Gaussian Filtering and Edges

Pablo Mesejo

pmesejo@go.ugr.es

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

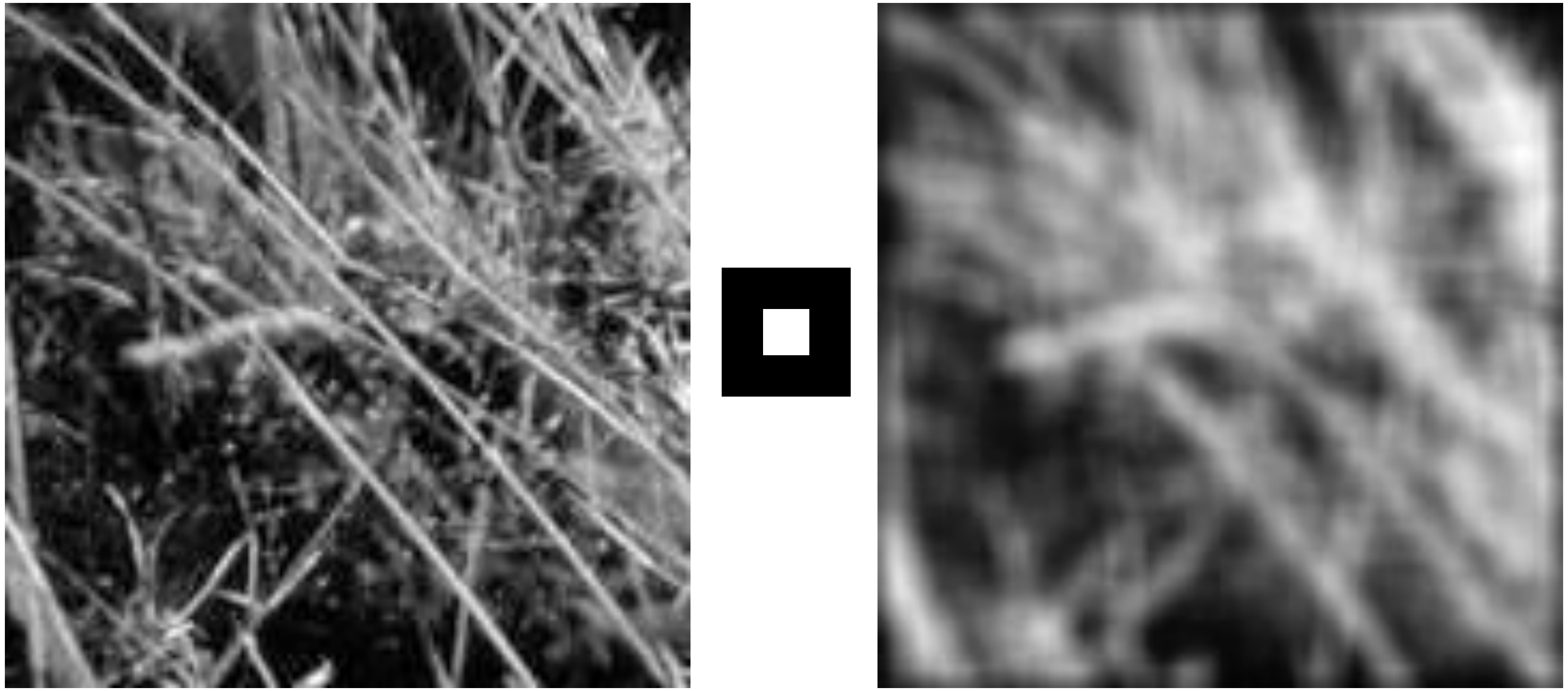
UNIVERSIDAD DE GRANADA

DECSAI

DaSCI

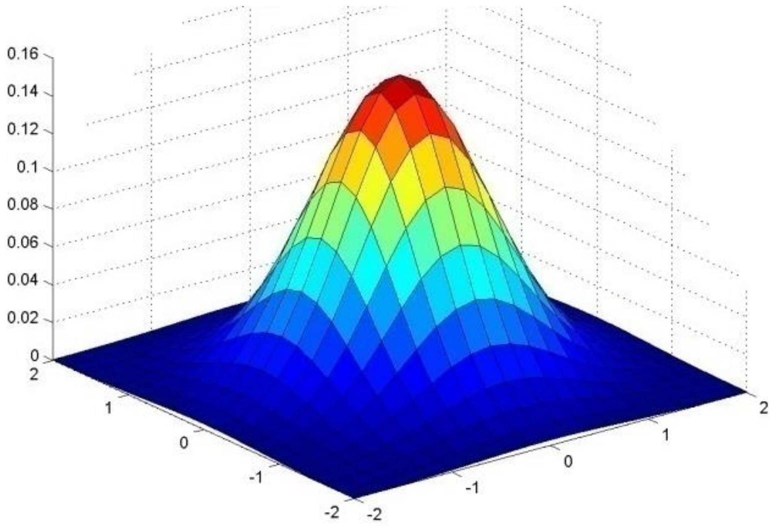
Instituto Andaluz de Investigación en **Data Science** and **Computational Intelligence**

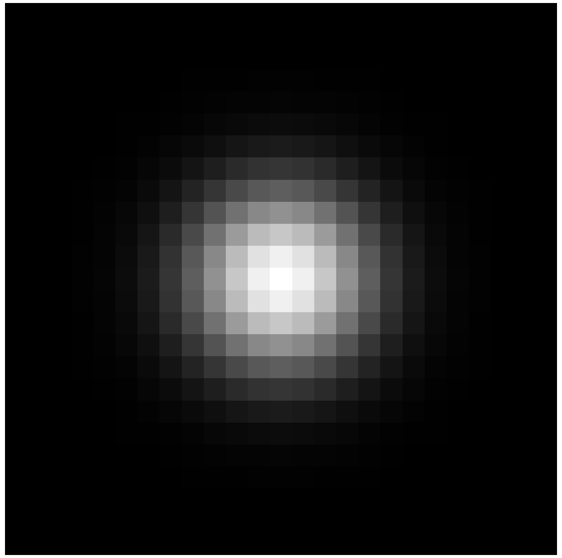
Smoothing with box filter



Source: D. Forsyth

Gaussian kernel



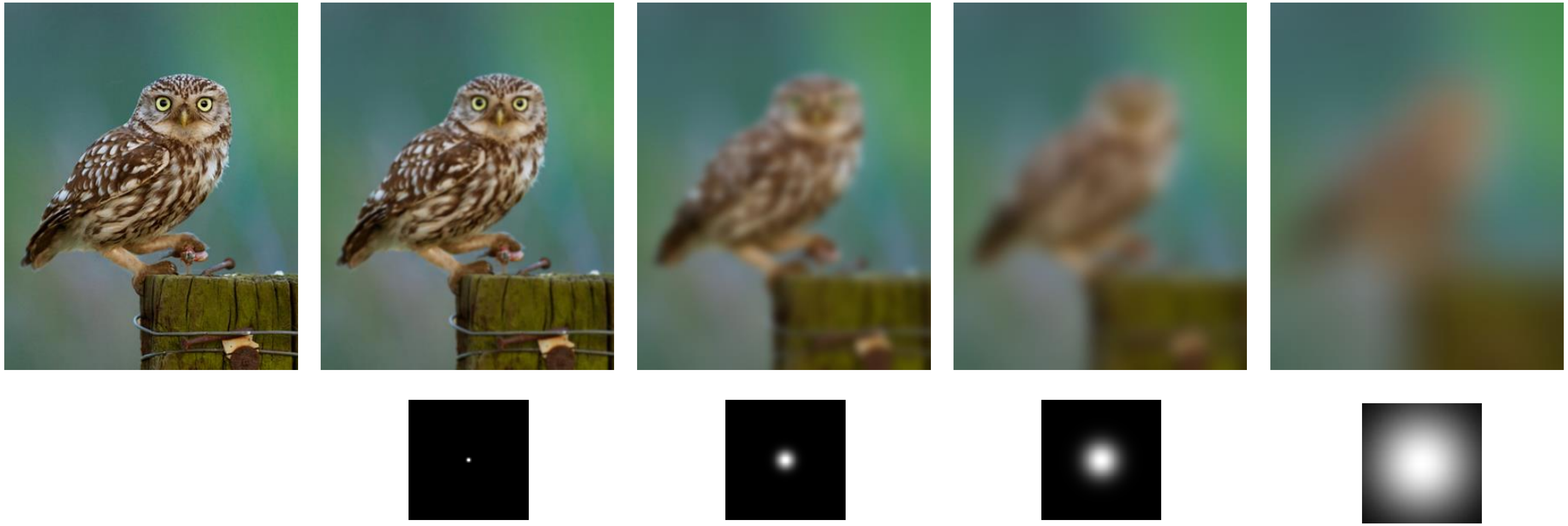


An isotropic (circularly symmetric) filter

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

- The coefficients are a 2D Gaussian.
- Gives more weight at the central pixels and less weight to the neighbors.
- The farther away the neighbors, the smaller the weight.

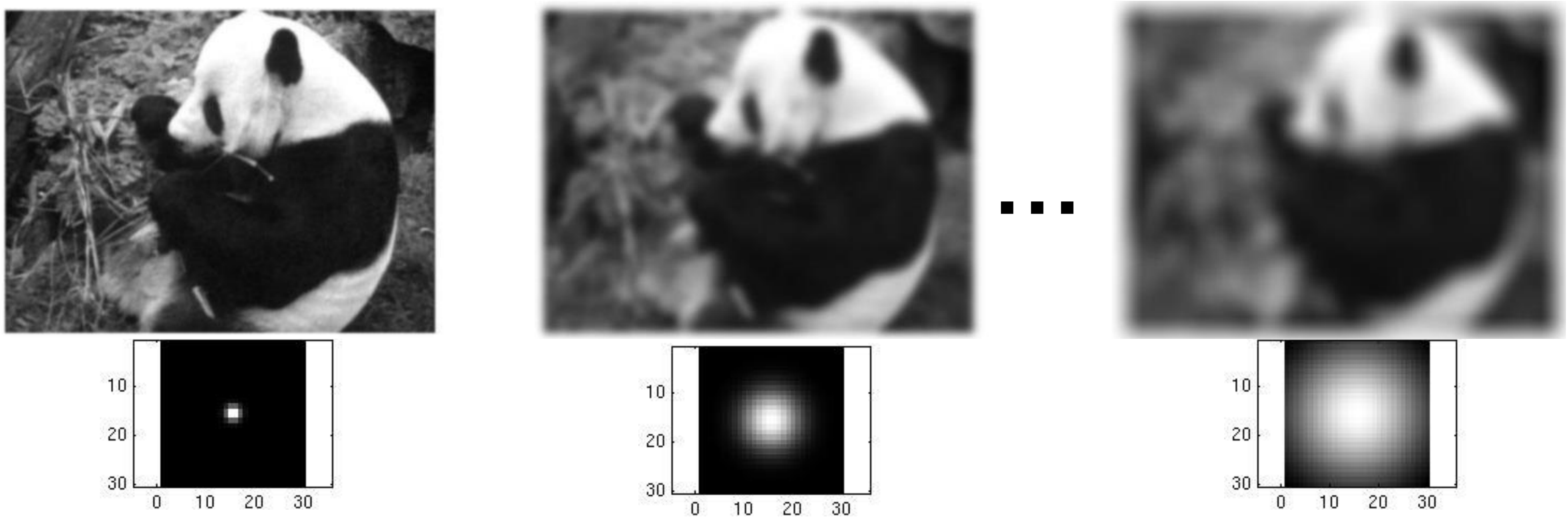
Gaussian filters



$\sigma = 1$ $\sigma = 5$ $\sigma = 10$ $\sigma = 30$

Smoothing with a Gaussian

Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

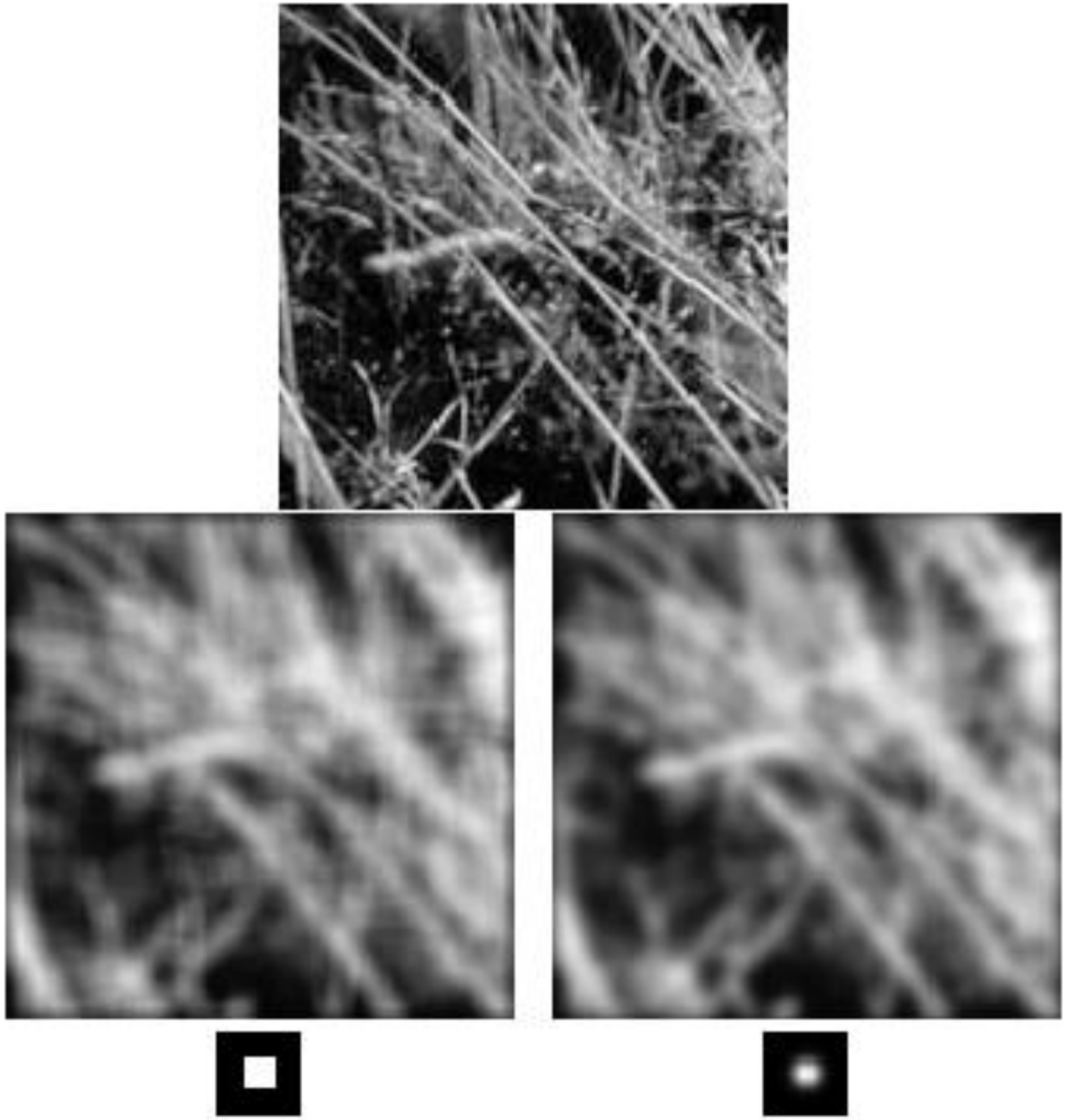



```
for sigma in np.arange(1, 10, 3):
        kernel=cv2.getGaussianKernel(ksize, sigma)
        outim=cv2.filter2D(im,-1,kernel) #correlation
        cv2.imshow(outim)
```

If you use Google Colab, `from google.colab.patches import cv2_imshow`

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

Mean vs. Gaussian filtering

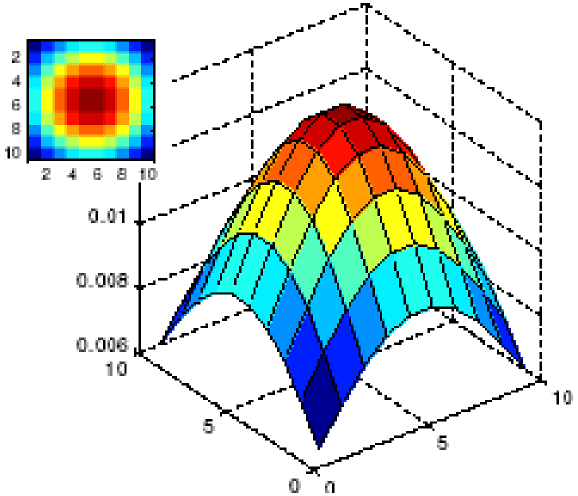


The box filter provides edgy artifacts!

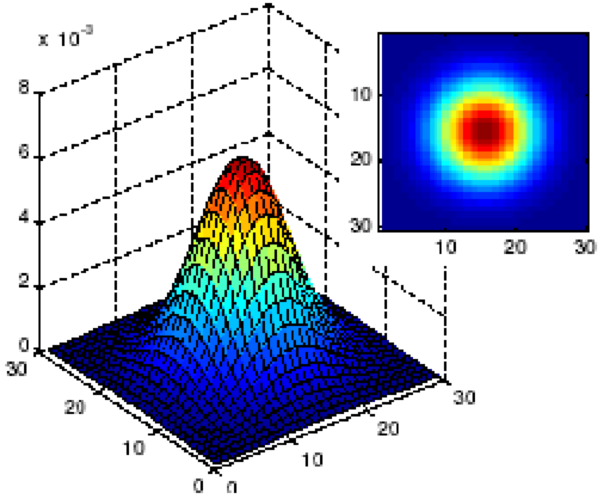
Gaussian is a better low-pass filter → it does not create high-frequency artifacts.

Gaussian filter

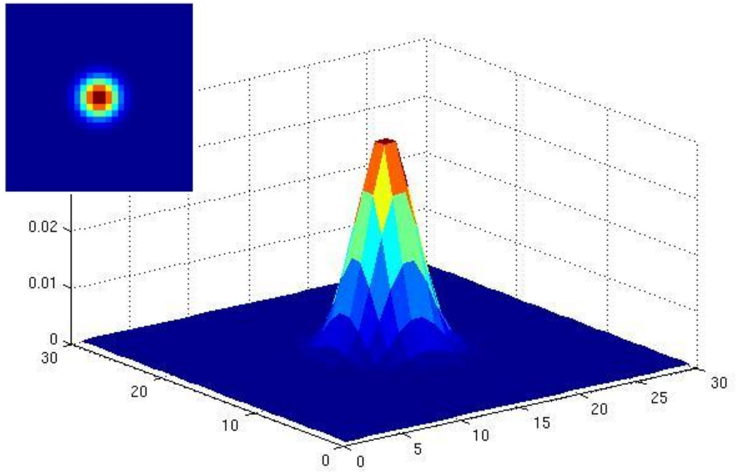
- What parameters matter here?
  - **Size** of kernel or mask
  - **Variance** of Gaussian determines extent of smoothing



σ = 5 with
10 x 10
kernel



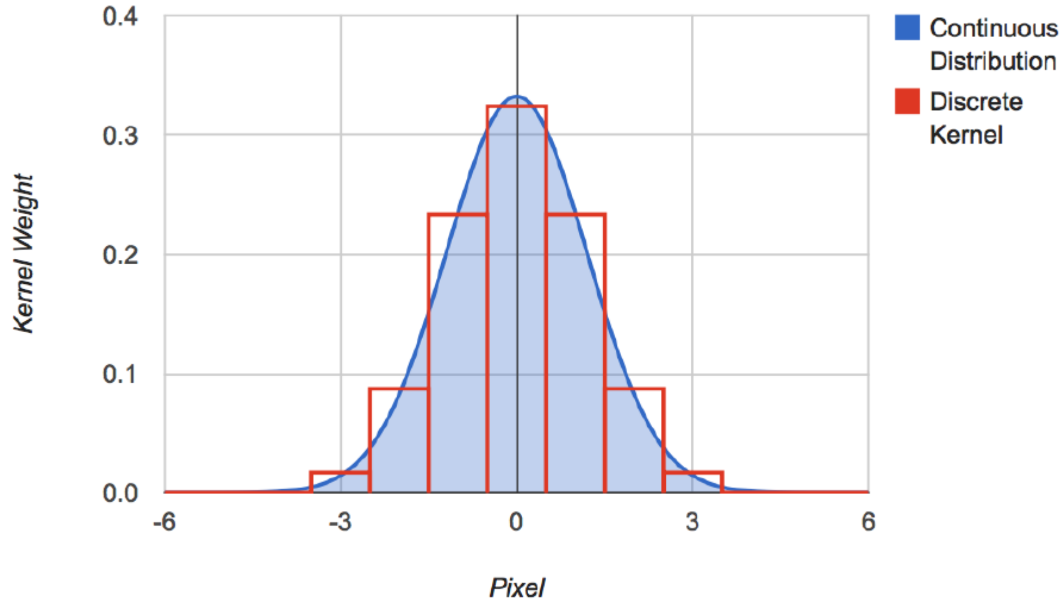
σ = 5 with
30 x 30
kernel



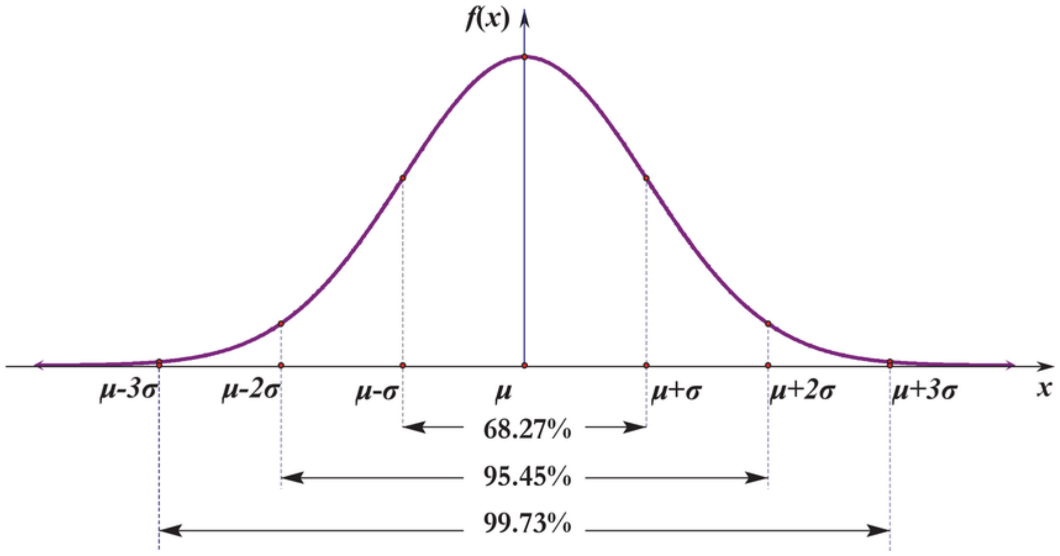
σ = 2 with
30 x 30
kernel

Gaussian discretization

We want to discretize a Gaussian.



We know that **almost all values are within 3 standard deviations of the mean** → It makes sense to use, **by default, 3σ.**



Gaussian discretization

$$f(x,\sigma) = c \cdot e^{-\frac{x^2}{2\sigma^2}}$$

We ignore this normalizing constant ($^1/_{\sigma\sqrt{2\pi}}$)**!** The shape of the filter is the same! We later normalize the kernel, making the sum of the coefficients equal to 1

Gaussian discrete mask 1D:
$[f(-k), f(-k+1), \cdots, f(0), \cdots, f(k-1), f(k)], \quad k \text{ an integer}$

If σ= 1 → kernel size is 7 (ie, *k* = 3)

According to the Gaussian properties, $\min(k) \geq 3\sigma$ provides samples covering more than 99% of the area under the curve

We enforce $\sum_{i=-k}^{k} f(i) = 1$

The mask must add up to 1 → you must divide your mask by the sum of its coefficients

Why?? 🤔

Gaussian discretization

We can go from σ to mask size (T), or viceversa, in order to get the Gaussian mask

If T is the mask size, T = 2·k+1 → k = (T-1)/2 → (T-1)/2 = 3·σ → 2·[3·σ] + 1 = T

Let's see an example. We have T=5:

- We'd have 5 positions: [f(-2), f(-1), f(0), f(1), f(2)]

- We compute the suitable sigma: (T-1)/2 = 3·σ → (5-1)/2 = 3·σ → 2 = 3·σ → σ = 2/3 = 0.67
- We compute the values of the mask substituting $x = \{-2, -1, 0, 1, 2\}$ on $f(x) = e^{-\frac{x^2}{2 \cdot 0.67^2}}$

Once we have σ and *k* we can discretize the mask by applying the Gaussian function (or its derivatives)

Gaussian discretization: finite approximation

Convolving a box filter with itself yields an approximation to a Gaussian kernel (easy way to construct a Gaussian kernel):

$$\frac{1}{2}\begin{bmatrix}1 & 1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 1\end{bmatrix} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

$$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

These are the odd rows of the binomial (Pascal's) triangle:

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & \boxed{1} & & \boxed{2} & & \boxed{1} & & \\ & 1 & & 3 & & 3 & & 1 & \\ \boxed{1} & & \boxed{4} & & \boxed{6} & & \boxed{4} & & \boxed{1} \end{array}$$

k=1, σ^2=0.5, σ=0.7

k=2, σ^2=1, σ=1

(2k+1)th row approximates Gaussian with σ^2=k/2

Check Stan Birchfield's notes for more information regarding the binomial and trinomial triangle to generate Gaussian approximations

Gaussian Properties

- Remove "high-frequency" components from the image (*low-pass filter*). All values are positive and sum up to 1.
  - Images become smoother.

- Completely described by 1st (mean) and 2nd (variance) order statistics (moments).

- The Fourier Transform of a Gaussian function is also a Gaussian

$$f(x) = e^{-\frac{x^2}{2\sigma^2}} \Longrightarrow \mathcal{F}[f(x)] = \sqrt{2\pi}\sigma e^{-2\pi^2\sigma^2\omega^2}$$

Gaussian Properties

- Closed under convolution (convolution of two Gaussians is another Gaussian)

  - When you convolve a Gaussian function (G_1) with another Gaussian function (G_2), the result is still a Gaussian function (G)
  - Convolving twice with Gaussian kernel of width σ is the same as convolving once with kernel of width $\sqrt{\sigma^2 + \sigma^2} = \sqrt{2\sigma^2} = \sigma\sqrt{2}$
  - In general, n repeated convolutions with σ Gaussian approximates single convolution with $\sigma\sqrt{n}$ Gaussian

$$\boxed{G(x, y|\sigma) = G\left(x, y \,\middle|\, \sqrt{\sigma_1^2 + \sigma_2^2}\right) = G_1(x, y|\sigma_1) * G_2(x, y|\sigma_2)}$$

https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables#Proof_using_convolutions

Gaussian Properties

- Separable (efficient)
  - Factors into product of two 1D Gaussians
  - Example:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \Longleftrightarrow$$

```python
import scipy.signal as sp
import numpy as np

np.outer(np.array([1,2,1]), np.array([1,2,1]))

sp.convolve2d(np.array([[0,1,0],[0,2,0], [0,1,0]]),
              np.array([[0,0,0],[1,2,1], [0,0,0]]), mode='same')
```

Outer product:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

Separability of the Gaussian

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

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

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Source: D. Lowe

Separability example

2D filtering (center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

$$\begin{aligned} &= 2 + 6 + 3 = 11 \\ &= 6 + 20 + 10 = 36 \\ &= 4 + 8 + 6 = 18 \\ &\overline{\quad 65 \quad} \end{aligned}$$

The filter factors into a product of 1D filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform filtering along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

Followed by filtering along the remaining column:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix} = \begin{bmatrix} & & \\ & 65 & \\ & & \end{bmatrix}$$

Source: K. Grauman

Separability example

We combine information from different rows
(kernel Y - Coefficients for filtering each column)

```
[1]
[2]
[1]
```

```
[ 0,  0,  0,  0,  0]
[ 0, 10, 20, 30,  0]
[ 0, 40, 50, 60,  0]
[ 0, 70, 80, 90,  0]
[ 0,  0,  0,  0,  0]
```

We combine information from different columns
(kernel X - Coefficients for filtering each row)

```
[1, 2, 1]
```

```
[   0,   0,   0,   0,   0]
[   0,  60,  90, 120,   0]
[   0, 160, 200, 240,   0]
[   0, 180, 210, 240,   0]
[   0,   0,   0,   0,   0]
```

```
[   0,   0,   0,   0,   0]
[   0, 210, 360, 330,   0]
[   0, 520, 800, 680,   0]
[   0, 570, 840, 690,   0]
[   0,   0,   0,   0,   0]
```

Input Image

```
[ 0,  0,  0,  0,  0]
[ 0, 10, 20, 30,  0]
[ 0, 40, 50, 60,  0]
[ 0, 70, 80, 90,  0]
[ 0,  0,  0,  0,  0]
```

Kernel

```
[1, 2, 1]
```

http://www.songho.ca/dsp/convolution/convolution2d_separable.html

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Separable Kernel

Separability example (technical note)

It is not necessary to always go through all the rows and columns (4 for-loops)!!!
For each 1D convolution, we only need to traverse the rows once!!!

```
[1. 1. 1. 1. 1.]
[2. 2. 2. 2. 2.]
[1. 1. 1. 1. 1.]
```

```
[ 0,  0,  0,  0,  0]
[ 0, 10, 20, 30,  0]
[ 0, 40, 50, 60,  0]
[ 0, 70, 80, 90,  0]
[ 0,  0,  0,  0,  0]
```

→

```
[   0,   0,   0,   0,   0]
[   0,  60,  90, 120,   0]
[   0, 160, 200, 240,   0]
[   0, 180, 210, 240,   0]
[   0,   0,   0,   0,   0]
```

↓ T

```
[   0   0   0   0   0]
[   0  60 160 180   0]
[   0  90 200 210   0]
[   0 120 240 240   0]
[   0   0   0   0   0]
```

```
[1. 1. 1. 1. 1.]
 2. 2. 2. 2.]
 1. 1. 1. 1.]
```

T

←

```
[   0,   0,   0,   0,   0]
[   0, 210, 360, 330,   0]
[   0, 520, 800, 680,   0]
[   0, 570, 840, 690,   0]
[   0,   0,   0,   0,   0]
```

Why is separability useful in practice?

- Separability means that a 2D convolution can be reduced to two 1D convolutions (one among rows and one among columns)
- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  – $O(n^2 m^2)$

We move the $m \times m$ filter through the whole image. So, we do $m \cdot m$ operations for each of the $n \cdot n$ pixels in the image: $m^2 \cdot n^2$ operations

- What if the kernel is separable?
  – $O(n^2 m)$

We move the $1 \times m$ filter through the whole image. So, we do m operations for each of the $n \cdot n$ pixels in the image (one per rows and another per columns): $m \cdot n \cdot n + m \cdot n \cdot n = 2 \cdot (m \cdot n^2)$

Example: 512x512 image, 11x11 filter
- 1 convolution 2D: 512*512*11*11 ~32M multiplications
- 2 convolutions 1D: 512*512*11 + 512*512*11 ~6M multiplications

More ideas about separability

- Allowed because convolution is associative ($f * (g_1 * g_2) = (f * g_1) * g_2$)
- Some 2D kernels can be directly decomposed into 1D kernels:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

$$\frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- A 2D kernel is *separable* iff all rows / columns are linearly dependent
  – This filter

```
0  1  0
1 -4  1
0  1  0
```

is not separable:

$$\alpha \cdot [0, 1, 0] + \beta \cdot [1, -4, 1] = [0, 0, 0] \leftrightarrow \alpha = 0 \land \beta = 0$$

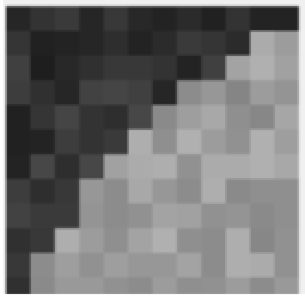
→ Linearly independent vectors!!

Note: any 2D-mask admits a SVD decomposition (*low-rank approximations*): sum of several separable kernels.
https://bartwronski.com/2020/02/03/separate-your-filters-svd-and-low-rank-approximation-of-image-filters/

Computing derivatives: high-frequency features (high-pass filters)

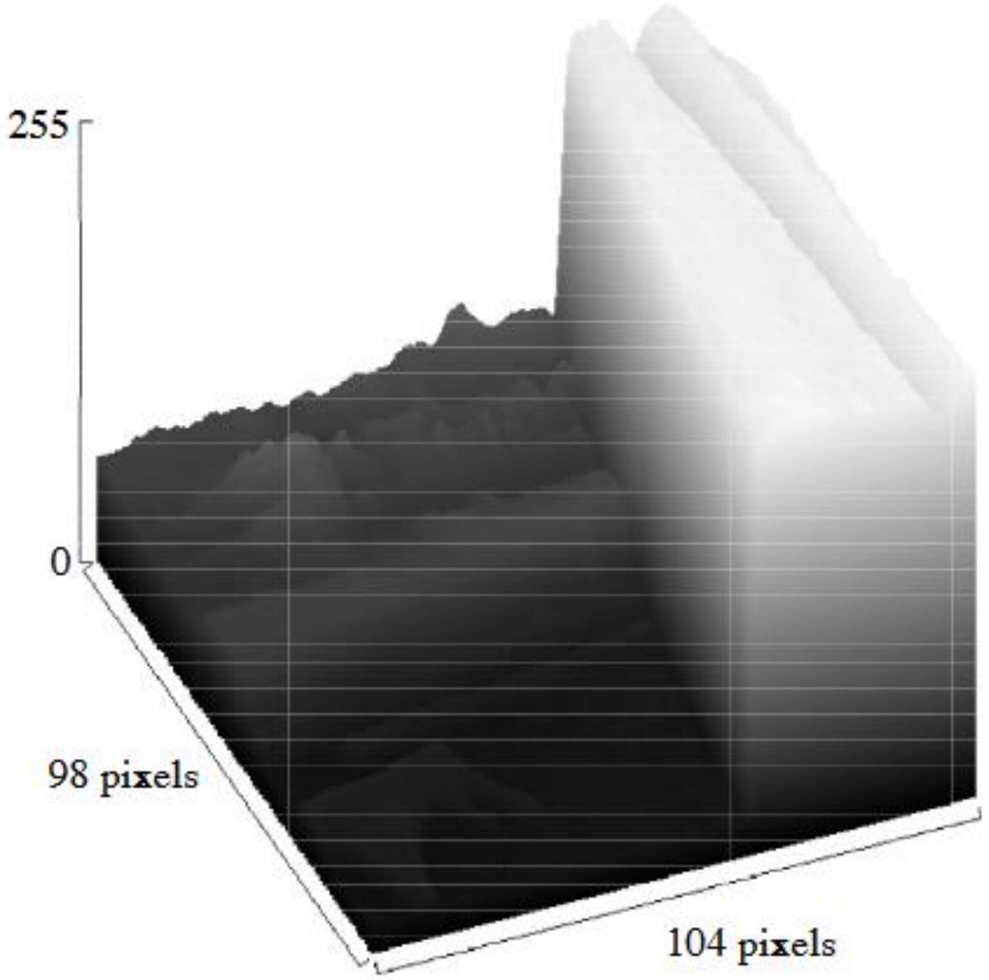
Edges

	24	25	26	27	28	29	30	31	32	33	34	35
24	15	20	23	15	22	16	14	17	13	20	14	14
25	21	13	14	15	18	14	17	23	21	17	67	61
26	25	12	16	19	22	22	20	14	25	64	69	63
27	24	19	15	26	27	25	15	55	59	52	61	58
28	13	21	27	20	18	28	54	62	66	56	53	65
29	13	14	24	20	23	68	56	69	61	57	69	62
30	14	28	18	28	63	67	68	57	67	67	69	66
31	24	18	22	59	55	66	59	55	68	54	56	56
32	26	23	23	58	55	57	64	63	57	58	54	56
33	19	22	68	61	56	64	69	56	55	68	53	57
34	22	55	62	57	62	59	59	64	55	68	67	57
35	19	54	60	60	57	55	61	56	57	61	55	60



https://cs.wellesley.edu/~cs332/doc/edgeNotes.pdf

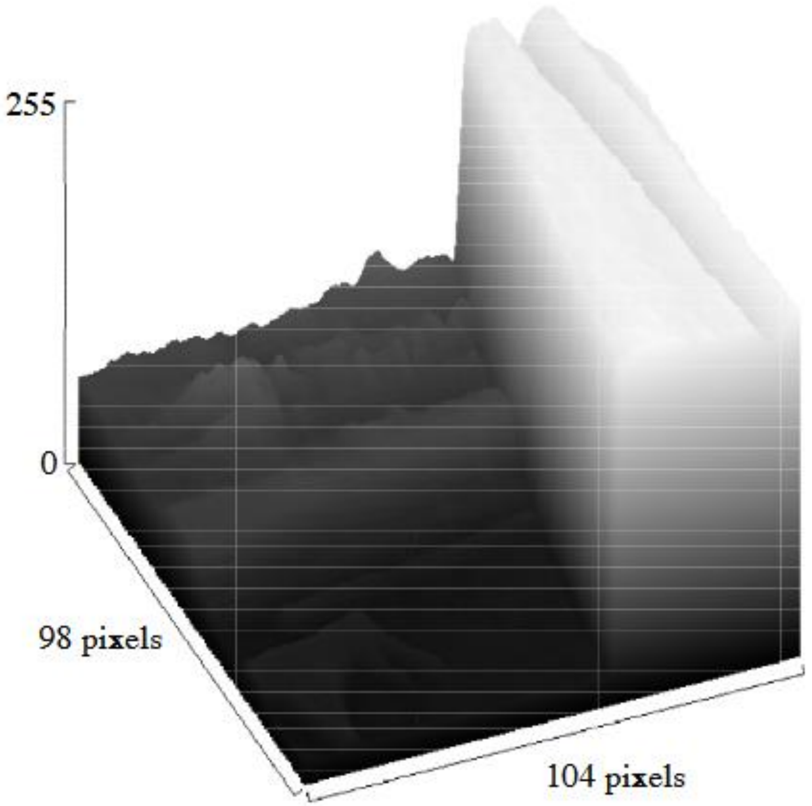
Edges



https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D02-Images.pdf

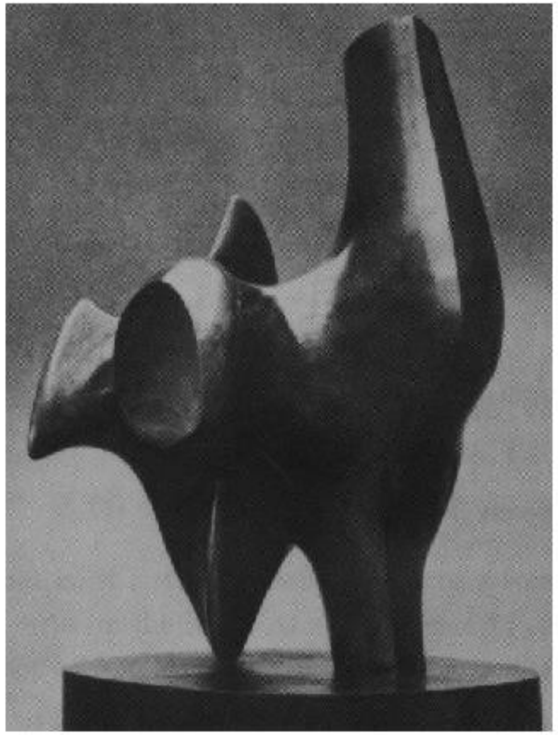
Edges and High Frequencies

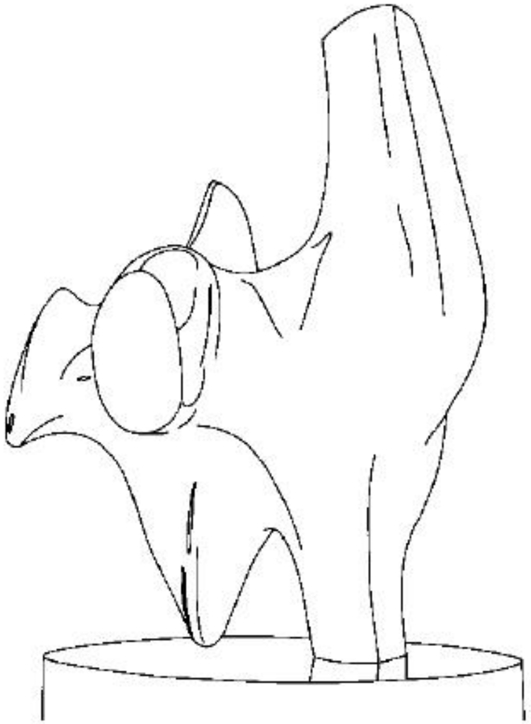
- Frequency in 2D images:
  - rate of change of gray levels in spatial terms
    - If it “involves” many pixels to make a change
      → low frequency
    - If it “involves” few pixels to make a change (i.e., the change is abrupt)
      → high frequency



- We’ll come back to this later!

Edge detection





- **Convert a 2D image into a set of curves**
  - **Extracts salient features of the scene**
  - **More compact than pixels**

Importance of intensity edges

from Walther et al., Simple line drawings suffice for functional MRI decoding of natural scene categories, PNAS 2011

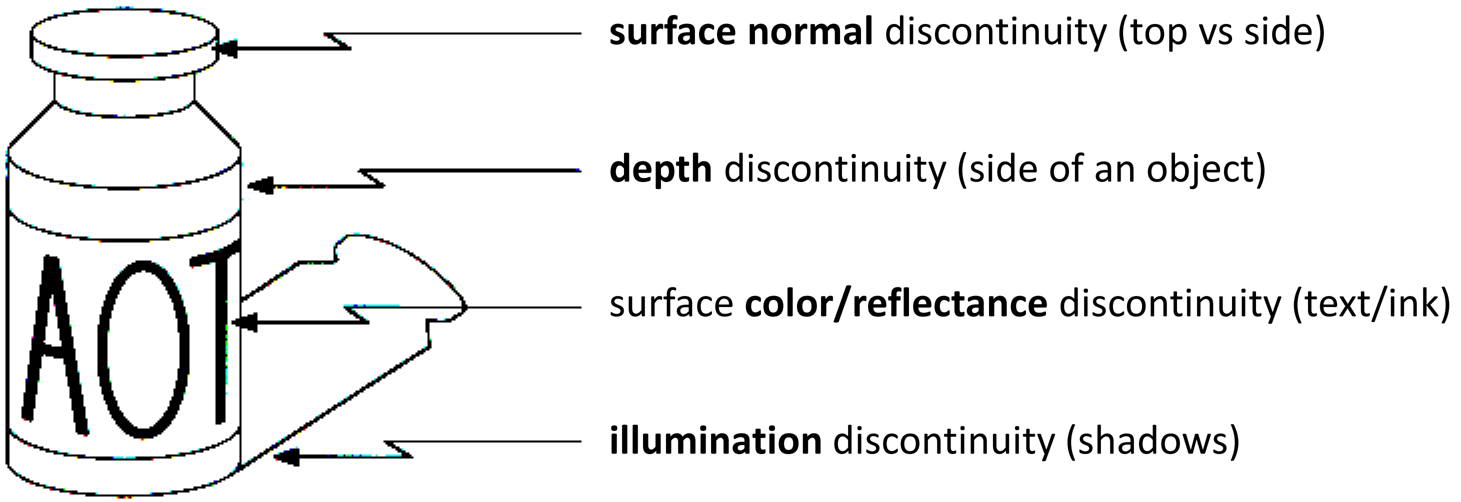
→ ***Much information is retained by the edges***

Importance of intensity edges

from Walther et al., Simple line drawings suffice for functional MRI decoding of natural scene categories, PNAS 2011

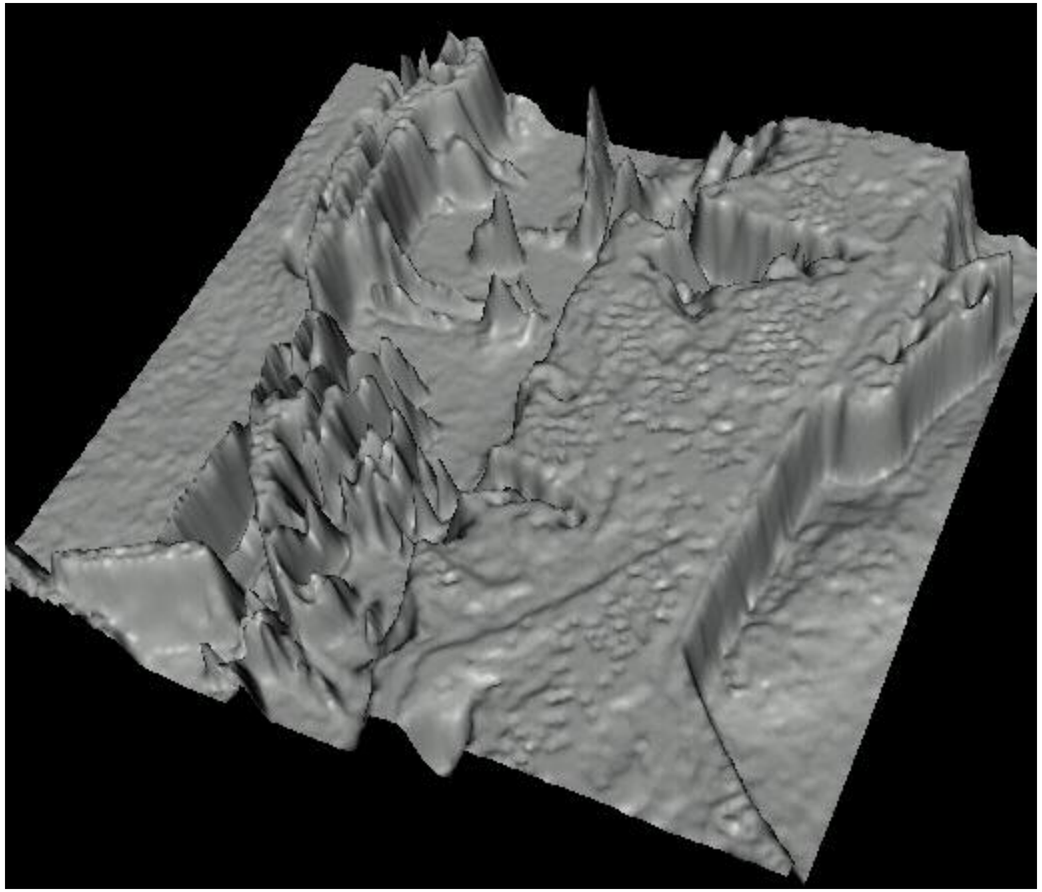
→ Much information is retained by the edges

Origin of edges



- Edges are caused by a variety of factors

Images as functions…

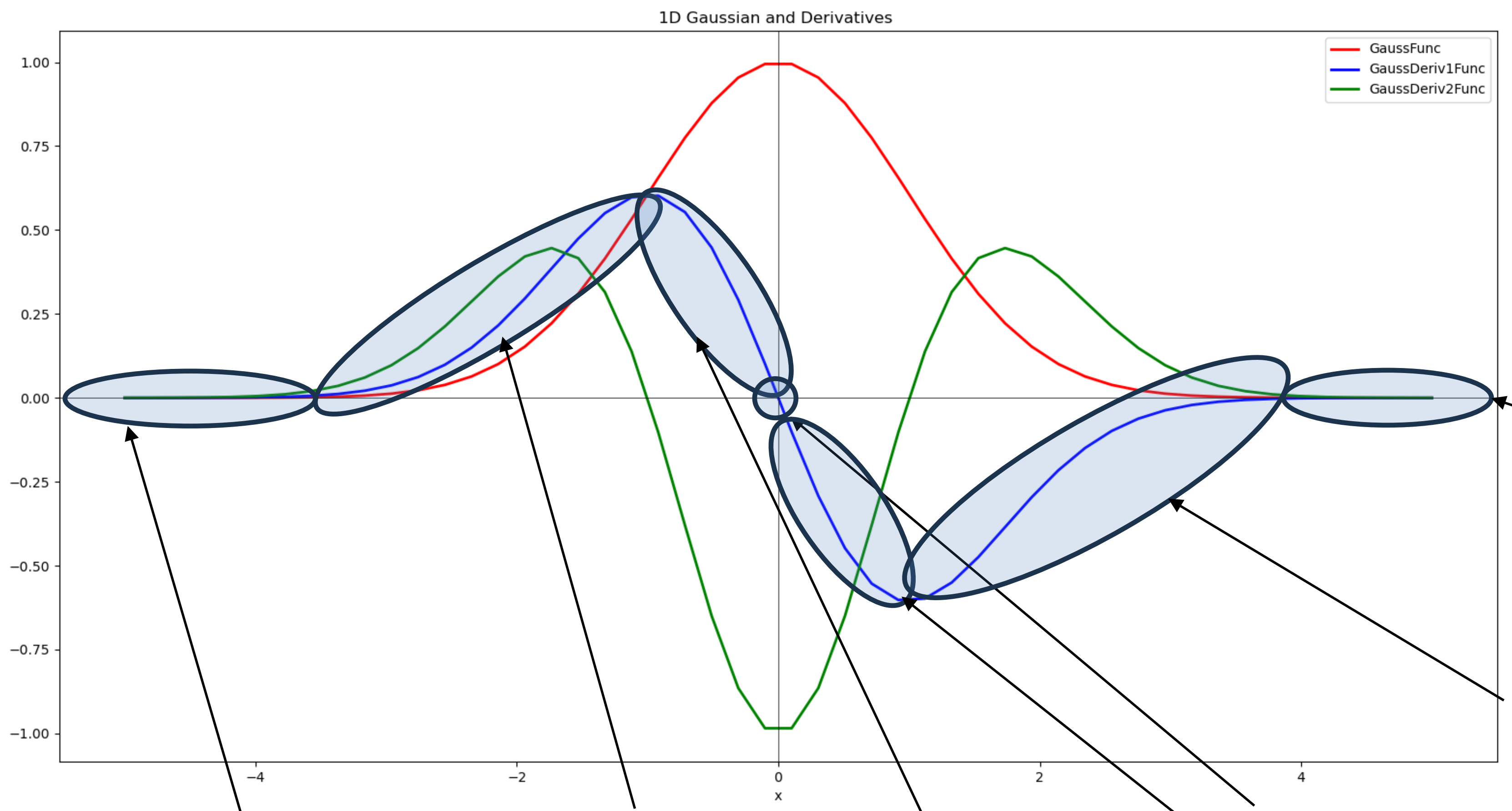


- Edges look like steep cliffs

Remembering the concept of derivative

The derivative shows the **sensitivity of change** of a function's output with respect to the input

Remember that **the derivative** of a function at a chosen input value, when it exists, **is the slope of the tangent line** of the function **at that point**.



No variation → derivative=0

Decreasing negative slope

Increasing positive slope

Slope at maximum is zero (No variation) → derivative=0

No variation → derivative=0

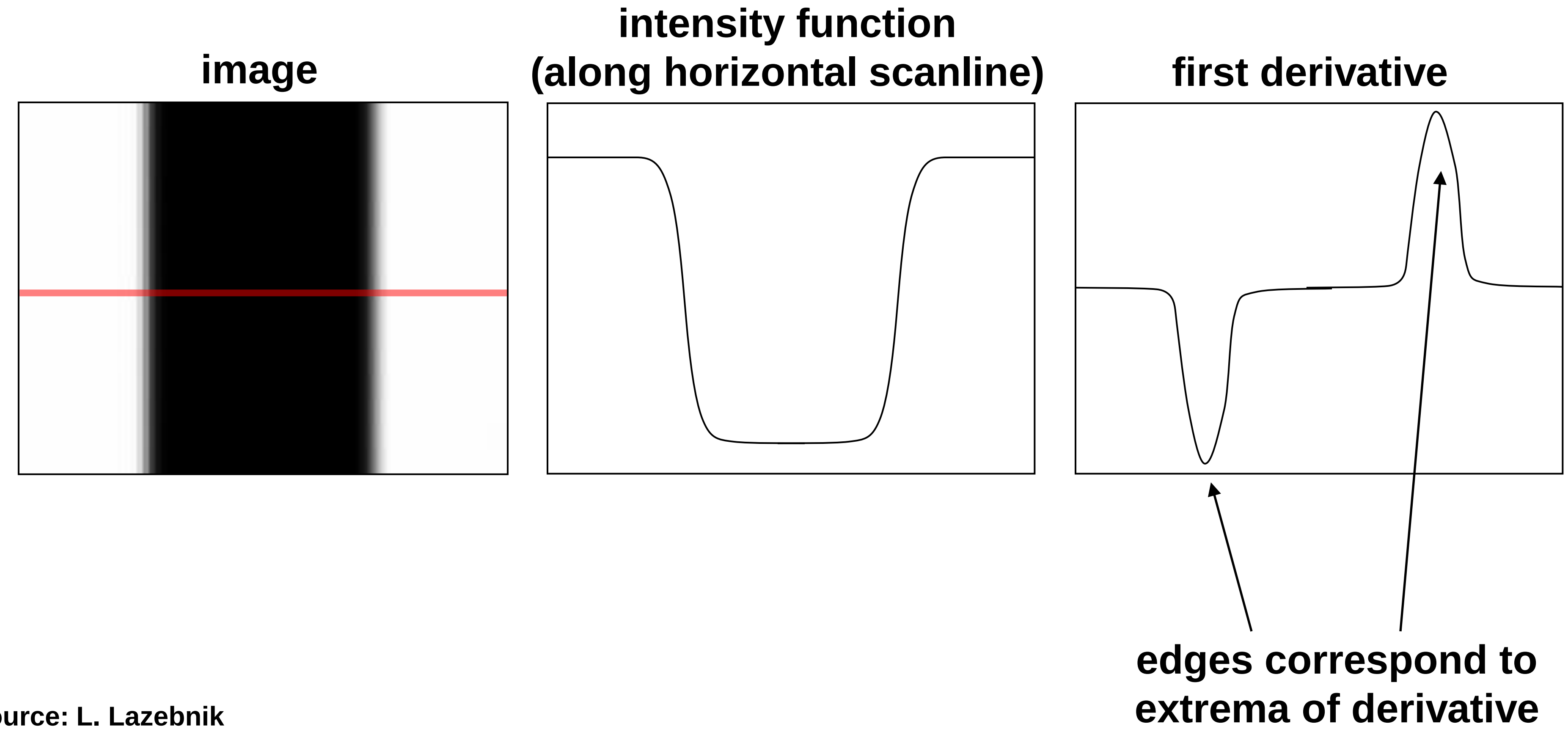
Decreasing positive slope

Increasing negative slope

Let's pay attention to the **blue** line (first derivative of the **red** one).
This will help us to better understand the next slides.

Characterizing edges

- **An edge is a place of *rapid change* in the image intensity function**



Source: L. Lazebnik

Image derivatives

For a 2D function, $f(x, y)$, the partial derivative is

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

For discrete data, we can approximate the derivative using finite differences:

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

We compute derivatives through the **difference between intensities of adjacent pixels**.

Image derivatives

- **How can we differentiate a digital image F[x,y]?**
  - **Take discrete derivative (finite difference)**

$$\frac{\partial f}{\partial x}[x,y] \approx F[x+1,y] - F[x,y]$$

How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:

1	-1	

H_x

$\frac{\partial f}{\partial y}$:

	-1	
	1	

H_y

Source: S. Seitz

Image derivatives

- Odd length avoids undesirable shifting of signal

$[1 \quad \underline{-1}]$

Forward difference

$[4 \quad 7 \quad 8 \quad 1 \quad 3] * [1 \quad \underline{-1}] = [3 \quad 1 \quad -7 \quad 2 \quad 0]$

(Don't forget to flip kernel before convolving)

The position to be replaced appears underlined.

In this example, we reflect the boundary. So, the value outside the image would be 3: -1*3+1*3=0

$[\underline{1} \quad -1]$

Backward difference

$[4 \quad 7 \quad 8 \quad 1 \quad 3] * [\underline{1} \quad -1] = [0 \quad 3 \quad 1 \quad -7 \quad 2]$

- To avoid undesirable shift, use central difference kernel

$\frac{1}{2}[1 \quad \underline{0} \quad -1]$

Central difference

$h = 2$

Forward	$f'(x) \approx \dfrac{f(x+h) - f(x)}{h}$
Backward	$f'(x) \approx \dfrac{f(x) - f(x-h)}{h}$
Central	$f'(x) \approx \dfrac{f(x+0.5h) - f(x-0.5h)}{h}$

Example:

$[4 \quad 7 \quad 8 \quad 1 \quad 3] * \frac{1}{2}[1 \quad \underline{0} \quad -1] = \frac{1}{2}[3 \quad 4 \quad -6 \quad -5 \quad 2]$

We don't really care about this factor.
We just want to detect differences!

Image derivatives

- Another perspective to get central differences:

$f_F{}'(x) \approx \frac{1}{h}(f(x+h) - f(x))$ **(forward)**

$f_B{}'(x) \approx \frac{1}{h}(f(x) - f(x-h))$ **(backward)**

  - First order central (average):

$$f_C{}'(x) \approx \frac{1}{2}(f_F{}' + f_B{}') = \frac{1}{2h}(f(x+h) - f(x-h)) \longrightarrow \frac{1}{2}\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline\end{array}$$

mask: *h*=1

  - Second order central:

$$f_c{}''(x) \approx \frac{f'(x+0.5h) - f'(x-0.5h)}{h} = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h}$$

$$= \frac{1}{h^2}(f(x+h) - 2f(x) + f(x-h)) \longrightarrow \begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline\end{array}$$

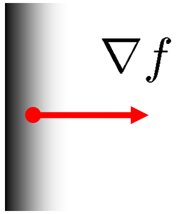
mask: *h*=1

In derivative masks, weight normalization is to 0

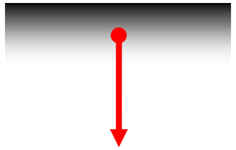
Image gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

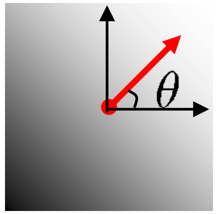
<u>The gradient points in the direction of most rapid increase in intensity</u>



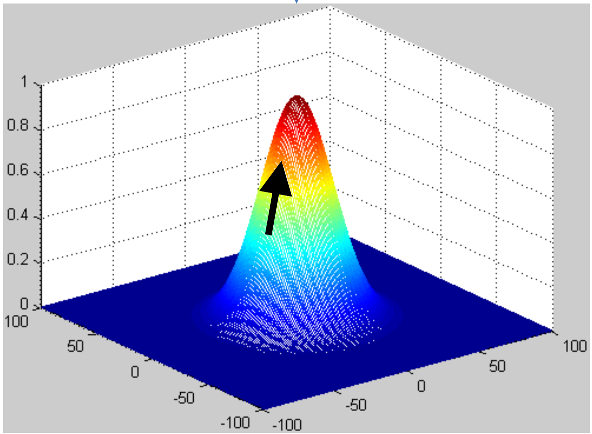
$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



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



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



The *edge strength* is given by the **gradient magnitude**:

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

The **gradient direction** is given by:

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

Source: Steve Seitz

Image gradient

Decomposing the Sobel filter

$$\underbrace{\begin{bmatrix}1\\2\\1\end{bmatrix}}_{\text{1D Gaussian filter}} * \underbrace{\begin{bmatrix}-1 & 0 & 1\end{bmatrix}}_{\text{x-derivative}} \Rightarrow \underbrace{\begin{bmatrix}-1 & 0 & +1\\-2 & 0 & +2\\-1 & 0 & +1\end{bmatrix}}_{\text{Sobel - x}}$$

Horizontal derivative

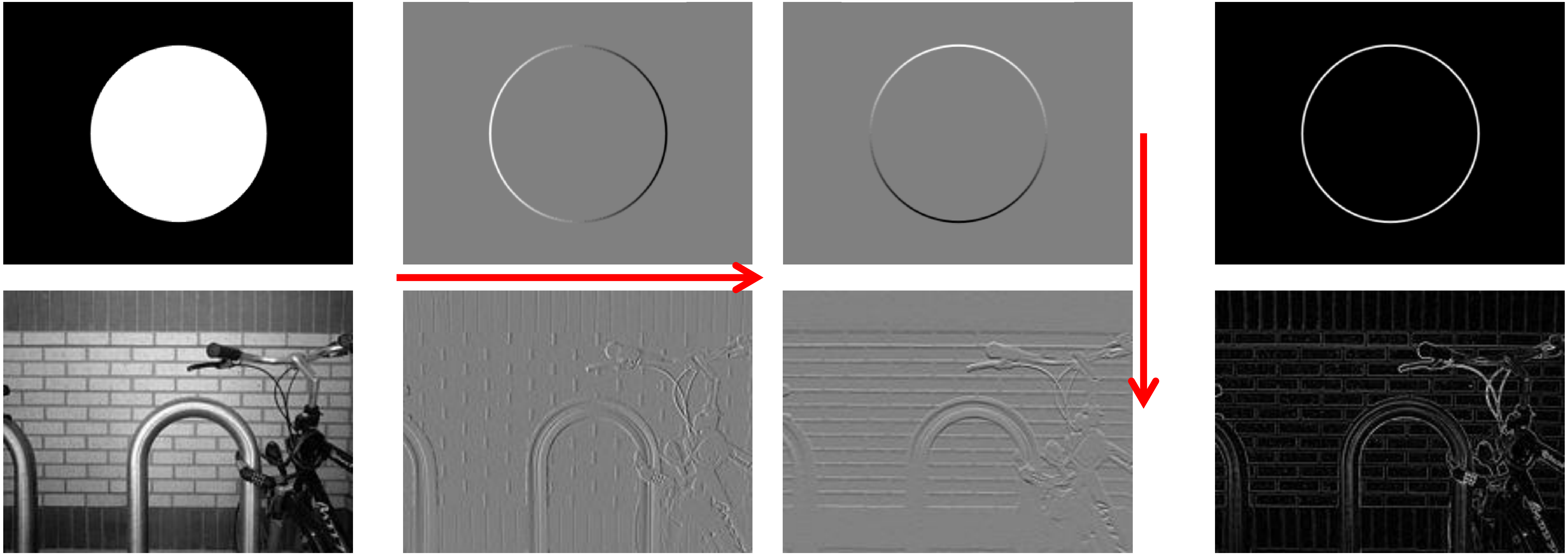
$$\mathbf{G_x} = \begin{bmatrix}-1 & 0 & +1\\-2 & 0 & +2\\-1 & 0 & +1\end{bmatrix}$$

Vertical derivative

$$\mathbf{G_y} = \begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\+1 & +2 & +1\end{bmatrix}$$

Gradient magnitude

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$



Visual examples taken from https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude and https://en.wikipedia.org/wiki/Sobel_operator

Note: These filters give negative values, because derivatives can be negative. Then, at coding level, you have to “transfer” those values to [0,255] or [0,1].

Image gradient

Is there any difference if we apply cross-correlation or convolution with this kernel?? 🤔

Horizontal derivative

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Vertical derivative

$$\mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Gradient magnitude

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

Gradient orientation

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

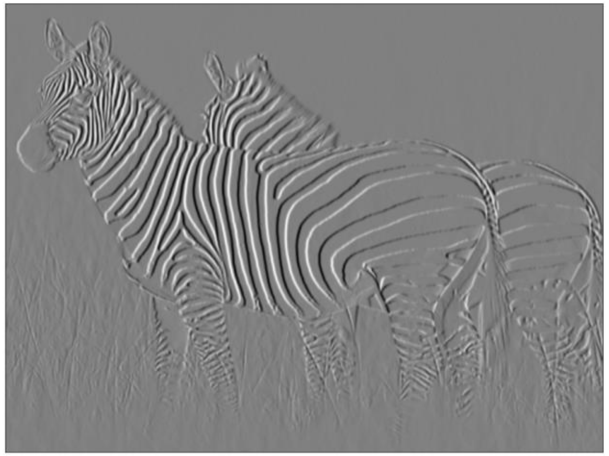
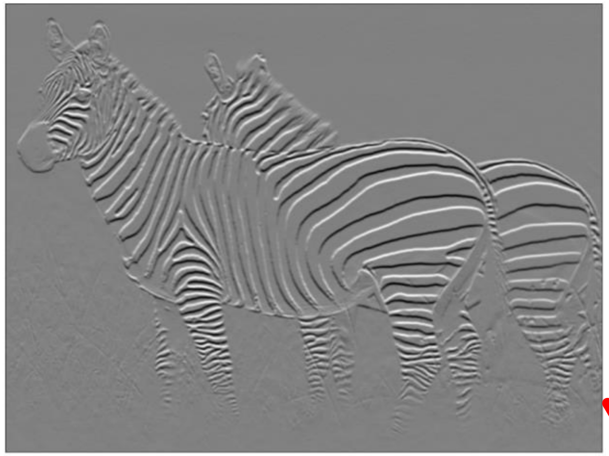
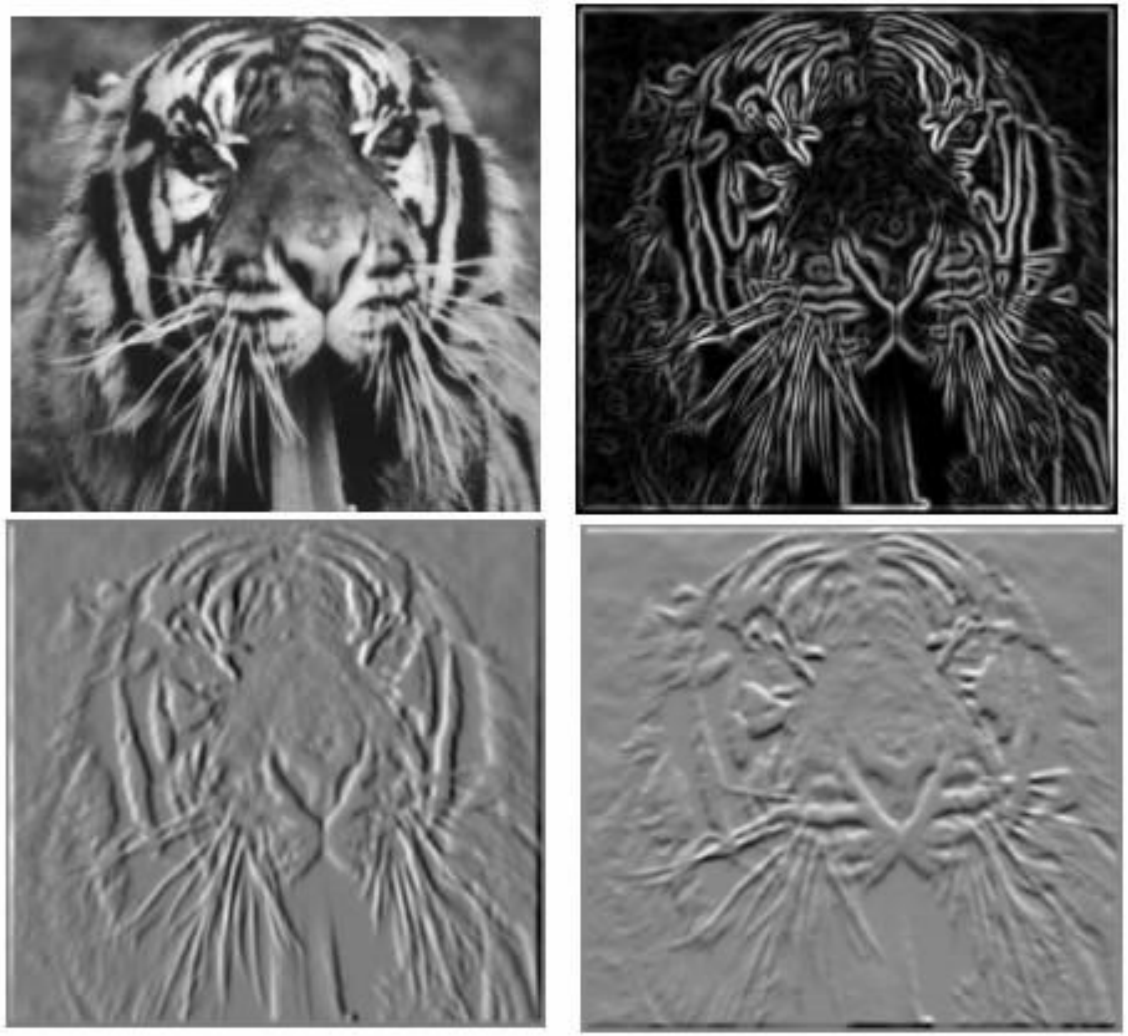



Image gradient



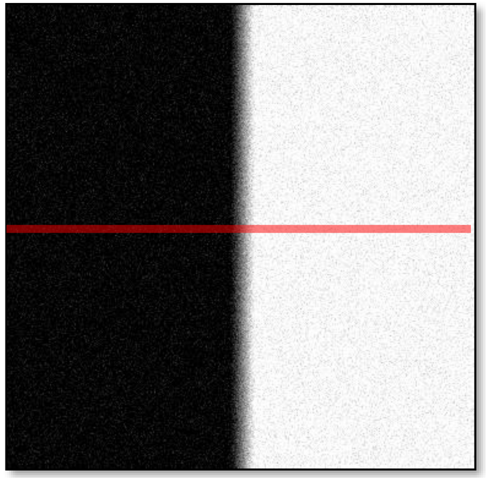
🤔

Which one represents the gradient magnitude?

Which one shows changes with respect to x?

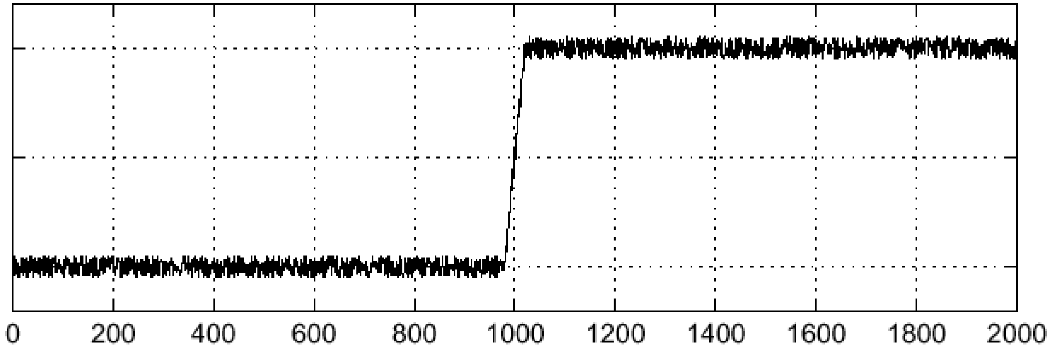
Which one shows changes with respect to y?

Effects of noise

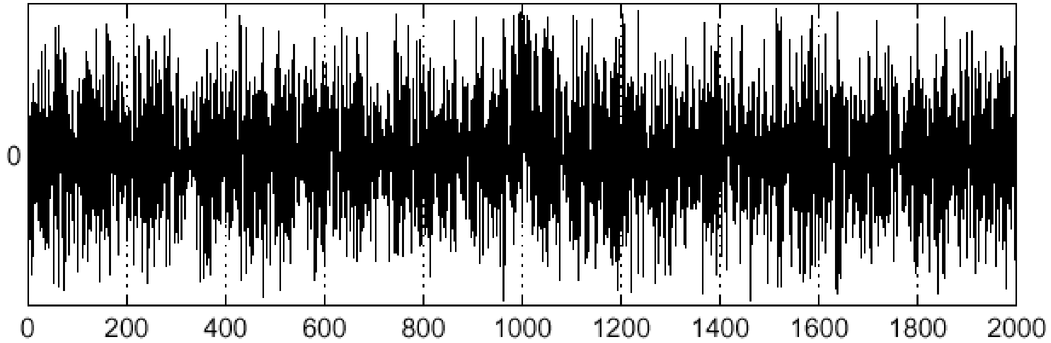


Noisy input image

$$f(x)$$



$$\frac{d}{dx}f(x)$$

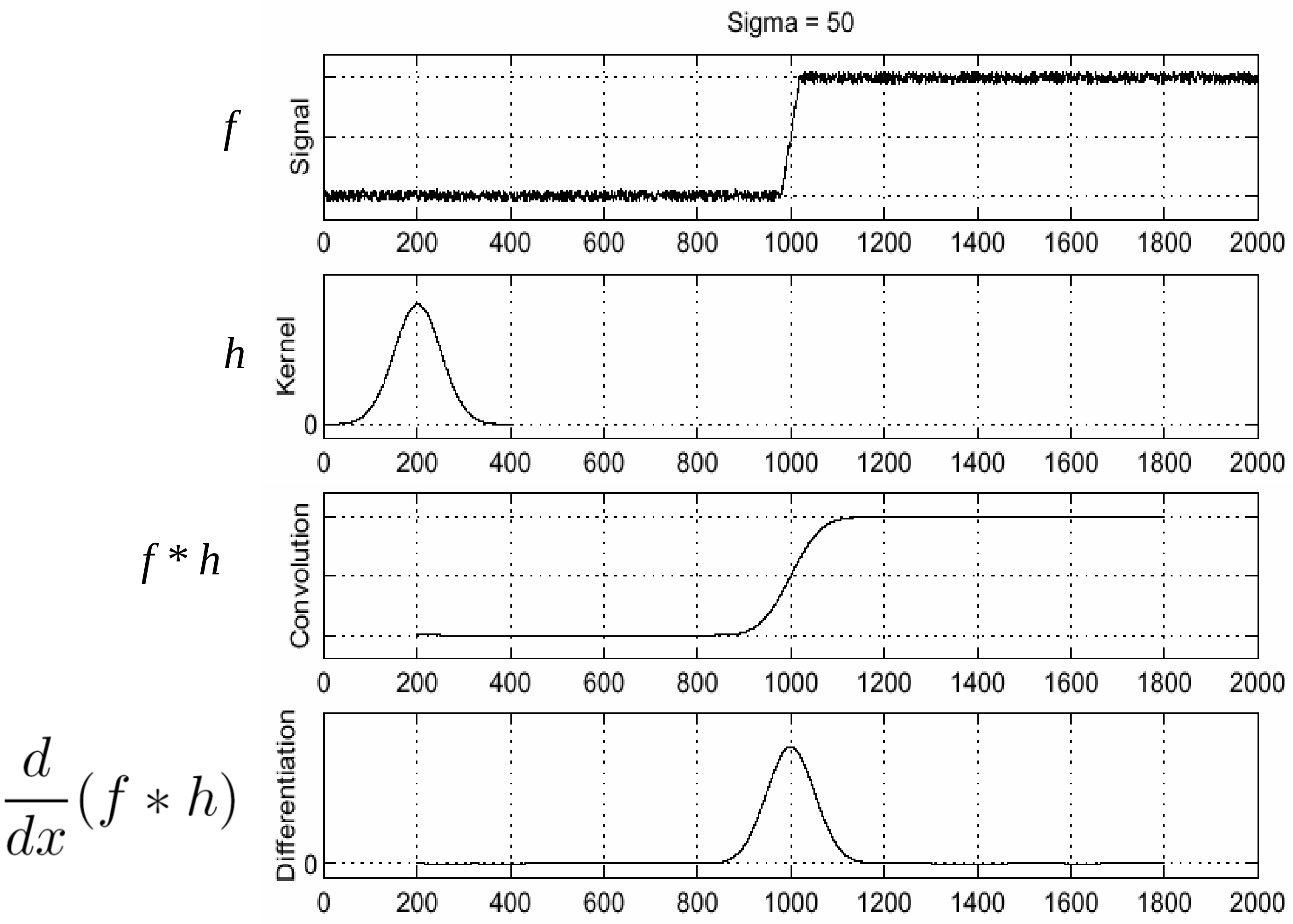


Derivatives amplify noise (high-frequency components)!

Where is the edge?

Source: S. Seitz

Solution: smooth first



How many convolutions do we have here??

To find edges, look for peaks $\frac{d}{dx}(f * h)$

Source: S. Seitz

Associative property of convolution

- Differentiation is convolution, and convolution is associative:

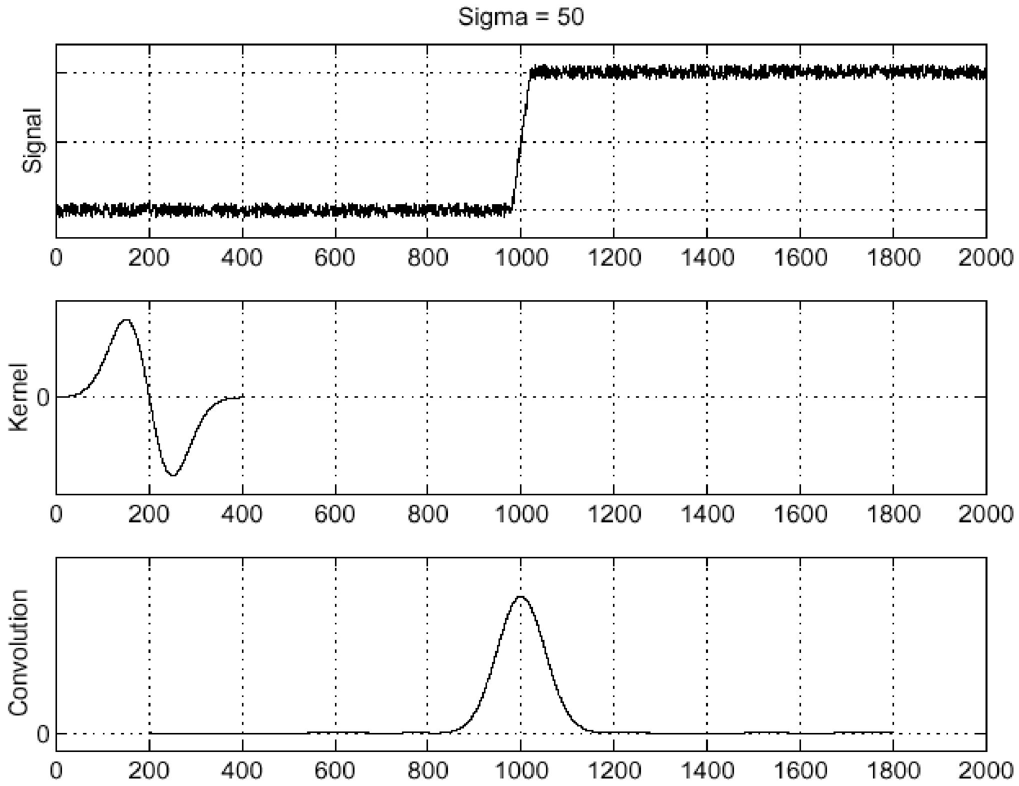
$$\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$$

- This saves us one operation:

f

$\frac{d}{dx}h$

$f * \frac{d}{dx}h$



Computing derivatives implies smoothing (in one direction) and differentiating (in the other)

Examples:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

1D Gaussian filter   x-derivative   Sobel - x

$$\tfrac{1}{2}\begin{bmatrix} 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 \end{bmatrix} = \tfrac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

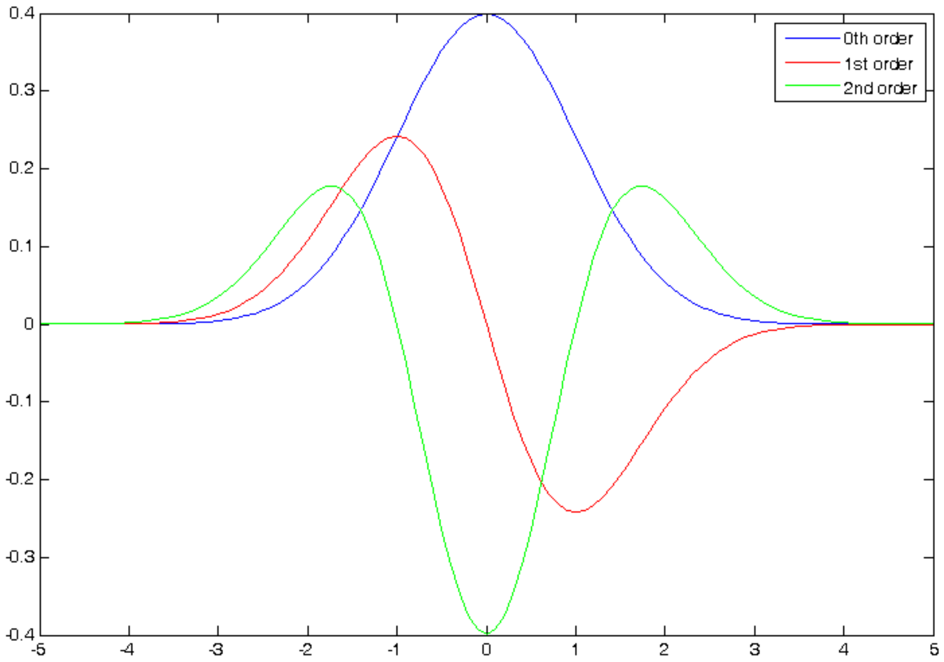
Source: S. Seitz

The 1D Gaussian and its derivatives

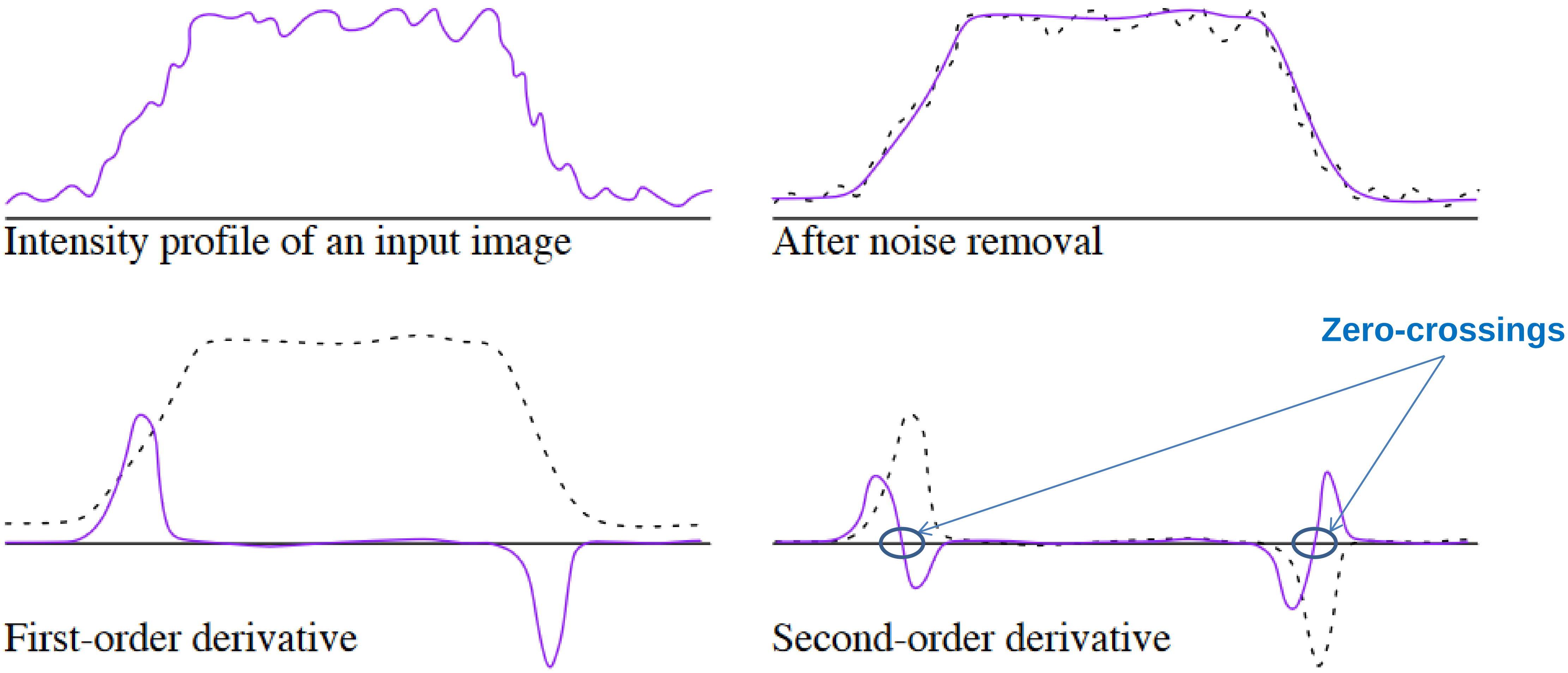
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)$$

$$g'_\sigma(x) = \frac{\partial g_\sigma(x)}{\partial x} = -\frac{x}{\sigma^2}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right) = -\frac{x}{\sigma^2}g_\sigma(x)$$

$$g''_\sigma(x) = \frac{\partial^2 g_\sigma(x)}{\partial x^2} = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right)\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right)g_\sigma(x)$$



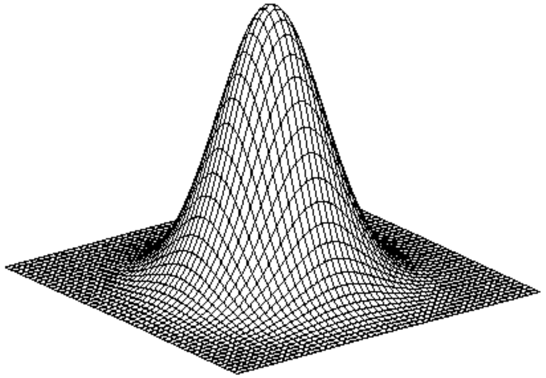
The 1D Gaussian and its derivatives



Main advantage of second derivatives? They allow you to locate the edge more precisely!

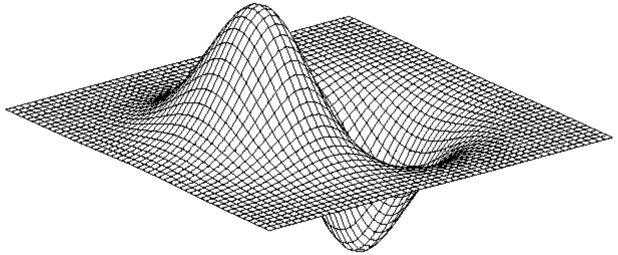
https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D06-LocalOp1.pdf

2D edge detection filters



Gaussian

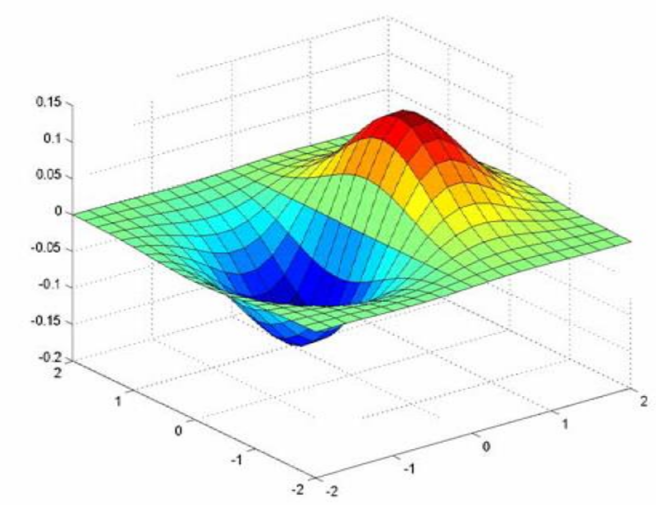
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



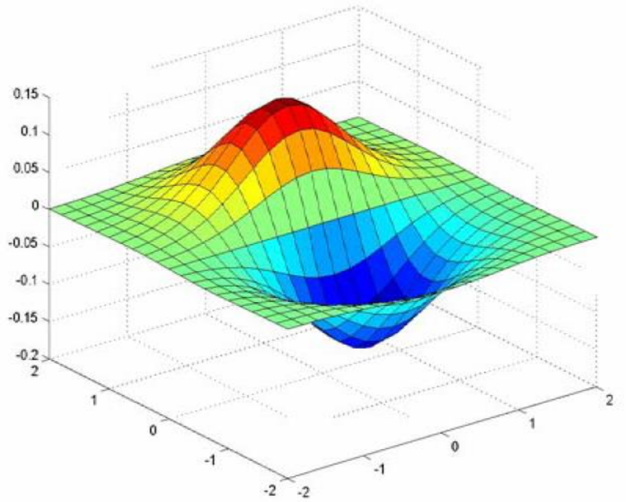
derivative of Gaussian (*x*)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

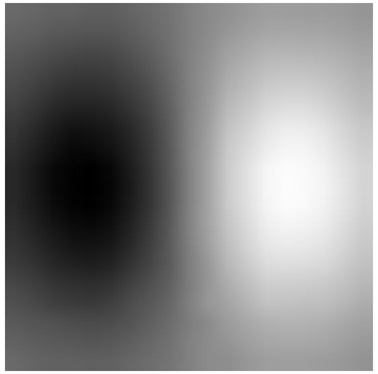
Derivative of Gaussian filter

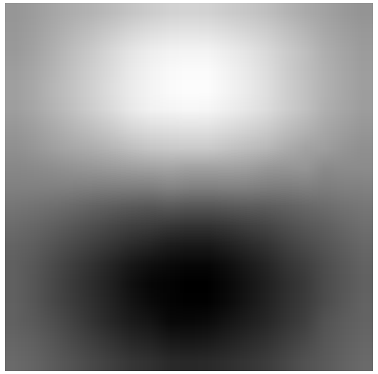


x-direction



y-direction

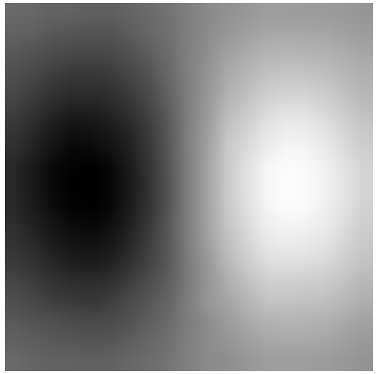




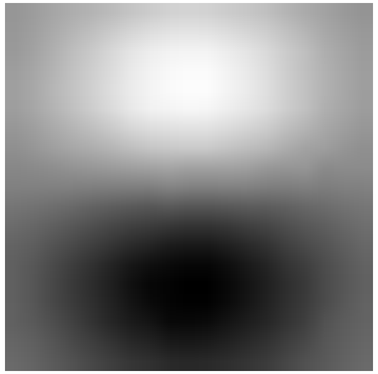
Derivative of Gaussian filter

Is the sign really important?? 🤔

***x*-direction**

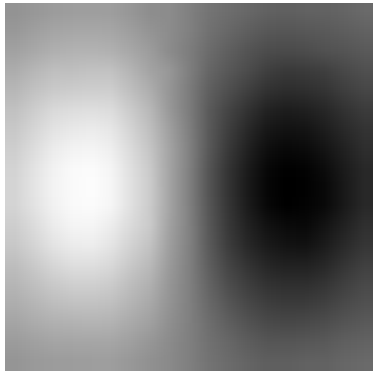


***y*-direction**

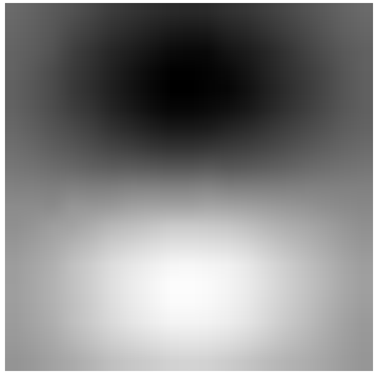


vs

***x*-direction**



***y*-direction**



Common 2D Gauss derivative kernels

Prewitt

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

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

Sobel

$$Sobel_x = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$$

$$Sobel_y = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$$

Scharr

$$Scharr_x = \frac{1}{16}\begin{bmatrix}3\\10\\3\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 0 & -3\\10 & 0 & -10\\3 & 0 & -3\end{bmatrix}$$

$$Scharr_y = \frac{1}{16}\begin{bmatrix}3 & 10 & 3\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 10 & 3\\0 & 0 & 0\\-3 & -10 & -3\end{bmatrix}$$

The standard definition of the Sobel operator omits the 1/8 term

- **doesn't make a difference for edge detection**

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

Creating larger derivative kernels

If we want a 5x5 Sobel kernel:

1	2	1	*	−1	0	1

```
sp.convolve(np.array([1,2,1]),
np.array([-1,0,1]), mode='full')
```

−1	−2	0	2	1

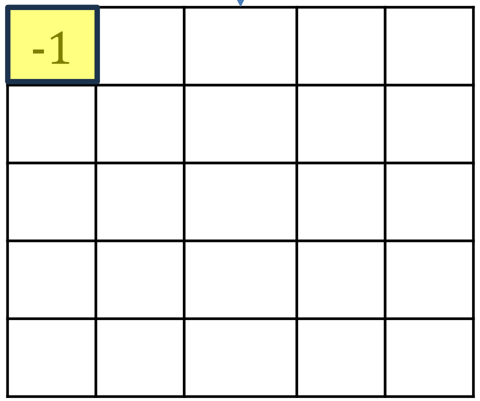
1	2	1	*	1	2	1

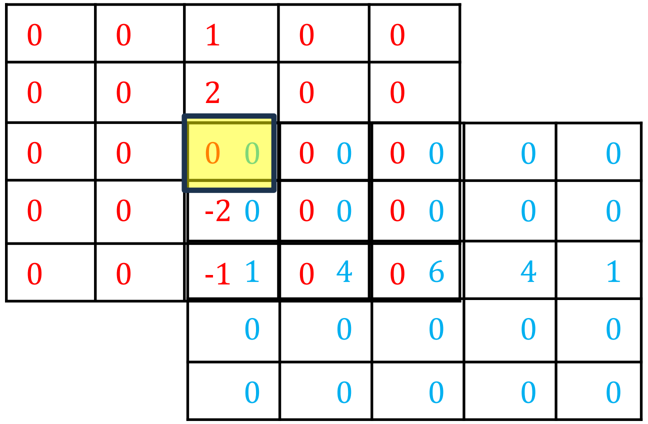
```
sp.convolve(np.array([1,2,1]),
np.array([1,2,1]), mode='full')
```

1	4	6	4	1

*

```
np.outer() or
sp.convolve2d()
```





We can **generate kernels recursively**, using the 1D central difference kernel and the binomial!

Creating larger derivative kernels

If we want a 5x5 Sobel kernel:

1	2	1

*

−1	0	1

```
sp.convolve(np.array([1,2,1]),
np.array([-1,0,1]), mode='full')
```

−1	−2	0	2	1

1	2	1

*

1	2	1

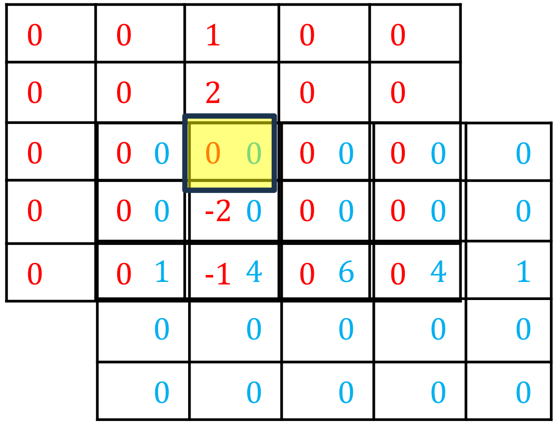
```
sp.convolve(np.array([1,2,1]),
np.array([1,2,1]), mode='full')
```

1	4	6	4	1

*

```
np.outer() or
sp.convolve2d()
```

-1	-4			



We can **generate kernels recursively**, using the 1D central difference kernel and the binomial!

Creating larger derivative kernels

If we want a 5x5 Sobel kernel:

1	2	1

*

−1	0	1

```
sp.convolve(np.array([1,2,1]),
np.array([-1,0,1]), mode='full')
```

−1	−2	0	2	1

1	2	1

*

1	2	1

```
sp.convolve(np.array([1,2,1]),
np.array([1,2,1]), mode='full')
```

1	4	6	4	1

*

```
np.outer() or
sp.convolve2d()
```

-1	-4	-6		

0	0	1	0	0
0	0	2	0	0
0 0	0 0	0 0	0 0	0 0
0 0	0 0	-2 0	0 0	0 0
0 1	0 4	-1 6	0 4	0 1
0	0	0	0	0
0	0	0	0	0

We can **generate kernels recursively**, using the 1D central difference kernel and the binomial!

Creating larger derivative kernels

If we want a 5x5 Sobel kernel:

1	2	1

*

−1	0	1

```
sp.convolve(np.array([1,2,1]),
np.array([-1,0,1]), mode='full')
```

−1	−2	0	2	1

1	2	1

*

1	2	1

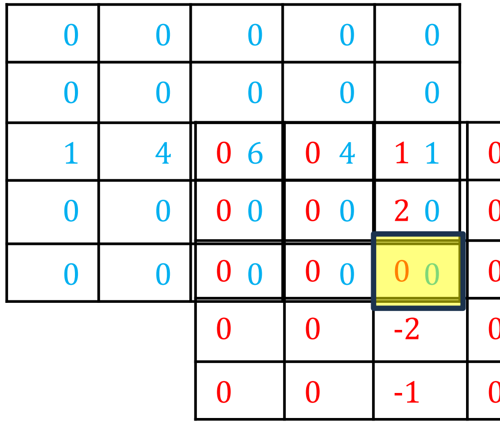
```
sp.convolve(np.array([1,2,1]),
np.array([1,2,1]), mode='full')
```

1	4	6	4	1

*

```
np.outer() or
sp.convolve2d()
```

-1	-4	-6	-4	-1
-2	-8	-12	-8	-2
0	0	0	0	0
2	8	12	8	2
1	4	6	4	1



We can **generate kernels recursively**, using the 1D central difference kernel and the binomial!

Creating larger derivative kernels

If we want a 7x7 Sobel kernel:

1	4	6	4	1

*

−1	0	1

1	4	6	4	1

*

1	2	1

−1	−4	−5	0	5	4	1

*

1	6	15	20	15	6	1

-1	-6	-15	-20	-15	-6	-1
-4	-24	-60	-80	-60	-24	-4
-5	-30	-75	-100	-75	-30	-5
0	0	0	0	0	0	0
5	30	75	100	75	30	5
4	24	60	80	60	24	4
1	6	15	20	15	6	1

OpenCV functions like ***`getDerivKernels()`*** or ***`getGaussianKernel()`*** can help you in this regard!

Separability

First derivative of 2D Gaussian is separable:

$$
\begin{aligned}
G(x,y) &= \frac{1}{2\pi\sigma^2}\exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right) \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y^2}{2\sigma^2}\right) \\
&= G_h(x)\cdot G_v(y) \\
\frac{\partial G(x,y)}{\partial x} &= -\frac{x}{\sigma^2}\cdot\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2}{2\sigma^2}\right)\exp\left(-\frac{y^2}{2\sigma^2}\right) \\
&= G'_h(x)\cdot G_v(y)
\end{aligned}
$$

horizontal 1D Gaussian derivative kernel

vertical 1D Gaussian kernel

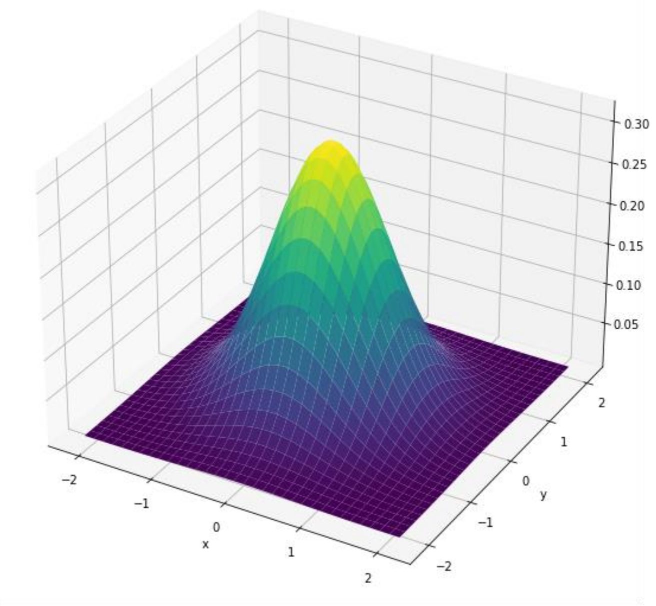
smooth in one direction, differentiate in the other

This allows us to know **which 1D kernels to apply, and in what order, to calculate the derivatives of an image.** In the previous example, the first derivative on X.

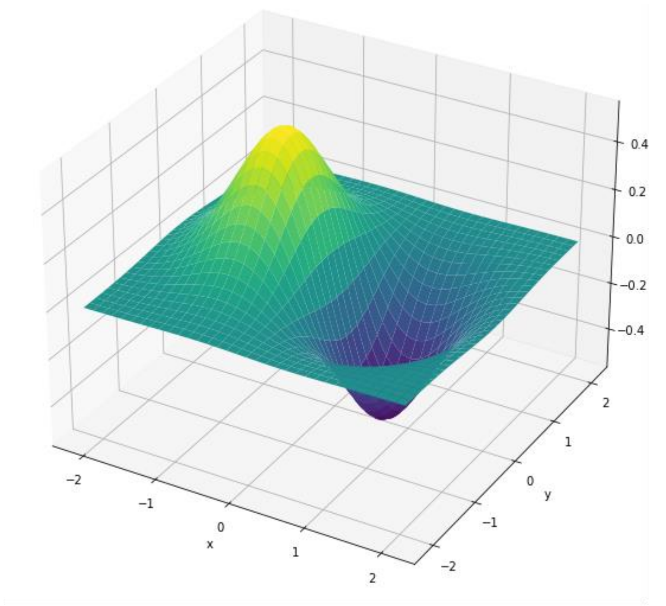
Laplacian of Gaussian

- How many 2nd derivative filters do we have? There are four 2nd partial derivative filters.
- In practice, it's handy to define a single 2nd derivative filter—the Laplacian
  - Instead of using $\frac{\partial^2(G_\sigma(x,y))}{\partial x^2} = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G_\sigma(x,y)$ and $\frac{\partial^2(G_\sigma(x,y))}{\partial y^2} = \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G_\sigma(x,y)$
  - We employ the Laplacian of Gaussian (LoG) filter

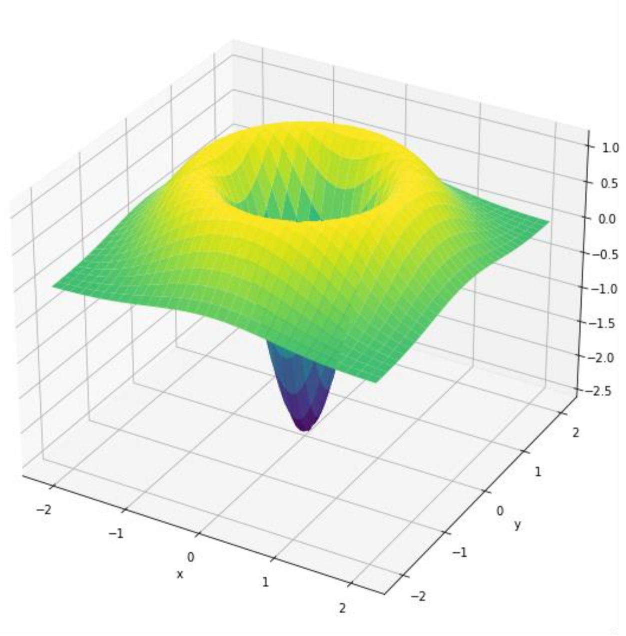
Gaussian



1st derivative of Gaussian (x)



LoG



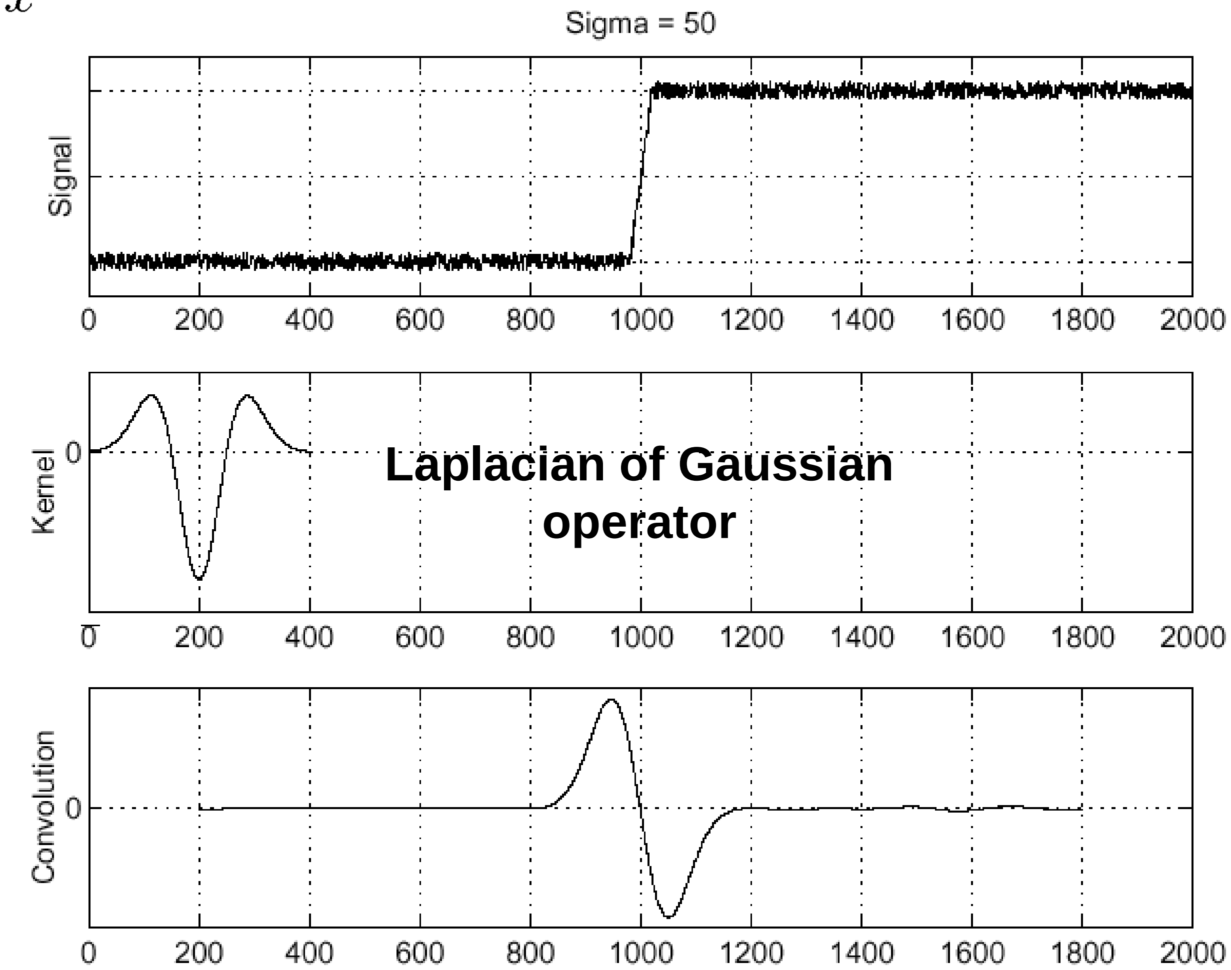
1D Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



Finding these zero-crossings is the goal of the Marr-Hildreth algorithm for edge detection.

Marr, D., & Hildreth, E. (1980). Theory of edge detection. *Procs. of the Royal Society of London*, *207*, 187-217.

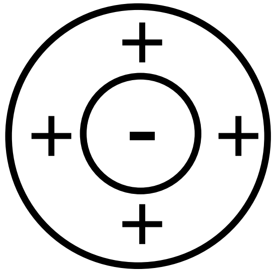
Where is the edge? **Zero-crossings of bottom graph**

Slide credit: Steve Seitz

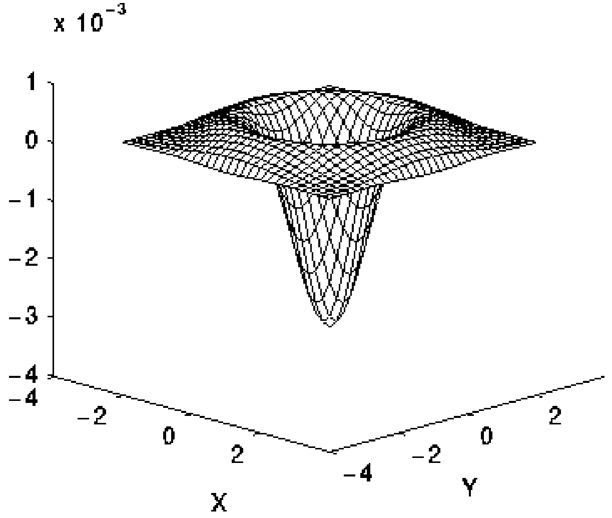
Laplacian of Gaussian

- Example of 3x3 Laplacian kernel:

0	1	0
1	-4	1
0	1	0



“inverted Mexican hat”



- This kernel is not separable, but…
  – we can calculate LoG as the sum of 2D convolutions that are separable
    - Because the Gaussian and its derivatives are!!!
    - In practice, we need 4 1D convolutions (and that is still less computationally expensive than doing a single 2D convolution; unless the kernel is very small)

if the kernel is 7×7 we need 49 multiplications and additions <u>per pixel</u> for the 2D kernel, or 4·7=28 multiplications and additions <u>per pixel</u> for the four 1D kernels; this difference grows as the kernel gets larger

Interesting references about LoG: https://stackoverflow.com/questions/53544983/how-is-laplacian-filter-calculated/53545480#53545480 and https://www.crisluengo.net/archives/1099/

Laplacian of Gaussian

The Laplacian operator is the divergence of the gradient, and is given by the sum of the second order derivatives (i.e. the trace of the Hessian matrix):

$$\boxed{\nabla^2} I = \nabla \cdot \nabla I = \left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \right] \left[\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y} \right]^T = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Laplacian of Gaussian

Associative property

Distributive property

$$\underbrace{\frac{\partial^2 (I \circledast G)}{\partial x^2} + \frac{\partial^2 (I \circledast G)}{\partial y^2}}_{\text{Laplacian of an image smoothed with a Gaussian kernel}} = \underbrace{I \circledast \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right)}_{\text{Image convolved with the Laplacian of a Gaussian}} = I \circledast \frac{\partial^2 G}{\partial x^2} + I \circledast \frac{\partial^2 G}{\partial y^2}$$

Laplacian of an image smoothed with a Gaussian kernel = **Image convolved with the Laplacian of a Gaussian**

Laplacian of Gaussian

How do we compute $I \circledast \frac{\partial^2 G}{\partial x^2} + I \circledast \frac{\partial^2 G}{\partial y^2}$ **?**

We know that the 1st derivative of a 2D Gaussian is separable:

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

Then:

$$\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = \boxed{G''_h(x) \cdot G_v(y)} + \boxed{G_h(x) \cdot G''_v(y)}$$

convolution in one direction with the 2nd derivative of the Gaussian, and then in the other direction, convolution with the Gaussian.

convolution in one direction with the Gaussian and then in the other, convolution with the 2nd derivative of the Gaussian.

Laplacian of Gaussian

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

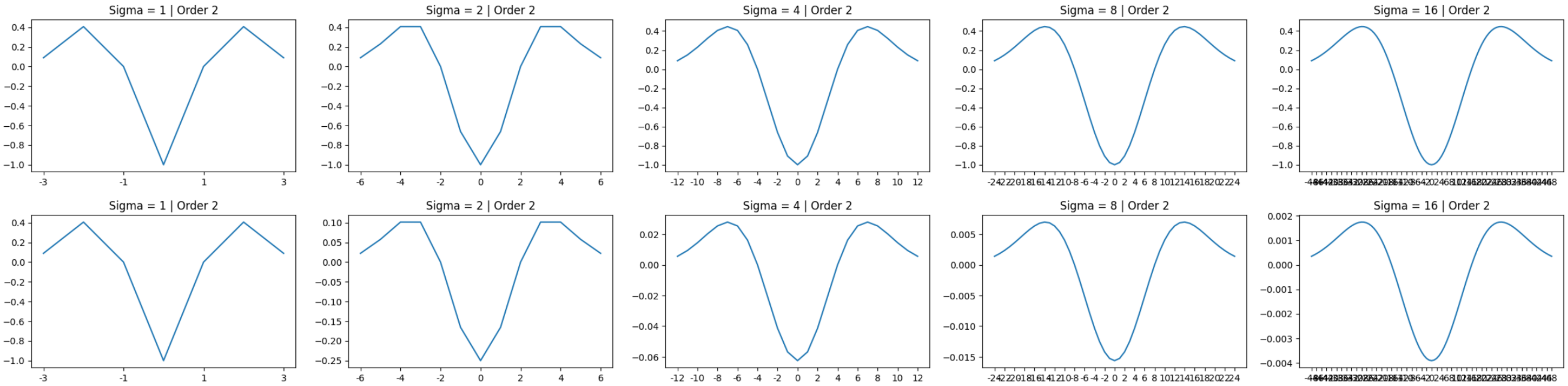
$$\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = G_h''(x) \cdot G_v(y) + G_h(x) \cdot G_v''(y)$$

Intuition: the Gaussian is a probability function, so it has area 1. If we squeeze it in the horizontal direction then we must expand it in the vertical direction to preserve area.

Why do we multiply by σ^2? 🤔

To **keep response the same (scale-invariant)**, must multiply Gaussian derivative by σ

Laplacian is the second Gaussian derivative, so it **must be multiplied by σ²**



Scale normalized

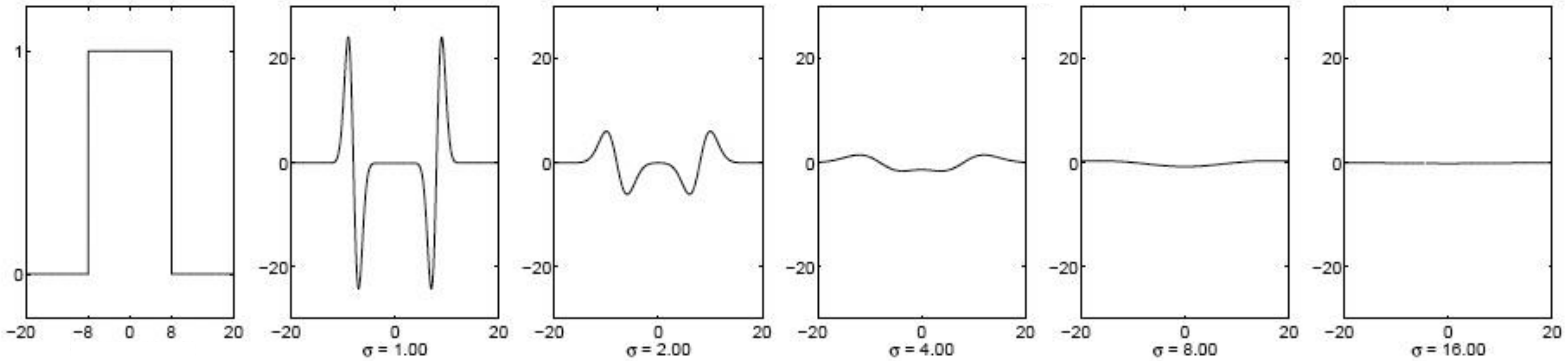
Scale NO normalized

Laplacian of Gaussian

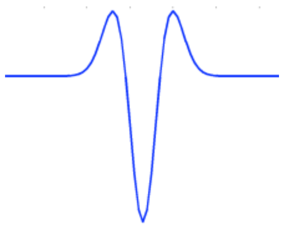
- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases
- To keep response the same (scale-invariant), must multiply Gaussian derivative by σ
- Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2

Original signal

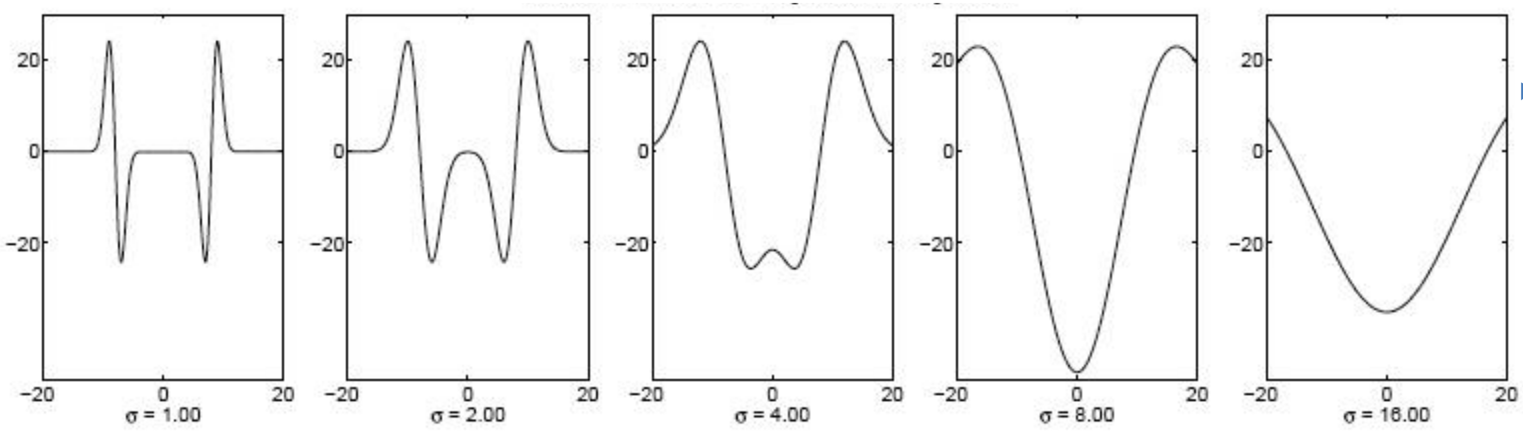
Unnormalized Laplacian response



Responses of convolving the original signal with the LoG (using different sigmas)



Scale-normalized Laplacian response



Laplacian of Gaussian approximation

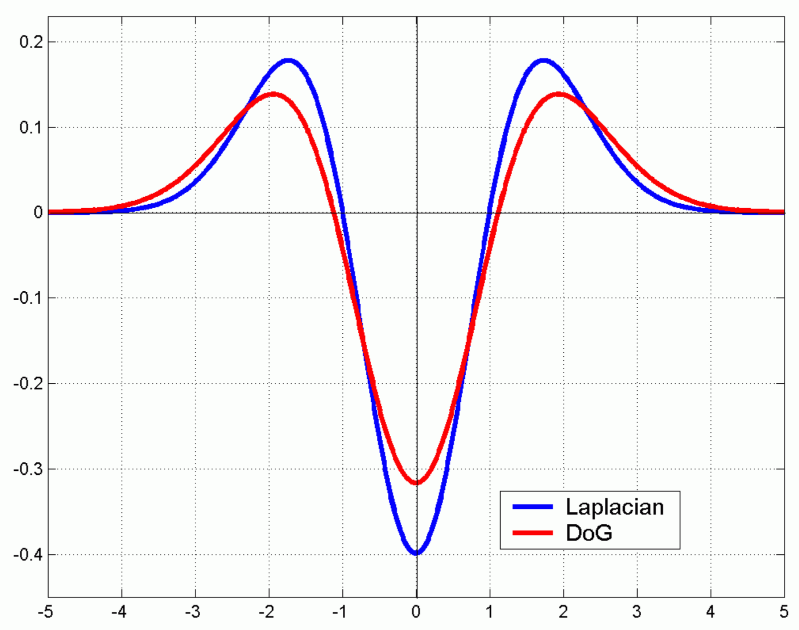
- Approximating the Laplacian with a difference of Gaussians:

$$L = \sigma^2\left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)\right)$$

(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)



Advantage: we don’t even need to implement the second derivative.

Laplacian of Gaussian approximation

- LoG and DoG are **band-pass filters**!

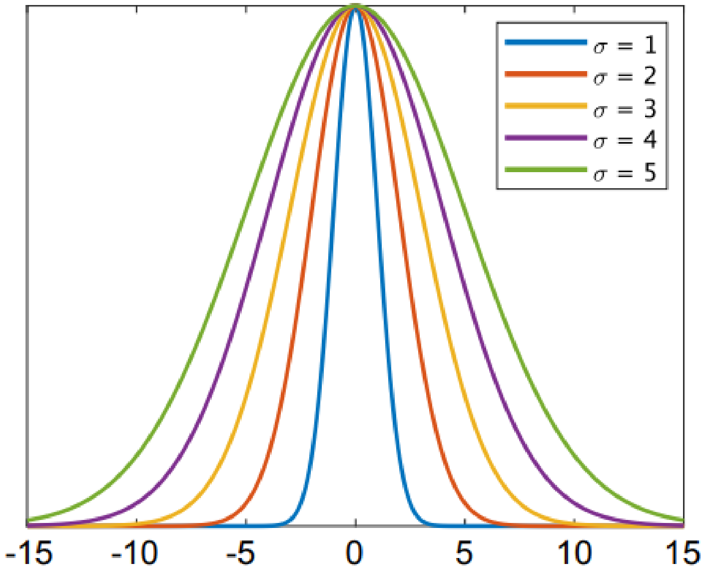
$$L = \sigma^2\left(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)\right)$$

(Laplacian)

$$DoG = G(x,y,k\sigma) - G(x,y,\sigma)$$

(Difference of Gaussians)

The Gaussian functions depicted here do not include the normalizing constant.



FT

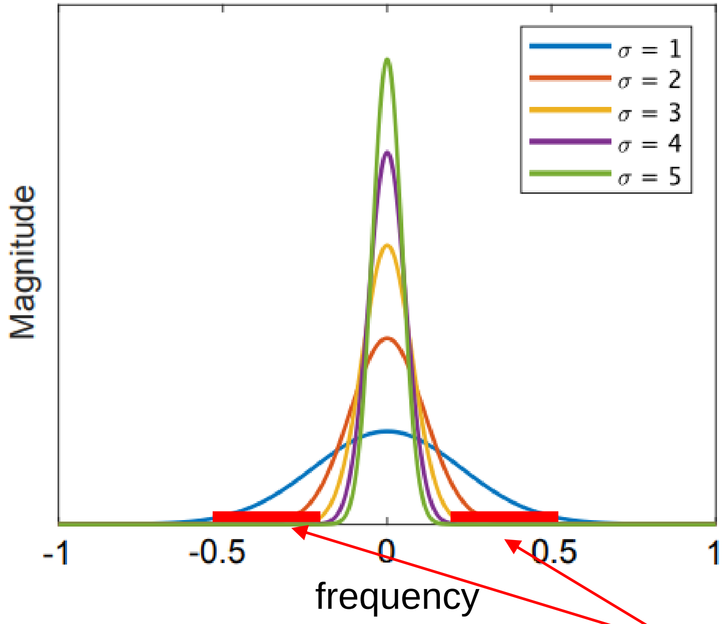


Image taken from Orive-Miguel et al. (2019). Improving localization of deep inclusions in time-resolved diffuse optical tomography. *Applied Sciences*, *9*(24), 5468.

The Gaussian filters involved are "killing" different high-frequencies. Their subtraction give us a range/band of (high) frequencies whose pass is allowed!

Note: We'll understand better this slide once we review the Frequency Domain!

Effect of σ on derivatives (scales)

σ = 1

σ = 3

The apparent structures differ depending on Gaussian’s scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

Summary: Two types of convolution kernels

Smoothing

$$\sum g_i = 1$$

(Smoothing a constant function should not change it)

Example: (1/4) * [1 2 1]

Low-pass filter (Gaussian)

Differentiating

$$\sum g_i = 0$$

(Differentiating a constant function should return 0)

Example: [-1 0 1]

High-pass filter (derivative of Gaussian)

… also bandpass filters (Laplacian of Gaussian)

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

Summary: Two types of convolution kernels

Two closely related problems:

blur
(to remove noise)

differentiate
(to highlight details)

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

Summary: Two types of convolution kernels

Two closely related problems:

blur
(to remove noise)

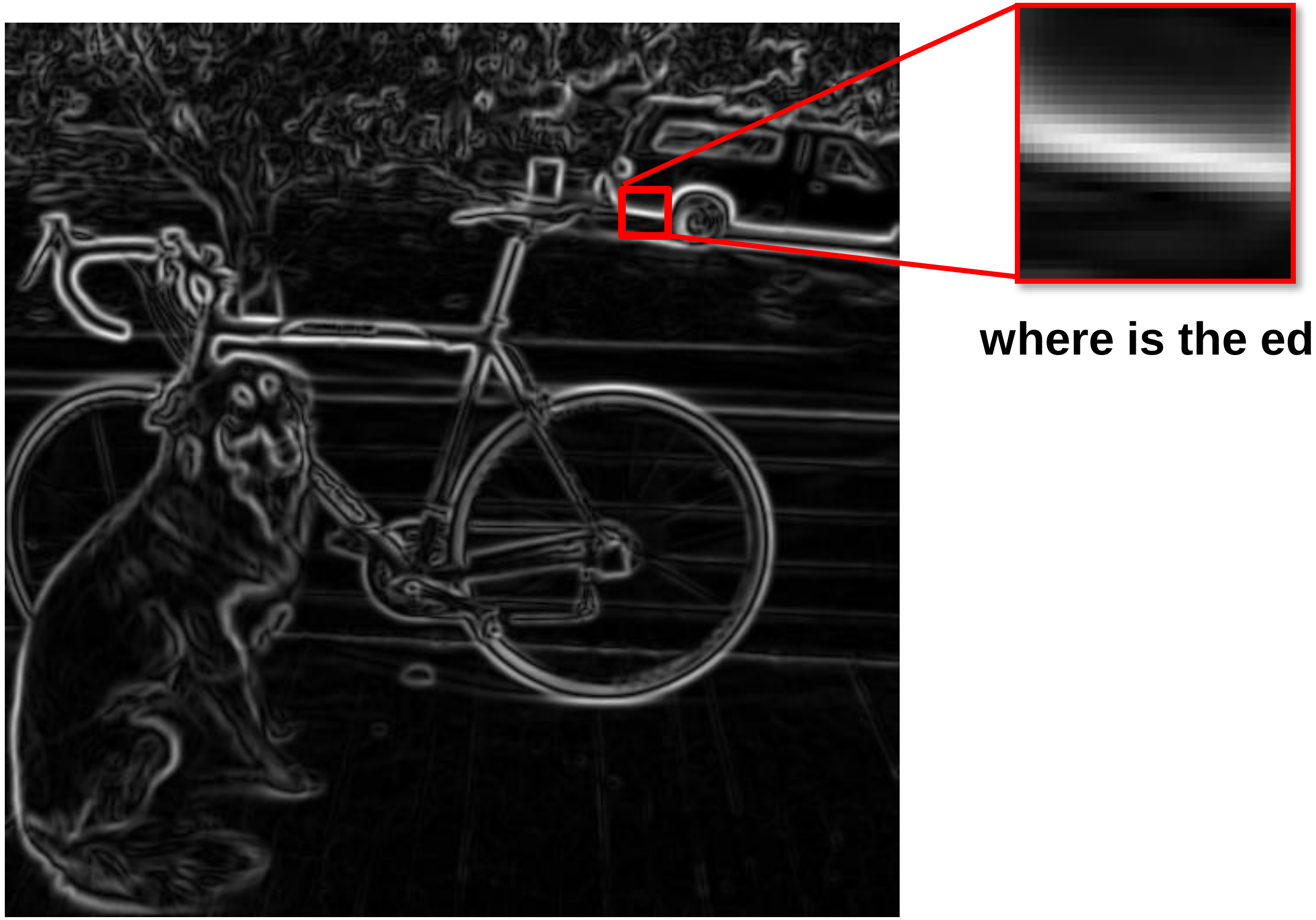
differentiate
(to highlight details)

Underlying math is the same!

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

Canny Edge Detector

- Sometimes we need to accurately locate the boundary....



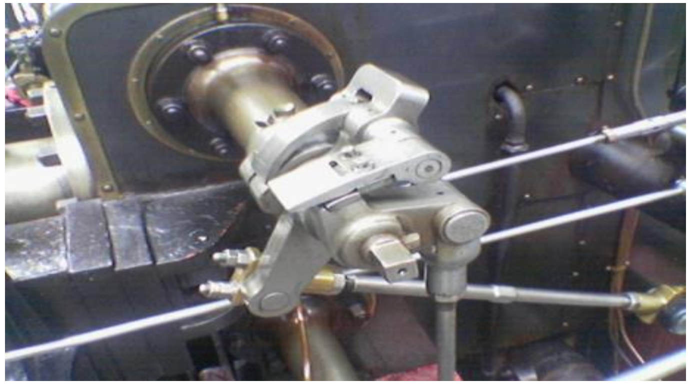
where is the edge?

Canny Edge Detector

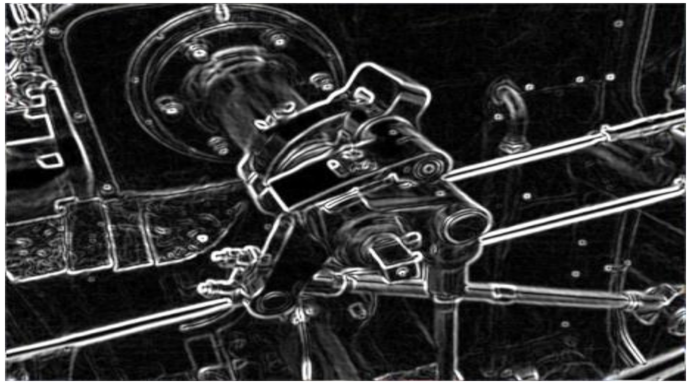
- Canny, J., "A Computational Approach To Edge Detection", IEEE Trans. on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986 (~43K citations in Google Scholar)
- Still popular because
  - easy to implement
  - small number of intuitive parameters
  - computationally efficient
  - good results

Python:
https://docs.opencv.org/4.8.0/da/d22/tutorial_py_canny.html

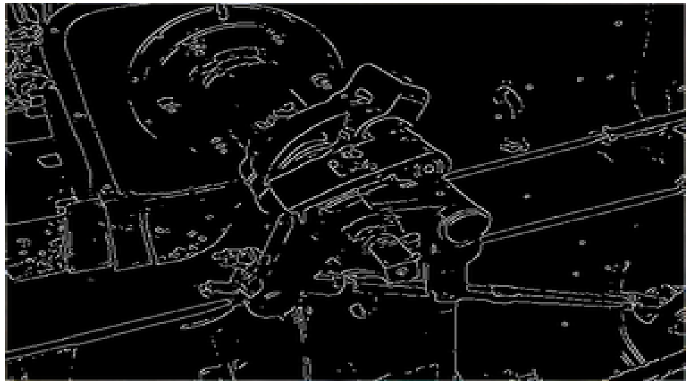
Original Image



Sobel Magnitude

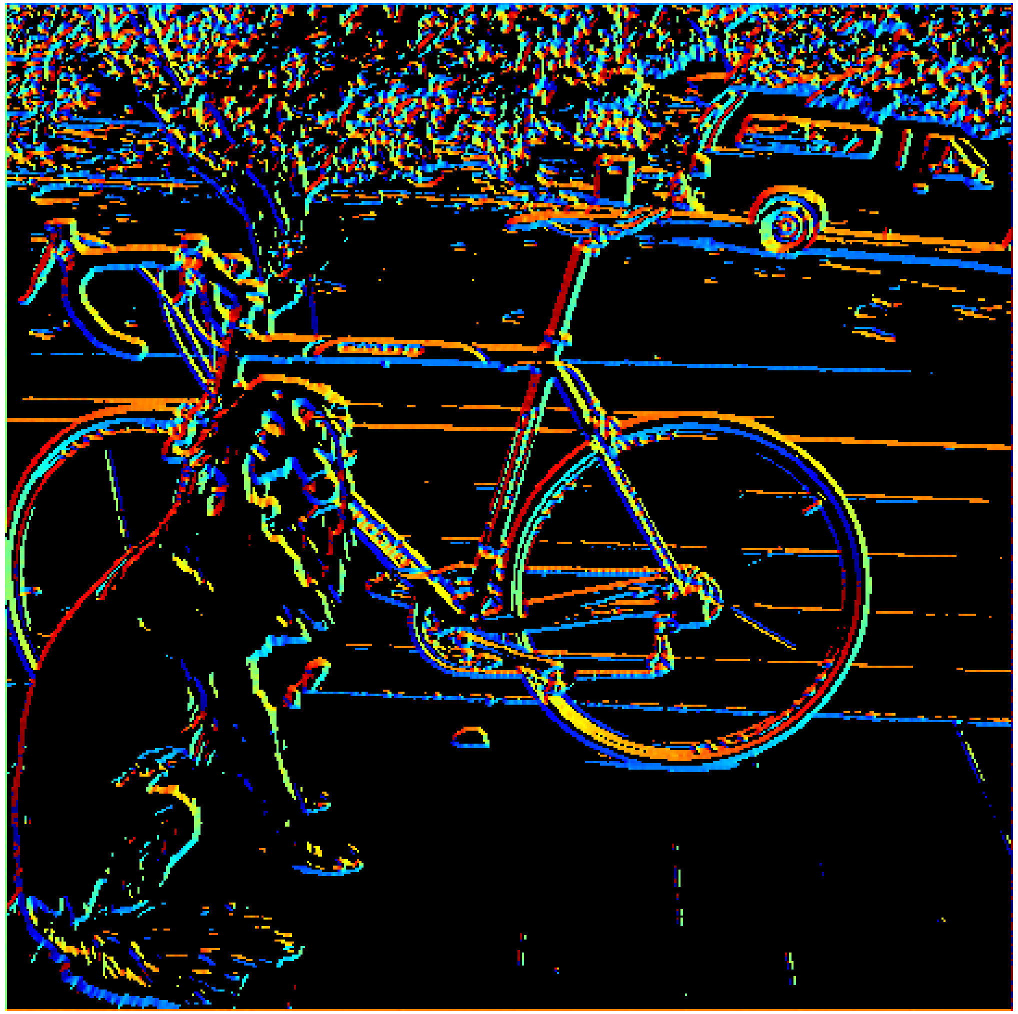


Canny Edge Detector

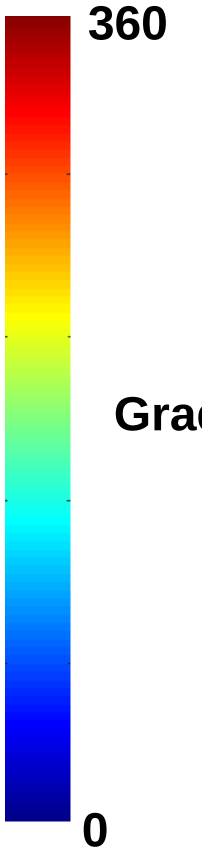


Canny Edge Detector

1) Filter image with derivative of Gaussian and compute the gradient (magnitude and orientation) at each pixel



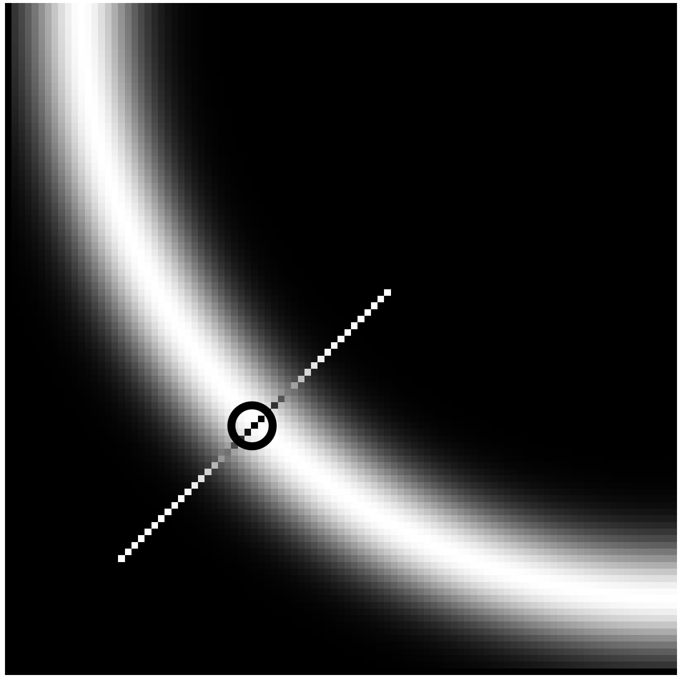
theta = atan2(gy, gx)

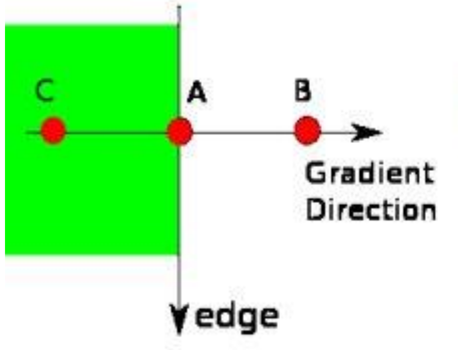


Canny Edge Detector

2) Non-maximum suppression

A full scan of the image is done to remove any unwanted pixels which may not constitute the edge.





For each point on the edge, we keep the highest (maximum) intensity point.
This makes the edges thinner (1 pixel wide).

- We check if every pixel is **local maximum in its neighborhood along gradient direction**
  - Point A is on the edge. Gradient direction is normal to the edge. Point B and C are in gradient directions.
  - Point A is checked with point B and C to see if it forms a local maximum.
  - If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

Before Non-max Suppression

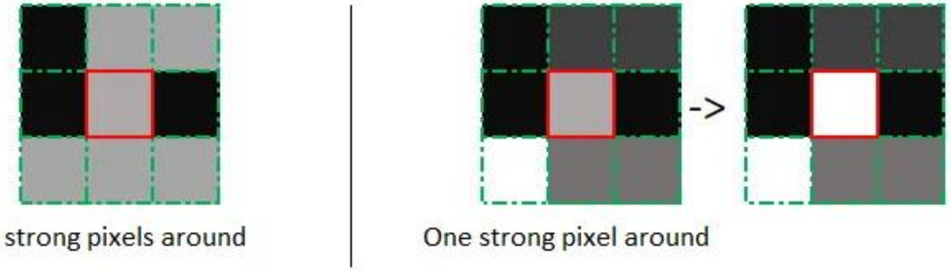
After Non-max Suppression

Canny Edge Detector

3) Linking and thresholding edges (hysteresis)

- We only want strong edges

- 2 thresholds (high, low), 3 cases (R: response):
  - R > high: strong edge!
  - R < low: no edge
  - R < high but R > low: weak edge
    - **Weak edges are edges iff they connect to strong ones**
    - Look in some neighborhood (usually 8 closest)



No strong pixels around

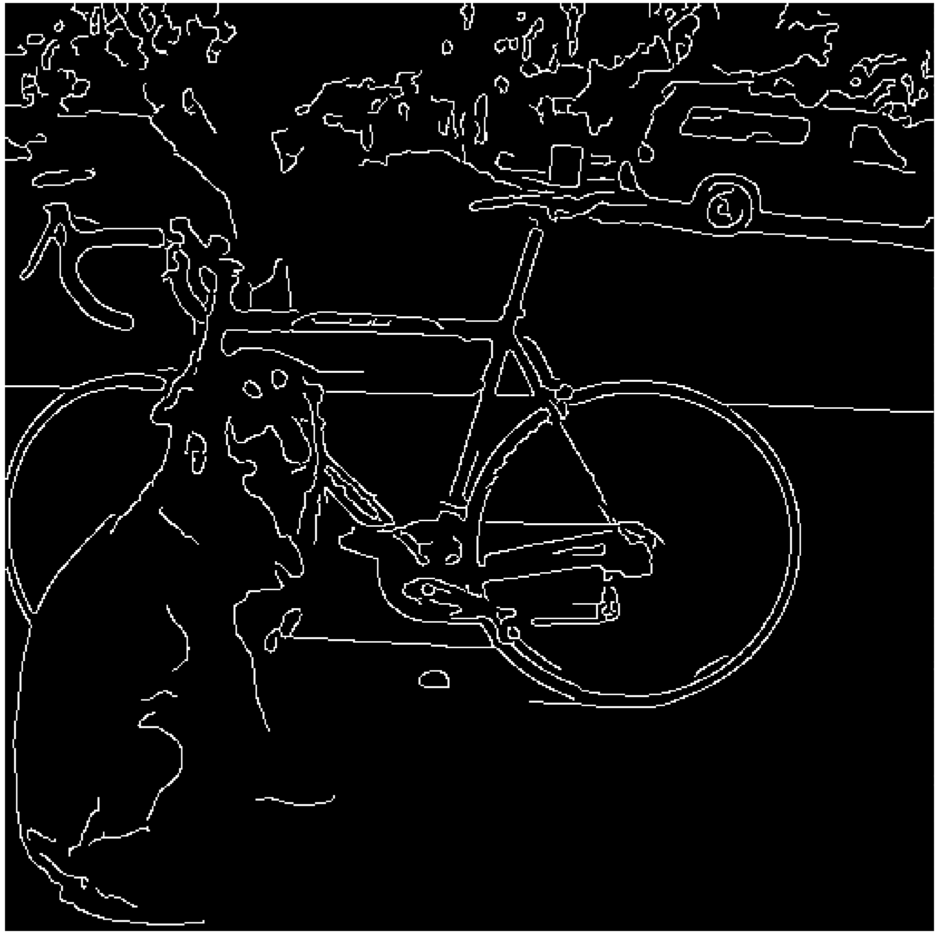
One strong pixel around

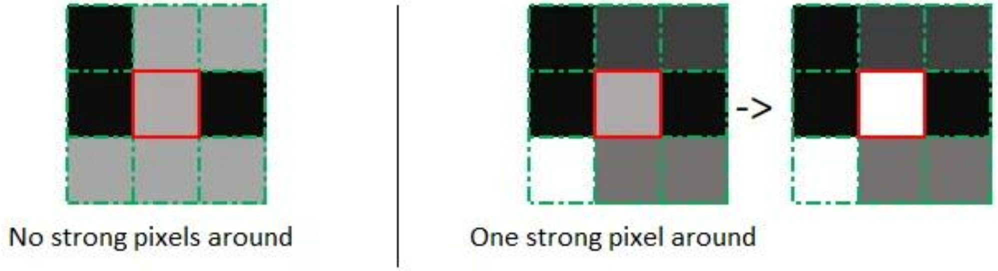
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Canny Edge Detector

3) Linking and thresholding edges (hysteresis)

- We only want strong edges
- 2 thresholds (high, low), 3 cases (R: response):
  - R > high: strong edge!
  - R < low: no edge
  - R < high but R > low: weak edge
    - **Weak edges are edges iff they connect to strong ones**
    - Look in some neighborhood (usually 8 closest)





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

Canny Edge Detector

1. Filter image with derivative of Gaussian

and find magnitude and orientation of gradient

2. Non-maximum suppression

3. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

Source: D. Lowe, L. Fei-Fei, J. Redmon

Canny Edge Detector

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - Large σ detects "large-scale" edges
  - Small σ detects fine edges

Parameters to tune:

high threshold

low threshold

σ : width of the Gaussian blur

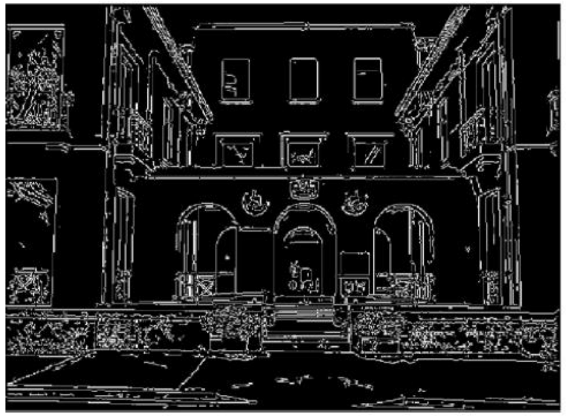
Source: S. Seitz

Canny Edge Detector

original image

high threshold
(strong edges)

low threshold
(weak edges)



hysteresis threshold

Source: L. Fei-Fei

Gaussian Filtering and Edges

Pablo Mesejo

pmesejo@go.ugr.es

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA

DECSAI

DaSCI

Instituto Andaluz de Investigación en

Data Science and **Computational Intelligence**