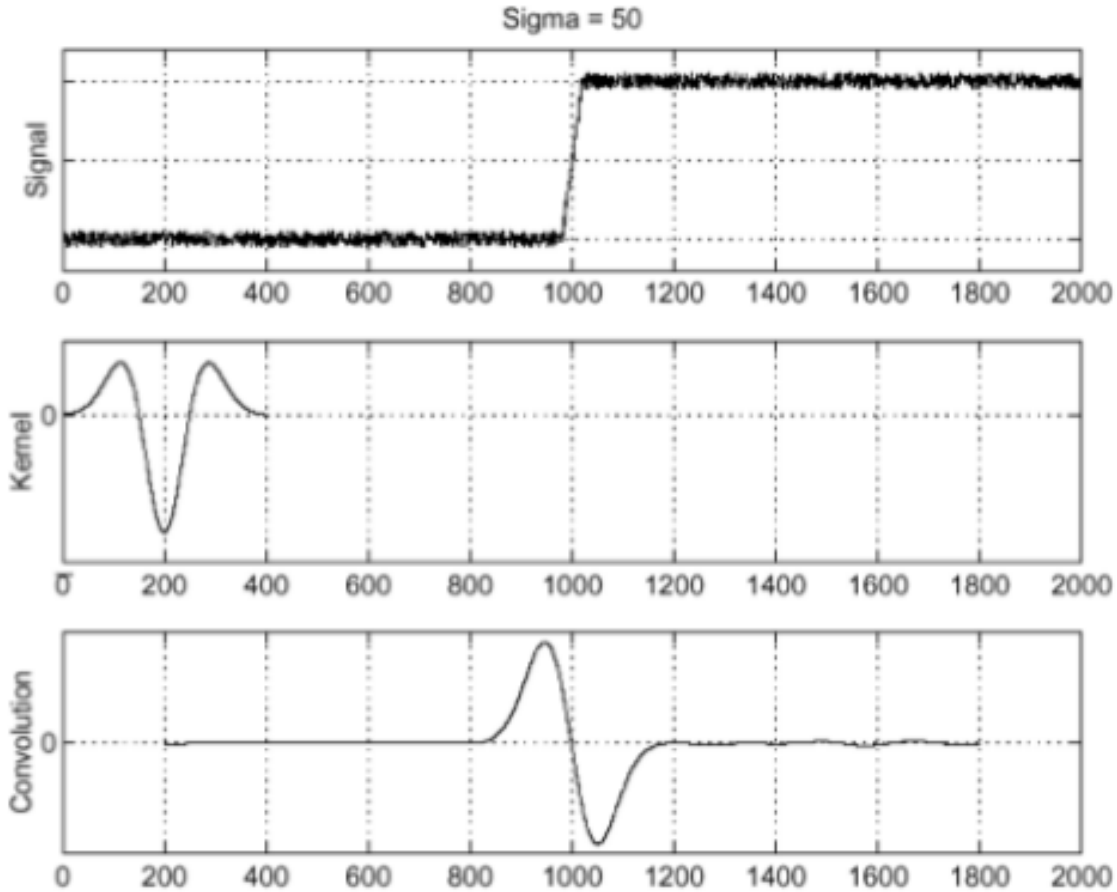
http://campar.in.tum.de/Chair/HaukeHeibelGaussianDerivatives

Laplacian of Gaussian (LoG) Filter

input

Laplacian of Gaussian

output



http://www.cs.cmu.edu/~16385/lectures/lecture2.pdf

2D Gaussian Derivatives

- Gaussian

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

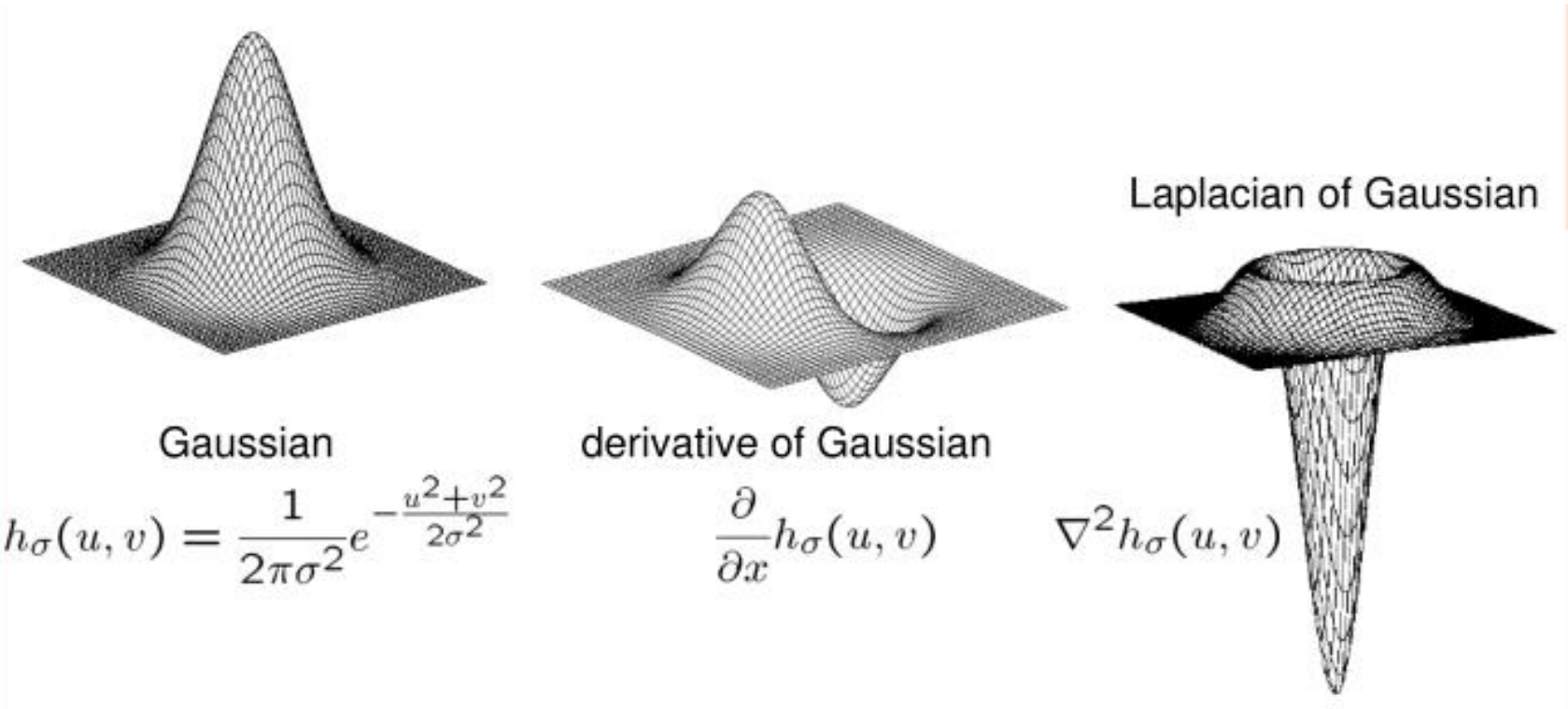
- First Derivatives

$$\frac{\partial G(x,y)}{\partial x} = \frac{-x}{4\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \quad \frac{\partial G(x,y)}{\partial y} = \frac{-y}{4\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- Laplacian

$$\frac{\partial G^2(x,y)}{\partial x^2}\frac{\partial G^2(x,y)}{\partial y^2} = -\frac{1}{4\pi\sigma^4}\left(1-\frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

Laplace of Gaussian (LoG) Filter



http://www.cs.cmu.edu/~16385/lectures/lecture2.pdf

`cv.Laplacian()`[1]

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

- Laplacian image를 계산한다.
  - `src`: input image
  - `dst`: output array of the same size and the same number of channels as `src`
  - `ddepth`: output image depth
  - `ksize`: aperture size of the second-derivative filters; the size must be positive and odd
  - `scale`: optional scale factor
  - `delta`: optional delta value
  - `borderValue`: pixel extrapolation method

$$\text{dst} = \frac{\partial^2 \text{src}}{\partial x^2} + \frac{\partial^2 \text{src}}{\partial y^2}$$

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d

Laplace Filter: Code 1

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load a grayscale image
img_orig = cv2.imread('lena.jpg', 0)

# Apply gradient filters
img_sobelX = cv2.Sobel(img_orig, cv2.CV_64F, 1, 0, ksize=3)
img_sobelY = cv2.Sobel(img_orig, cv2.CV_64F, 0, 1, ksize=3)
img_laplace = cv2.Laplacian(img_orig, cv2.CV_64F, ksize=3)

# Take absolute values and scale
img_sobelX = cv2.convertScaleAbs(img_sobelX)
img_sobelY = cv2.convertScaleAbs(img_sobelY)
img_sobel = cv2.addWeighted(img_sobelX, 1, img_sobelY, 1, 0)
img_laplace = cv2.convertScaleAbs(img_laplace)

# Display results
titles = ['Original', 'Sobel', 'Laplace']
images = [img_orig, img_sobel, img_laplace]
```

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

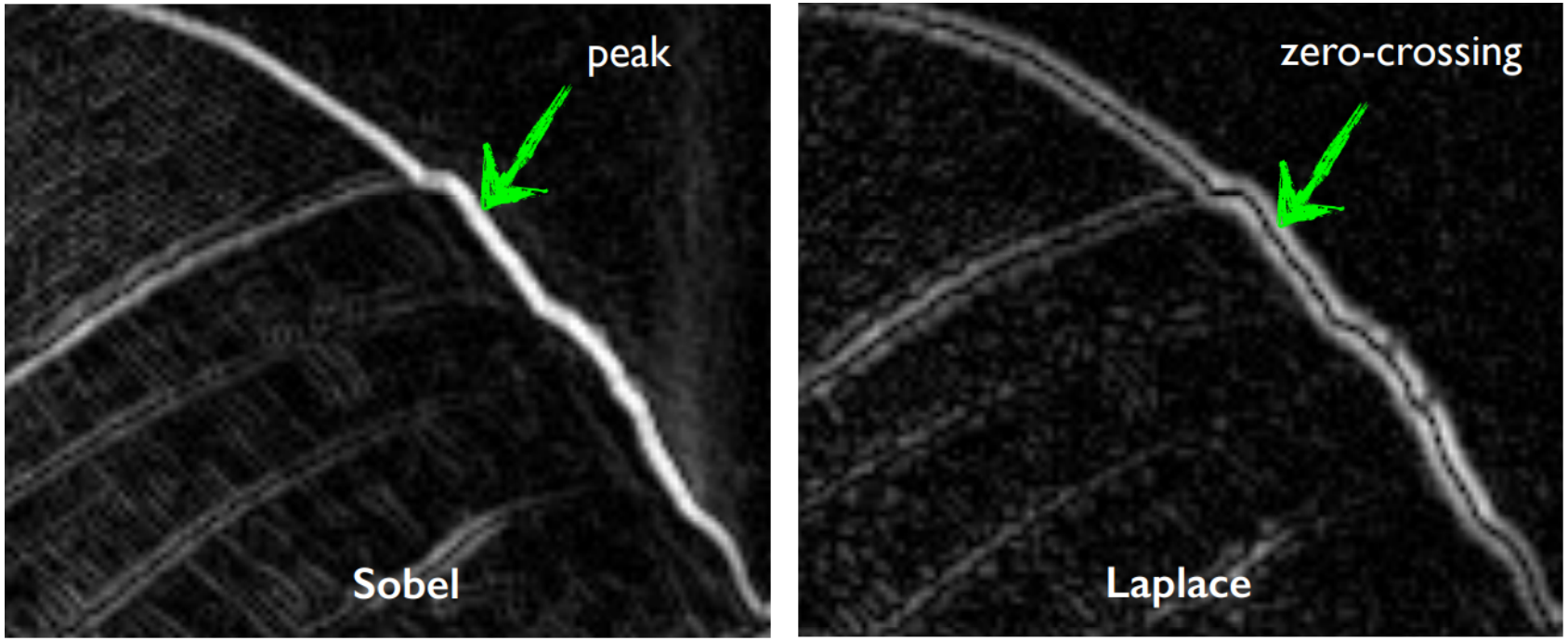
Laplace Filter: Result 1

Original

Sobel

Laplace

Sobel vs. Laplace Filter



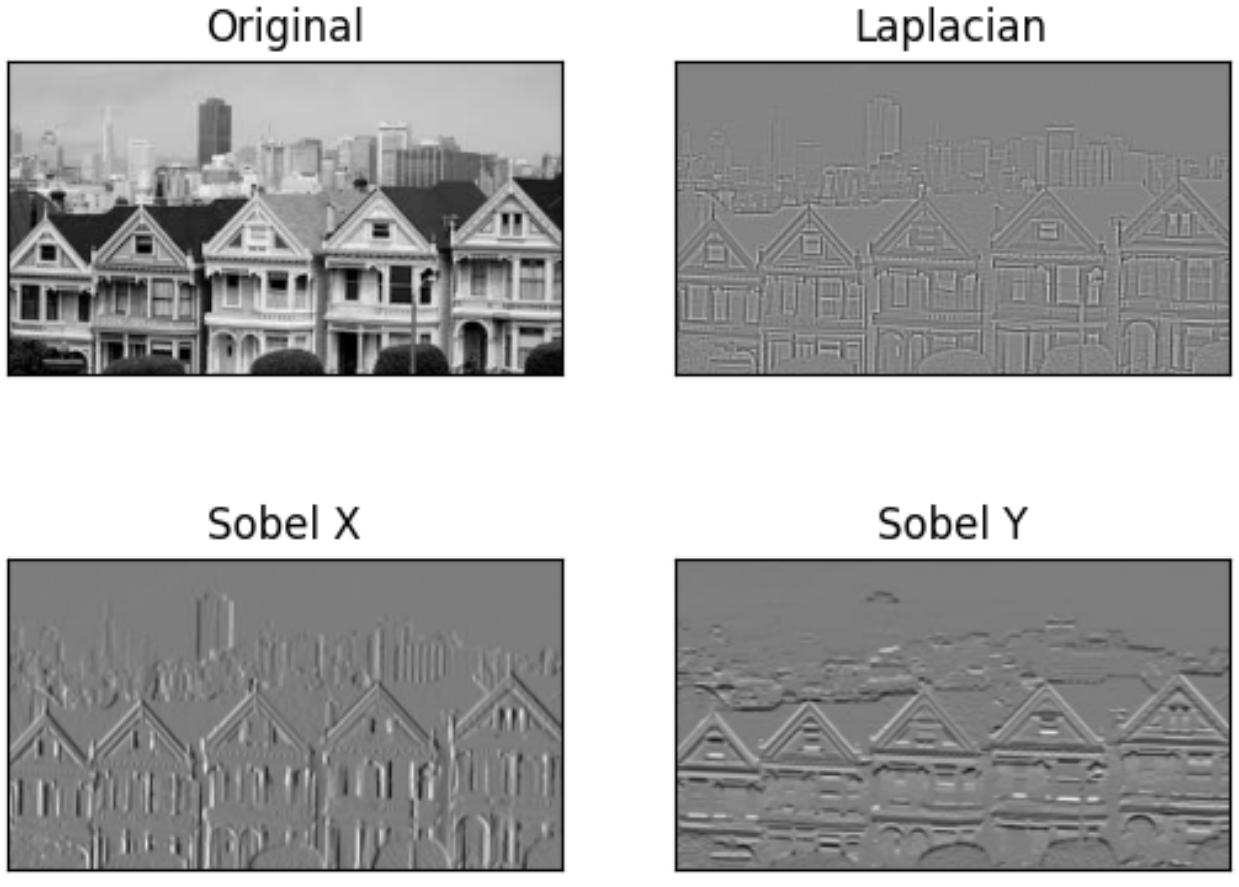
http://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf

Laplace Filter: Code 2

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load an image
img_orig = cv2.imread('SanFrancisco.jpg', 0)
#img_orig = cv2.imread('windows.jpg', 0)

# Convolute with proper kernels
img_sobelX = cv2.Sobel(img_orig, cv2. CV_64F, 1, 0, ksize=3)  # x
img_sobelY = cv2.Sobel(img_orig, cv2. CV_64F, 0, 1, ksize=3)  # y
img_laplace = cv2.Laplacian(img_orig, cv2.CV_64F, ksize=3)

# Display results
titles = ['Original', 'Laplacian', 'Sobel X', 'Sobel Y']
images = [img_orig, img_laplacian, img_sobelX, img_sobelY]

for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
```

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

Laplacian Filter: Result 2



Canny Edge Detection

Canny Edge Detection

- The most popular edge detection algorithm
- Developed by John F. Canny in 1986

- 5 Steps
  1. Noise Reduction
  2. Image Gradient
  3. Non-Maximum Suppression
  4. Double THresholding
  5. Hysteresis Thresholding

Canny Edge Detection: Result

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

(Step 1) Noise Reduction: Algorithm

- 5×5 Gaussian Filter

$$K = \frac{1}{256} \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

(Step 1) Noise Reduction: Result

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

(Step 2) Image Gradient: Algorithm

- Sobel Filter

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

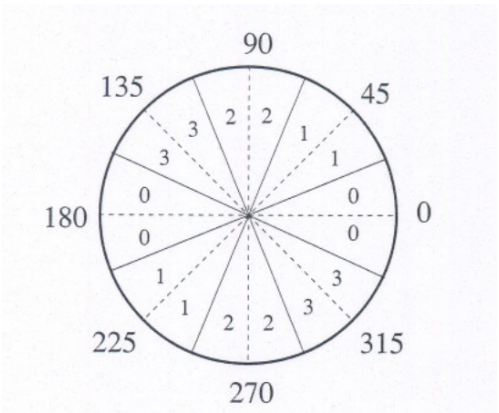
Gy

- Edge Gradient

$$G = \sqrt{G_x^2 + G_y^2} \text{ or } G = |G_x| + |G_y|$$

- Edge Direction

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$



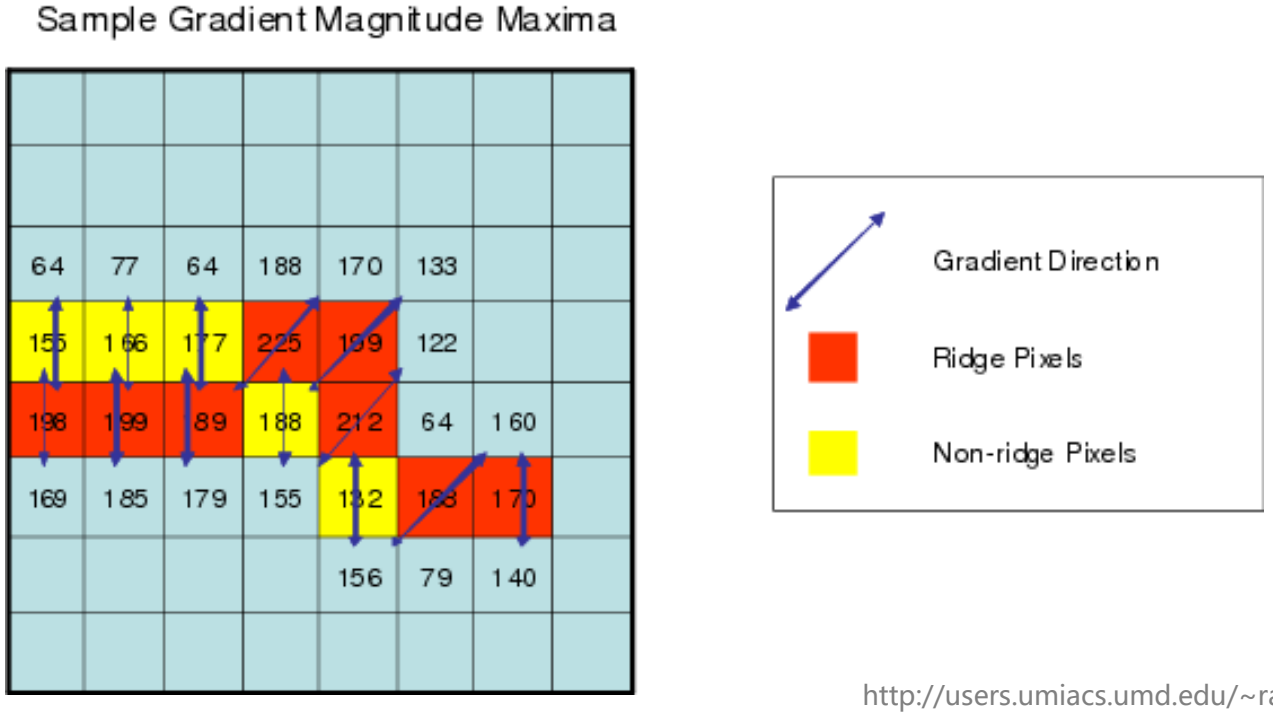
https://github.com/tanay-bits/cvlib/blob/master/Canny Edge Detector

(Step 2) Image Gradient: Result

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

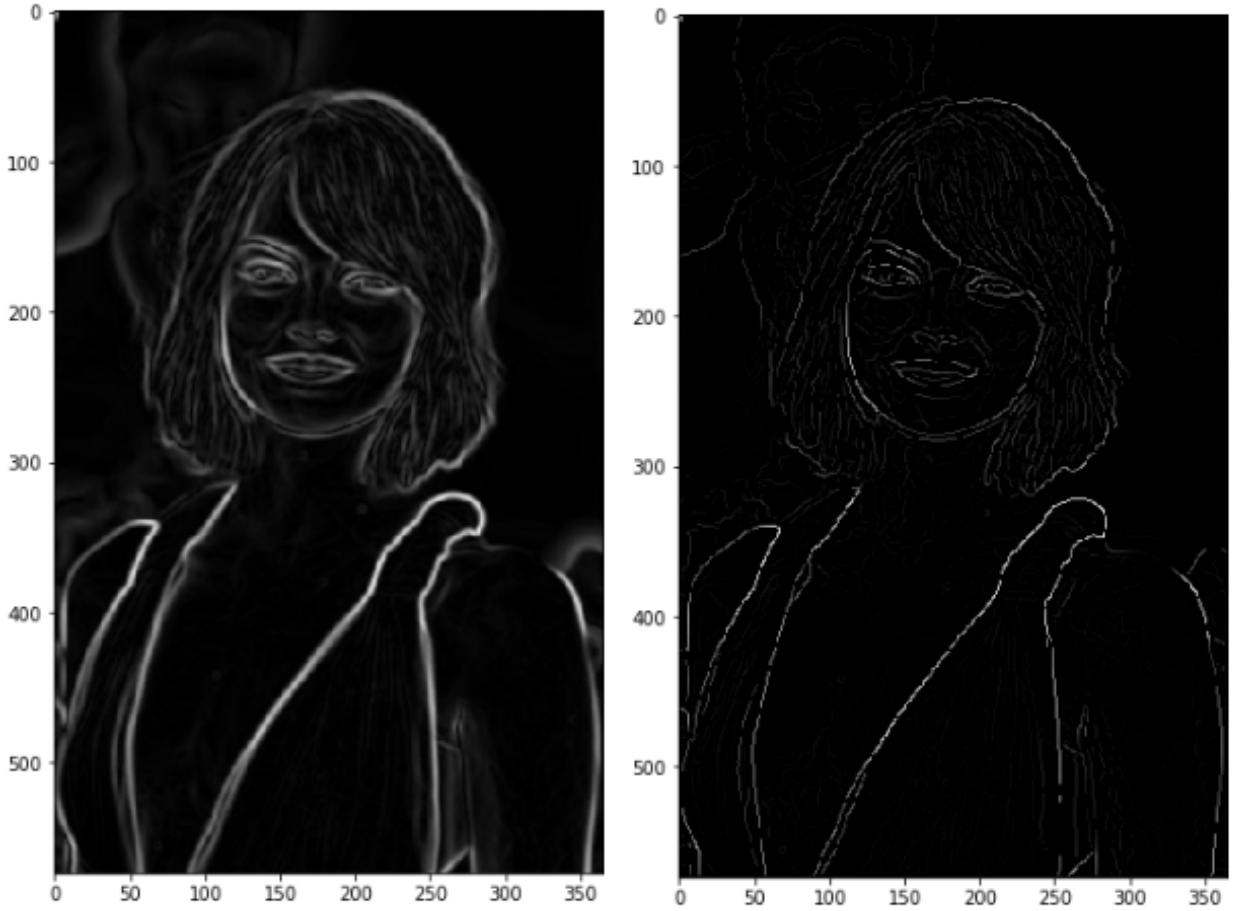
(Step 3) Non-Maximum Suppression: Algorithm

- 문제점: edge가 너무 두껍게 찾아진다. edge를 얇게 만들 수는 없을까? (Edge Thining)
  1. 각 pixel에 대해서 gradient direction(positive or negative)의 인접한 픽셀을 찾는다
  2. 각 pixel의 gradient 값이 인접한 pixel의 값보다 크면 local maximum, 크지 않으면 non-maximum이라고 하자.
  3. 각 pixel의 gradient 값이 non-maximum이면 gradient 값을 0으로 바꾼다. (suppression)



http://users.umiacs.umd.edu/~ramani/pubs/luo_gpu_canny_fin_2008.pdf

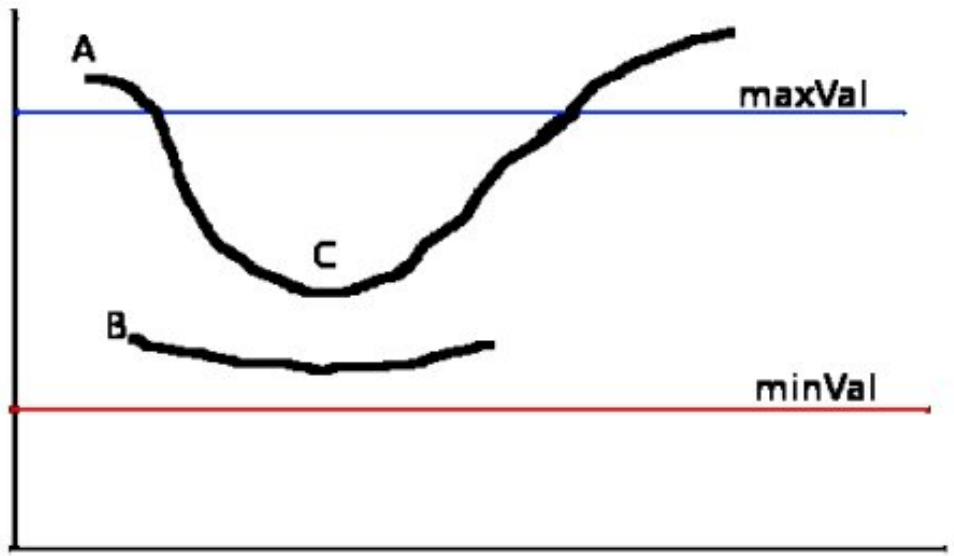
(Step 3) Non-Maximum Suppression: Result



https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

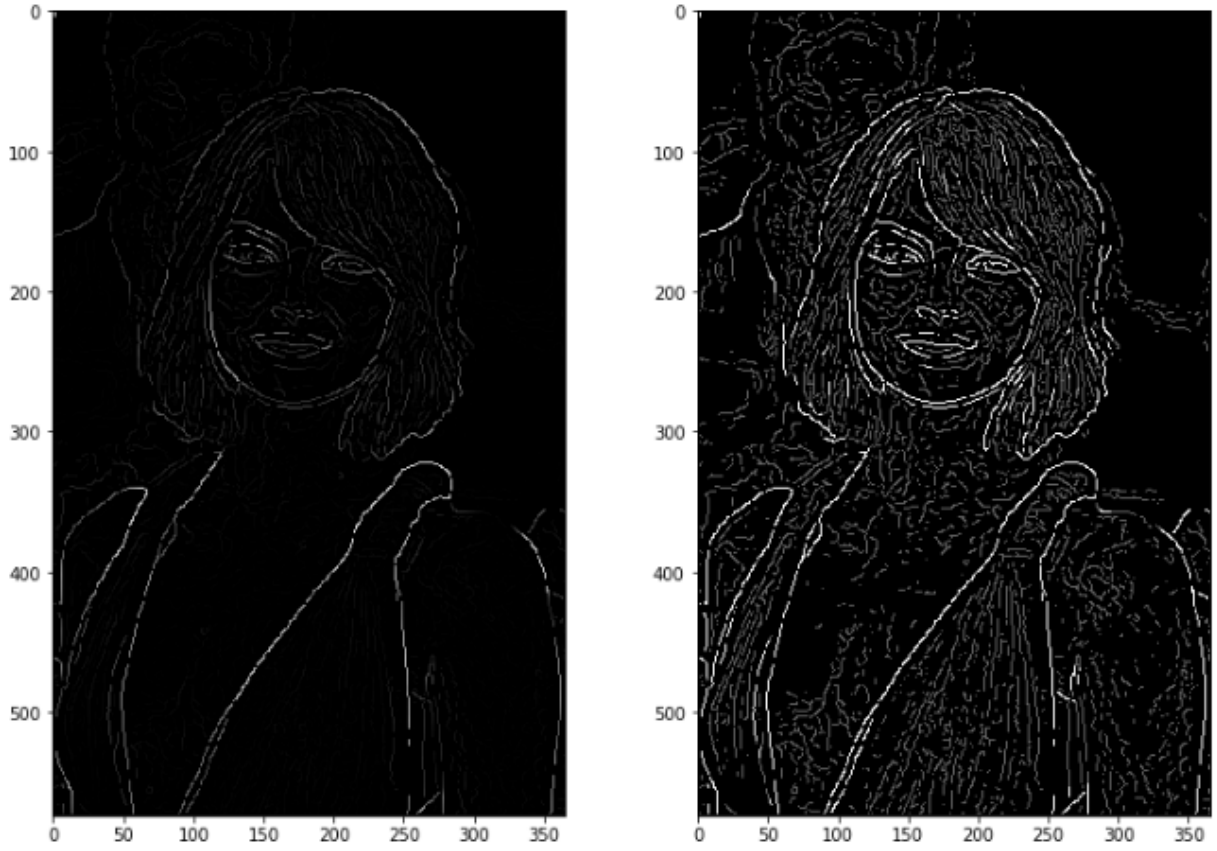
(Step 4) Double Thresholding: Algorithm

- 문제점: 옅은 edge도 찾아진다. 강한 edge만 남길 수 없을까?
  1. 각 pixel에 대해서 gradient 값이 `maxVal`보다 크면 edge로 남겨둔다. (strong edges)
  2. 각 pixel에 대해서 gradient 값이 `minVal`보다 작으면 edge를 버린다. (non-edges)
  3. 각 pixel에 대해서 gradient 값이 그 사이면 다음 단계를 위해서 남겨둔다. (weak edges)



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

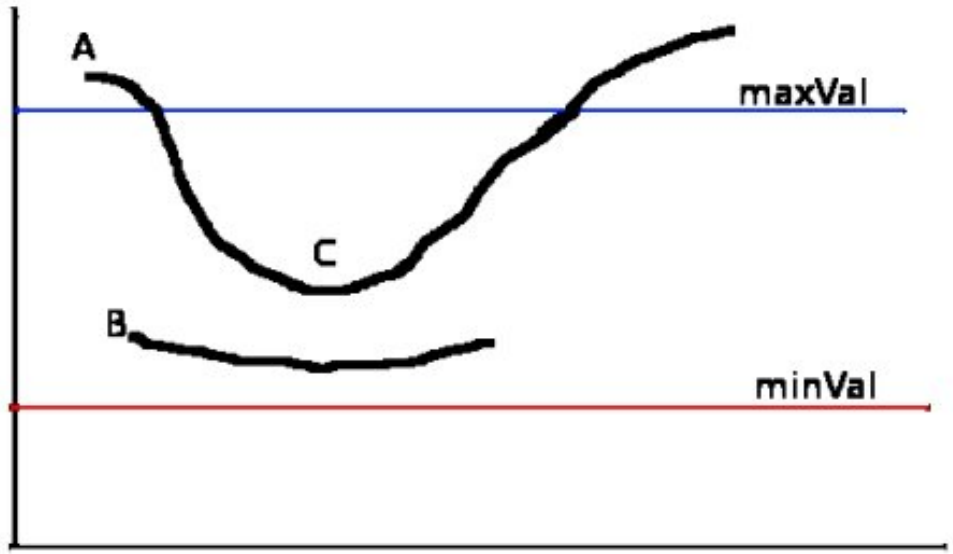
(Step 4) Double Thresholding: Result



https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

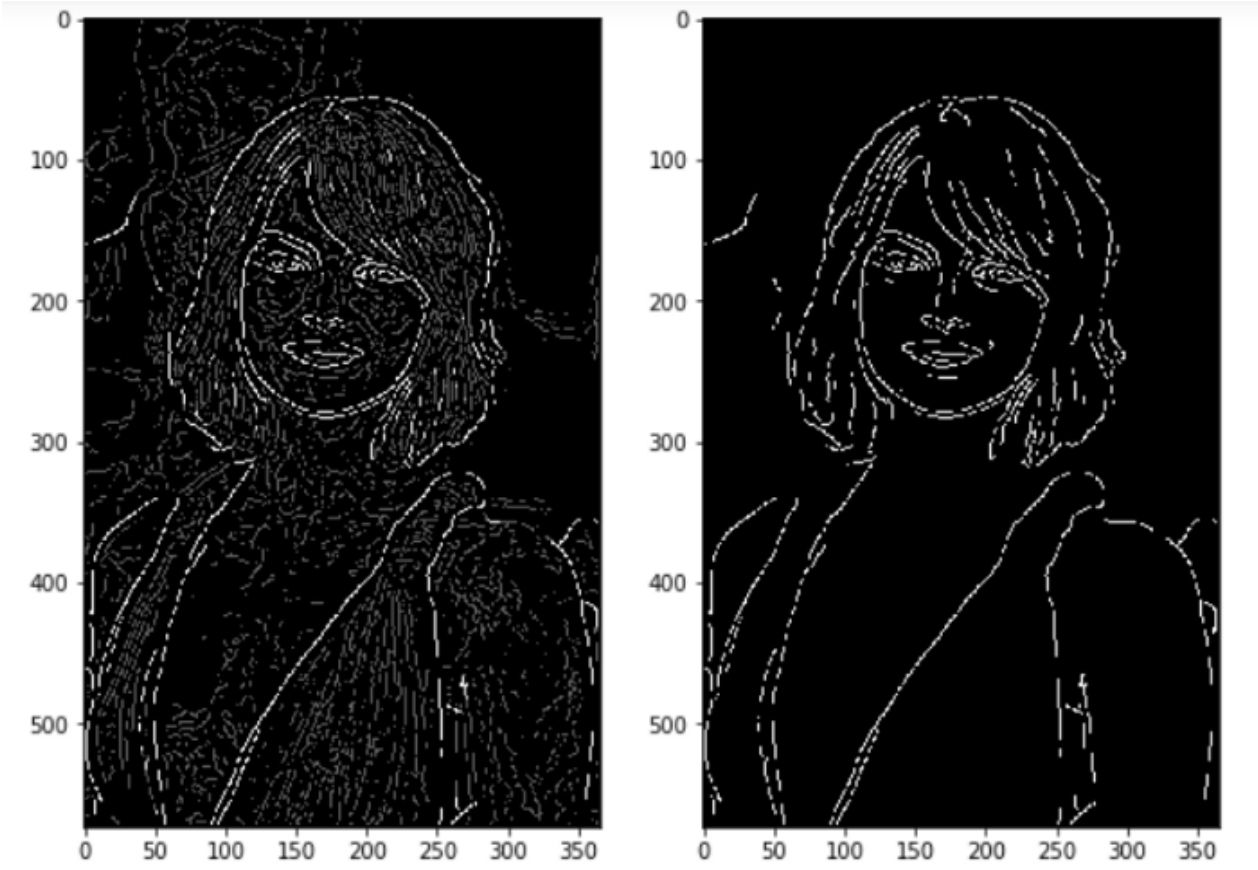
(Step 5) Hysteresis Thresholding: Algorithm

- 문제점: 짧은 edge도 찾아진다. 긴 edge만 남길수는 없을까?
  1. 각 pixel에 대해서 gradient 값이 `maxVal`보다 크면 edge로 남겨둔다. (strong edges)
  2. 각 pixel에 대해서 gradient 값이 `minVal`보다 작으면 edge를 버린다. (non-edges)
  3. 각 pixel에 대해서 gradient 값이 그 사이면 다음 단계를 위해서 남겨둔다. (weak edges)
     1. Strong edge에 연결된 weak edge는 남겨둔다.
     2. Strong edge에 연결되지 않은 weak edge는 버린다.



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

(Step 5) Hysteresis Thresholding: Result



https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

`cv.Canny()`[1]

```
edges = cv.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])
edges = cv.Canny(dx, dy, threshold1, threshold2[, edges[, L2gradient]])
```

- Canny 알고리즘을 이용해서 edge를 찾는다.
  - `image`: 8-bit input image
  - `dx`: 16-bit x derivative of input image (`CV_16SC1` or `CV_16SC3`)
  - `dy`: 16-bit y derivative of input image (`CV_16SC1` or `CV_16SC3`)
  - `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
  - `apertureSize`: aperture size for the Sobel operator
  - `L2gradient`: a flag, indicating whether a more accurate L2 norm should be used to calculate the image gradient magnitude (`L2gradient`=true), or whether the default L1 norm is enough (`L2gradient`=false)

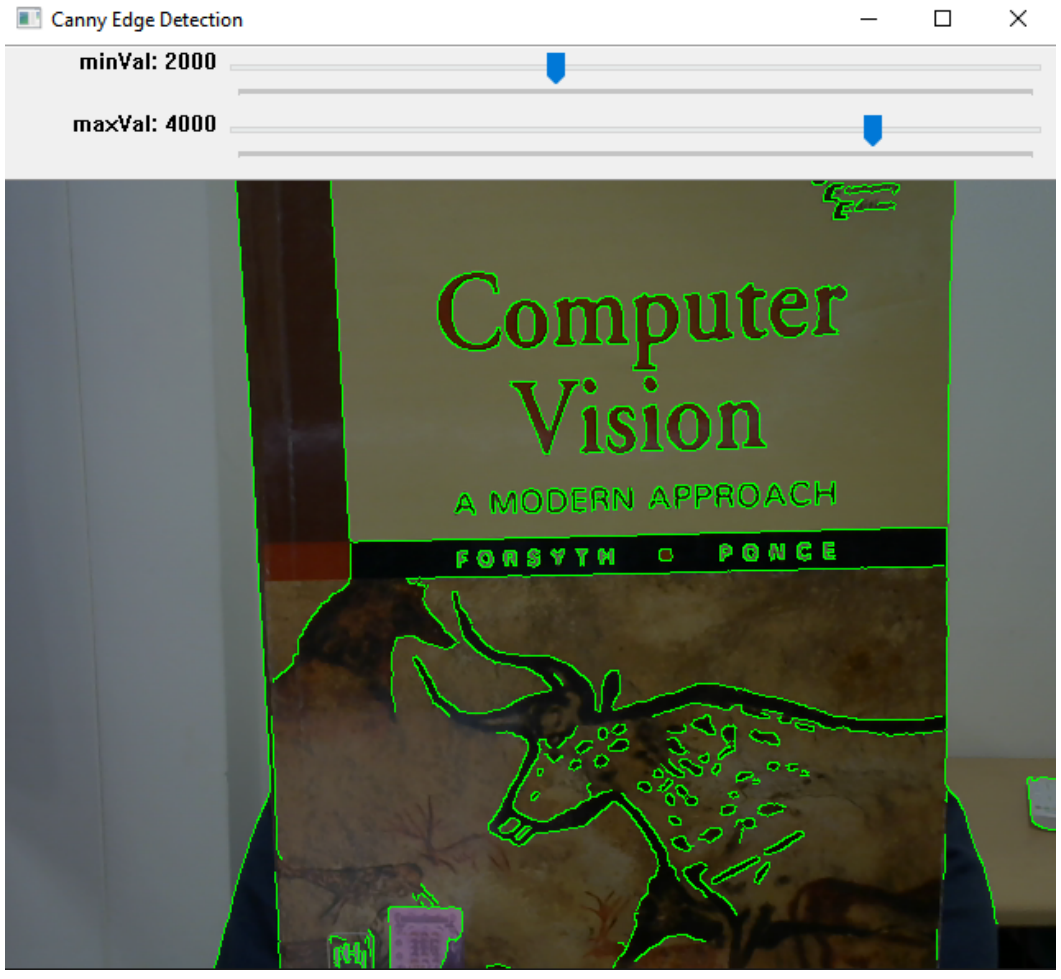
1. https://docs.opencv.org/4.4.0/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de

Canny Edge Detection: Code

```
import cv2 as cv
import numpy as np

def do_nothing(x):
    pass

# Create a video capture object
capture = cv.VideoCapture(0)

# Create a window and trackbars
winname = 'Canny Edge Detection'
cv.namedWindow(winname)
cv.createTrackbar('minVal', winname, 2000, 5000, do_nothing)
cv.createTrackbar('maxVal', winname, 4000, 5000, do_nothing)

# Infinite loop
while True:
    # Read a video frame
    ret, frame = capture.read()

    if ret is False:
```

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

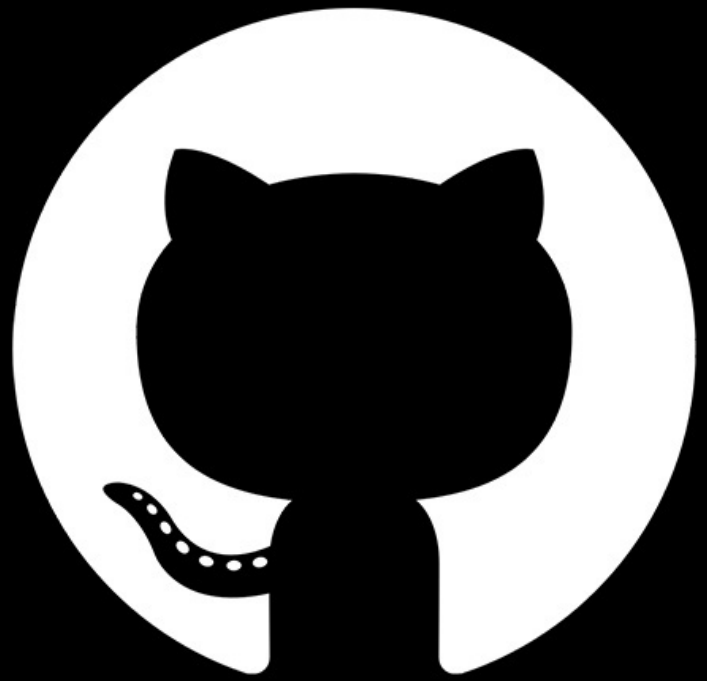
Canny Edge Detection: Result



Summary

- Smoothing: Kernel의 합이 1이다 (Weighted Average)
- Gradient: Kernel의 합이 0이다 (Constant $\Rightarrow$ 0)

Push Code to GitHub



References

References

- OpenCV Python Tutorials
  - Core Operations
    - Basic Operations on Images
    - Arithmetic Operations on Images
  - Image Processing
    - Image Thresholding
    - Smoothing Images
    - Morphological Transformations
    - Image Gradients