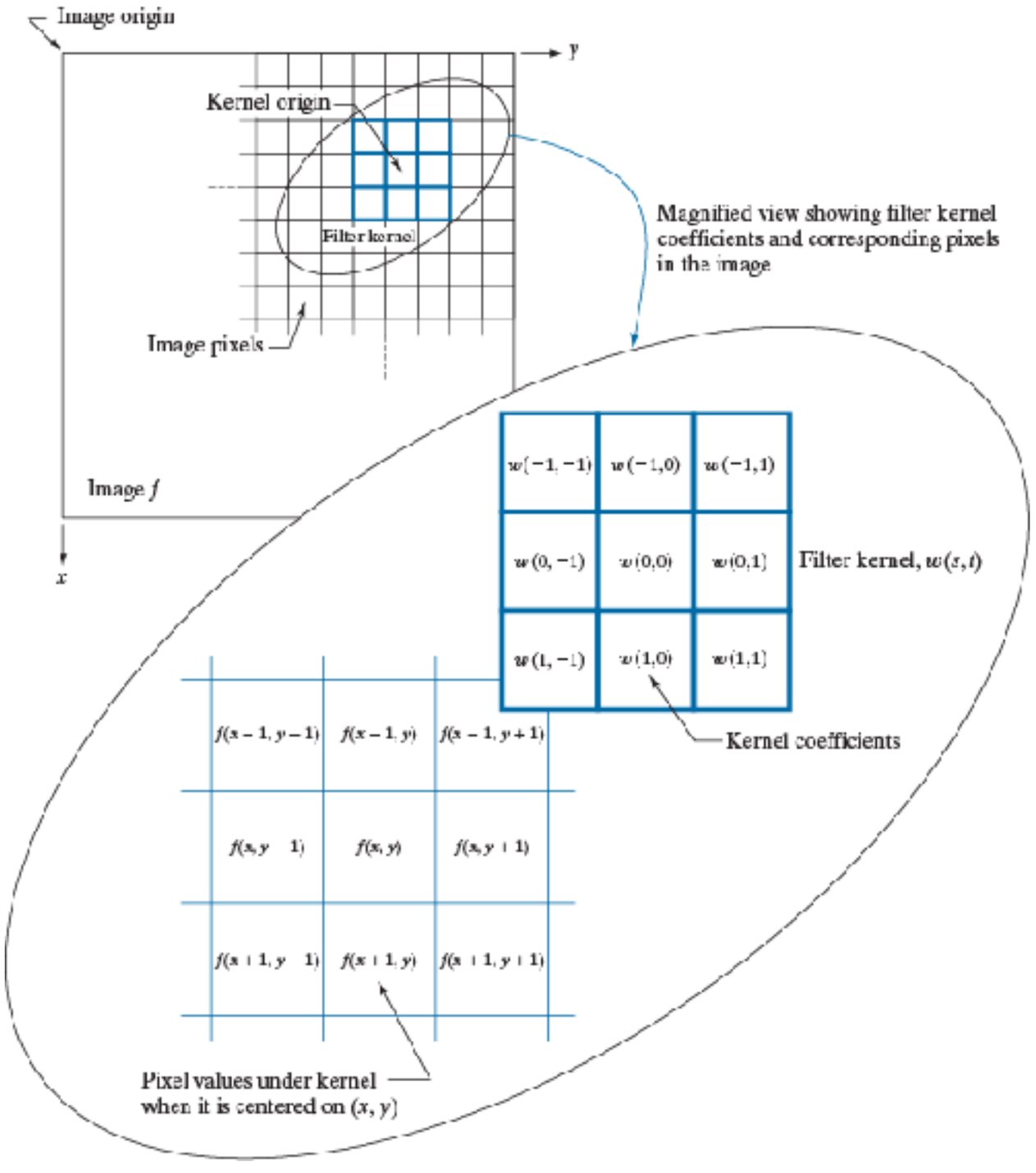
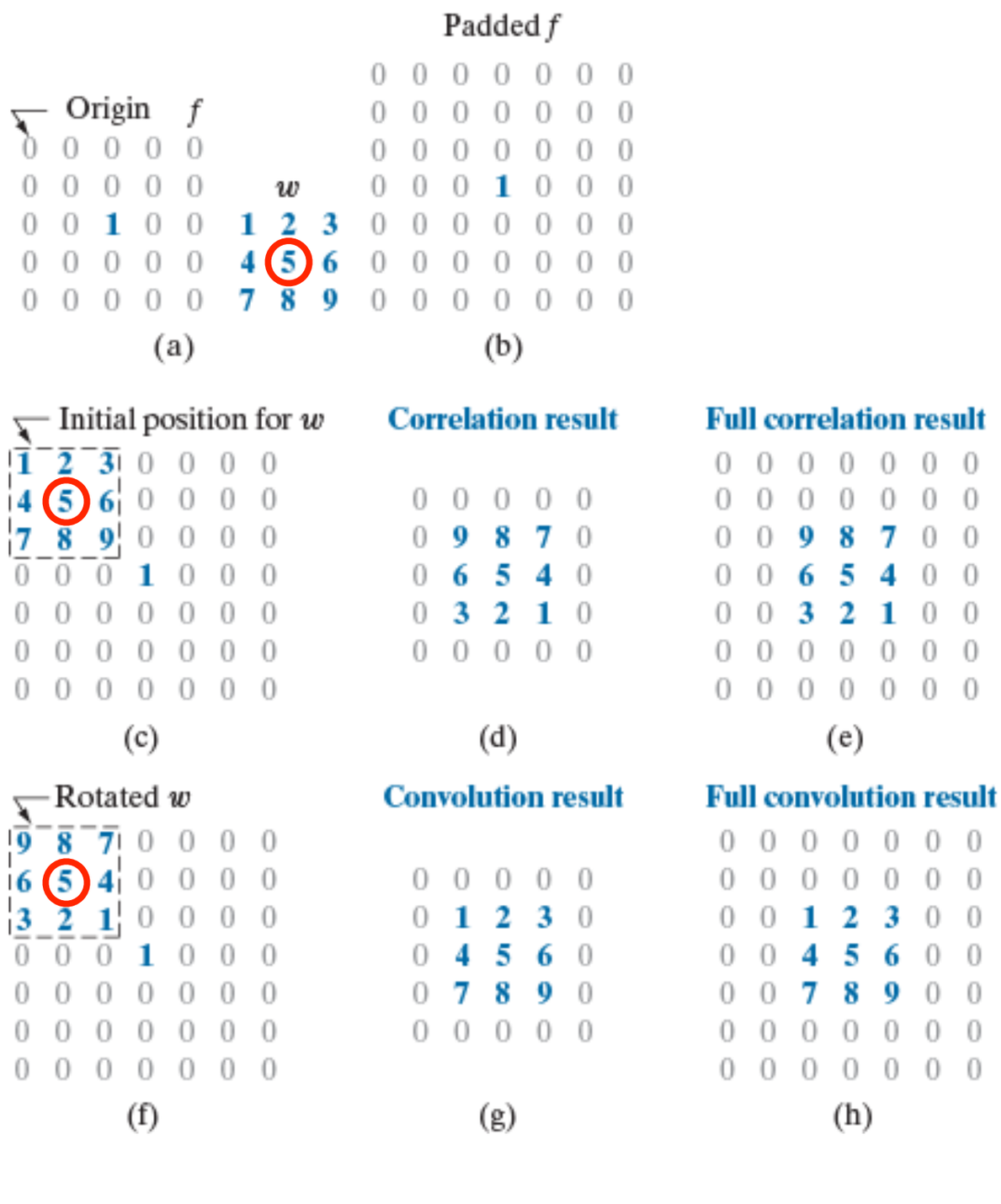
Spatial **Neighborhood** Filtering

Part two: Sharpening

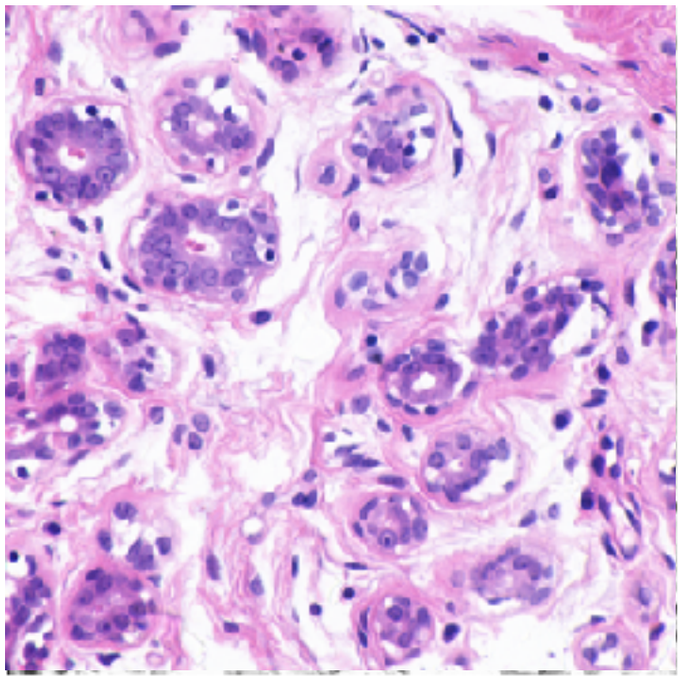
Smoothening **Averaging**

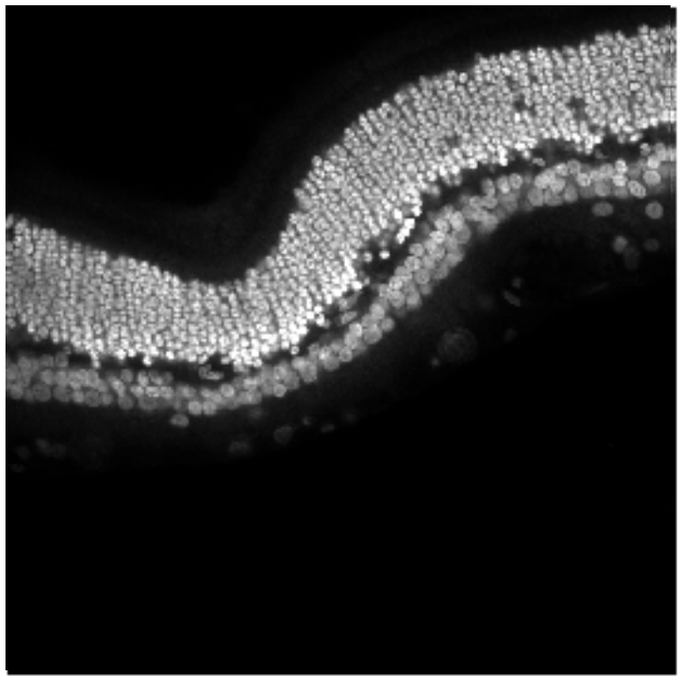
Last time…

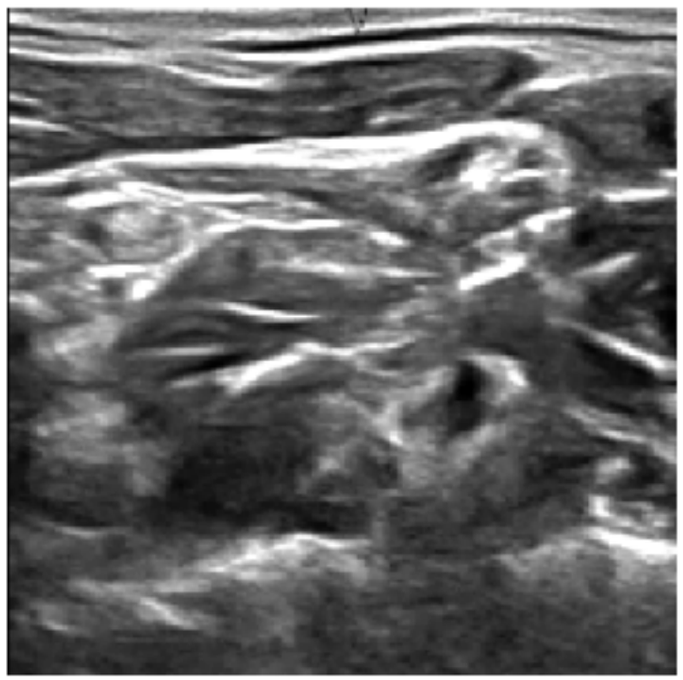


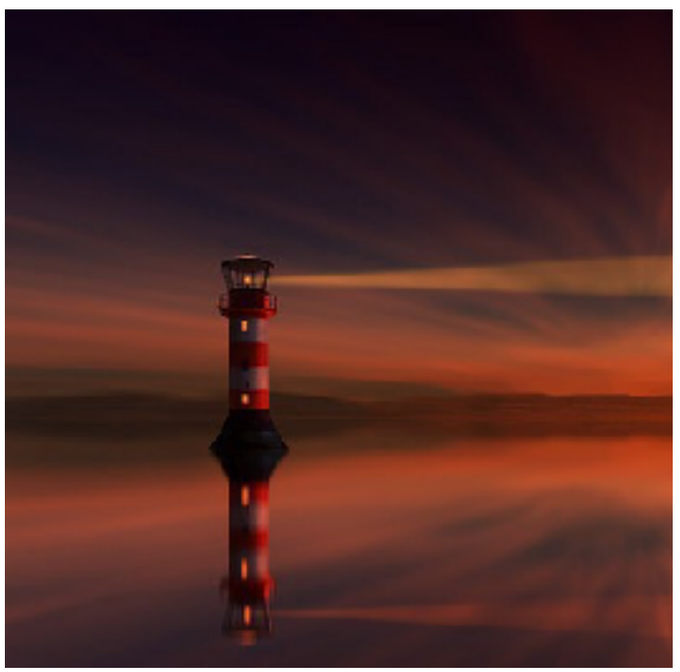


Smoothening…

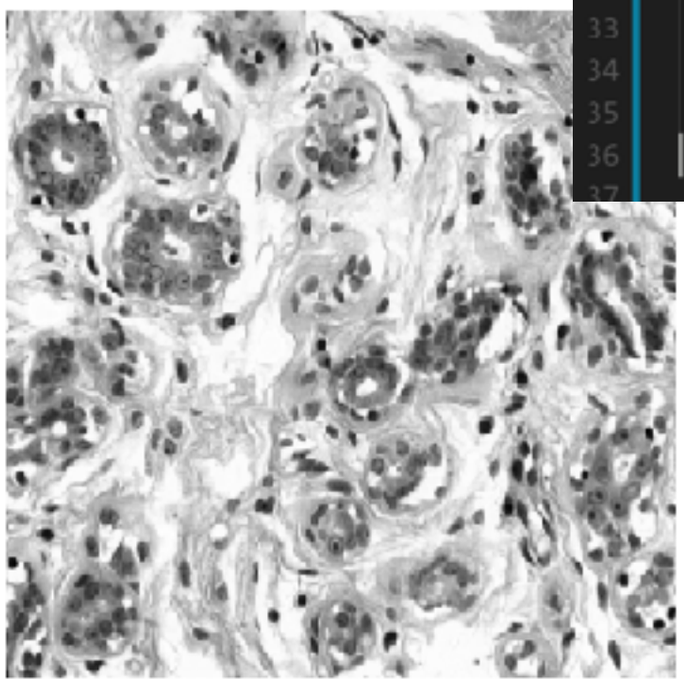


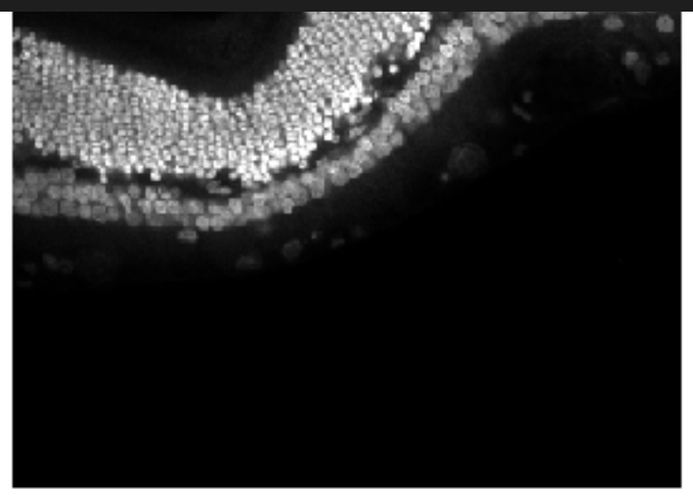


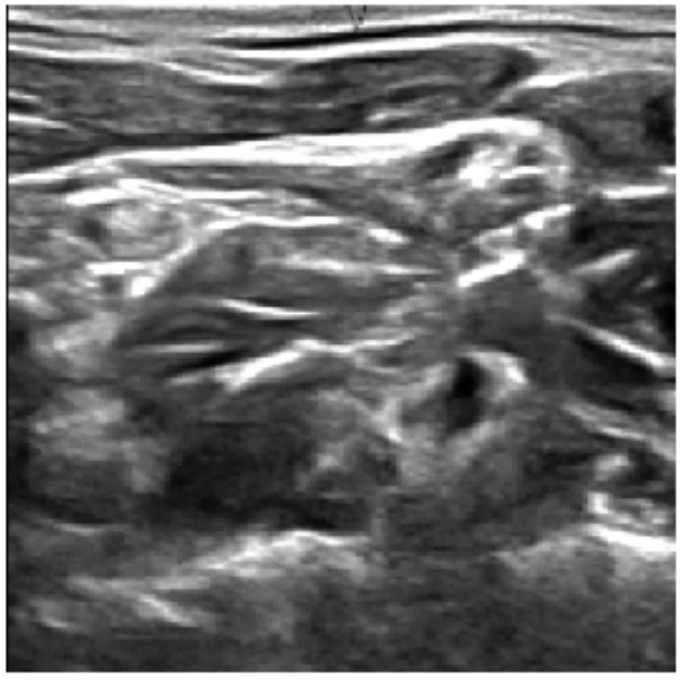


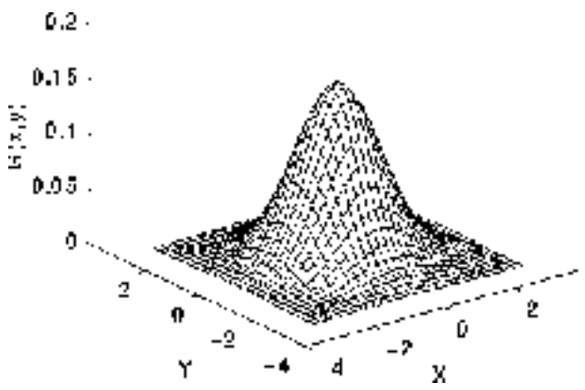


```python
# 1. Identity
kernerl = np.array([
    [0,0,0],
    [0,1,0],
    [0,0,0]
])

for x in range(len(one)):
    temp = convolve2d(one[x,:,:],kernerl,mode='same')
    plt.axis('off')
    plt.imshow(temp,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```

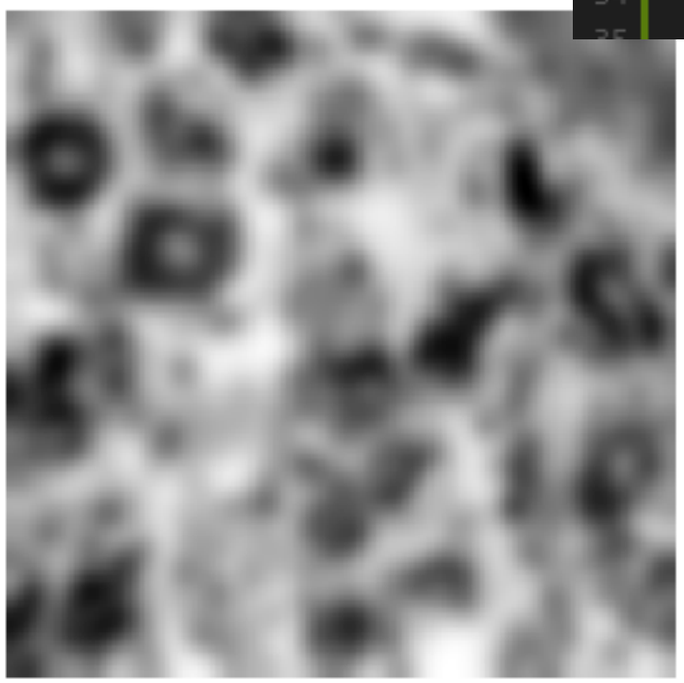


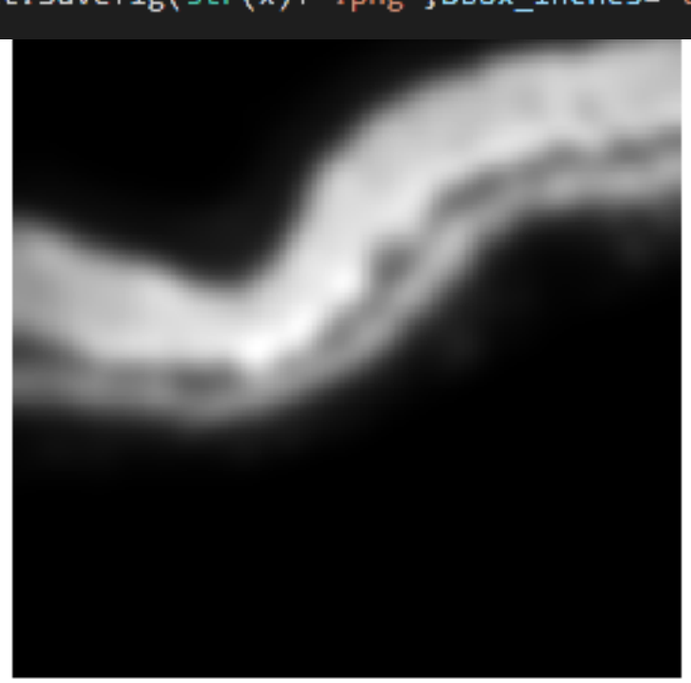


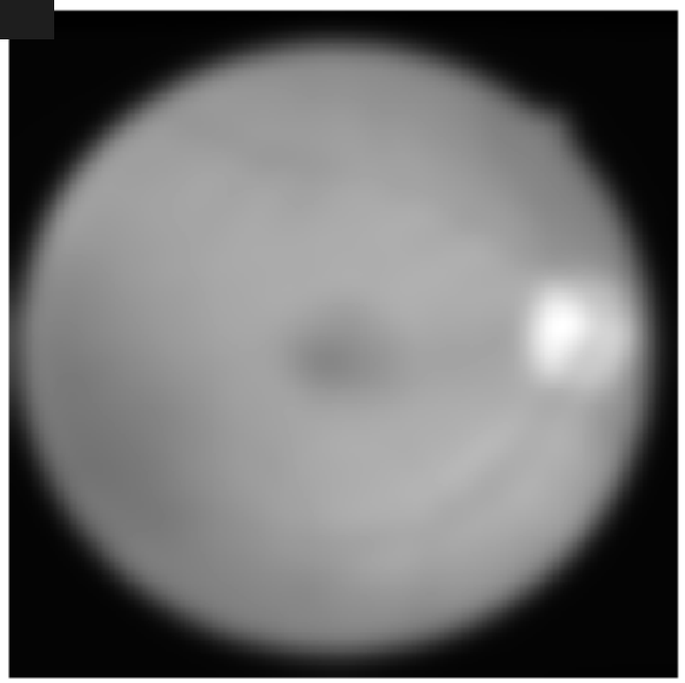


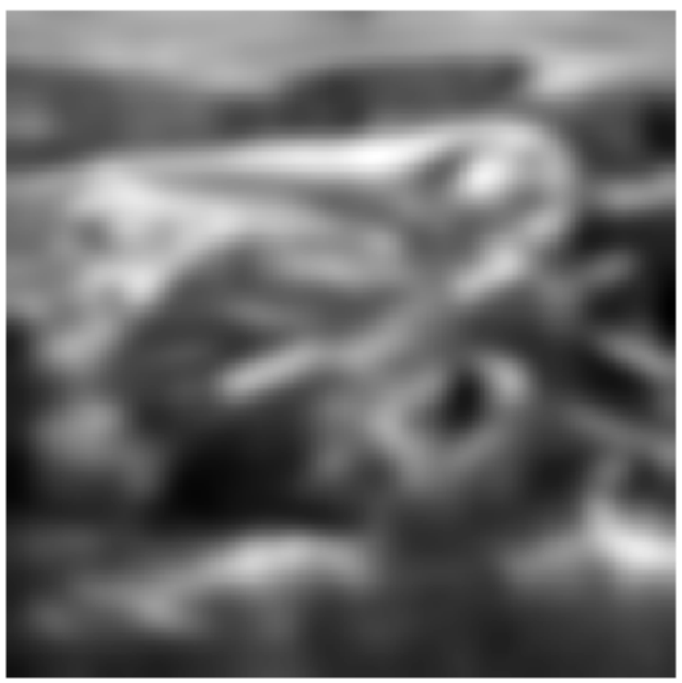


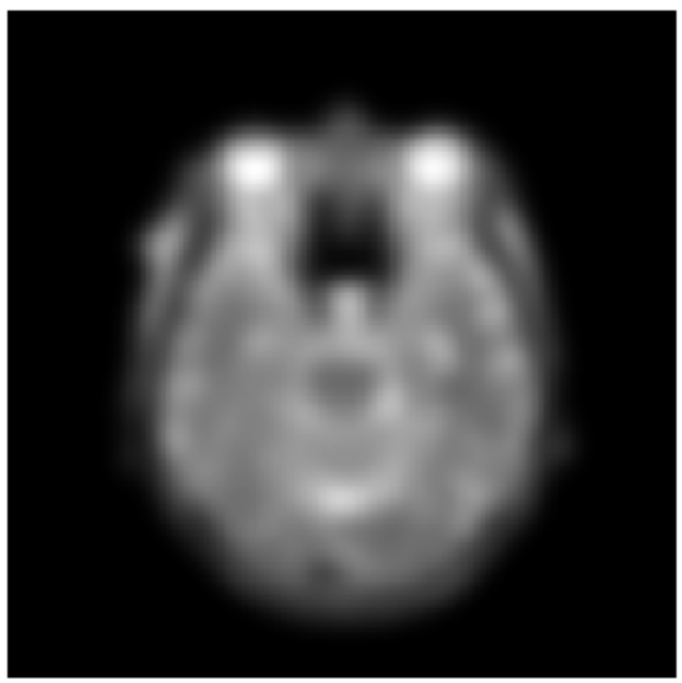

```python
# 8. Guassian Blur
for x in range(len(one)):
    temp = scipy.ndimage.filters.gaussian_filter(
        one[x,:,:],
        sigma = 10
        )
    plt.axis('off')
    plt.imshow(temp,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```











Smoothing by box averaging

Smoothing with a Gaussian

Linear vs. Non-linear filters / kernels

And now some linear intuition…

- An operator H (or system) is linear if two properties hold ($f1$ and $f2$ are some functions, a is a constant):
- Additivity (things sum):
  - $H(f1 + f2) = H(f1) + H(f2)$ (looks like distributive law)
- Multiplicative scaling (Homogeneity of degree 1) (constant scales):
  - $H(a \cdot f1) = a \cdot H(f1)$

Because it is sums and multiplies, the "filtering" operation we were doing are linear.

Correlation vs Convolution

Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

$$G = H \otimes F$$

Flip in both dimensions
(bottom to top, right to left)

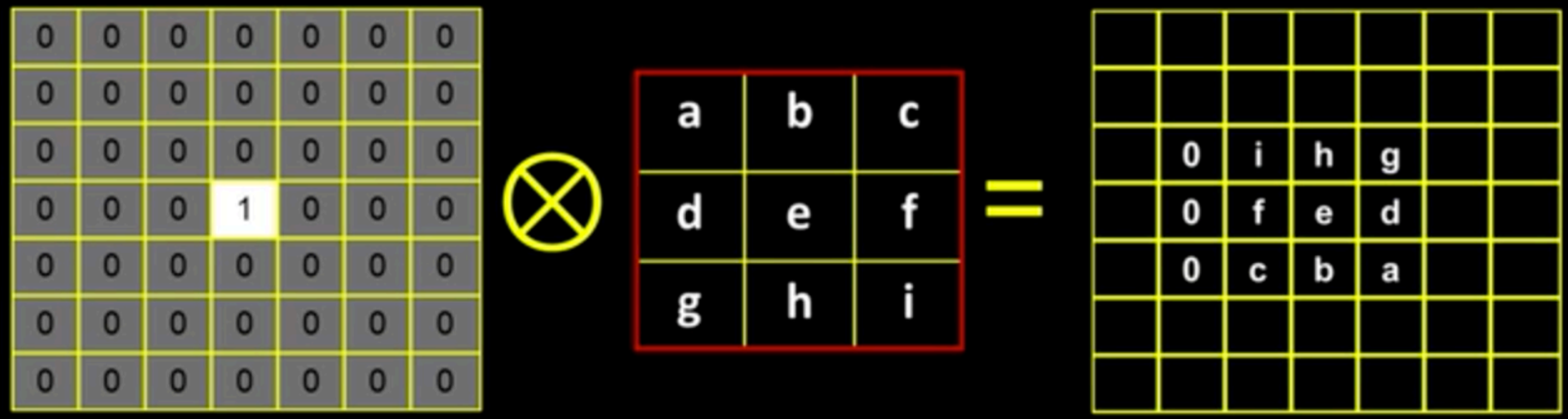
Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

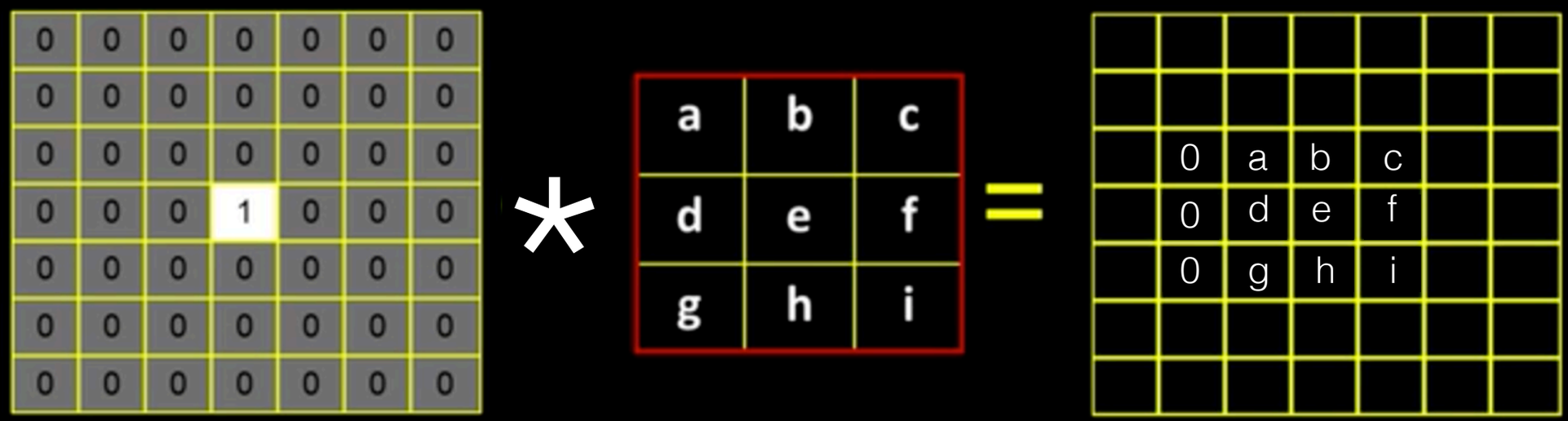
K. Grauman

Correlation:



F(x,y) *H(u,v)* *G(x,y)*

Convolution:



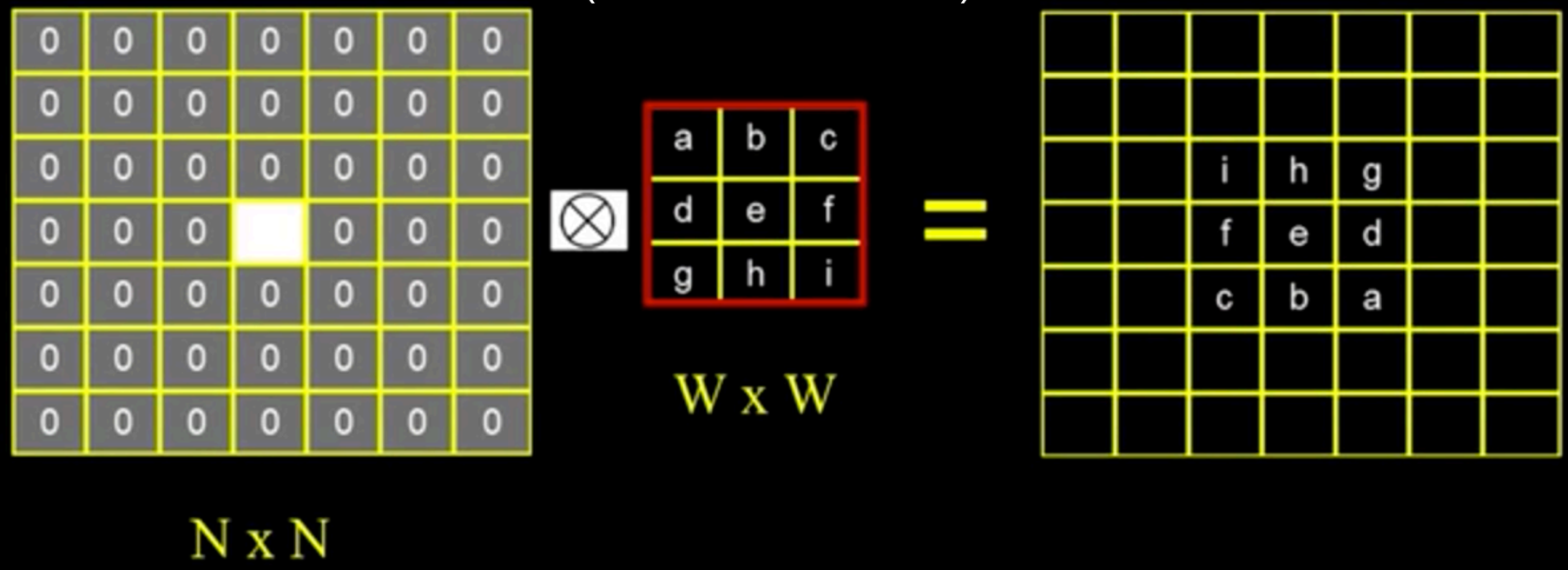
F(x,y) *H(u,v)* *G(x,y)*

Separability

Computational Complexity

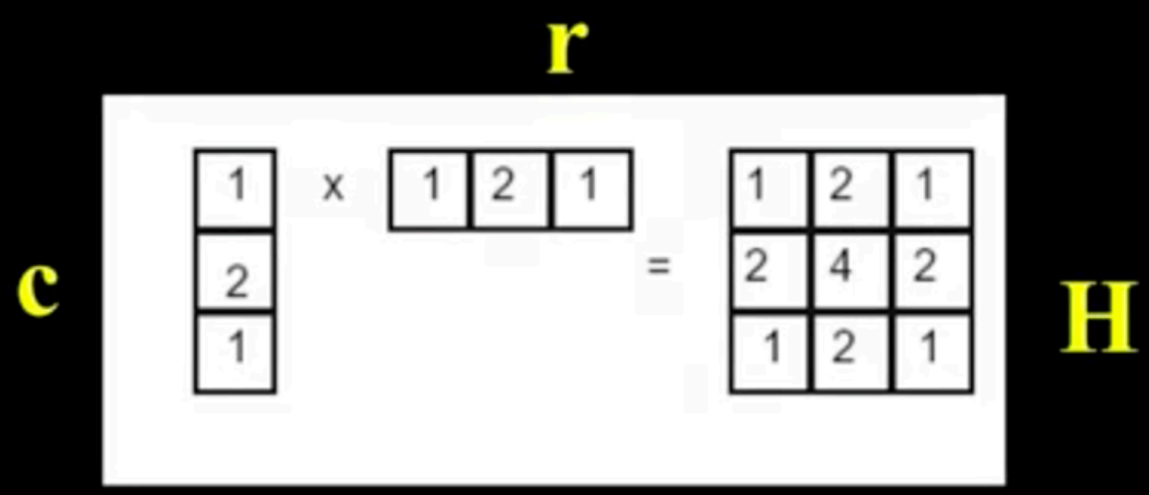
- If an image is *NxN* and a kernel (filter) is WxW, how many multiplies do you need to compute a convolution?

(or correlation)



- You need N*N*W*W = N^2W^2
  - which can get big (ish)

Separability

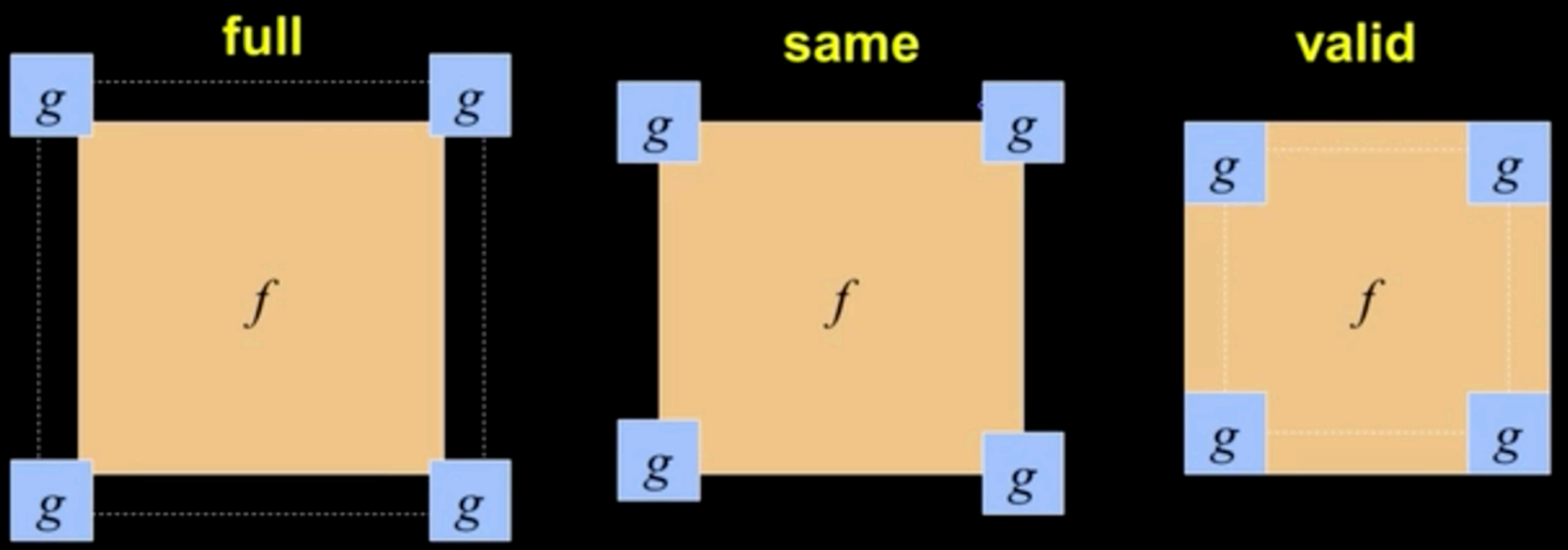


$$G = H * F = (C * R) * F = C * (R * F)$$

- So we do two convolutions but each is W*N*N. So this is useful if W is big enough such that $2 \cdot W \cdot N^2 << W^2 \cdot N^2$
- Used to be ***very*** important. Still, if W=31, save a factor of 15.

Boundary issues (**Padding**)

- What is the size of the output?
- Using old Matlab nomenclature we have three choices:



Source: S. Lazebnik

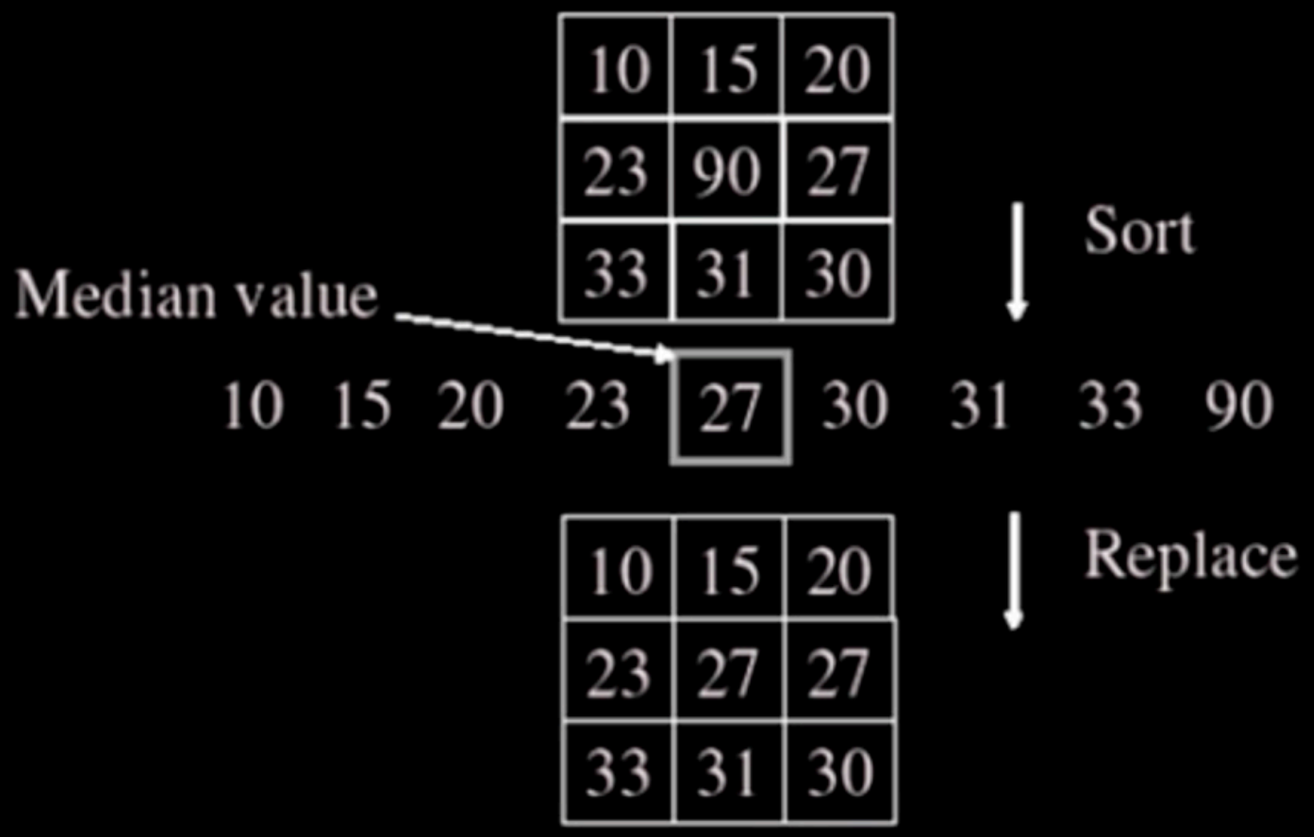
Copy edge / Replicate

Clip / Constant

Reflect edge / Symmetric

Wrap / Circular

Median filter

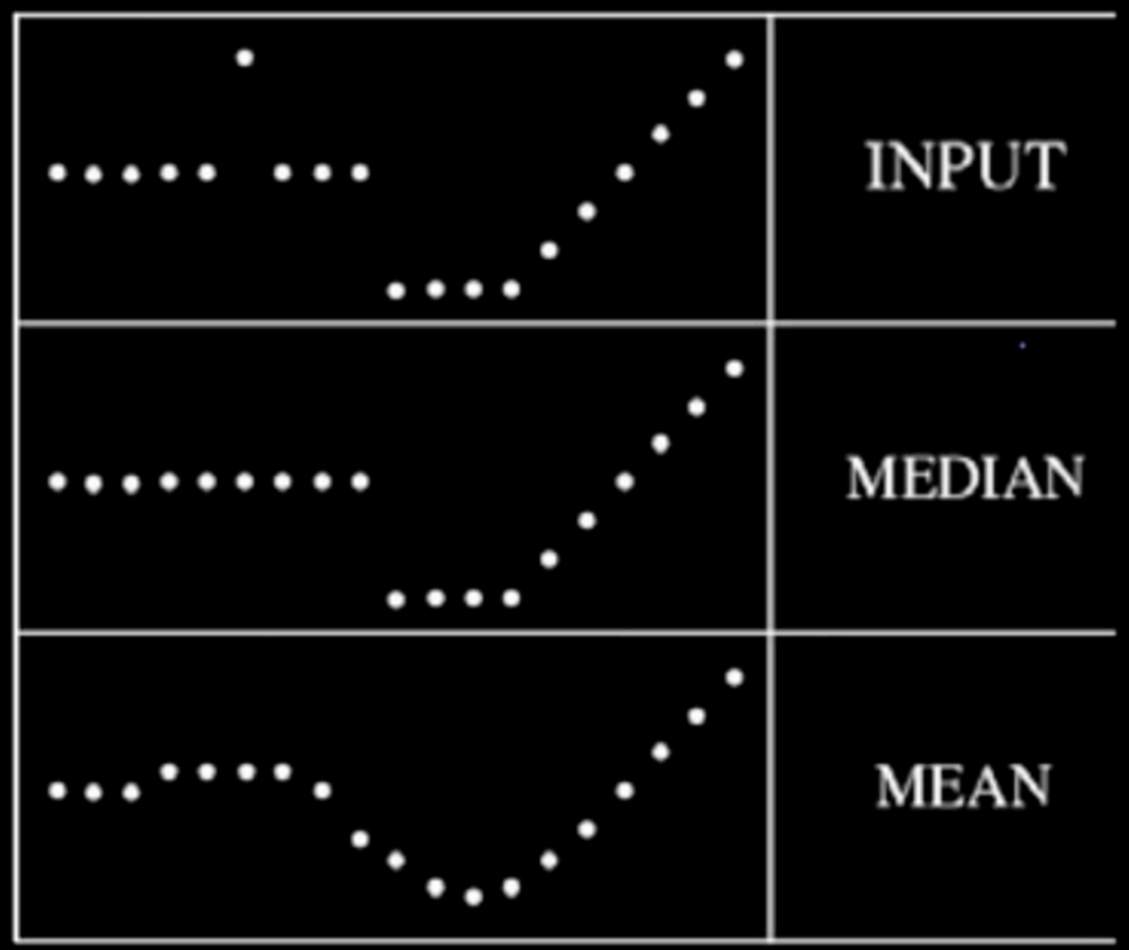


- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Linear? **NO**

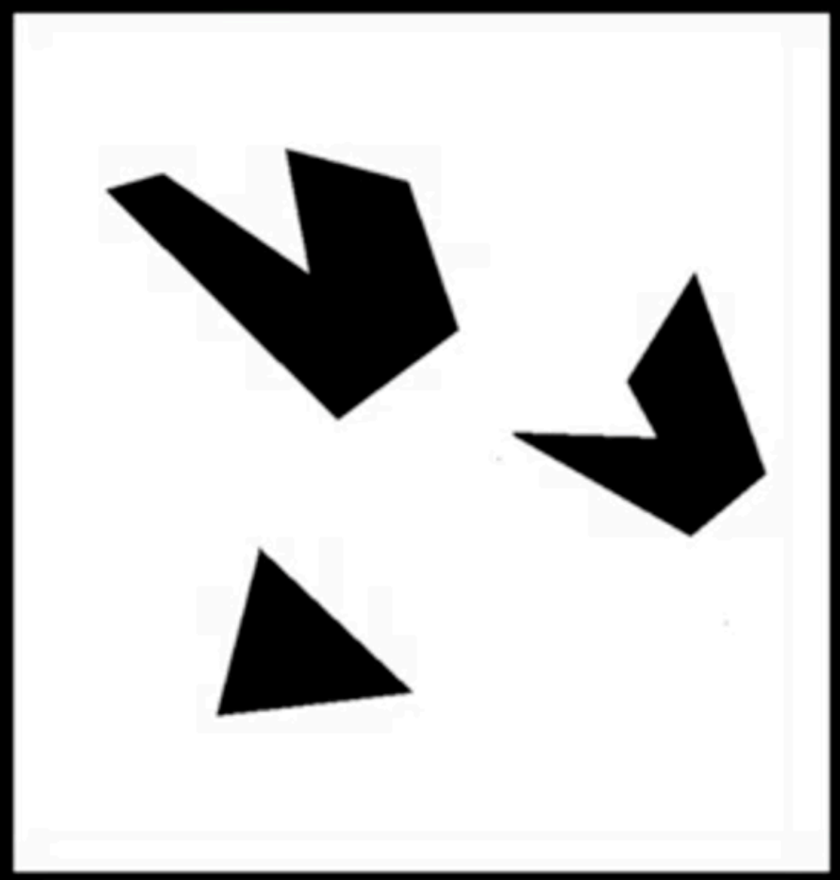
K. Grauman

Median filter

Median filter is edge preserving



Template matching



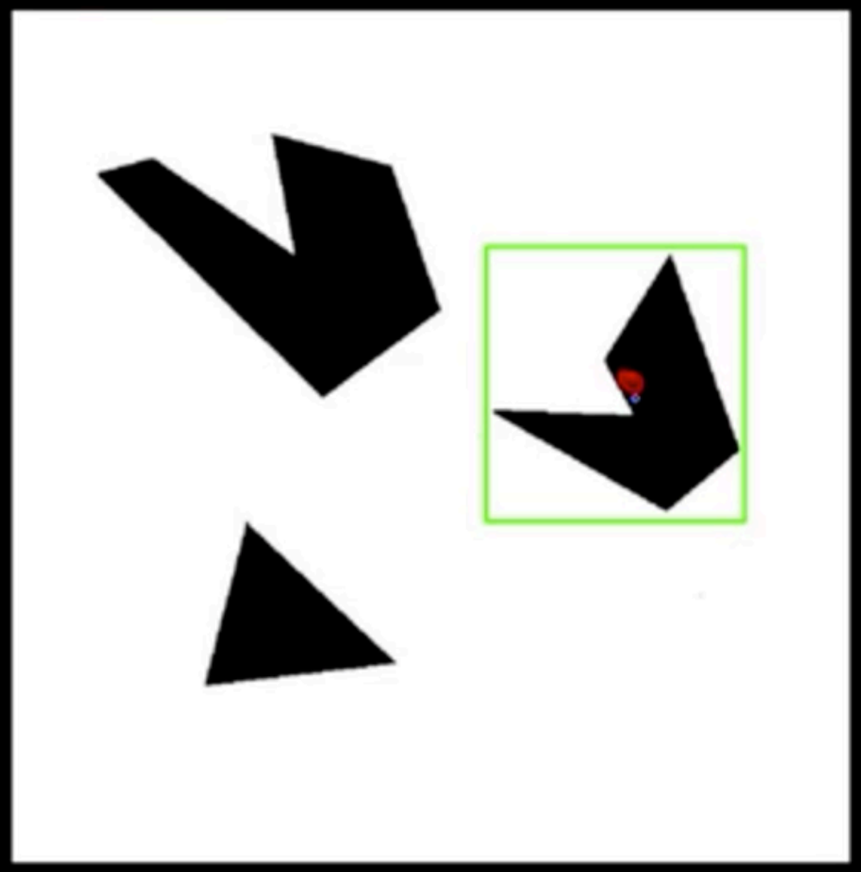
Scene

A toy example

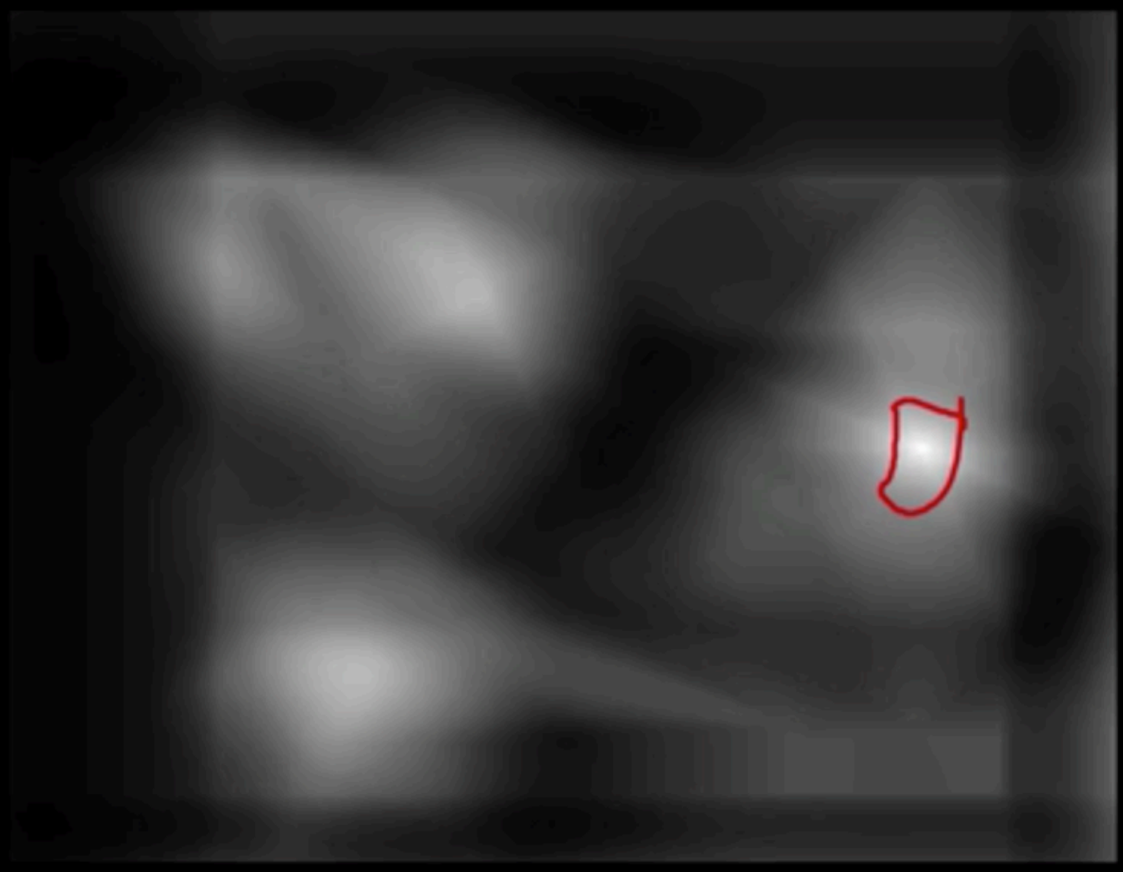
Template (mask)

K. Grauman

Template matching



Detected template

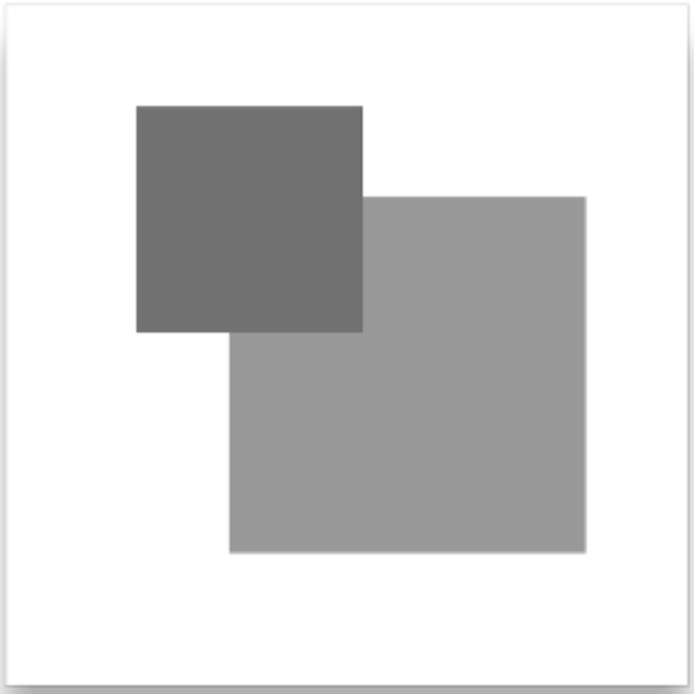


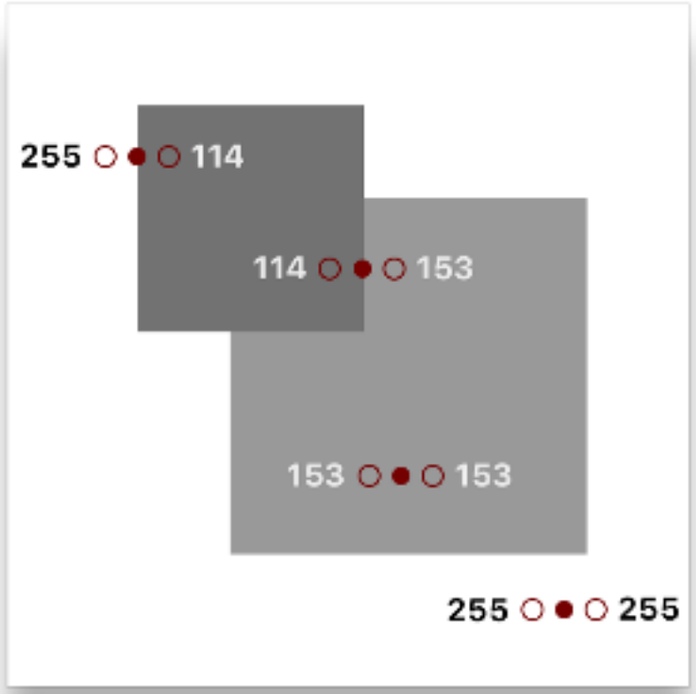
Correlation map

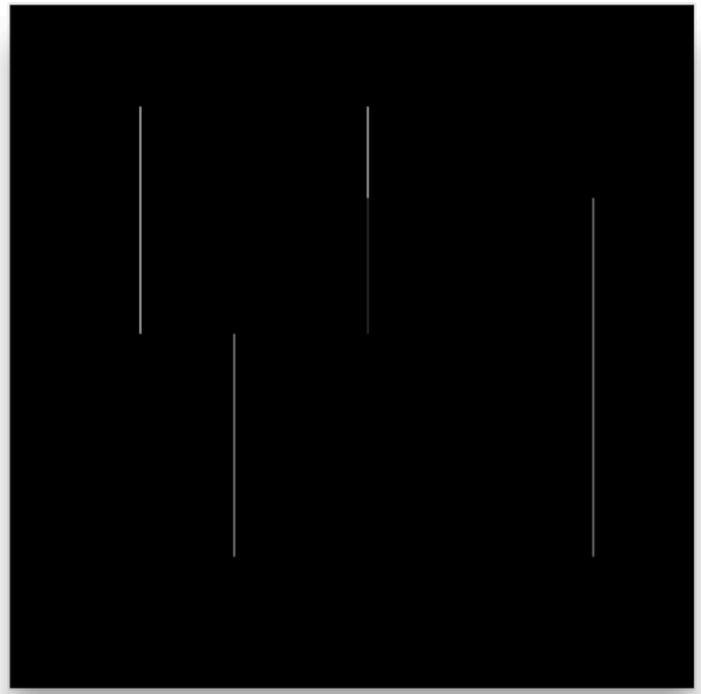
Sharpening

Today…

Differences







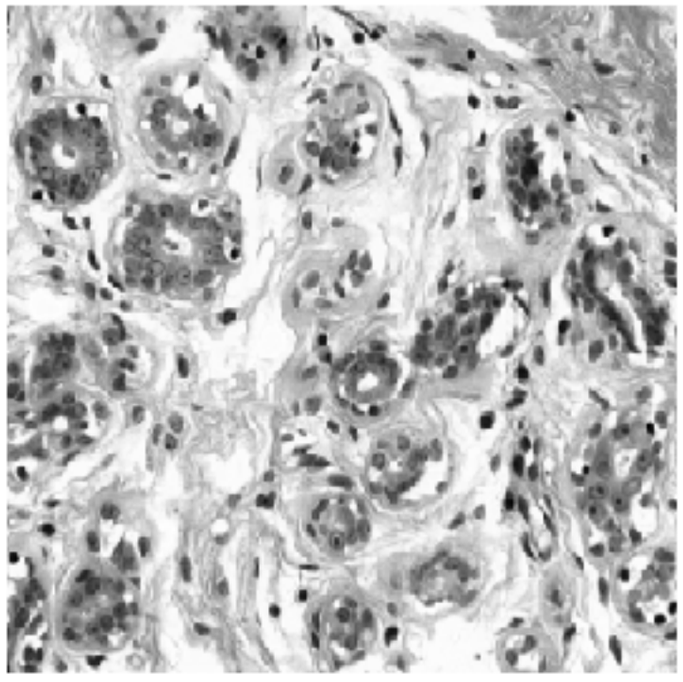
```
kernel = [1  0 -1]
```

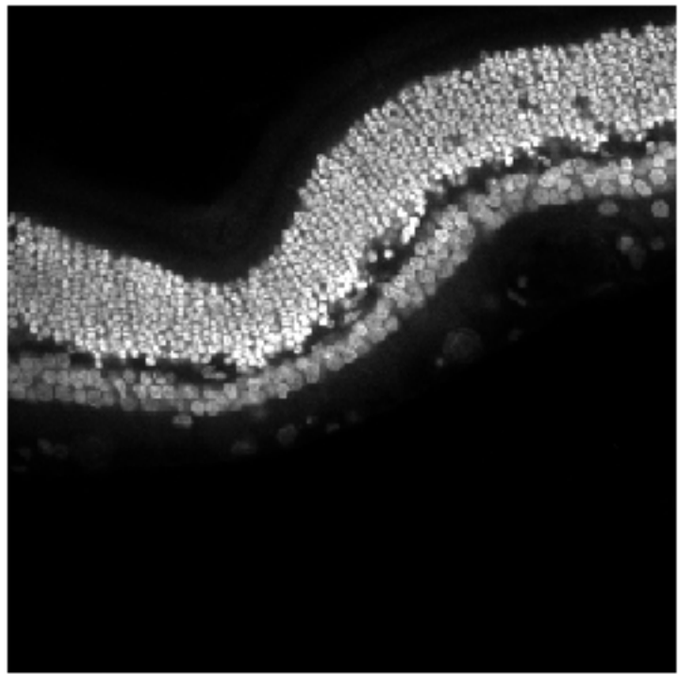
```
kernel = [
  [0  0  0]
  [1  0 -1]
  [0  0  0]
]
```

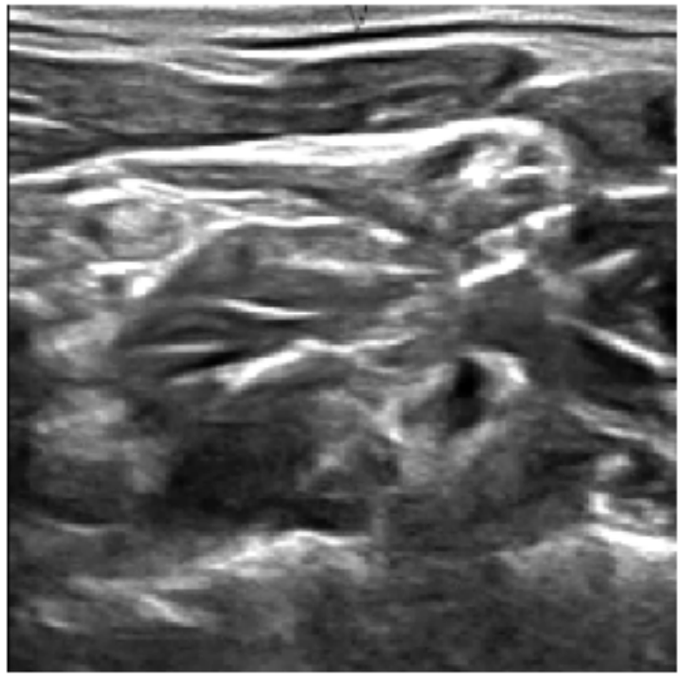
255 × 0	255 × 0	255 × 0	114	114	114	114
255 × 1	255 × 0	255 × -1	114	114	114	114
255 × 0	255 × 0	255 × 0	114	114	114	114
255	255	255	114	114	114	114
255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255

0				

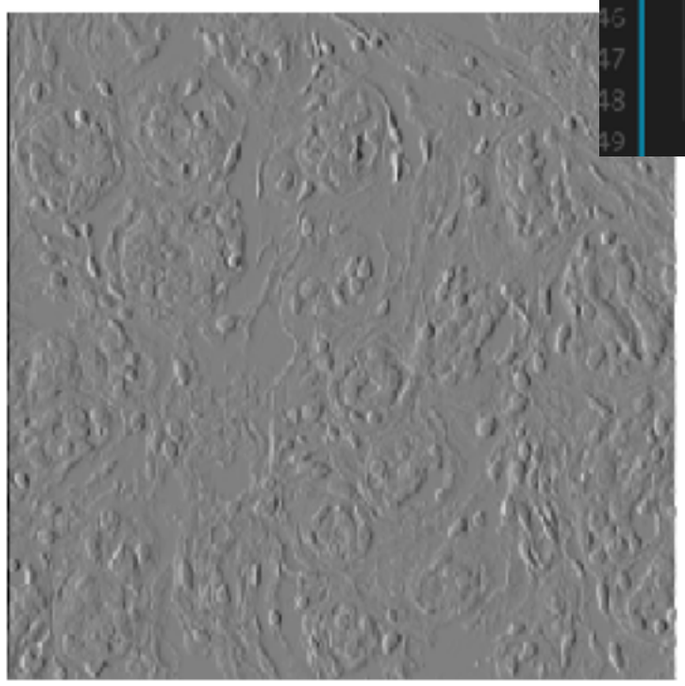
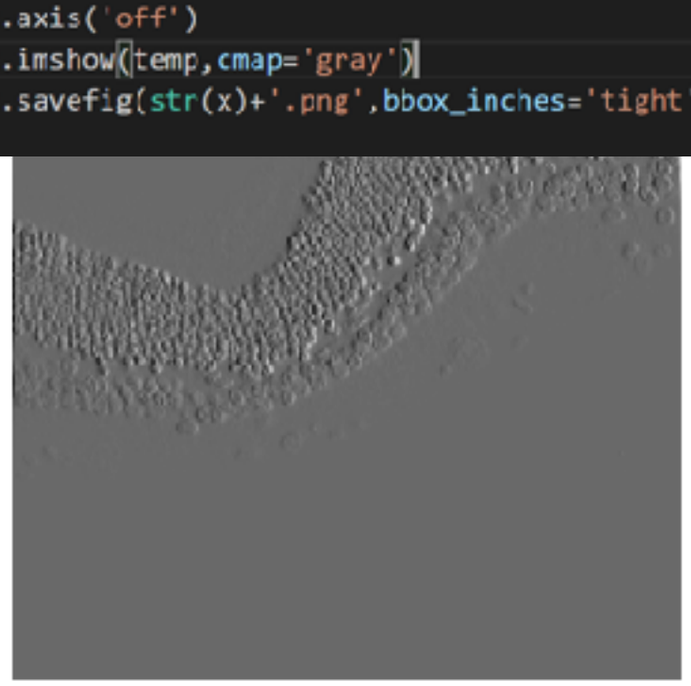
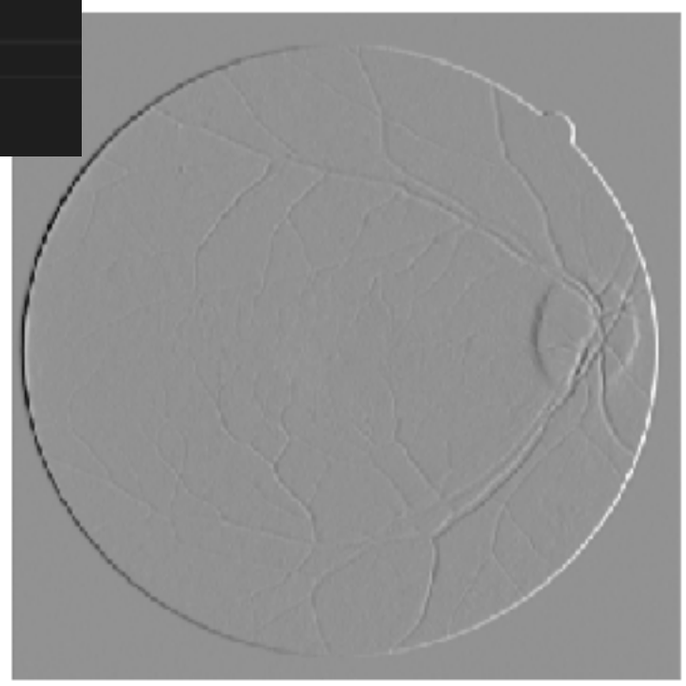
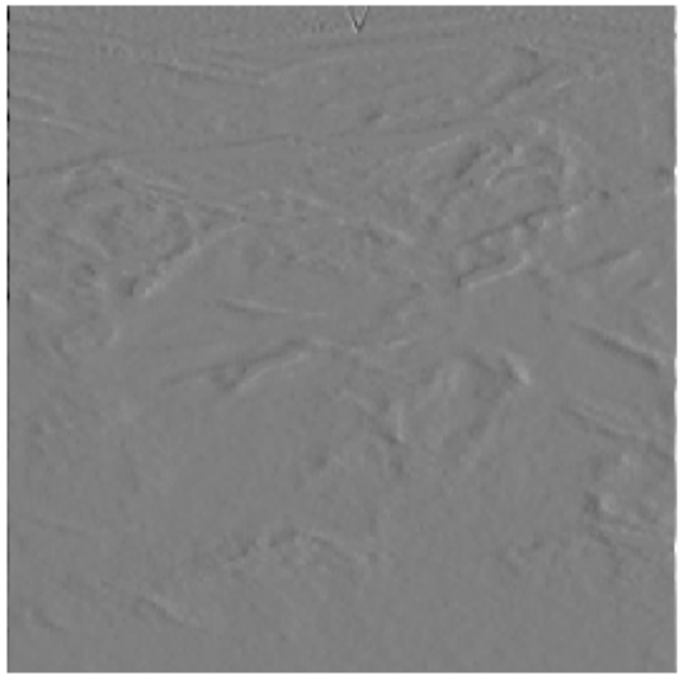
Sharpening…

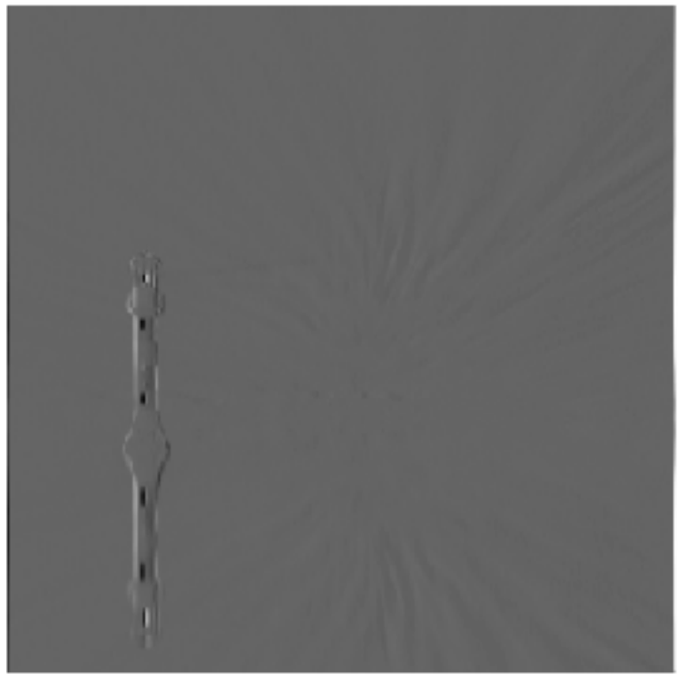




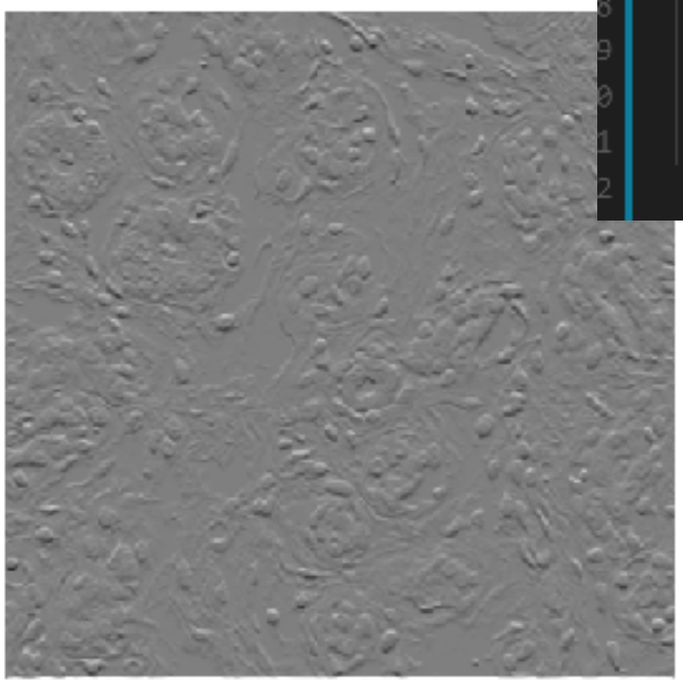
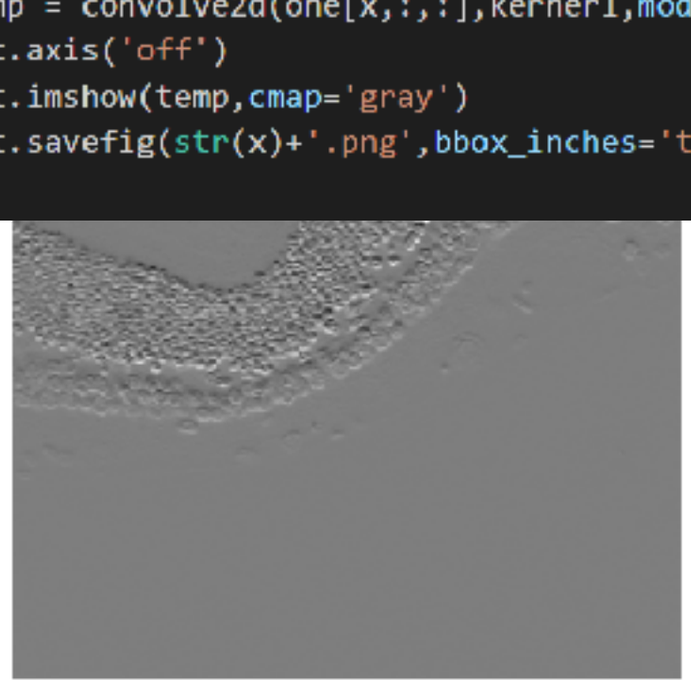
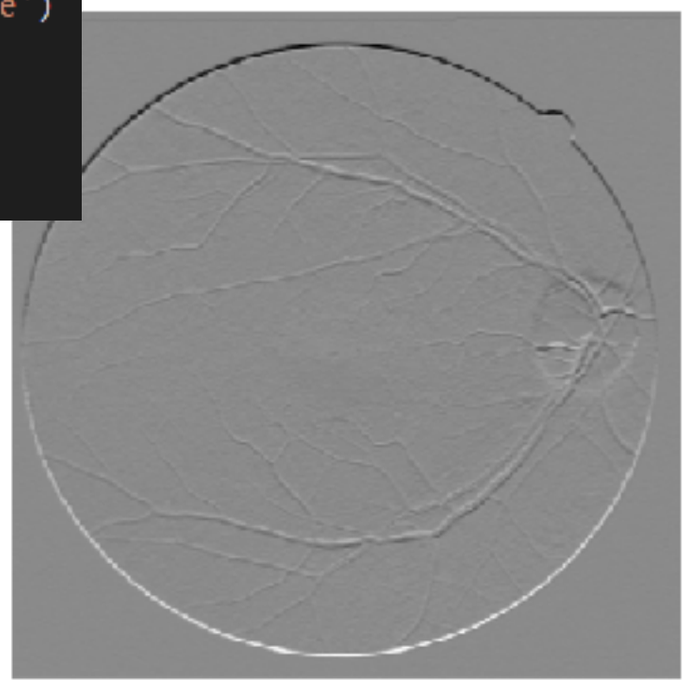
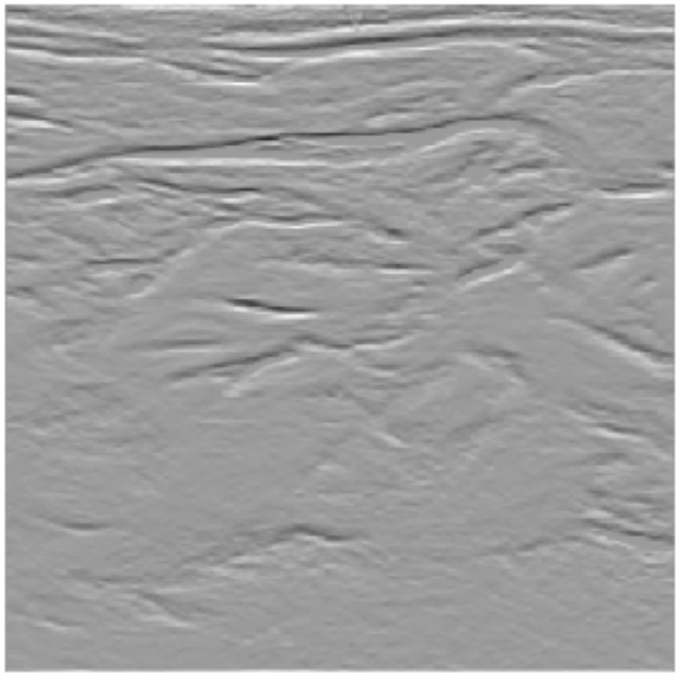


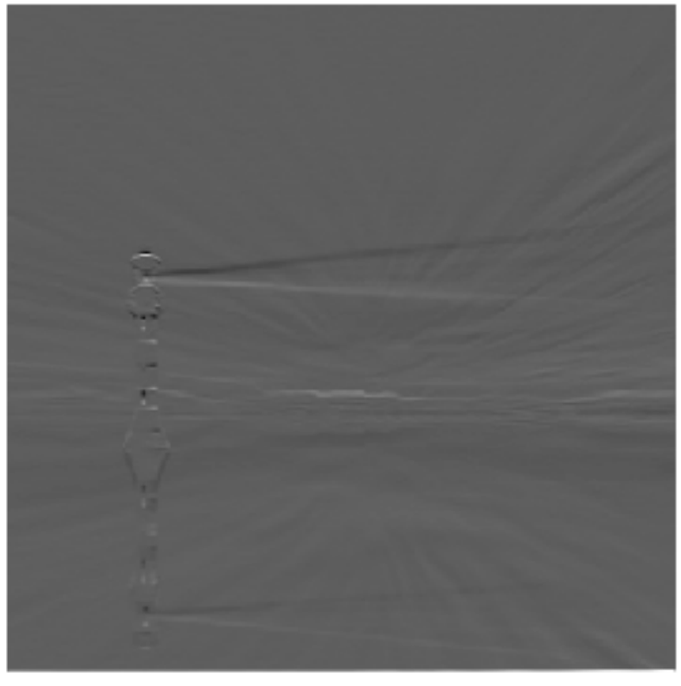
```python
# 2. Edge Detection (Horizontal)
kernerl = np.array([
    [-1,0,1],
    [-1,0,1],
    [-1,0,1]
])

for x in range(len(one)):
    temp = convolve2d(one[x,:,:],kernerl,mode='same')
    plt.axis('off')
    plt.imshow(temp,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```



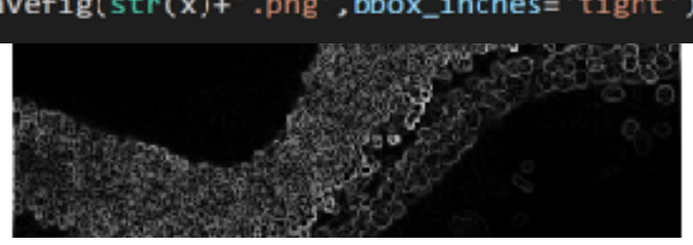
```python
# 3. Edge Detection (Vertical)
kernerl = np.array([
    [-1,-1,-1],
    [0,0,0],
    [1,1,1]
])

for x in range(len(one)):
    temp = convolve2d(one[x,:,:],kernerl,mode='same')
    plt.axis('off')
    plt.imshow(temp,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```

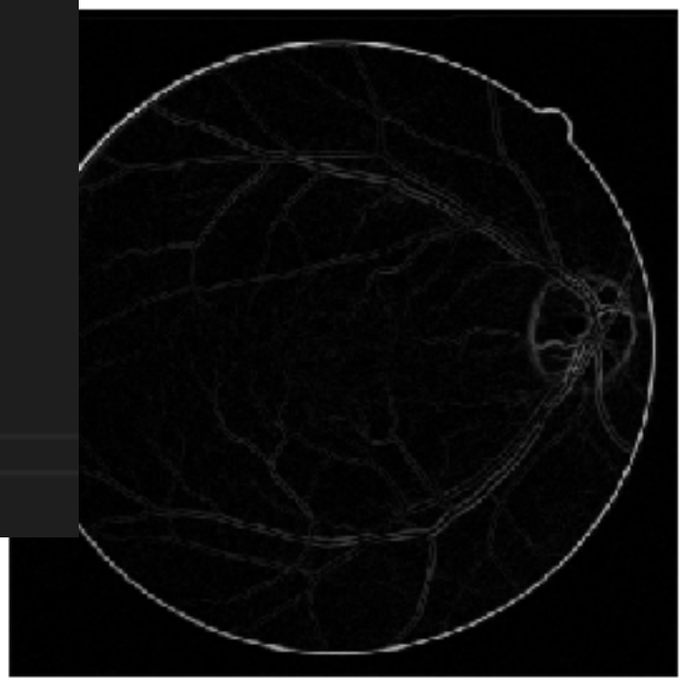





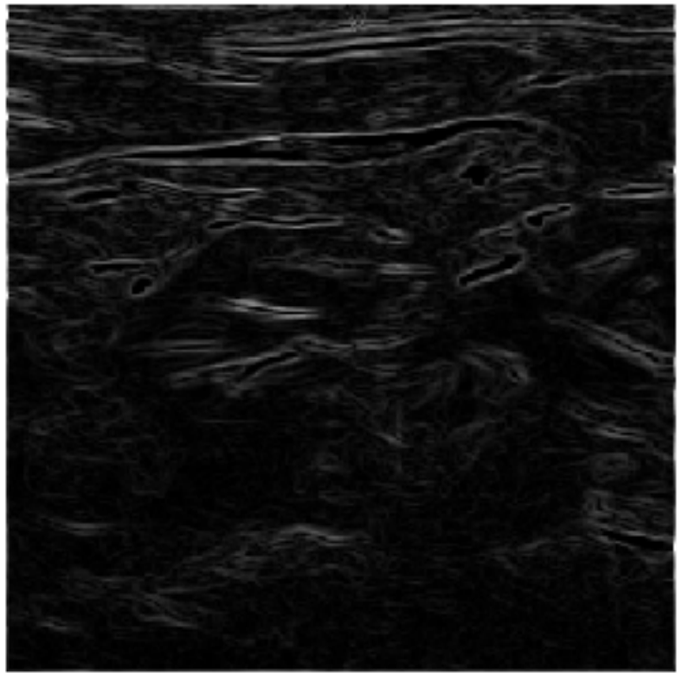


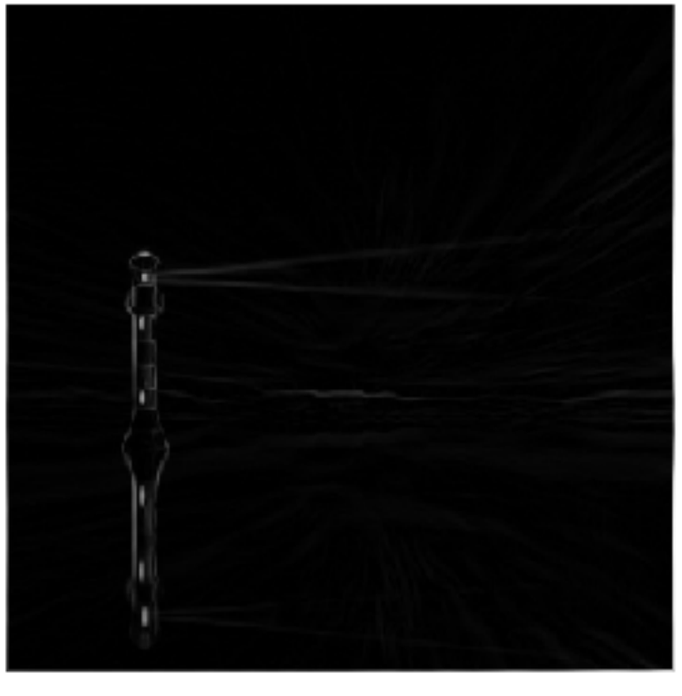
$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

```python
# 4. Gradient Magnitude
kernerl1 = np.array([
    [-1,-1,-1],
    [0,0,0],
    [1,1,1]
])
kernerl2 = np.array([
    [-1,0,1],
    [-1,0,1],
    [-1,0,1]
])

for x in range(len(one)):
    temp1 = convolve2d(one[x,:,:],kernerl1,mode='same')
    temp2 = convolve2d(one[x,:,:],kernerl2,mode='same')

    temp3 = np.sqrt(temp1**2 + temp2**2)

    plt.axis('off')
    plt.imshow(temp3,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```



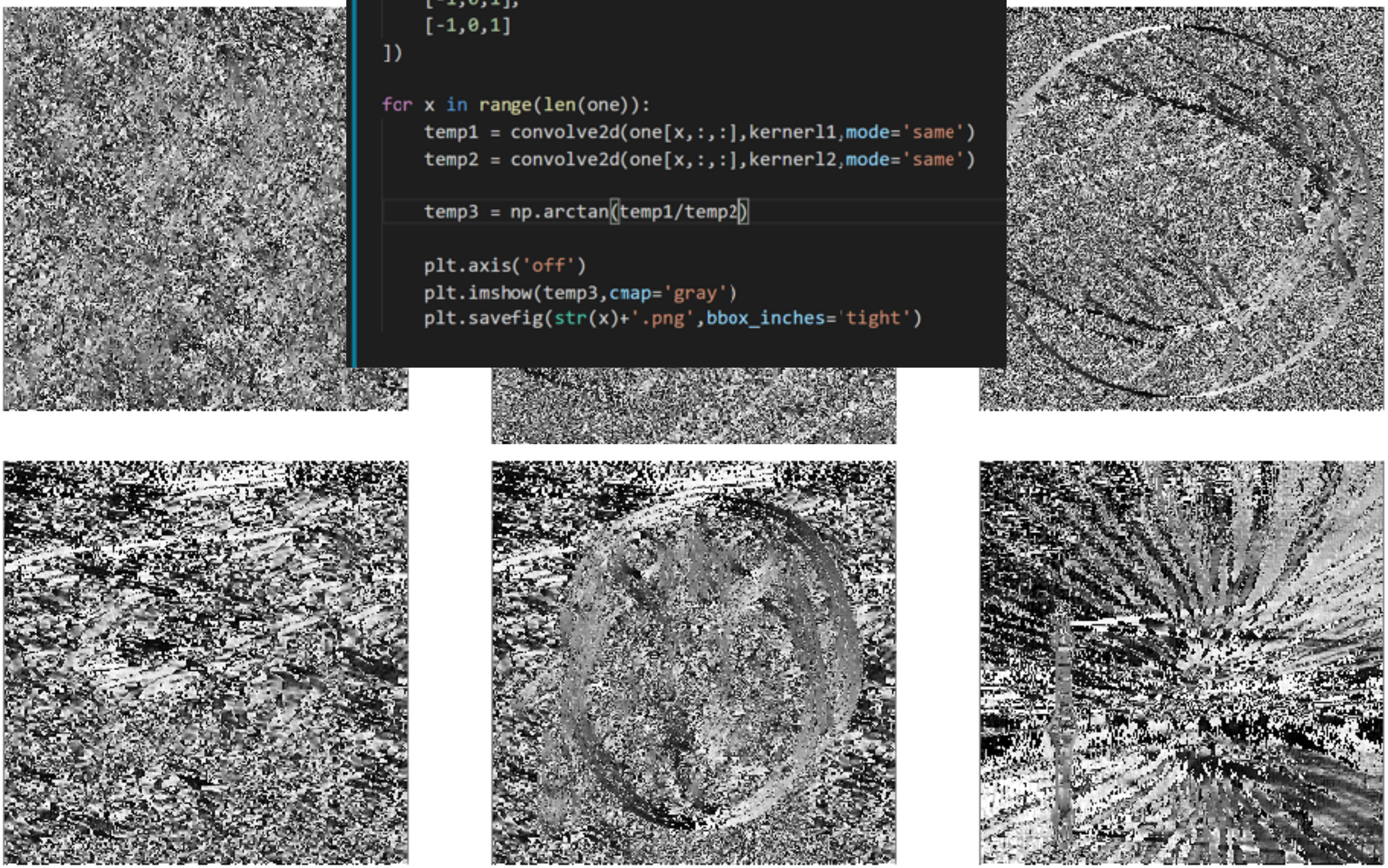






$$\Theta = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

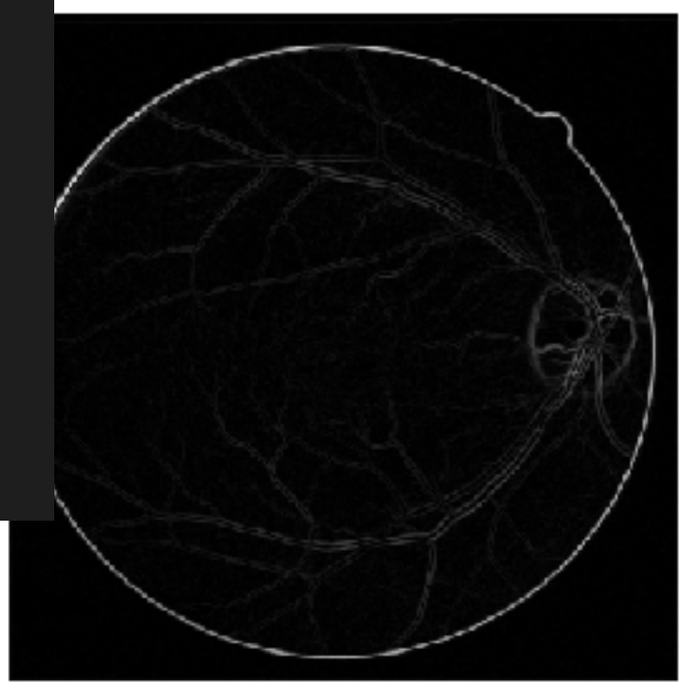
```python
# 5. Gradient Direction
kernerl1 = np.array([
    [-1,-1,-1],
    [0,0,0],
    [1,1,1]
])
kernerl2 = np.array([
    [-1,0,1],
    [-1,0,1],
    [-1,0,1]
])

for x in range(len(one)):
    temp1 = convolve2d(one[x,:,:],kernerl1,mode='same')
    temp2 = convolve2d(one[x,:,:],kernerl2,mode='same')

    temp3 = np.arctan(temp1/temp2)

    plt.axis('off')
    plt.imshow(temp3,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```

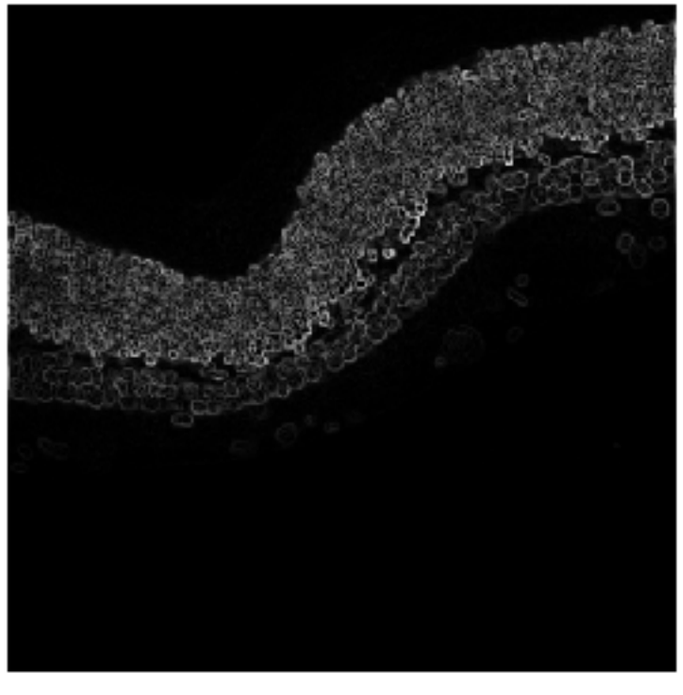
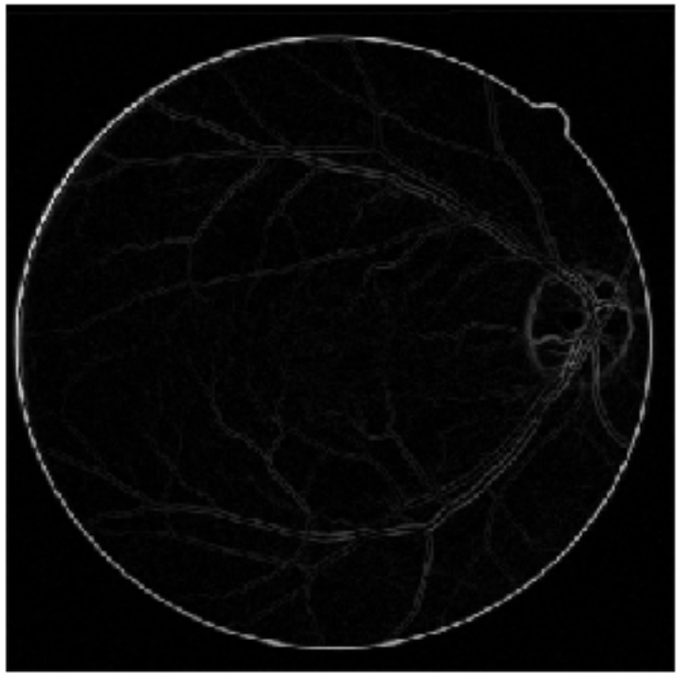


Sobel

```python
# 6. Sobel Gradient Magnitude
kernerl1 = np.array([
    [1,2,1],
    [0,0,0],
    [-1,-2,-1]
])
kernerl2 = np.array([
    [1,0,-1],
    [2,0,-2],
    [1,0,-1]
])

for x in range(len(one)):
    temp1 = convolve2d(one[x,:,:],kernerl1,mode='same')
    temp2 = convolve2d(one[x,:,:],kernerl2,mode='same')

    temp3 = np.sqrt(temp1**2 + temp2**2)

    plt.axis('off')
    plt.imshow(temp3,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```

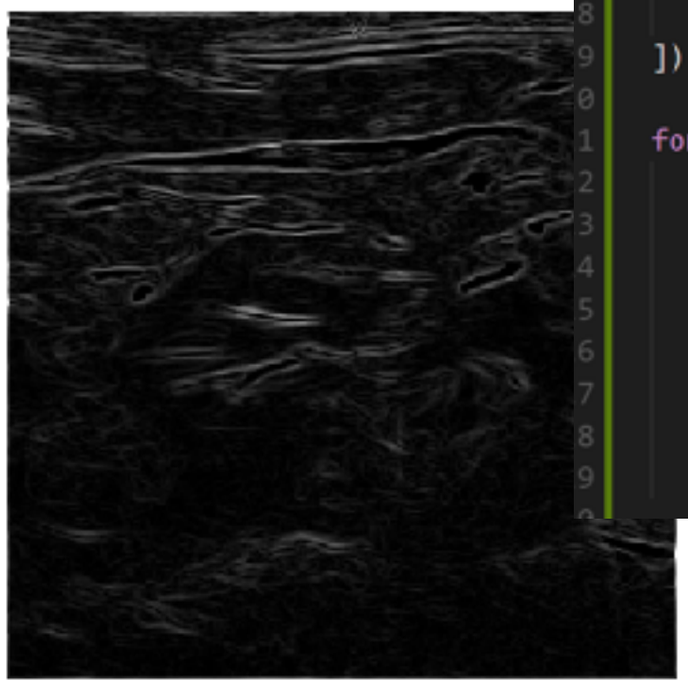


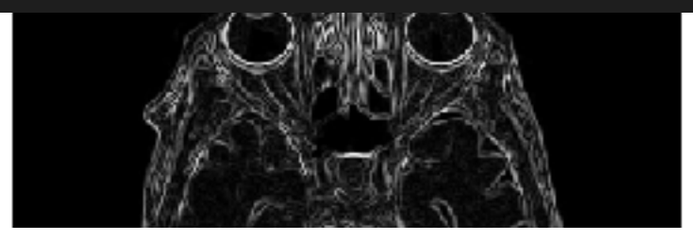


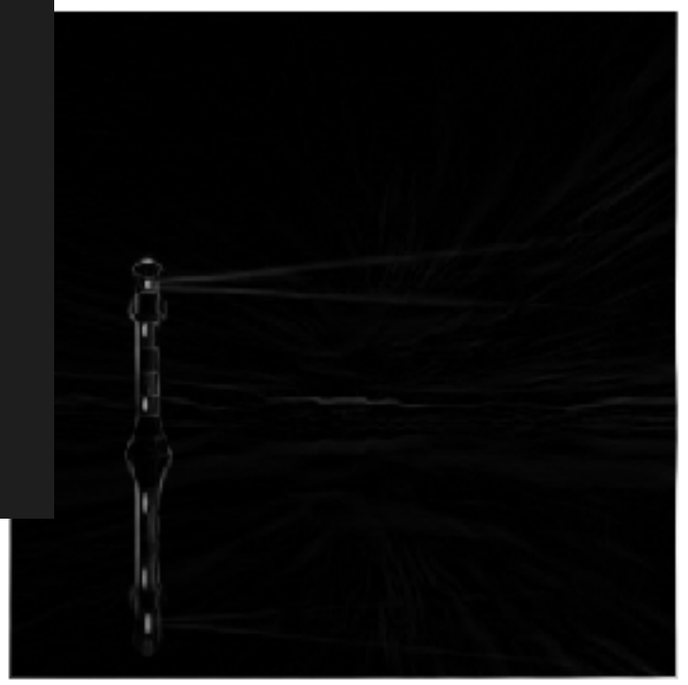



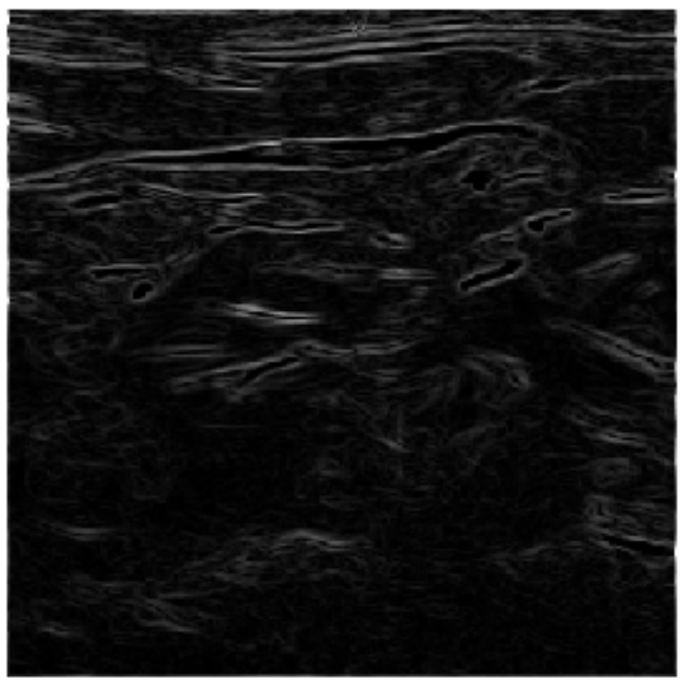
Sobel

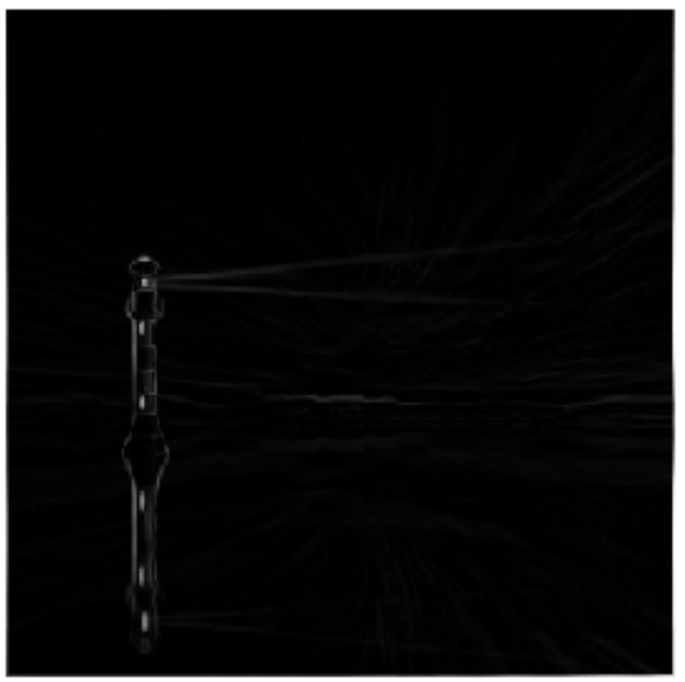
```python
# 6. Sobel Gradient Magnitude
kernerl1 = np.array([
    [1,2,1],
    [0,0,0],
    [-1,-2,-1]
])
kernerl2 = np.array([
    [1,0,-1],
    [2,0,-2],
    [1,0,-1]
])

for x in range(len(one)):
    temp1 = convolve2d(one[x,:,:],kernerl1,mode='same')
    temp2 = convolve2d(one[x,:,:],kernerl2,mode='same')

    temp3 = np.sqrt(temp1**2 + temp2**2)

    plt.axis('off')
    plt.imshow(temp3,cmap='gray')
    plt.savefig(str(x)+'.png',bbox_inches='tight')
```



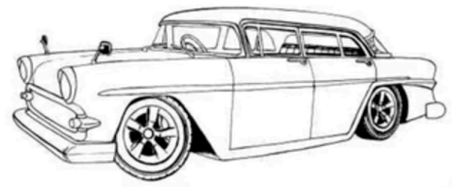


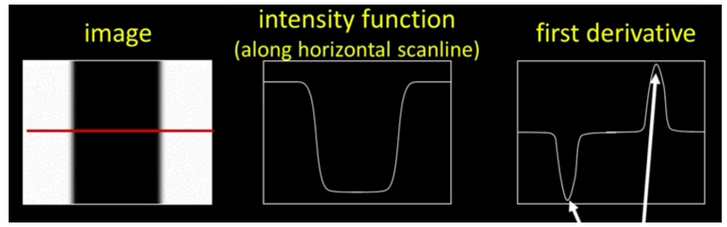


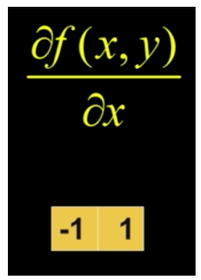


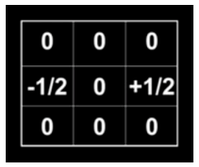


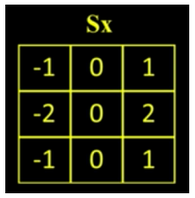
Today…

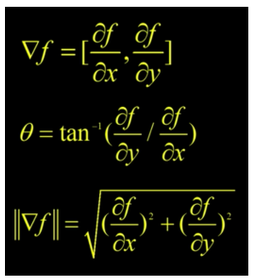




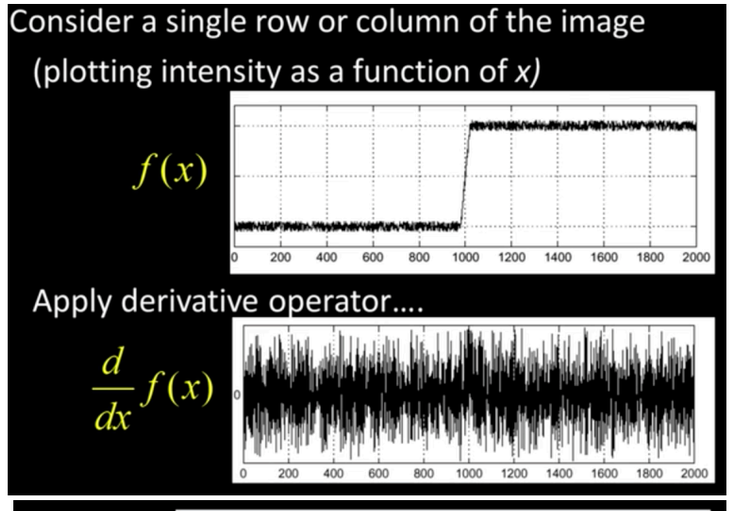








- «Reduced» images tells a lot
- Changes in intensity, Edge detection, derivatives (finite differences) and gradients.
- Noise (Smooth first)
- Derivative of Gaussian, sigma, scale
- Canny edge detector
- 2nd order derivatives, laplacian



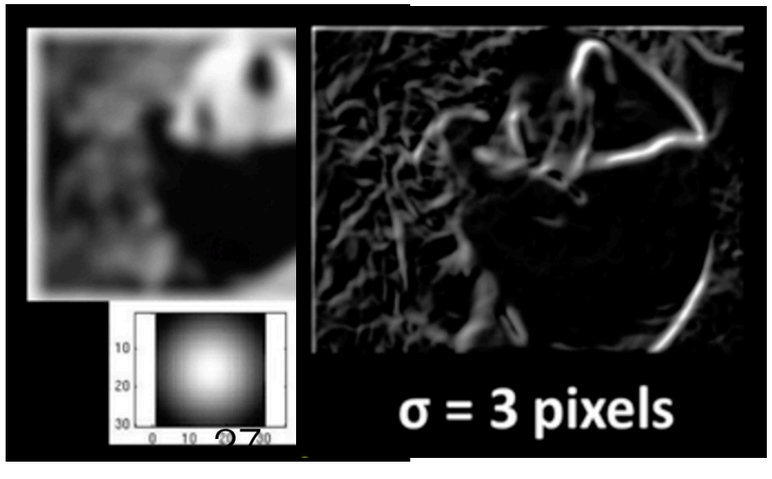
$$\frac{\partial^2 f(x)}{\partial x^2} \approx f(x-1) - 2f(x) + f(x+1)$$

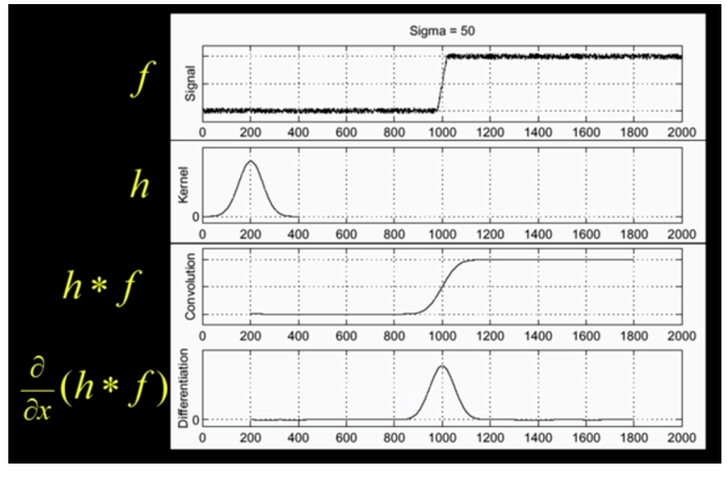
1	-2	1

Filter mask

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

0	1	0
1	-4	1
0	1	0

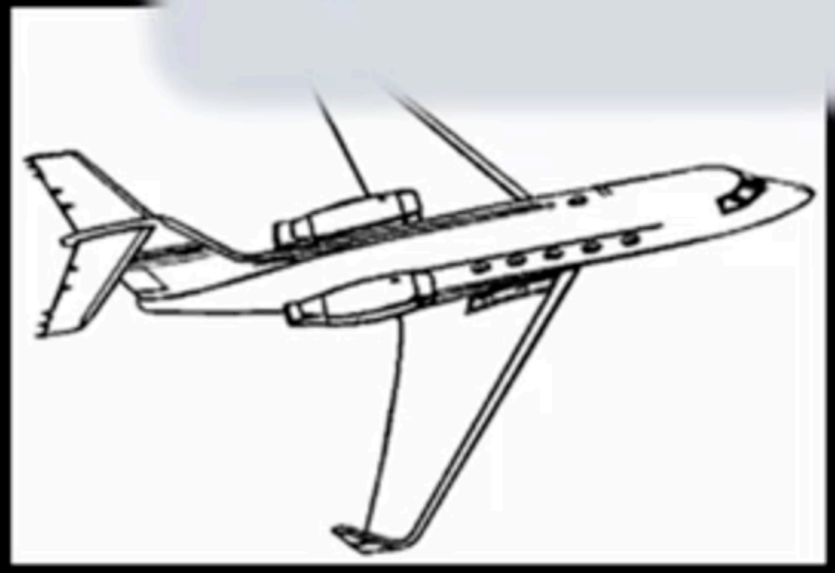




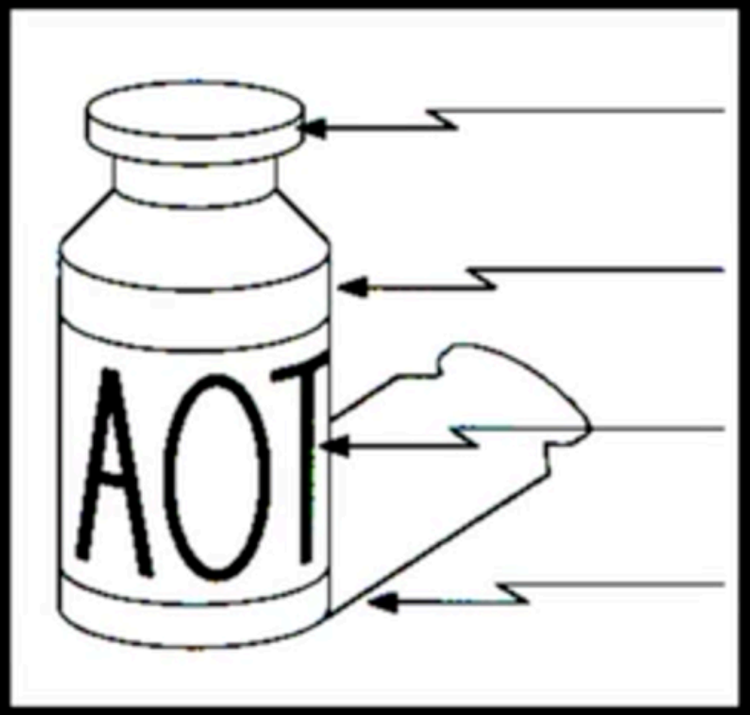
Edge detection - Gradients

Reduced images

Edges seem to be important…



Origin of Edges



surface normal discontinuity

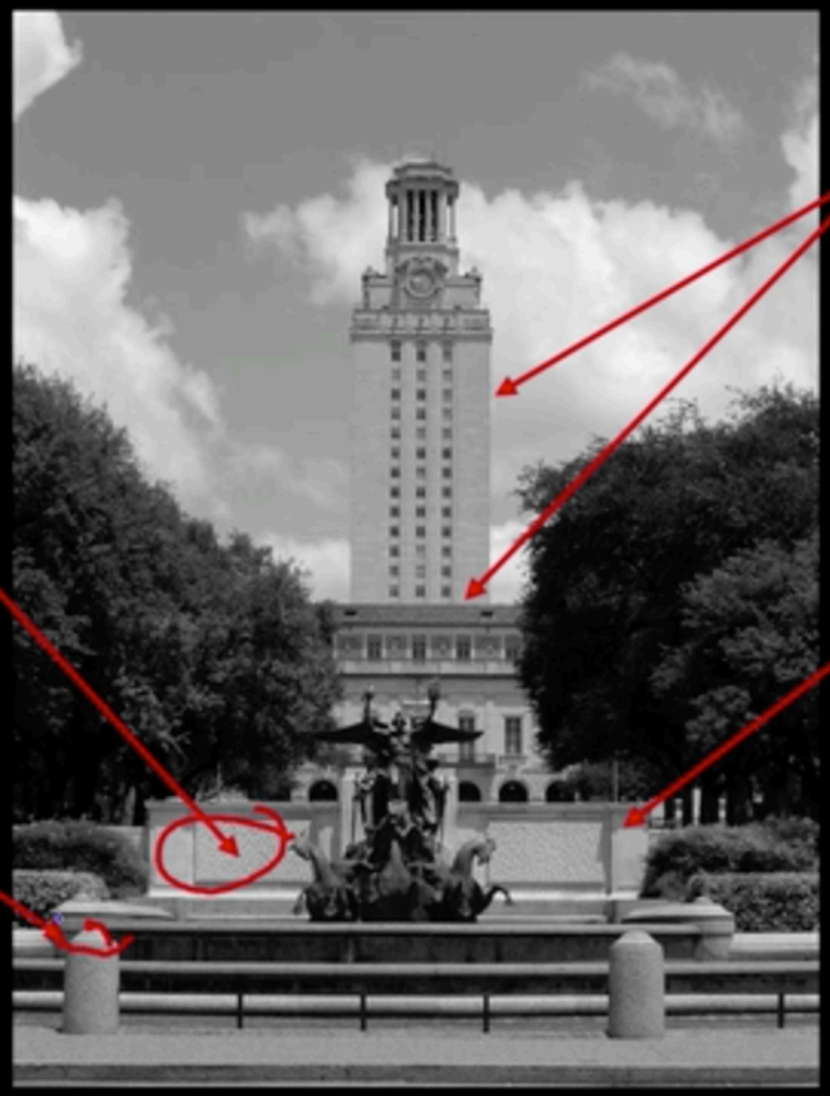
depth discontinuity

surface color discontinuity

illumination discontinuity

In a real image

In a real image



Reflectance change: appearance information, texture

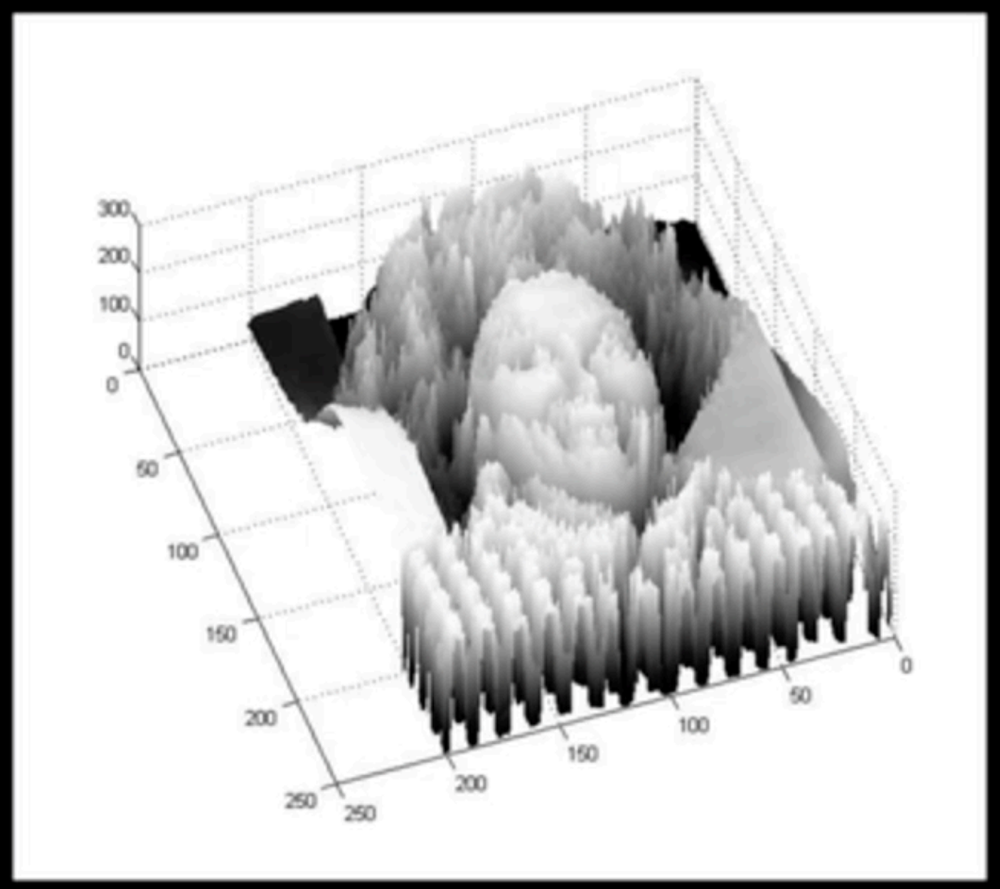
Depth discontinuity: object boundary

Discontinuous change in surface orientation

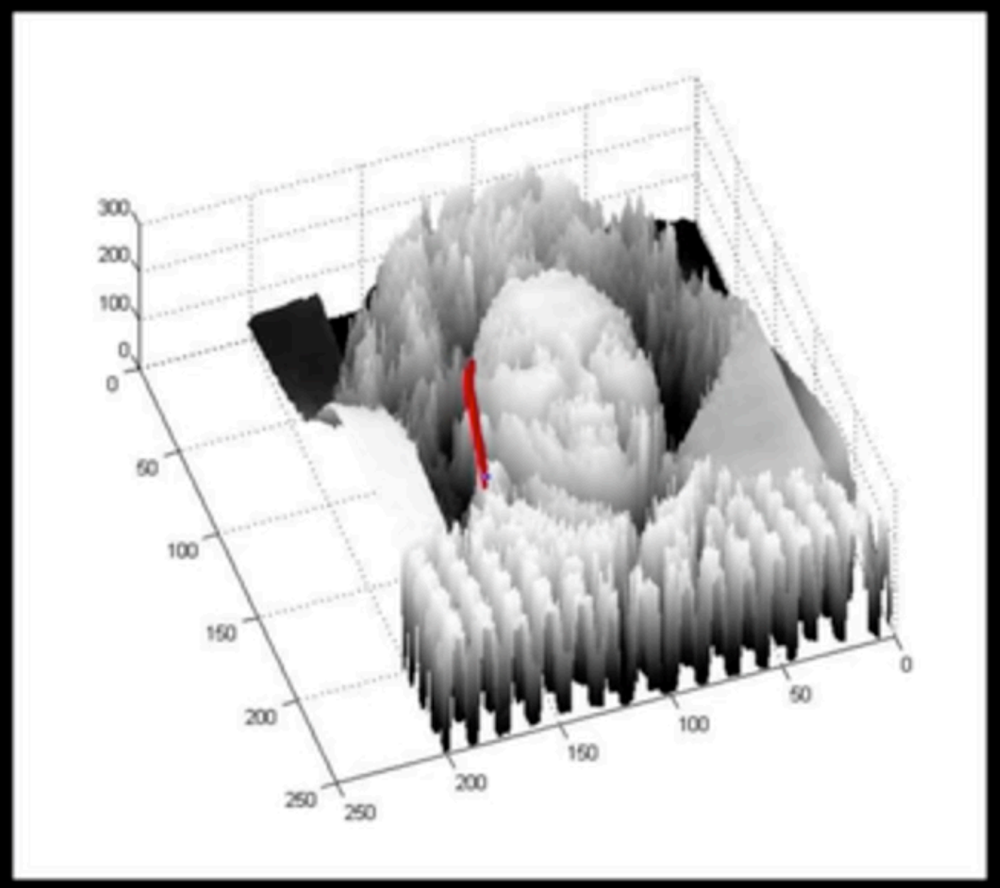
Cast shadows

Edge detection

Recall images as functions…



Recall images as functions…



Edges look like steep cliffs

Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

- neighborhood size
- how to detect change

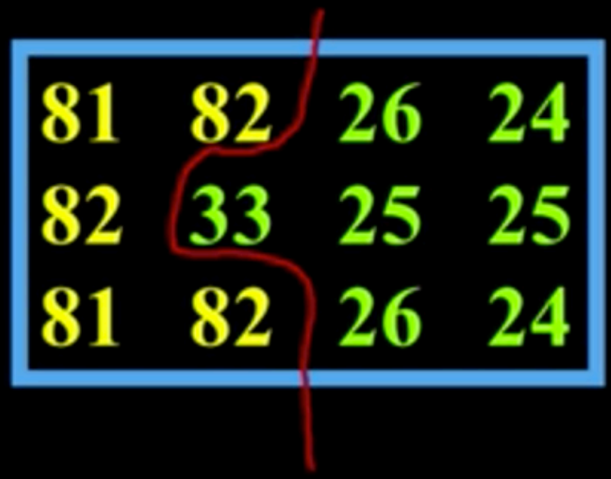
81	82	26	24
82	33	25	25
81	82	26	24

Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

- neighborhood size
- how to detect change



Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

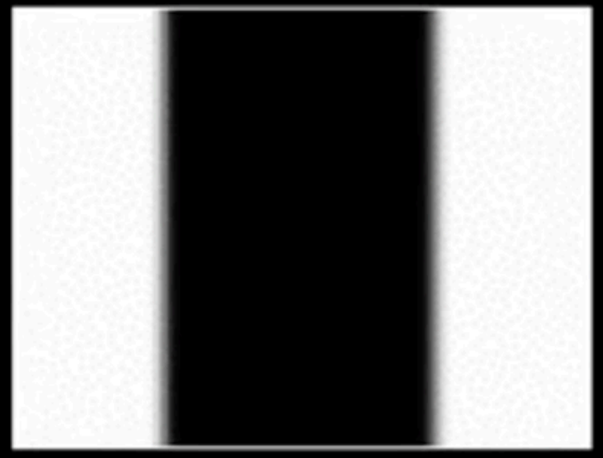
- neighborhood size
- how to detect change

81	82	26	24
82	33	25	25
81	82	26	24

Derivatives and edges

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

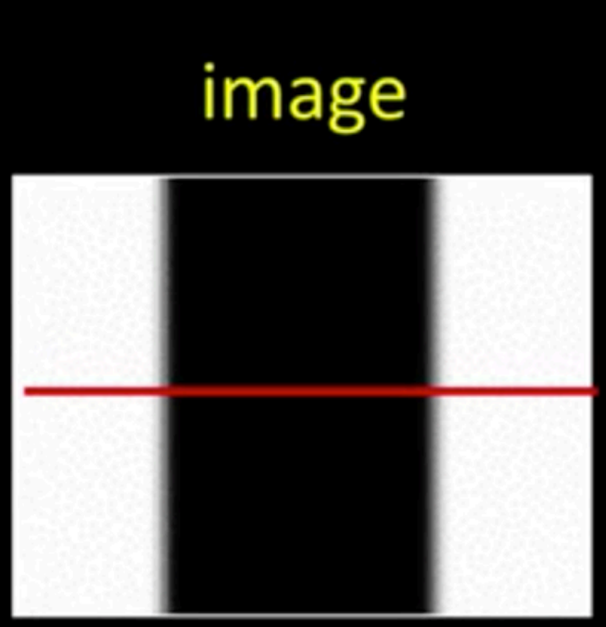
image

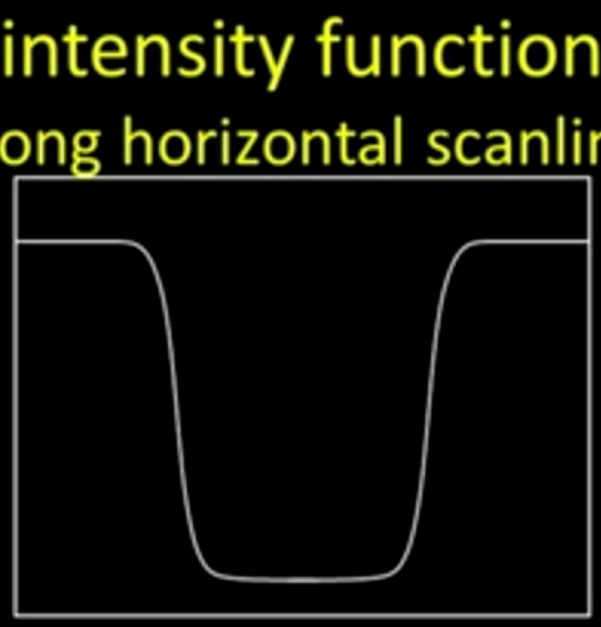


Source: S. Lazebnik

Derivatives and edges

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

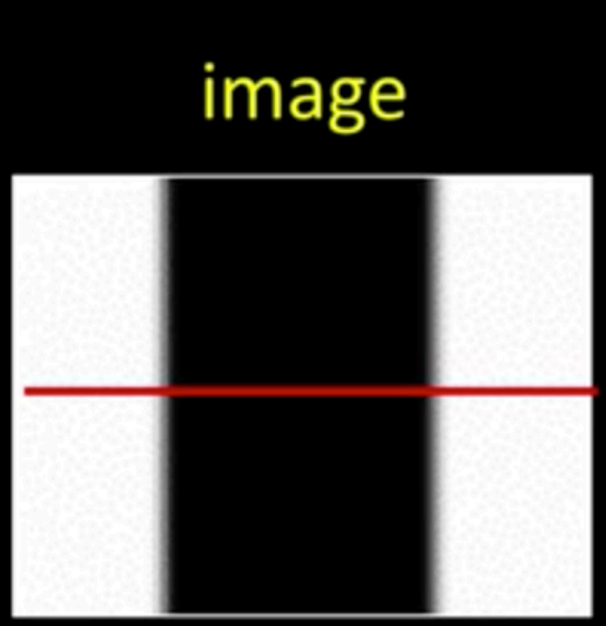




Source: S. Lazebnik

Derivatives and edges

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



Source: S. Lazebnik

Differential Operators

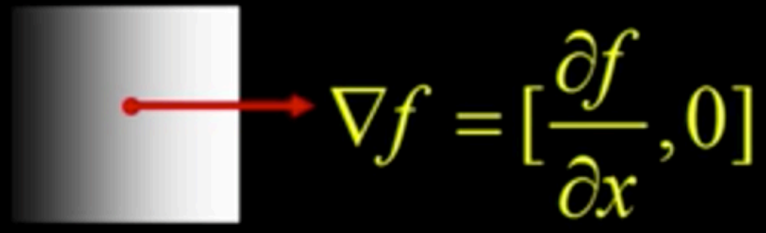
- Differential operators –when applied to the image returns some derivatives.
- Model these “operators” as masks/kernels that compute the image gradient function.
- Threshold the this gradient function to select the edge pixels.

Image gradient

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

Image gradient

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



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

Image gradient

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

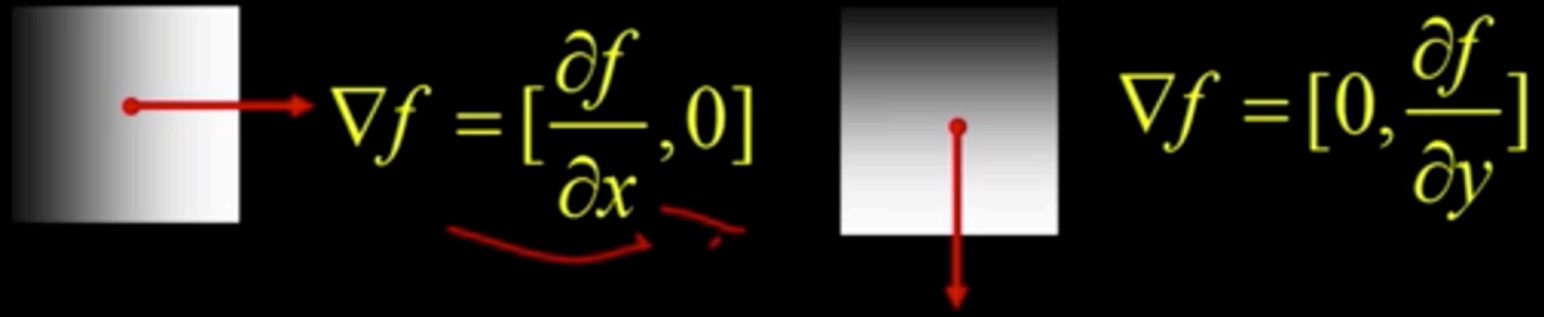
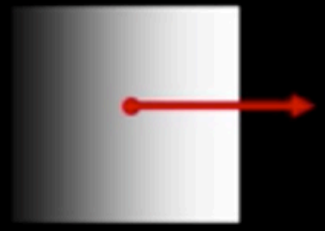
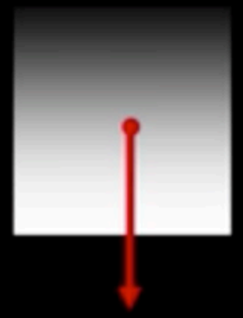


Image gradient

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



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

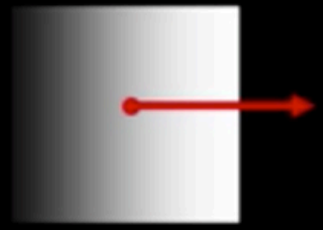


$$\nabla f = [0, \frac{\partial f}{\partial y}]$$

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

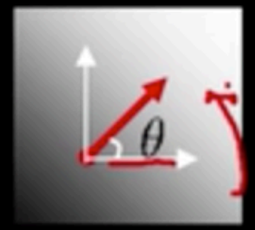
Image gradient

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



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

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



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

The gradient points in the direction of most rapid increase in intensity

Image gradient

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

The gradient direction is given by: $$\theta = \tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$$

The *amount of change* is given by the gradient magnitude: $$\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

Q&A 01

Quiz

What does it mean when the magnitude of the image gradient is zero?

a) The image is constant over the entire neighborhood.

b) The underlying function $f(x,y)$ is at a maximum.

c) The underlying function $f(x,y)$ is at a minimum.

d) Either (a), (b), or (c).

Quiz

What does it mean when the magnitude of the image gradient is zero?

a) The image is constant over the entire neighborhood.

b) The underlying function $f(x,y)$ is at a maximum.

c) The underlying function $f(x,y)$ is at a minimum.

d) Either (a), (b), or (c).

Discrete gradient

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

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

Discrete gradient

For discrete data, we can approximate using finite differences:

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

Discrete gradient

For discrete data, we can approximate using finite differences:

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

$$\approx f(x+1,y)-f(x,y)$$

"right derivative" But is it???

Discrete gradient

For discrete data, we can approximate using finite differences:

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

$$\approx f(x+1,y)-f(x,y)$$

"right derivative" But is it???

-1	1

-f(x,y) + f(x+1,y)

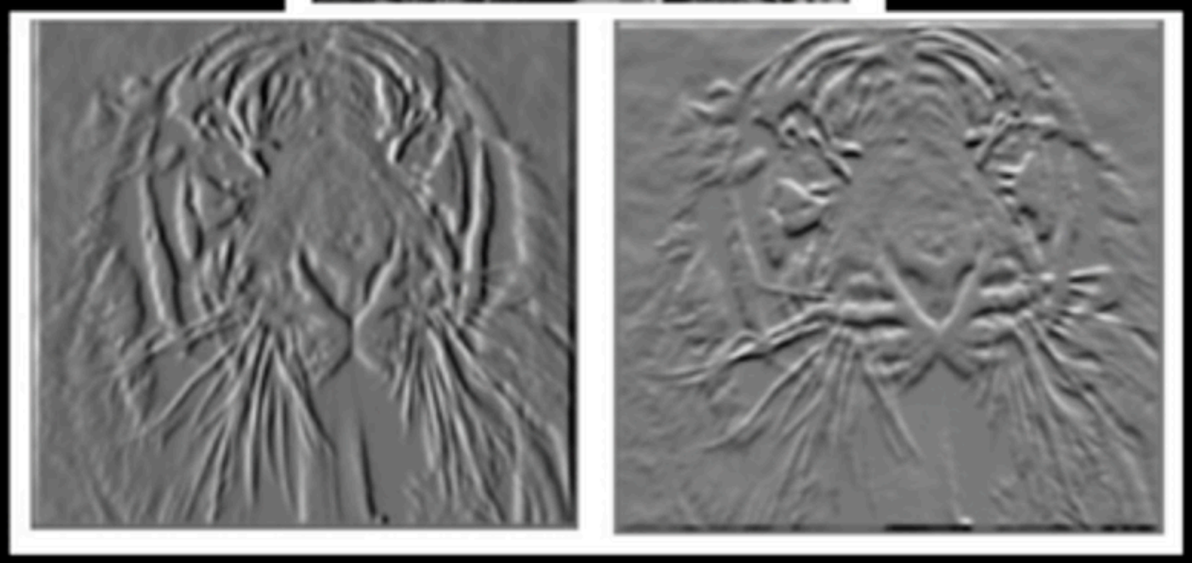
Finite differences – x or y?

Source: D. Forsyth

Partial derivatives of an image

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$



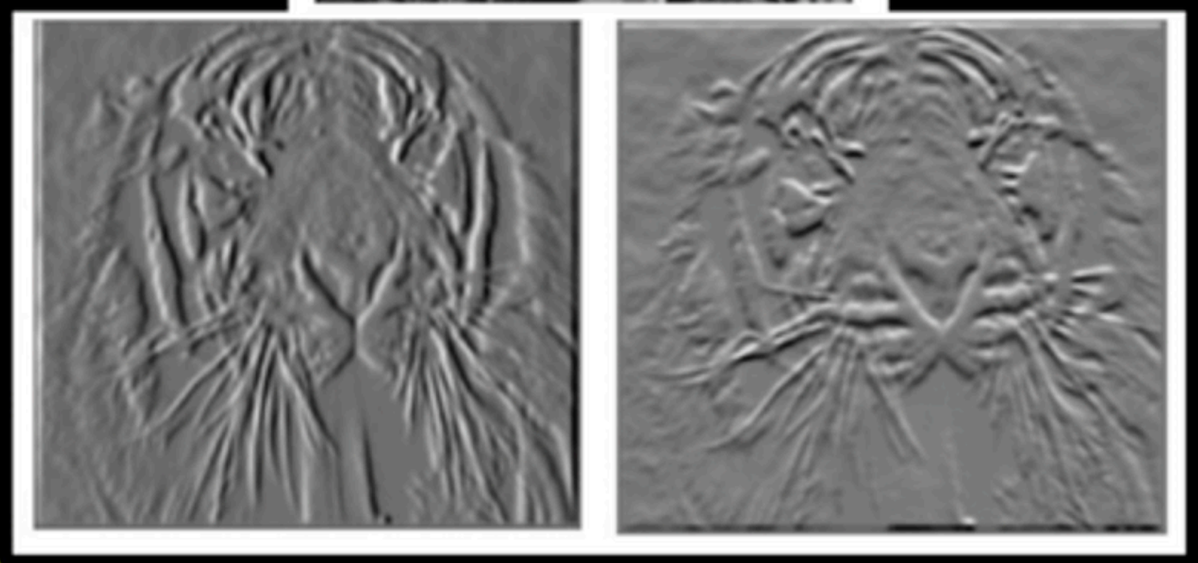
$$\frac{\partial f(x,y)}{\partial y}$$

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

-1	1

(correlation filters)



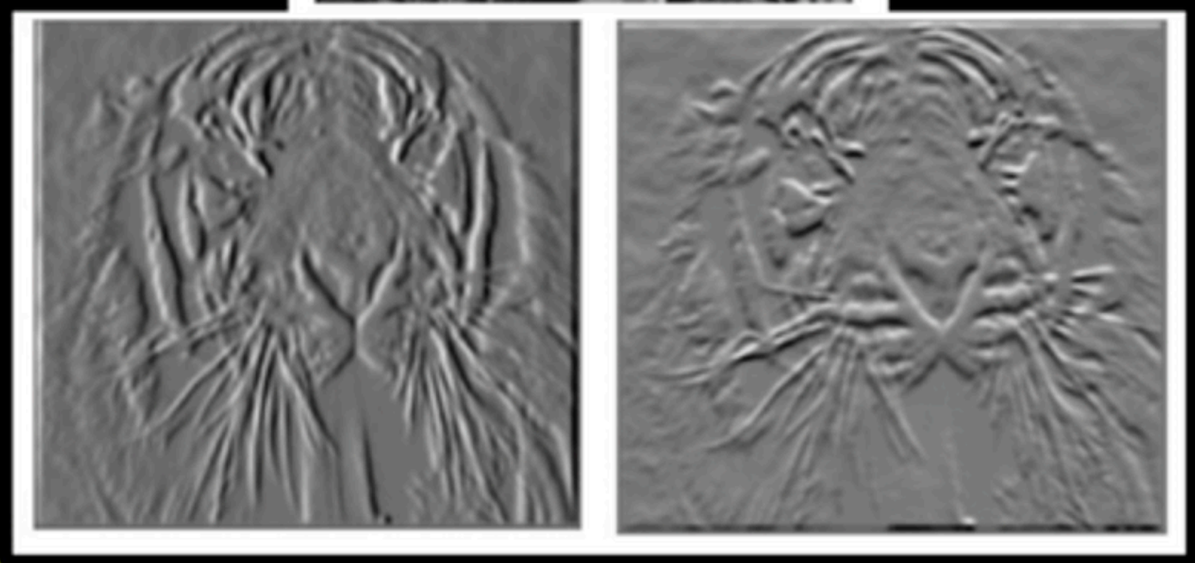
$$\frac{\partial f(x,y)}{\partial y}$$

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

-1	1

(correlation filters)



$$\frac{\partial f(x,y)}{\partial y}$$

-1
1

?

or

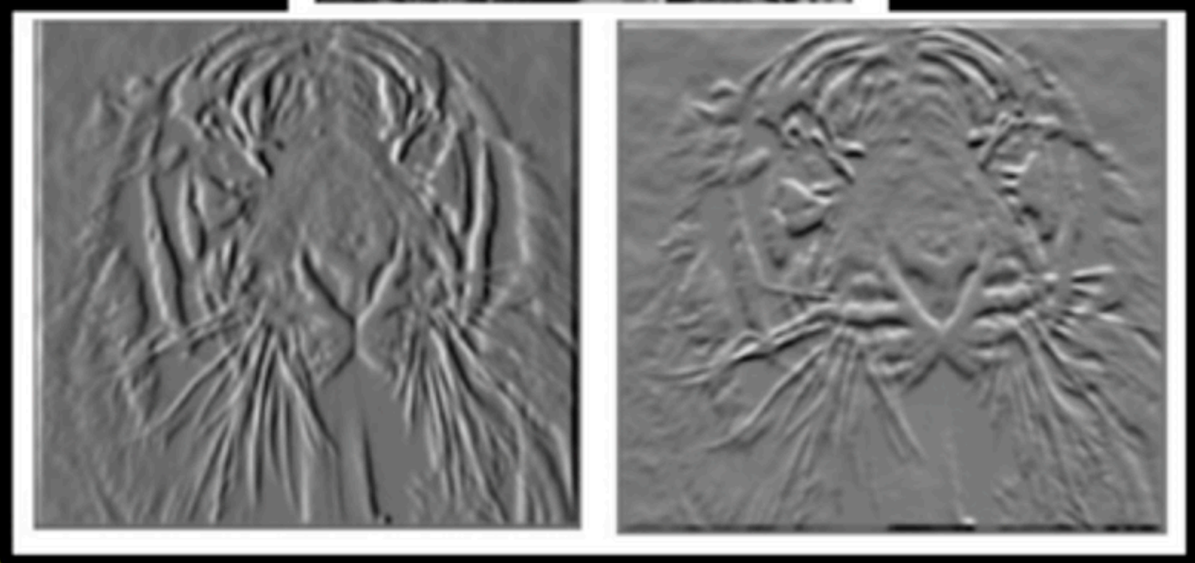
1
-1

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

-1	1

(correlation filters)



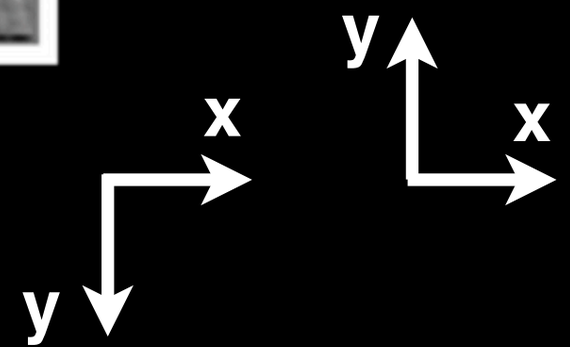
$$\frac{\partial f(x,y)}{\partial y}$$

-1
1

?

or

1
-1



The discrete gradient

- We want an “operator” (mask/kernel) that we can apply to the image that implements:

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

The discrete gradient

0	0
-1	+1
0	0

H

The discrete gradient

0	0
-1	+1
0	0

H

Not symmetric around image point; which is "middle" pixel?

The discrete gradient

0	0
-1	+1
0	0

H

Not symmetric around image point; which is "middle" pixel?

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

H

The discrete gradient

0	0
-1	+1
0	0

H

Not symmetric around image point; which is "middle" pixel?

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

H

Average of "left" and "right" derivative . See?

The discrete gradient

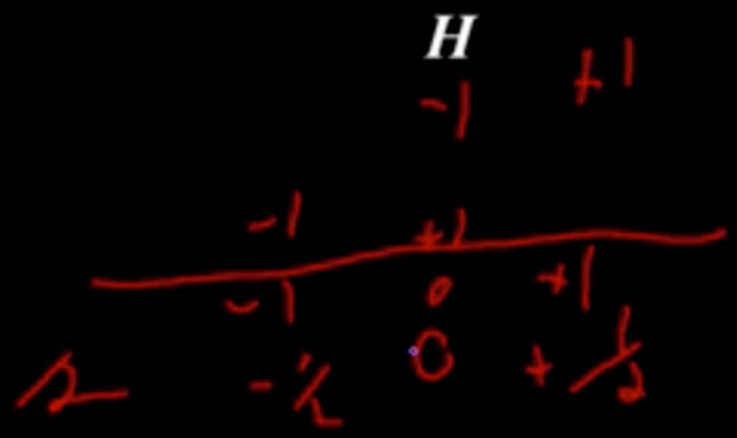
0	0
-1	+1
0	0
0	0

H

Not symmetric around image point; which is "middle" pixel?

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

H



Average of "left" and "right" derivative . See?

Some Well-Known Gradients Masks

- Sobel:

Sx

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

Sy

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

- Prewitt:

-1	0	1
-1	0	1
-1	0	1

1	1	1
0	0	0
-1	-1	-1

- Roberts:

0	1
-1	0

1	0
0	-1

Q&A 02

Quiz

It is better to compute gradients using:

a) ***Convolution*** since that's the right way to model filtering so you don't get flipped results.

b) ***Correlation*** because it's easier to know which way the derivatives are being computed.

c) Doesn't matter.

d) Neither since I can just write a for-loop to compute the derivatives.

Quiz

It is better to compute gradients using:

a) ***Convolution*** since that's the right way to model filtering so you don't get flipped results.

b) ***Correlation*** because it's easier to know which way the derivatives are being computed.

c) Doesn't matter.

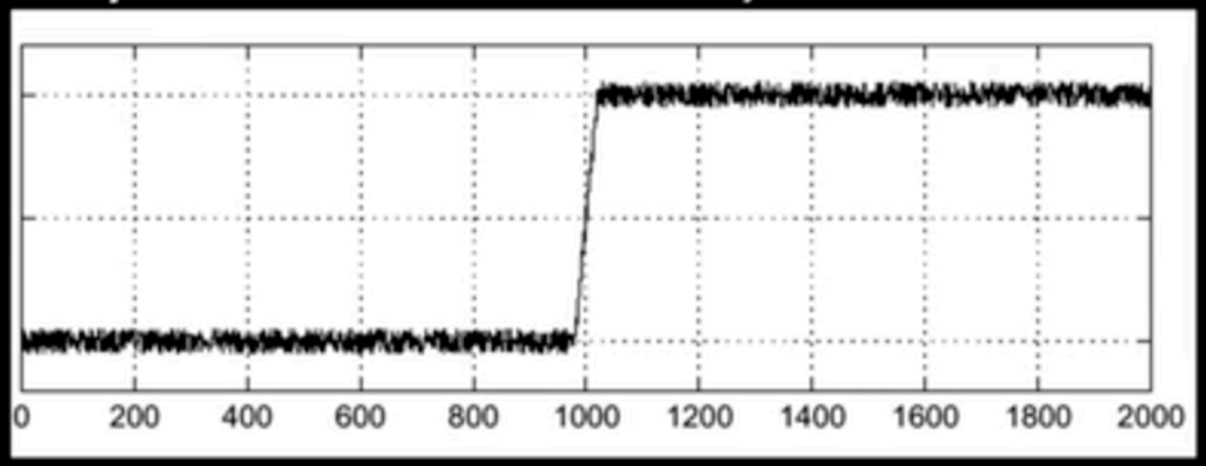
d) Neither since I can just write a for-loop to compute the derivatives.

But in the real world…

Consider a single row or column of the image

(plotting intensity as a function of *x)*

$f(x)$

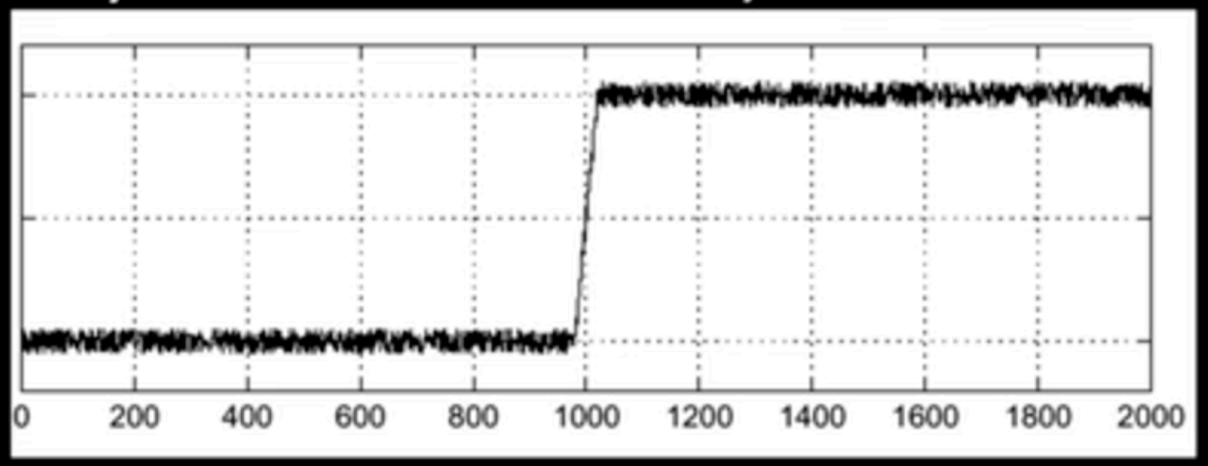


But in the real world…

Consider a single row or column of the image

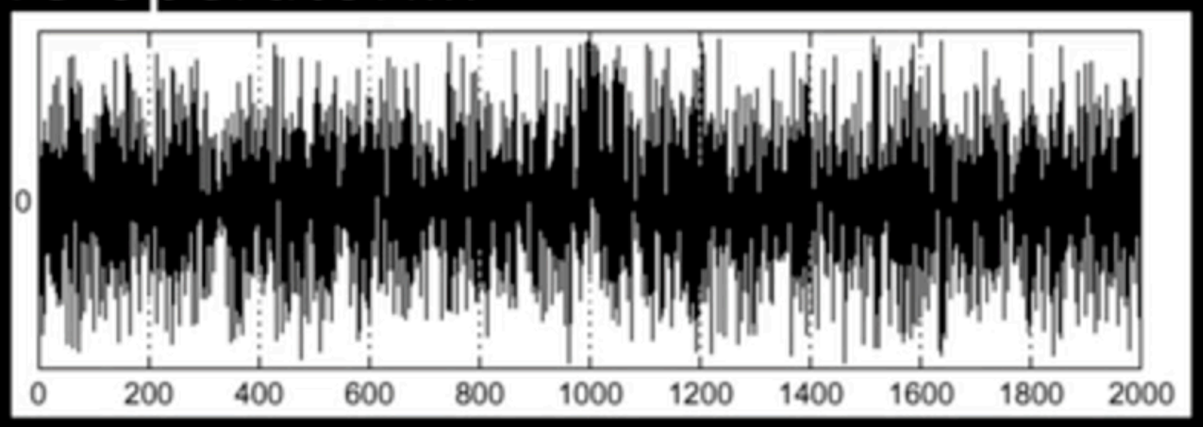
(plotting intensity as a function of *x)*

$f(x)$



Apply derivative operator….

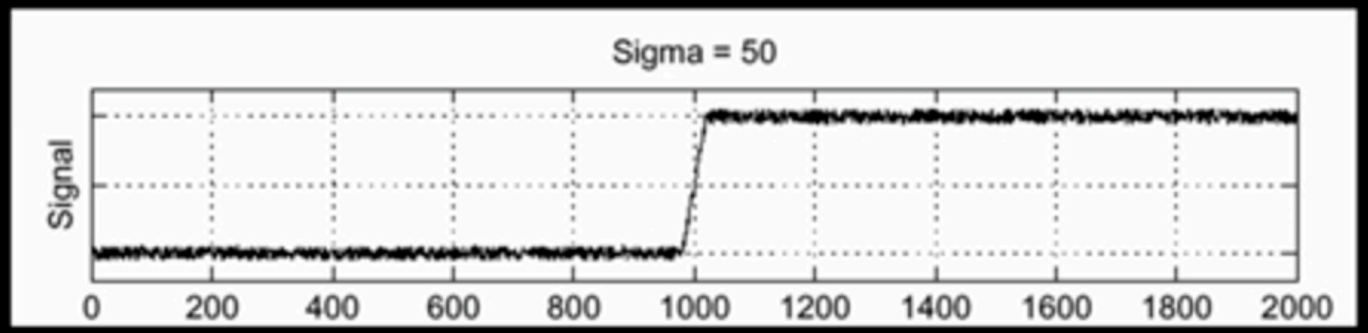
$$\frac{d}{dx} f(x)$$



Uh, where's the edge?

Solution: smooth first

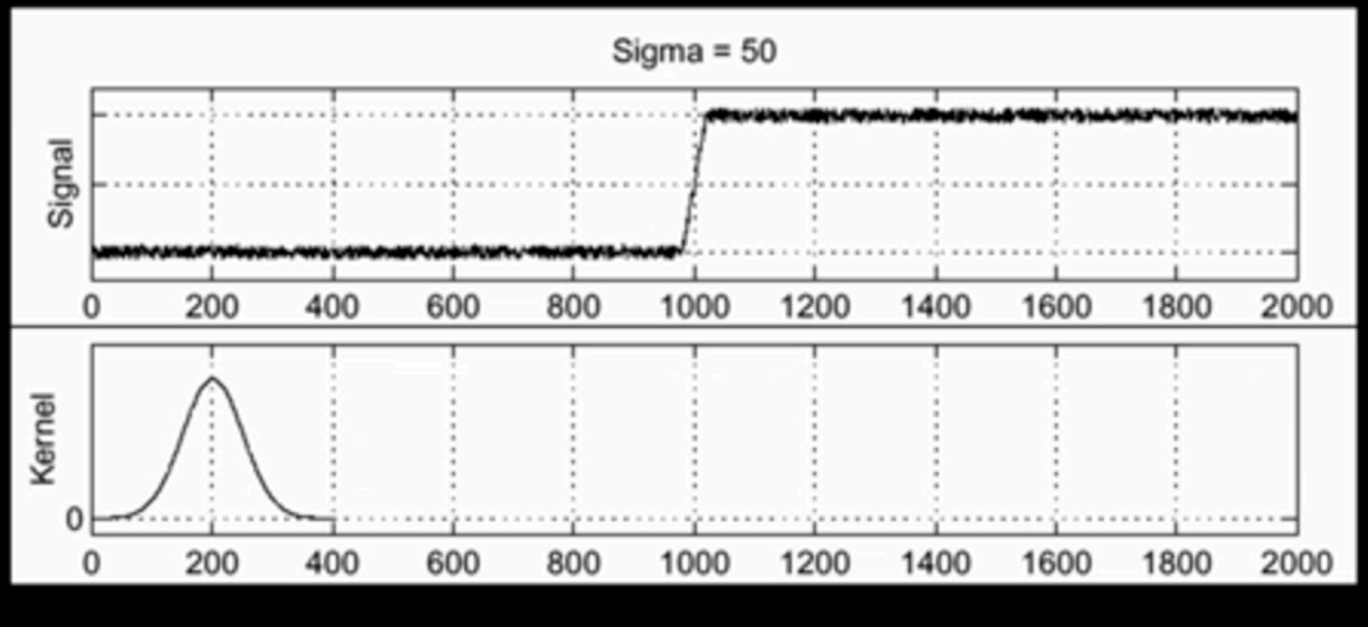
f



Solution: smooth first

f

h

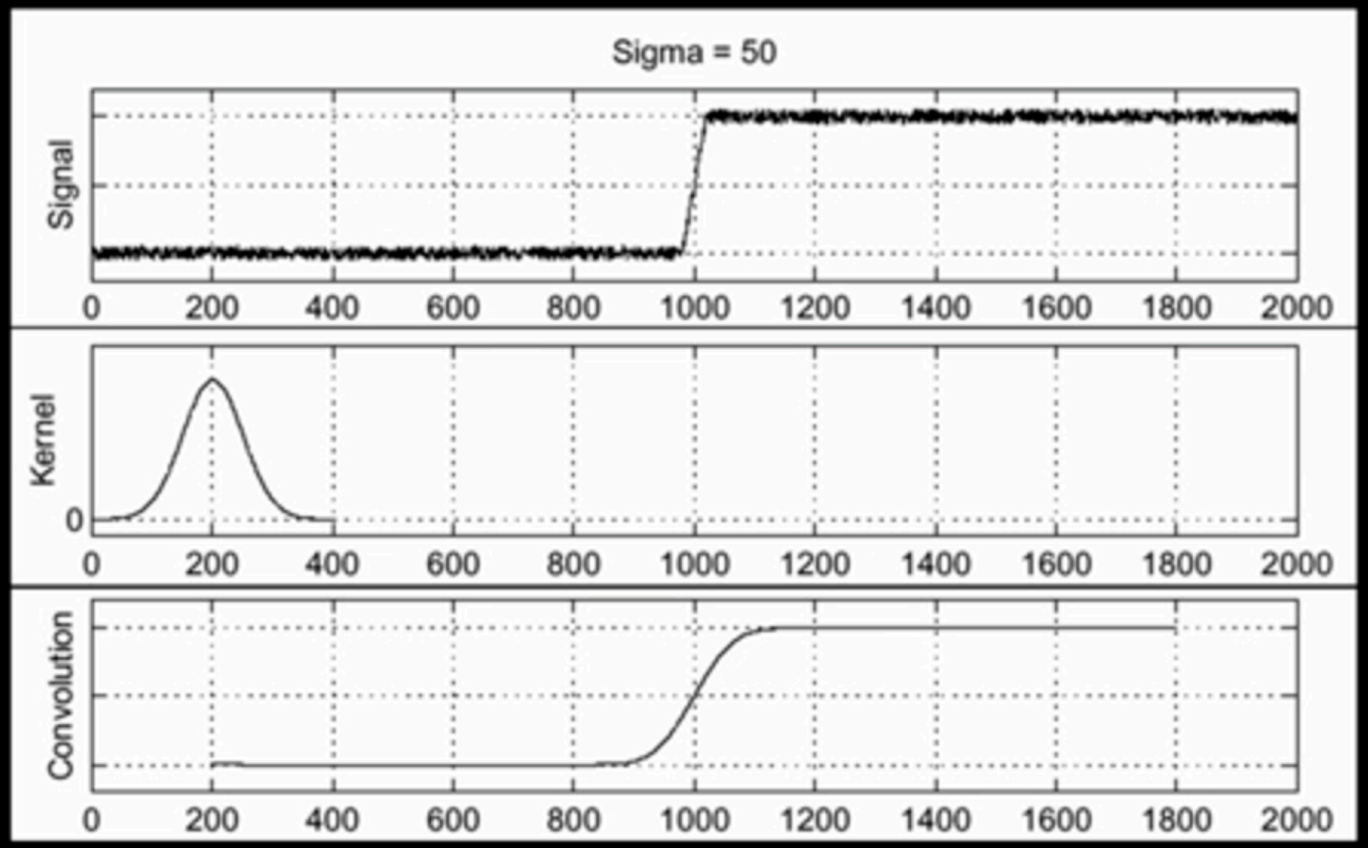


Solution: smooth first

f

h

$h * f$



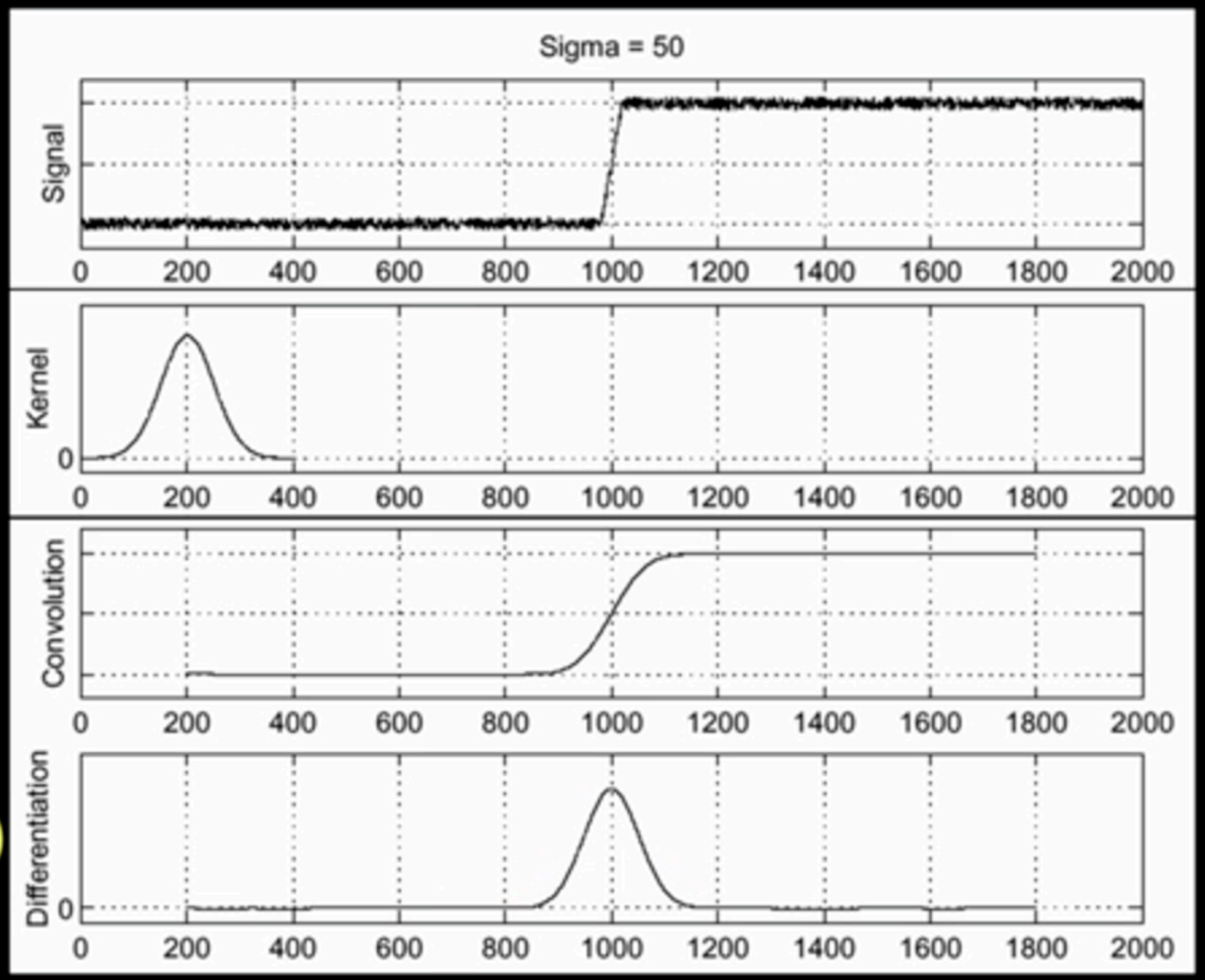
Solution: smooth first

f

h

$h * f$

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



Where is the edge?

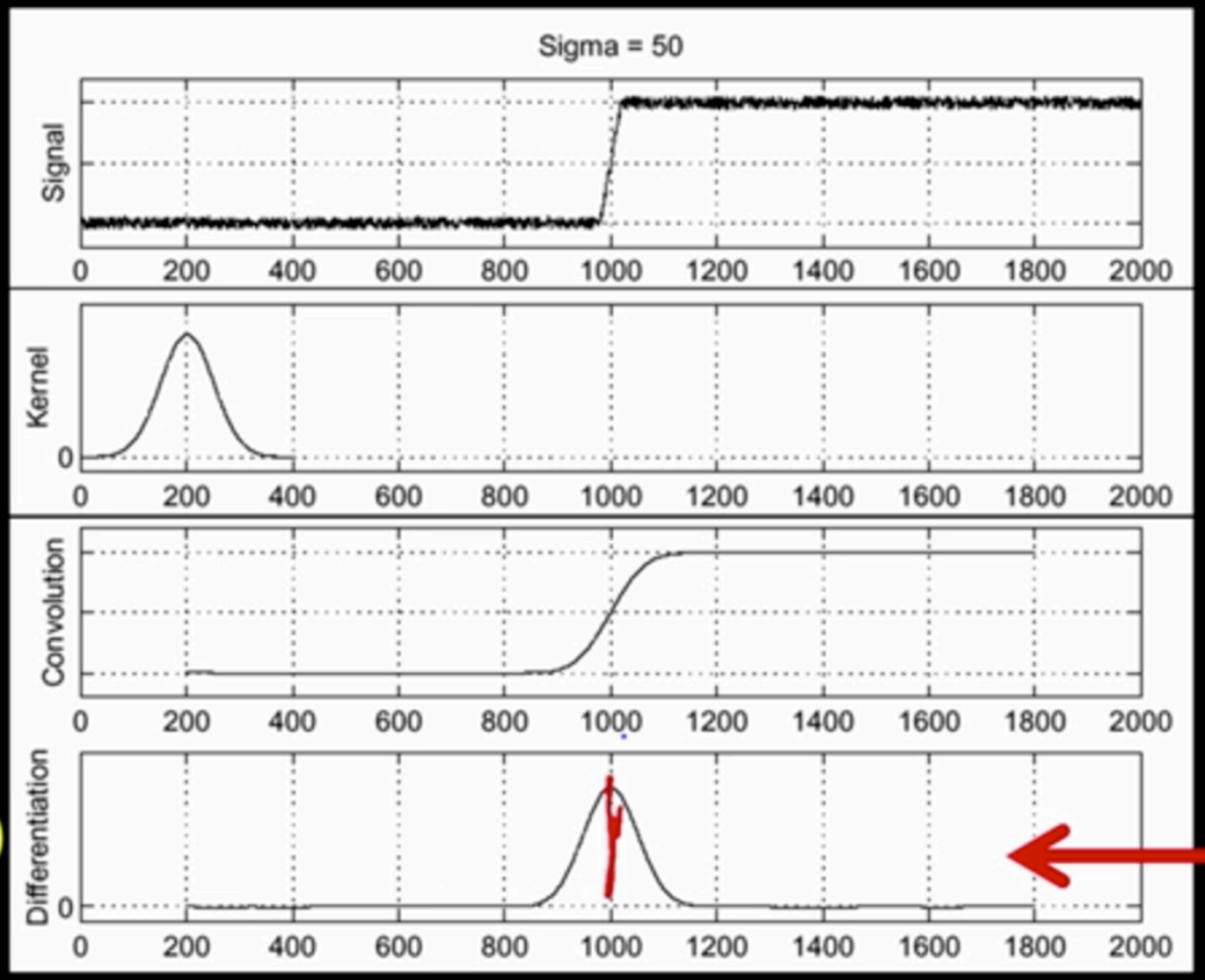
Solution: smooth first

f

h

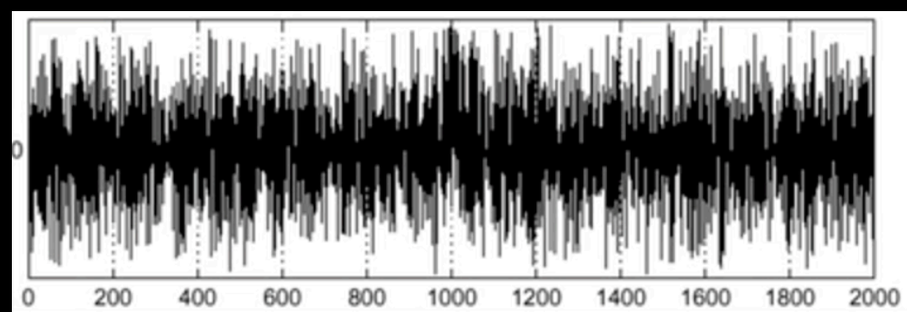
$h * f$

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



Where is the edge?

Look for peaks



Derivative theorem of convolution

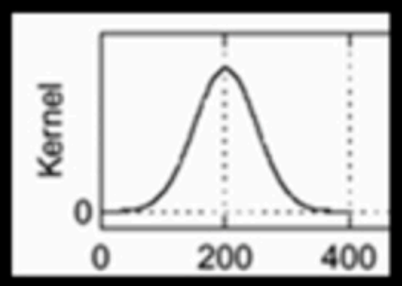
This saves us one operation: $$\frac{\partial}{\partial x}(h * f) = (\frac{\partial}{\partial x} h) * f$$

Derivative theorem of convolution

This saves us one operation: $\frac{\partial}{\partial x}(h * f) = (\frac{\partial}{\partial x} h) * f$

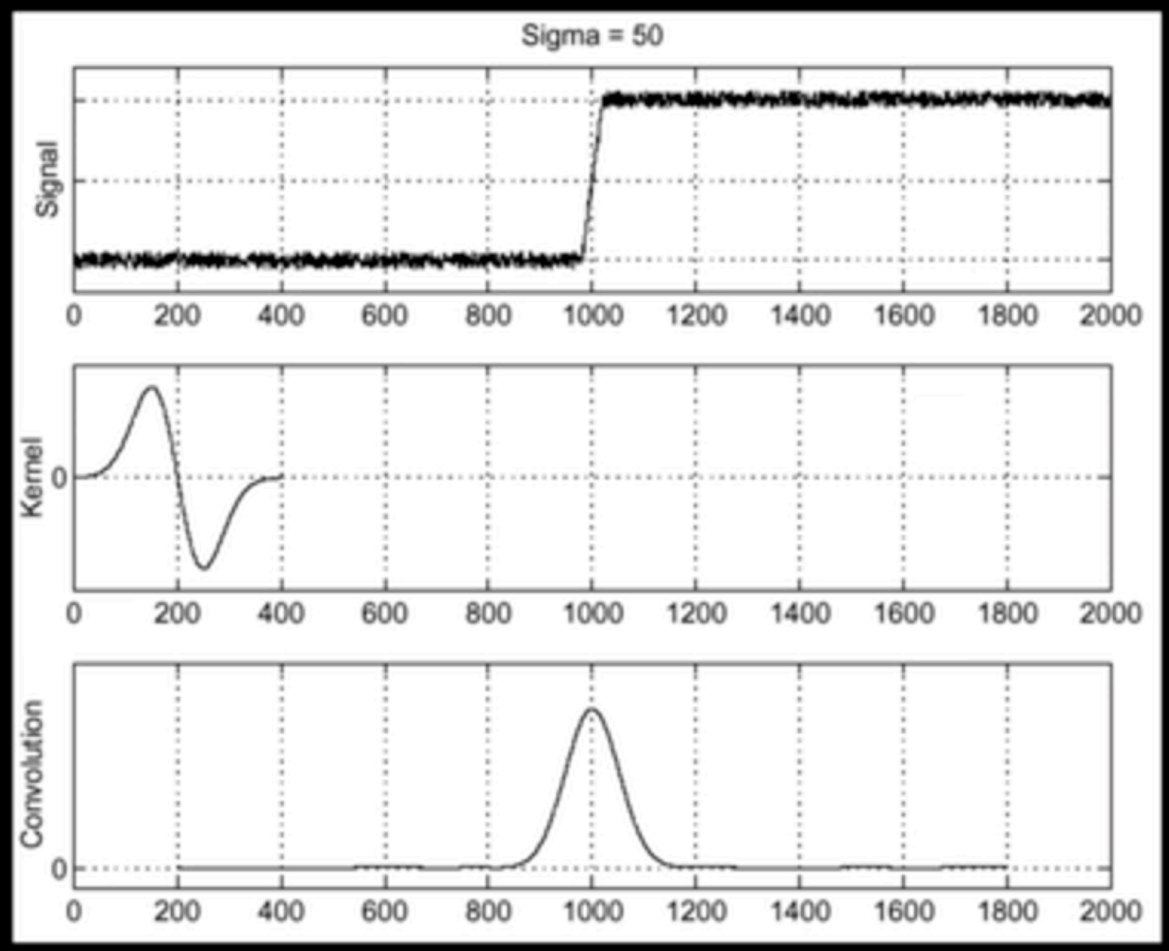
f

h



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

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

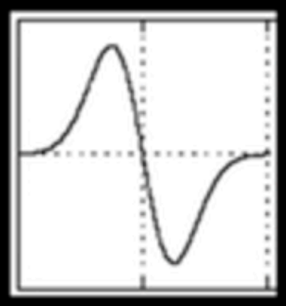


2nd derivative of Gaussian

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

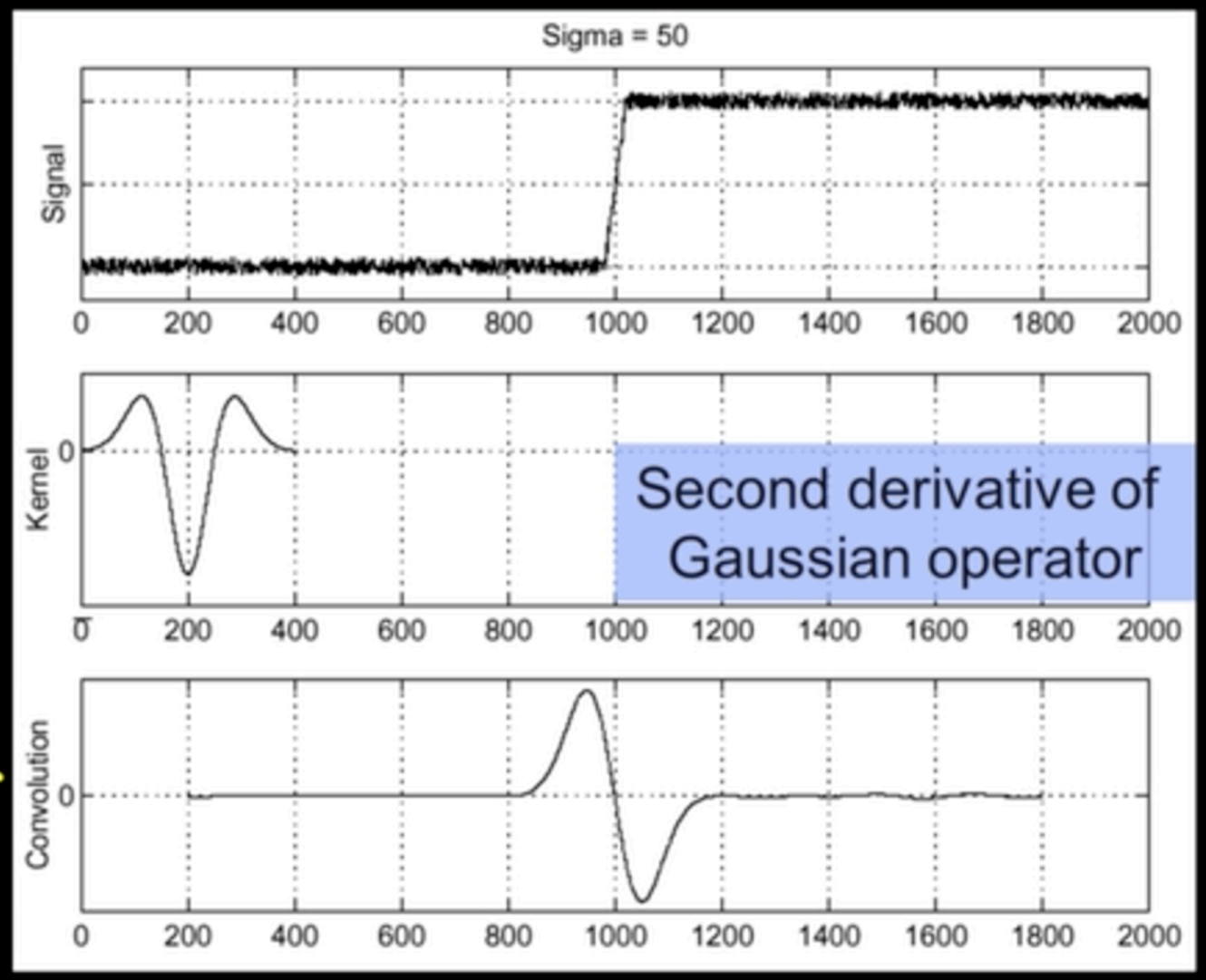
f

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



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

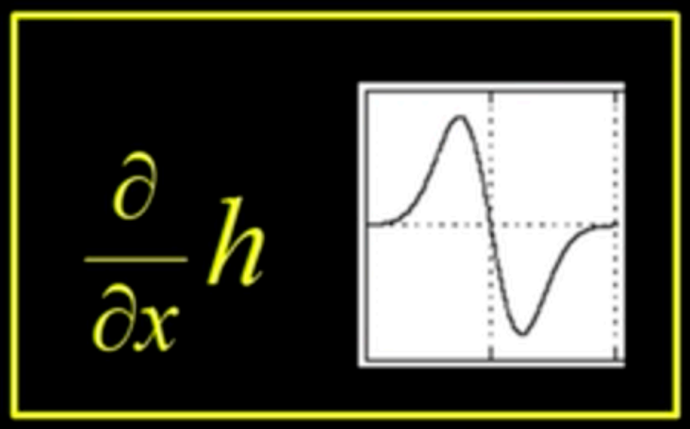
$(\frac{\partial^2}{\partial x^2} h) * f$



2nd derivative of Gaussian

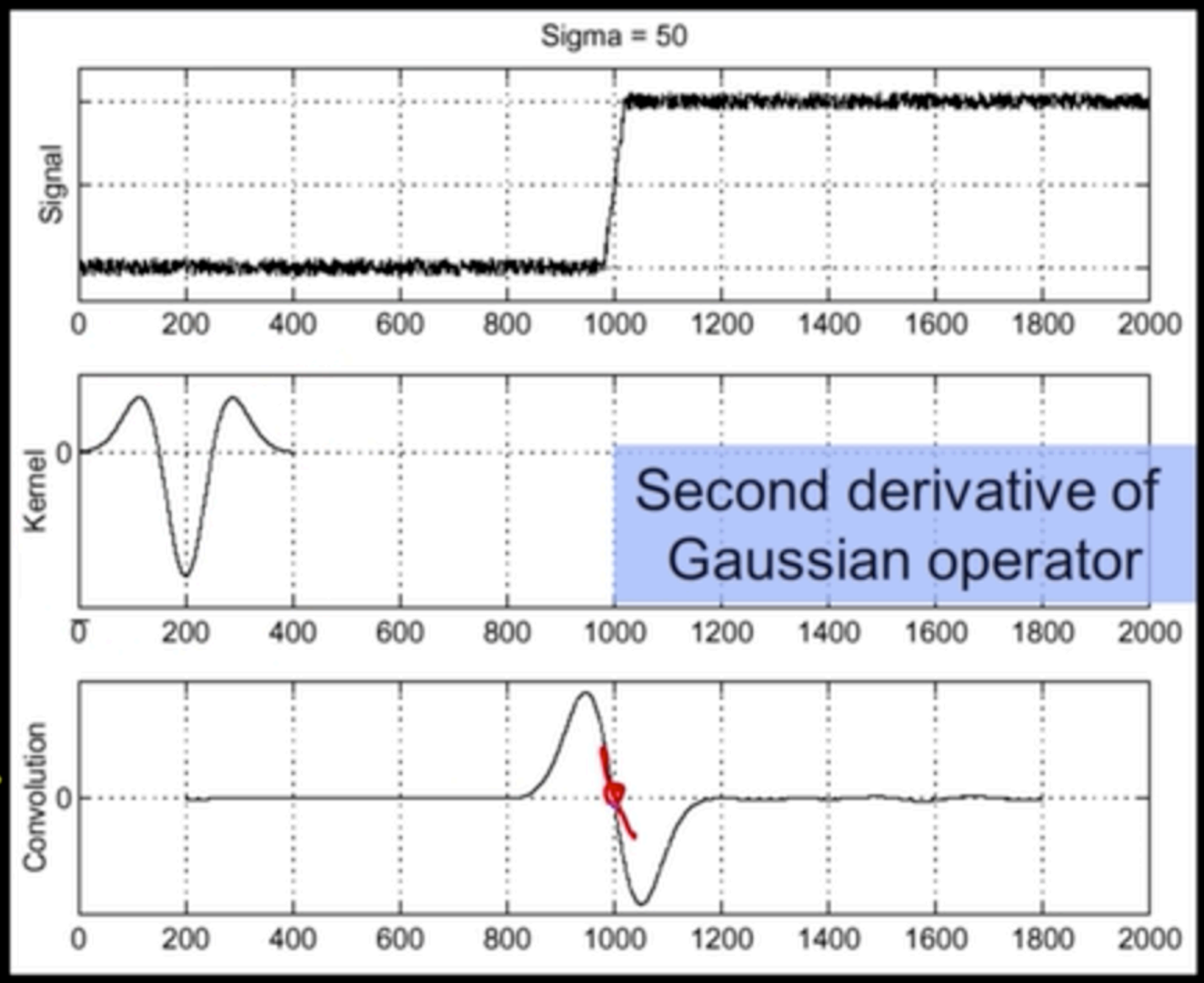
Consider $\frac{\partial^2}{\partial x^2}(h * f)$

f



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

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



Q&A 02

Quiz

Which linearity property did we take advantage of to first take the derivative of the kernel and then apply that?

a) associative
b) commutative
c) differentiation
d) (a) and (c)

Quiz

Which linearity property did we take advantage of to first take the derivative of the kernel and then apply that?

a) associative
b) commutative
c) differentiation
d) (a) and (c)

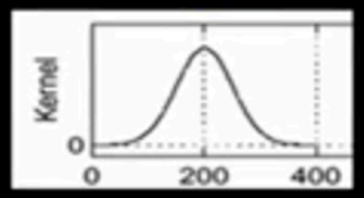
Edge detection - 2D operators

Derivative theorem of convolution - 1D

- This saves us one operation:

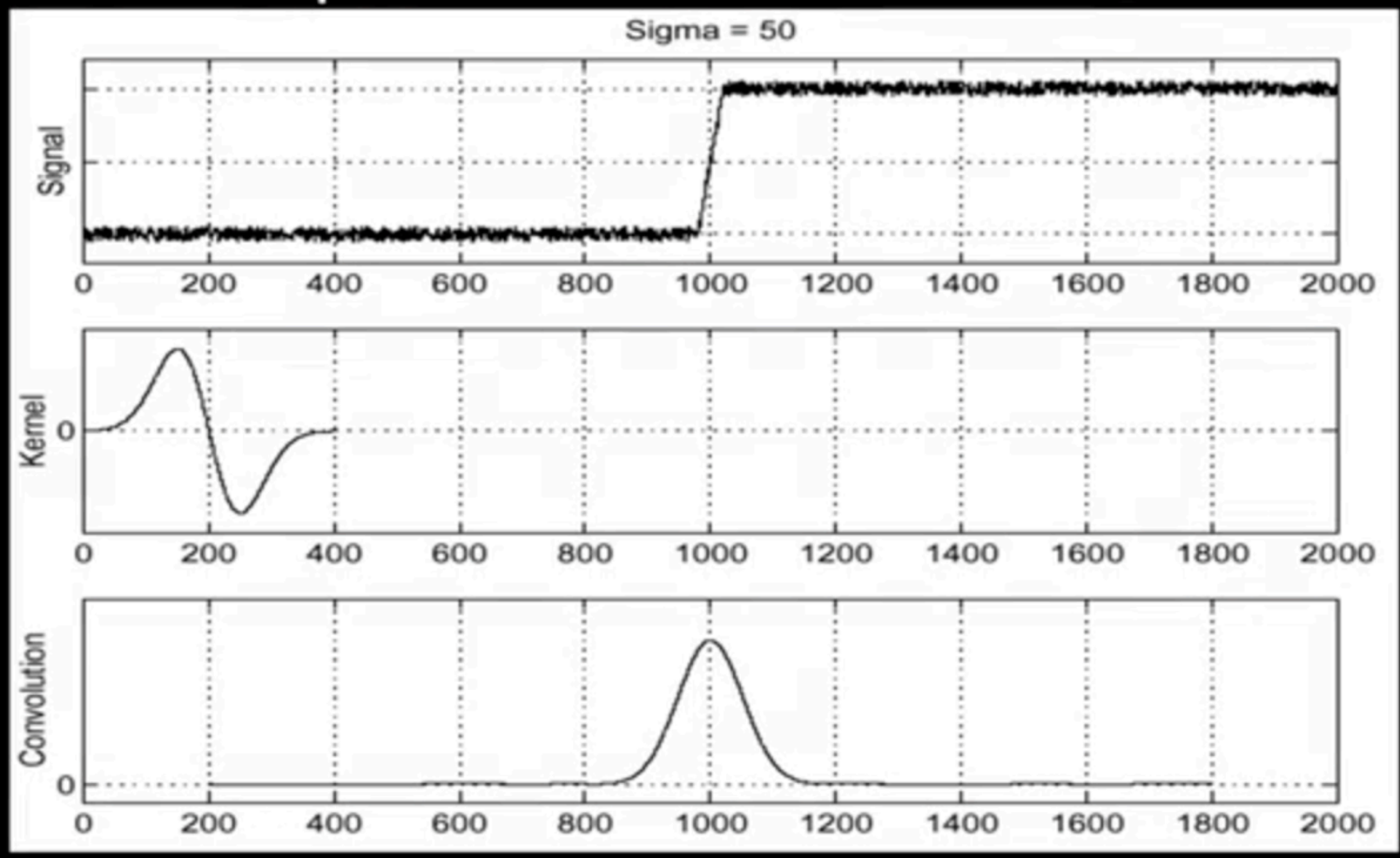
f

h



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

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

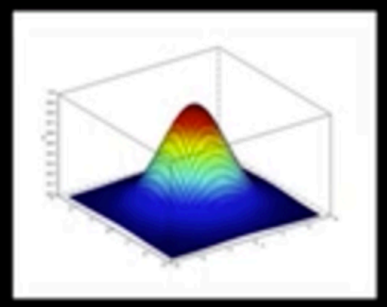


Derivative of Gaussian filter – 2D

$$(I \otimes g) \otimes h_x = I \otimes (g \otimes h_x)$$

Derivative of Gaussian filter – 2D

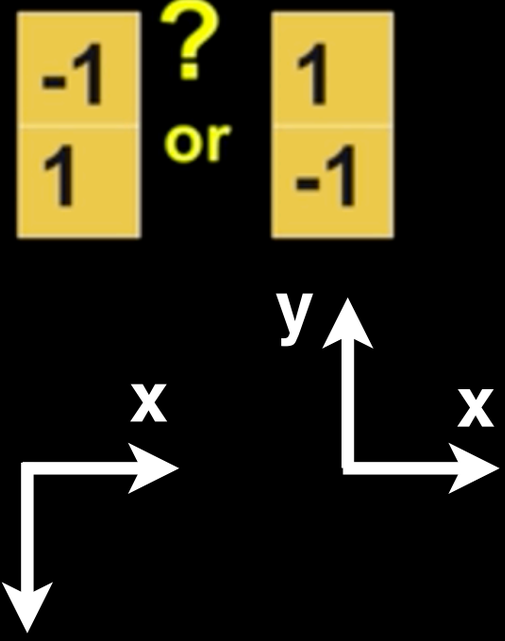
$$(I \otimes g) \otimes h_x = I \otimes (g \otimes h_x)$$

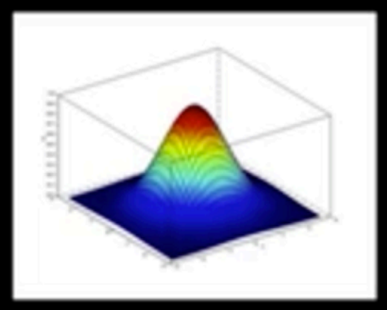


$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \end{bmatrix} =$$

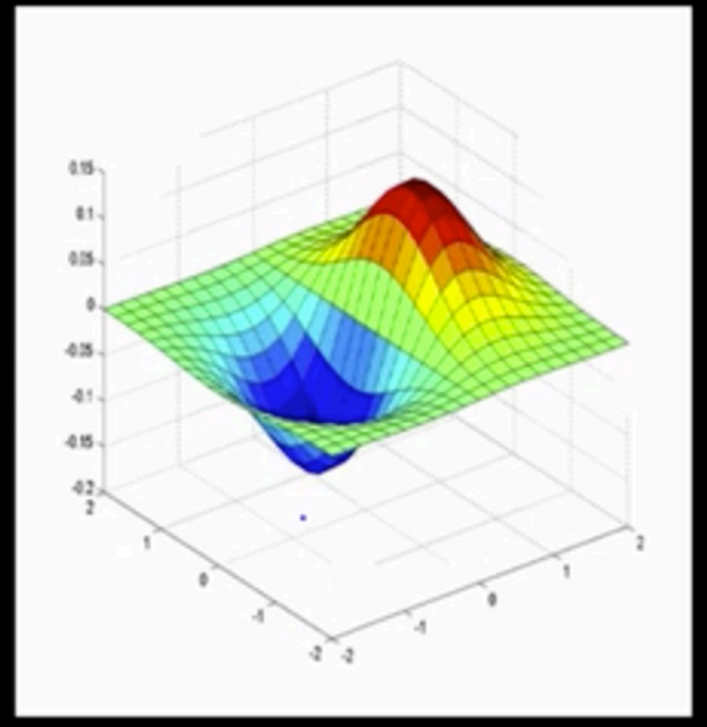
Derivative of Gaussian filter –

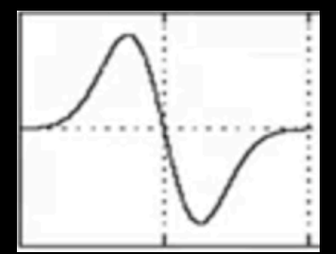
$$(I \otimes g) \otimes h = I \otimes (g \otimes h)$$



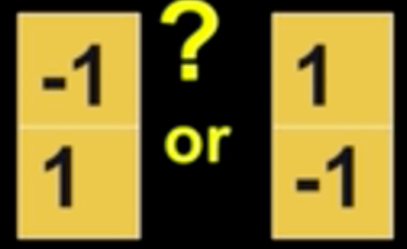


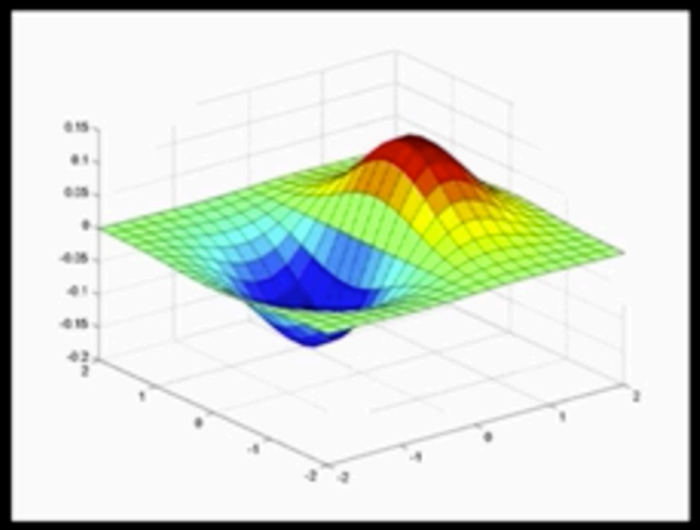
$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \end{bmatrix} =$$



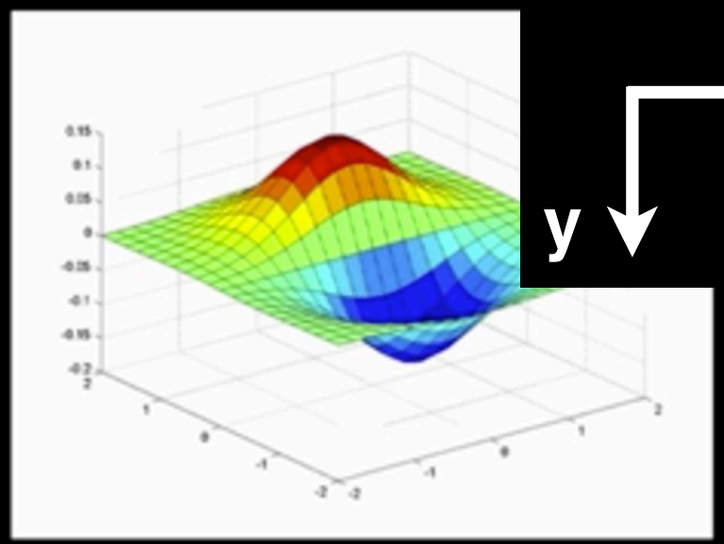


Derivative of Gaussian filter

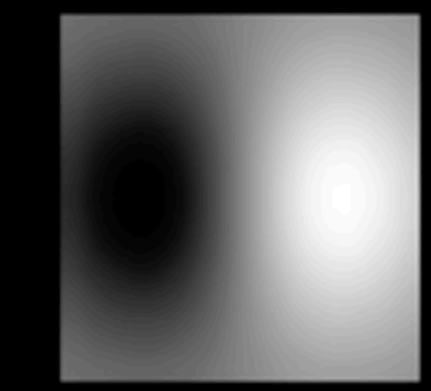


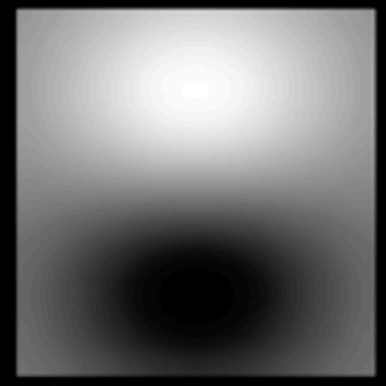


x-direction



y-direction

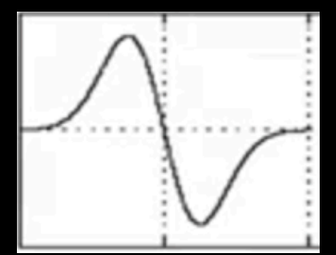




Correlation or convolution?

And for y *it's always a problem!*

Source: S. Lazebnik



Q&A 03

Quiz

Why is it preferable to apply *h* to the smoothing function *g* and apply the result to the Image.

a) It's not – they are mathematically equivalent.

b) Since *h* is typically smaller we take fewer derivatives so it's faster.

c) The smoothed derivative operator is computed once and you have it to use repeatedly.

d) B & C

Quiz

Why is it preferable to apply *h* to the smoothing function *g* and apply the result to the Image.

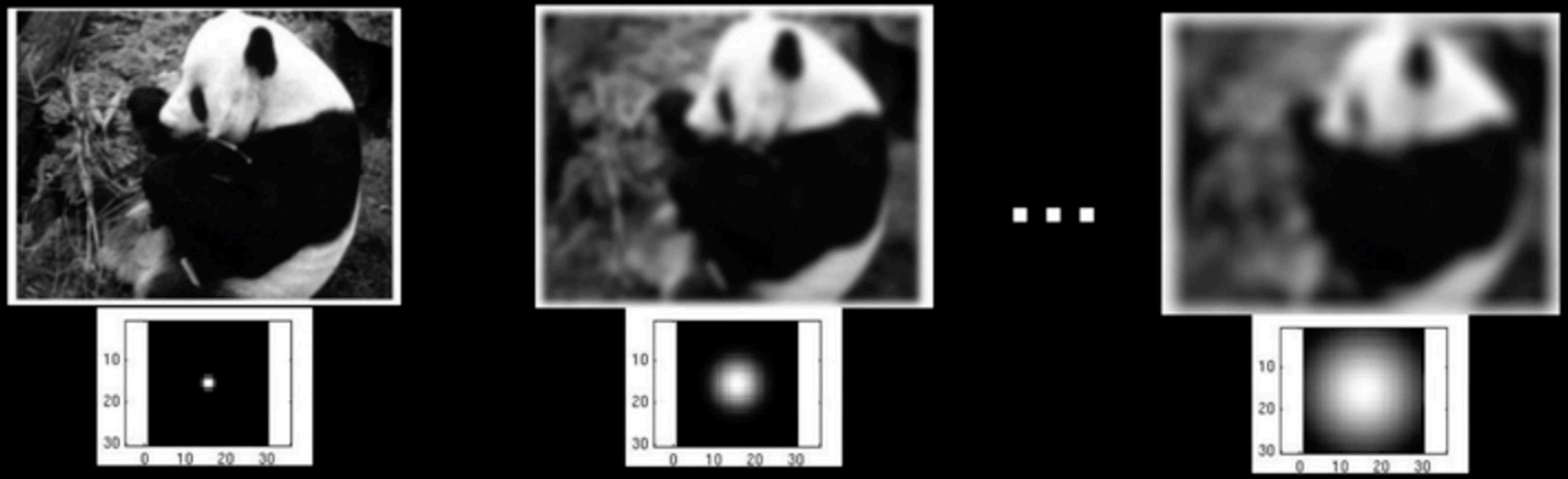
a) It's not – they are mathematically equivalent.

b) Since *h* is typically smaller we take fewer derivatives so it's faster.

c) The smoothed derivative operator is computed once and you have it to use repeatedly.

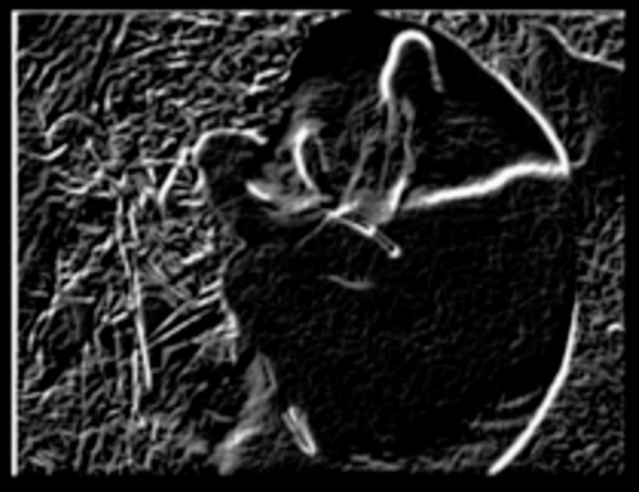
d) B & C

Smoothing with a Gaussian




```
for sigma=1:3:10
 h = fspecial('gaussian', fsize, sigma);
 out = imfilter(im, h);
 imshow(out);
 pause;
end
```

Effect of σ on derivatives



σ = 1 pixel

σ = 3 pixels

Smaller values: finer features detected

Larger values: larger scale edges detected

Gradients -> edges

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.

Gradients -> edges

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.

2. Threshold to find regions of "significant" gradient.

Gradients -> edges

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.

2. Threshold to find regions of "significant" gradient.

3. "Thin" to get localized edge pixels

Gradients -> edges

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.
2. Threshold to find regions of "significant" gradient.
3. "Thin" to get localized edge pixels
4. And link or connect edge pixels.

NB: Will be treated more extensively in the segmentation part of this course, this is just a heads up.

Canny edge operator

1. Filter image with derivative of Gaussian

Source: D. Lowe, L. Fei-Fei

Canny edge operator

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient

Source: D. Lowe, L. Fei-Fei

Canny edge operator

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:

   Thin multi-pixel wide “ridges” down to single pixel width

Source: D. Lowe, L. Fei-Fei

Canny edge operator

4. 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

MATLAB: edge(image, 'canny');
>>doc edge *(or help edge* if doc is not supported)

Source: D. Lowe, L. Fei-Fei

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

- Large σ detects large scale edges
- Small σ detects fine features

The choice of σ depends on desired behavior

Q&A 04

Quiz

The Canny edge operator is probably quite sensitive to noise.

a) True – derivatives accentuate noise
b) False – the gradient is computed using a derivative of Gaussian operator which removes noise.
c) Mostly false – it depends upon the σ chose.

Quiz

The Canny edge operator is probably quite sensitive to noise.

a) True – derivatives accentuate noise

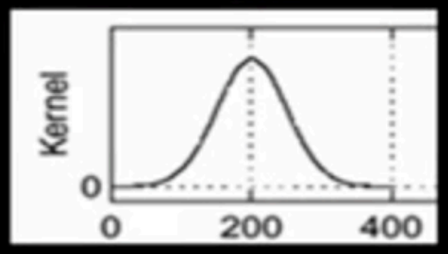
b) False – the gradient is computed using a derivative of Gaussian operator which removes noise.

c) Mostly false – it depends upon the σ chose.

Recall 1D 2nd derivative of Gaussian

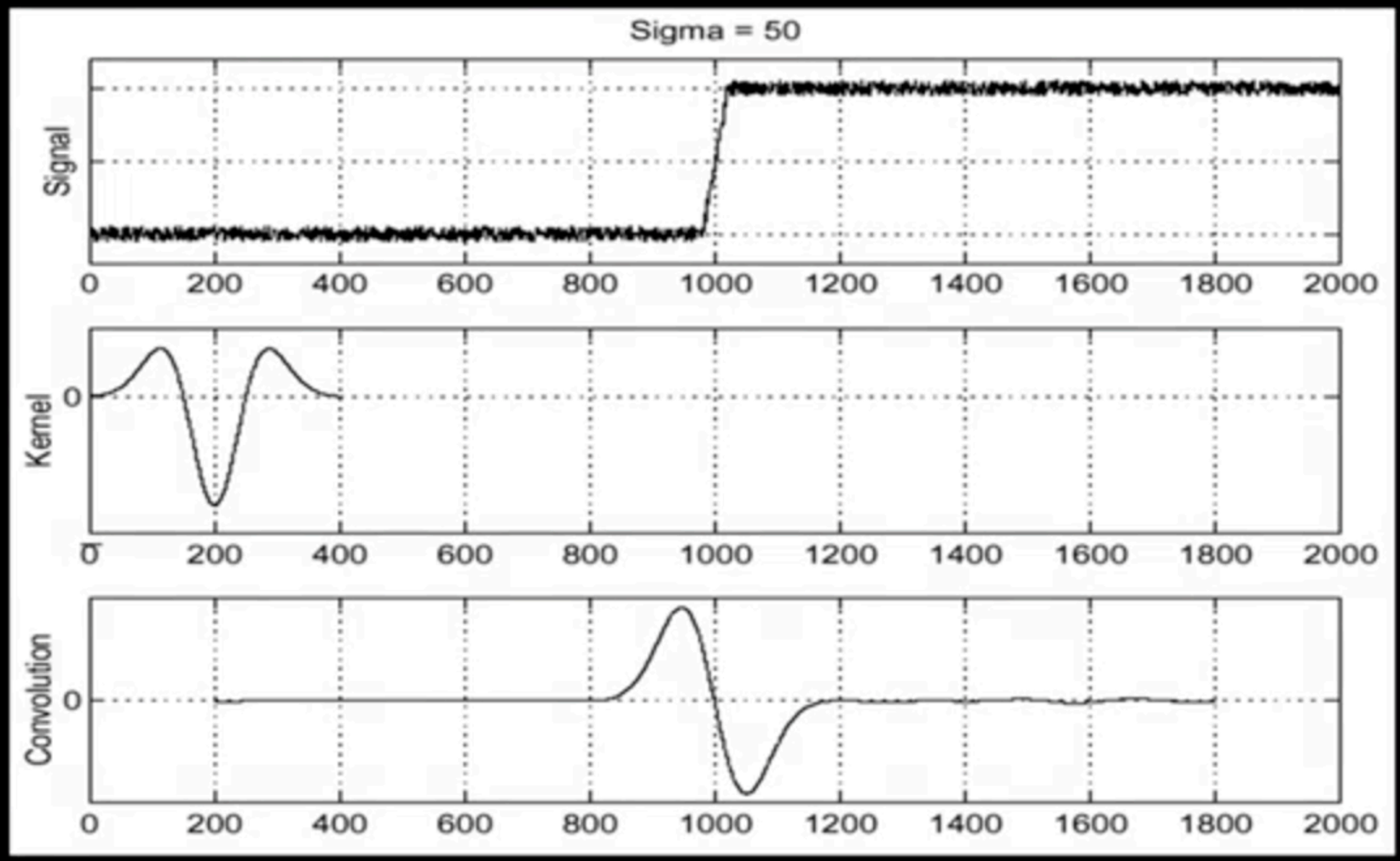
$$f$$

$$h$$

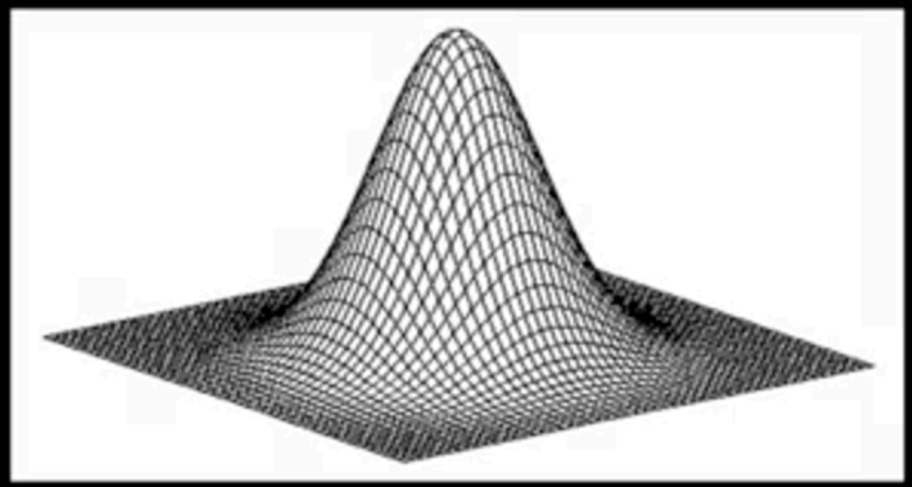


$$\frac{\partial^2}{\partial x^2} h$$

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

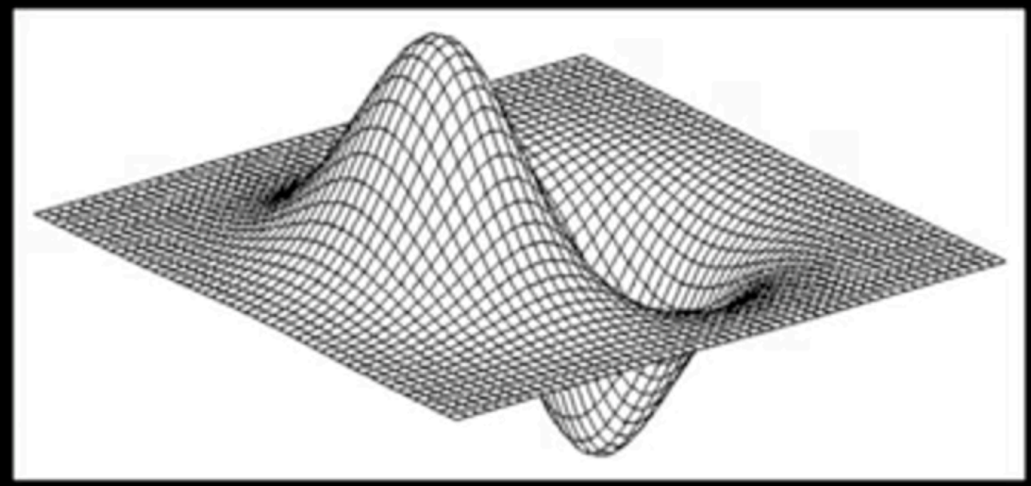


Single 2D edge detection filter



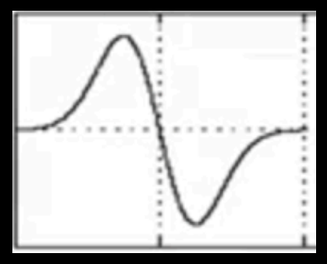
Gaussian

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



derivative of Gaussian

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

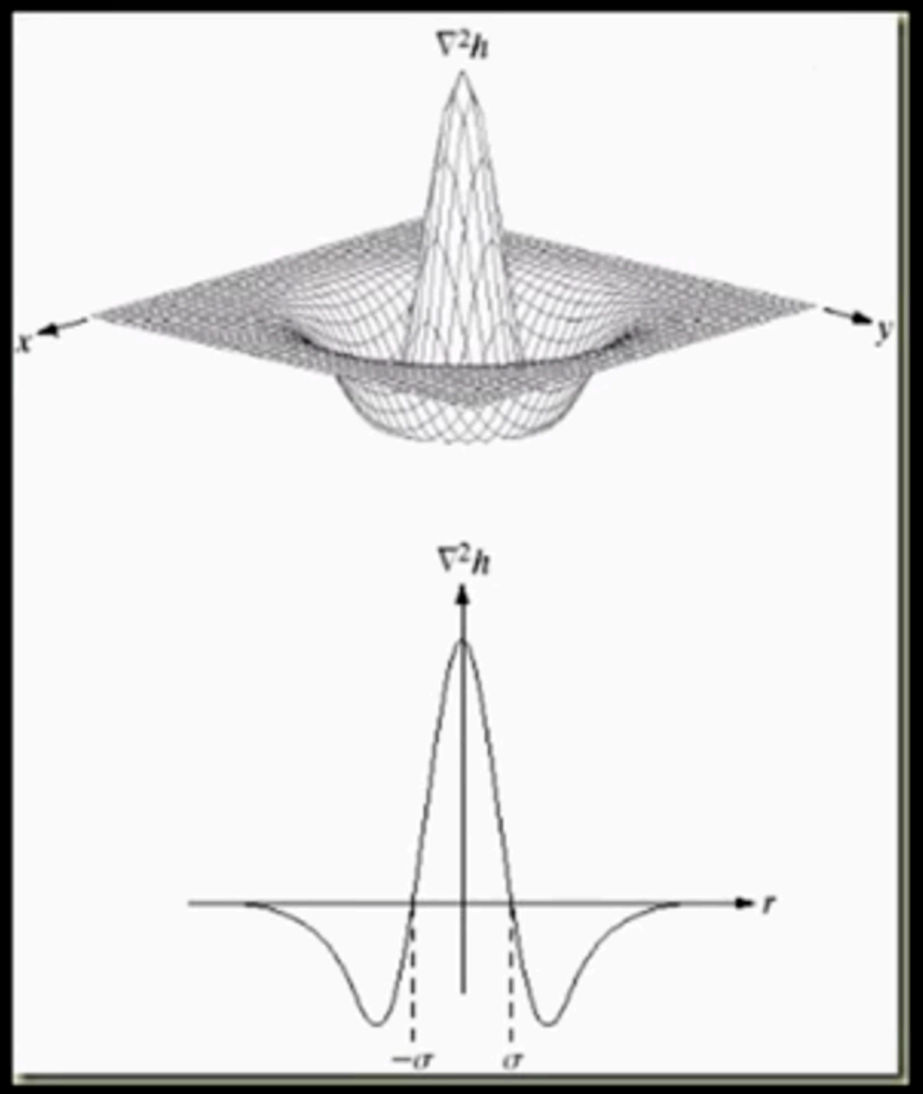


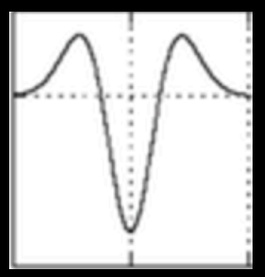
Single 2D edge detection filter

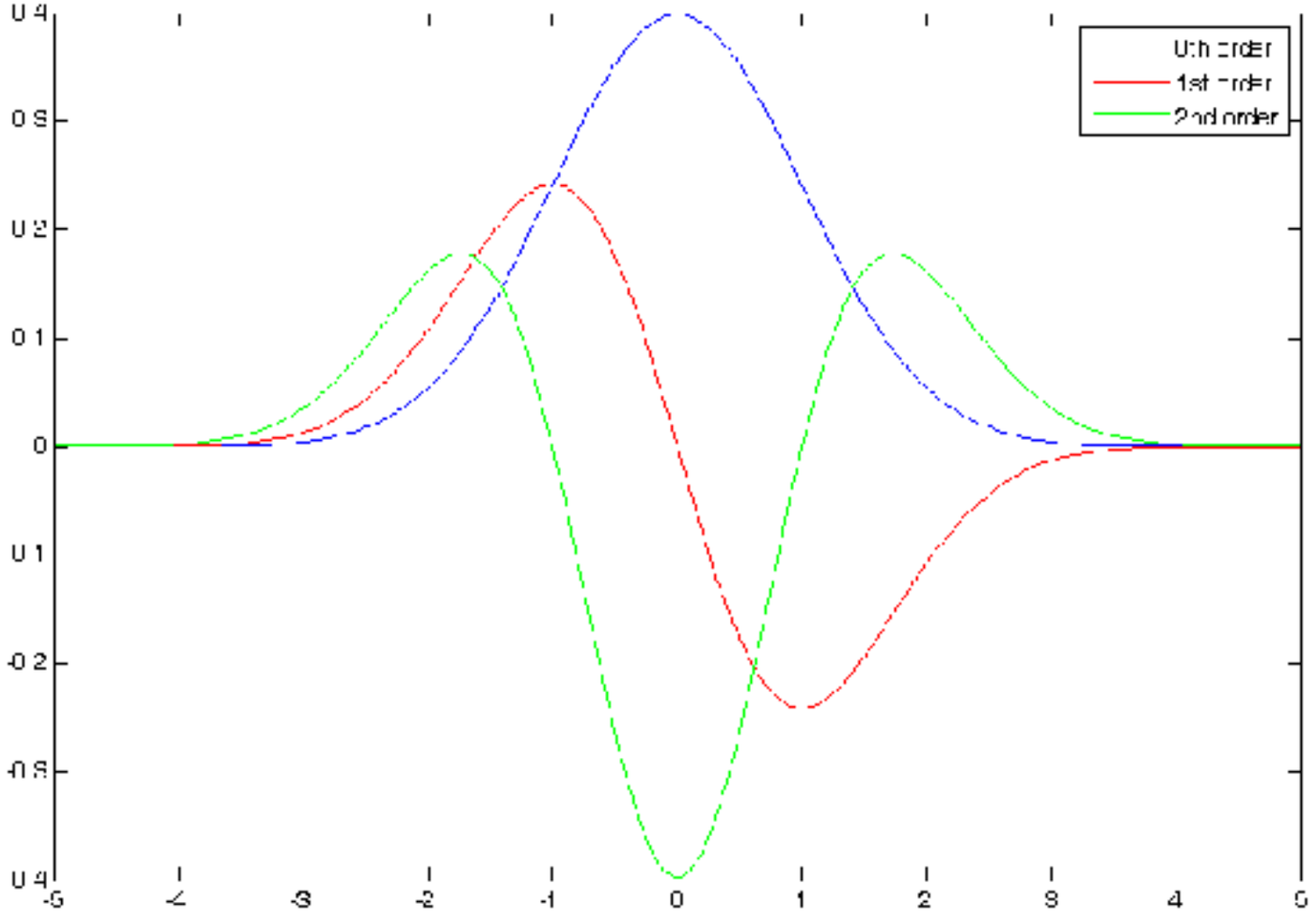
$$\nabla^2 h = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$\nabla^2$is the **Laplacian** operator,

And the zero-crossings are the edges.

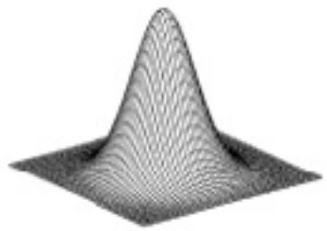






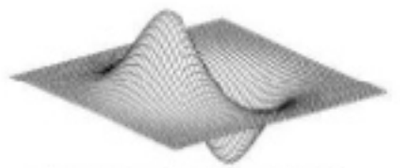
Correct!

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

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

Laplacian of Gaussian

$$\nabla^2 h_\sigma(u,v)$$

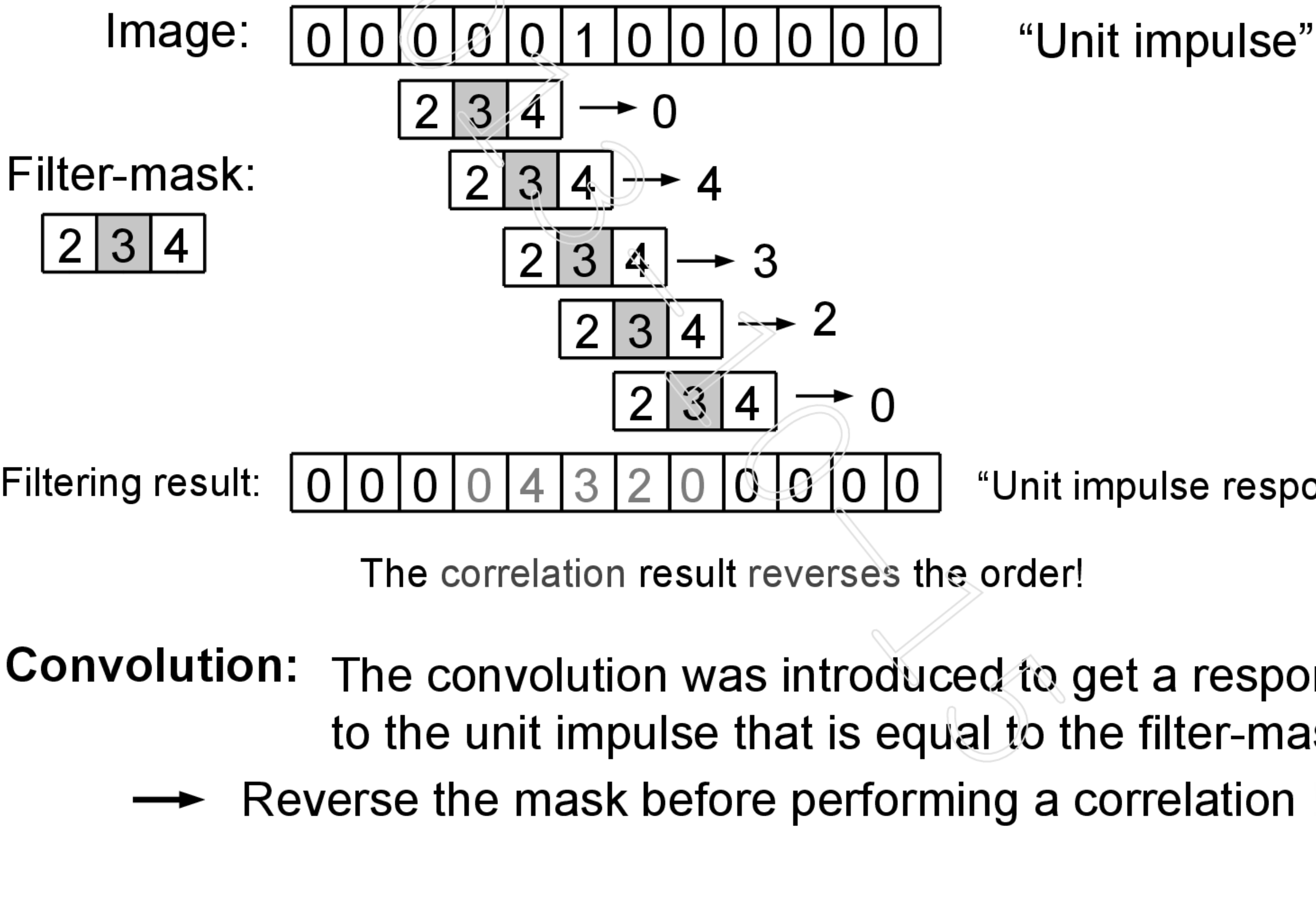
∇^2 is the **Laplacian** operator:

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

Additional material

1D Correlation / Convolution

Correlation:



The correlation result reverses the order!

Convolution: The convolution was introduced to get a response to the unit impulse that is equal to the filter-mask.

→ Reverse the mask before performing a correlation !

Convolution

- 1D continuous convolution:

$$g(n) = \int_{-\infty}^{\infty} w(t)\, f(n-t)\, dt$$

$$g = w * f$$

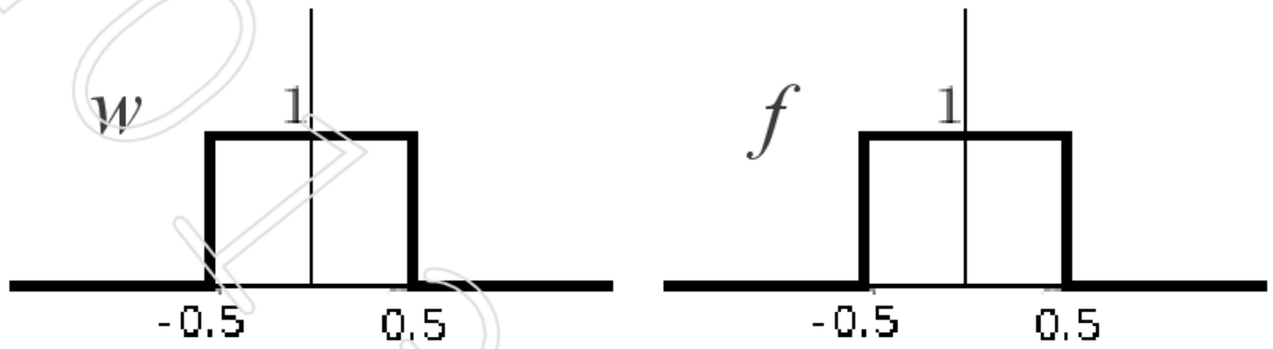
- 1D discrete convolution:

$$g(n) = \sum_{k \in D} w(k)\, f(n-k)$$

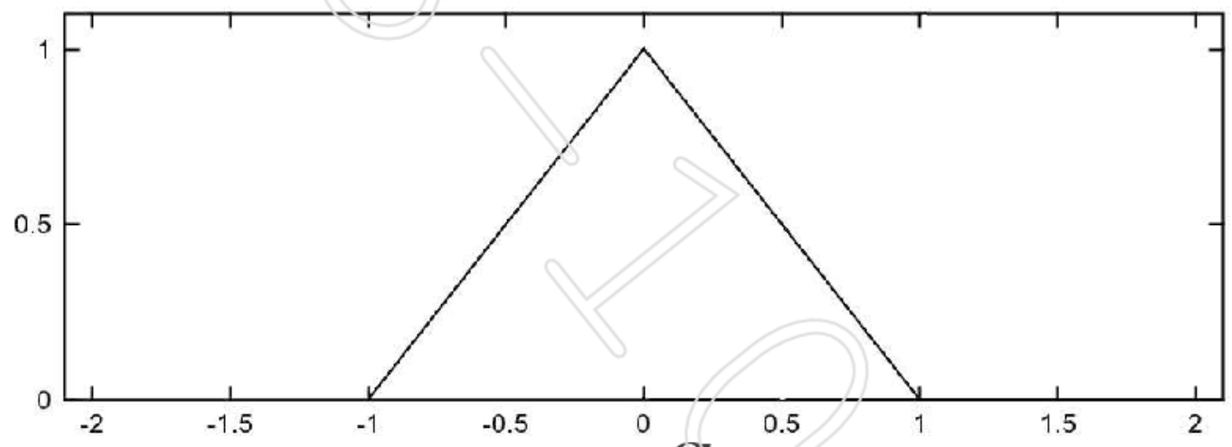
"Convolution of w and f"

Example: Convolution

Continuous:

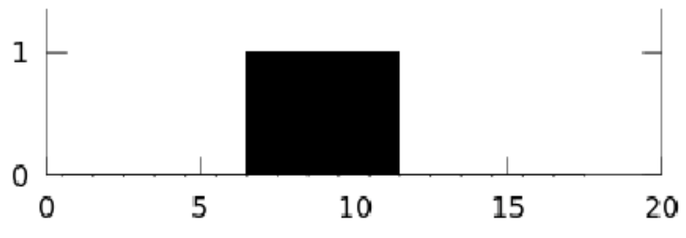


$g = w * f$



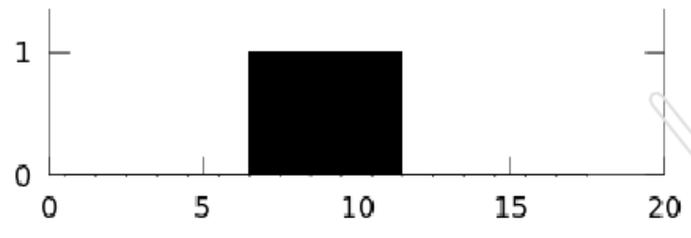
g

Discrete:



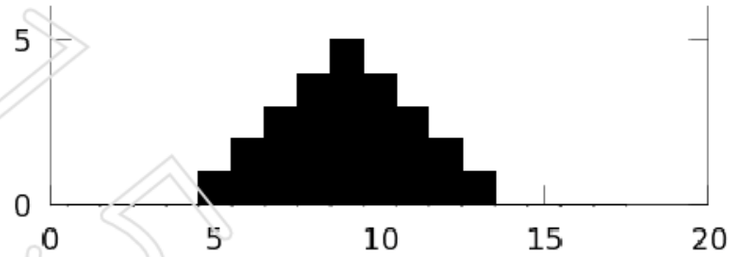
[0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]

w



[0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]

f



[0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0]

g

Convolution

- When the filter mask is symmetric then correlation and convolution results are the same!

→ Most of the filters we use **are** symmetric !

- Note: Convolution is a linear filtering operation.
- The terms **convolution filter**, **convolution mask** and **convolution kernel** are used in the literature to denote spatial filtering (so, can mean both correlation or convolution).

Creating a Filter Mask

- Take for Example an (unnormalized) 2D Gaussian function:

$$h(x,y) = e^{\frac{-x^2+y^2}{2\sigma}}$$ with standard deviation $\sigma = 1$

We just evaluate (sample) the function around its center to obtain for example a 3x3 filter mask.

$$\begin{pmatrix} h(-1,-1) & h(0,-1) & h(1,-1) \\ h(-1,0) & h(0,0) & h(1,0) \\ h(-1,1) & h(0,1) & h(1,1) \end{pmatrix} = \begin{pmatrix} 0.36788 & 0.60653 & 0.36788 \\ 0.60653 & 1.00000 & 0.60653 \\ 0.36788 & 0.60653 & 0.36788 \end{pmatrix}$$

After normalizing the sum of elements to 1.0:

$$\begin{pmatrix} 0.075114 & 0.123841 & 0.075114 \\ 0.123841 & 0.204180 & 0.123841 \\ 0.075114 & 0.123841 & 0.075114 \end{pmatrix}$$

"In practice one mostly use an approximation of this filter mask"

Spatial Smoothing Filters

- Smoothing filters blur the image and are typically used for the removal of small details and for noise reduction.

Smoothing can be obtained by **linear** and **nonlinear filtering**

NOT TO BE REPRODUCED

Spatial Sharpening Filters

- The aim of sharpening filters is to highlight transitions in intensity.
- Typically sharpening operators can be obtained by using principles from discrete differentiation.

Approximation of 1D first derivative:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x} = f(x+1)-f(x)$$

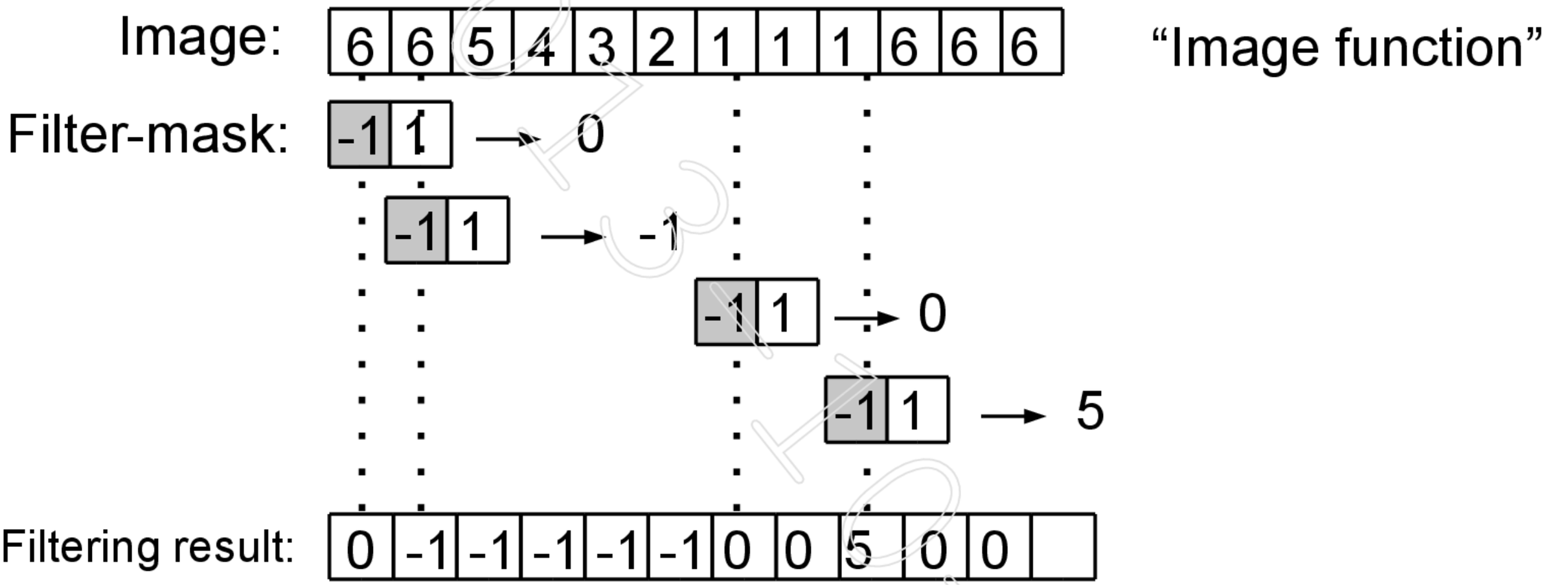
→ Difference between subsequent gray-values

-1	+1

Filter mask

1D Derivation

Correlation:



It behaves like we expect from a derivative:

Zero in constant areas
Nonzero at long ramps
Nonzero at steps

Second order Derivative

Approximation of the second derivative:

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

1	-2	1

Filter mask

→ Uses the values of the current position f(x) and the values from before f(x-1) and after f(x+1) the current value.

Note: Second derivatives are easier to implement in 2D than first derivatives. (→ we focus first on second derivatives!)

Derivatives: Illustration

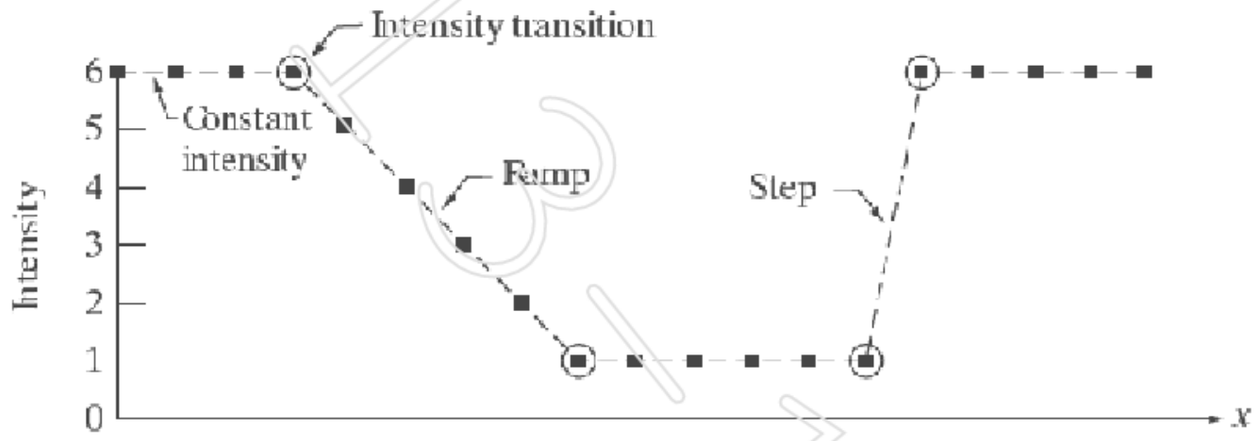
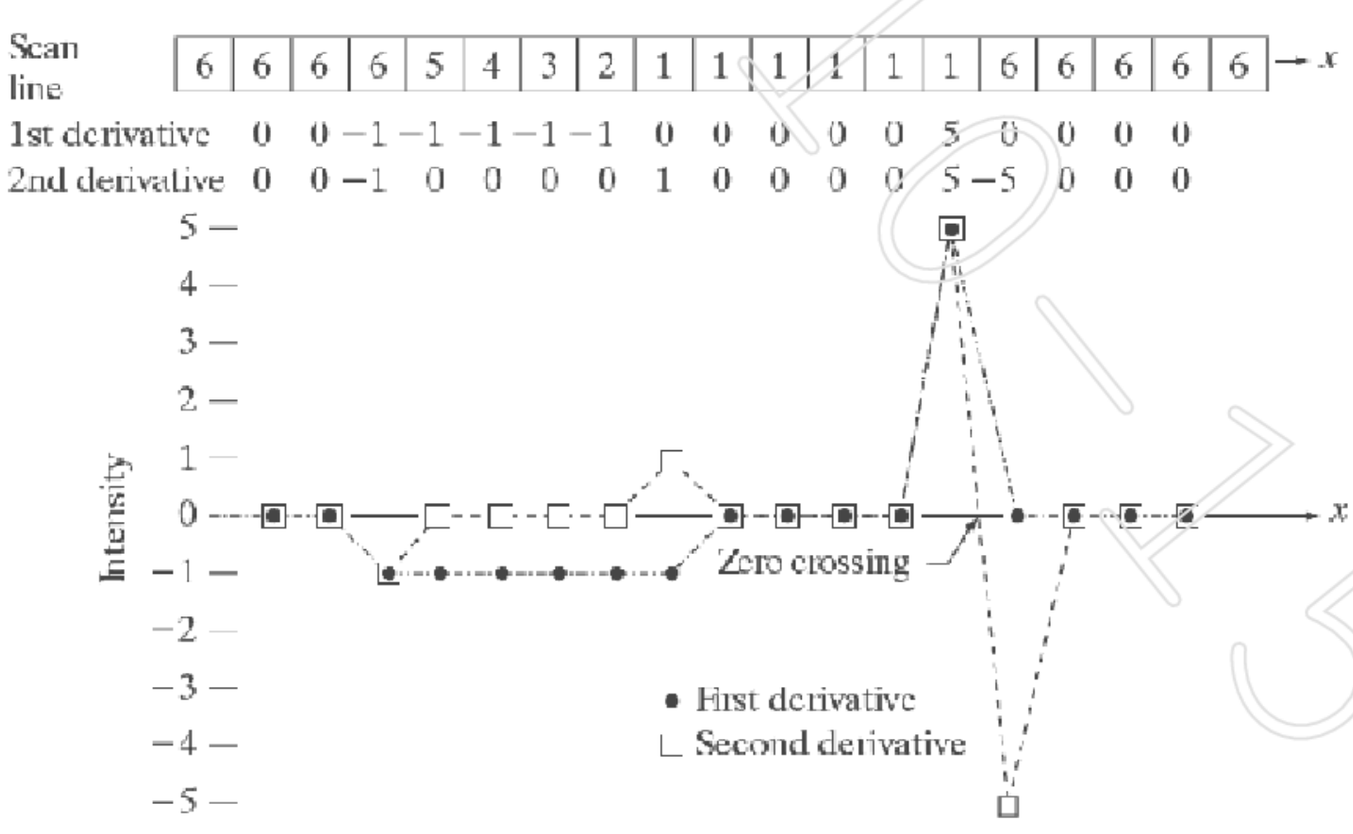


Image Function



First and second derivative

The Laplacian

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

Two parts:

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

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

Combined (discrete Laplacian):

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

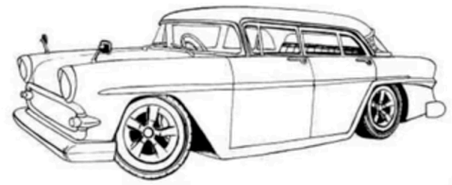
Note:"derivatives of any order are linear operations"

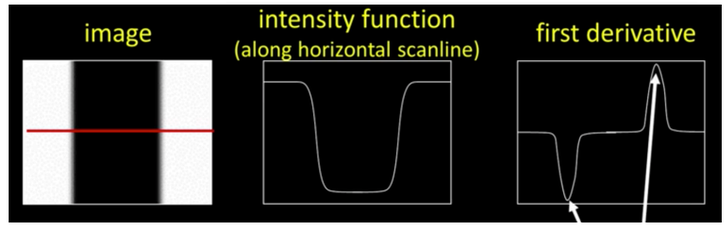
→ Linear filter

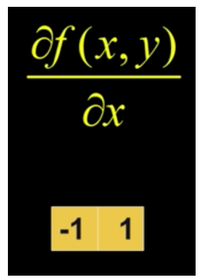
0	1	0
1	-4	1
0	1	0

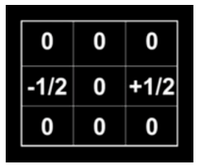
Laplacian filter mask

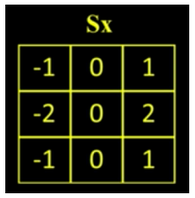
Summary

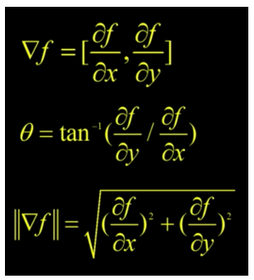




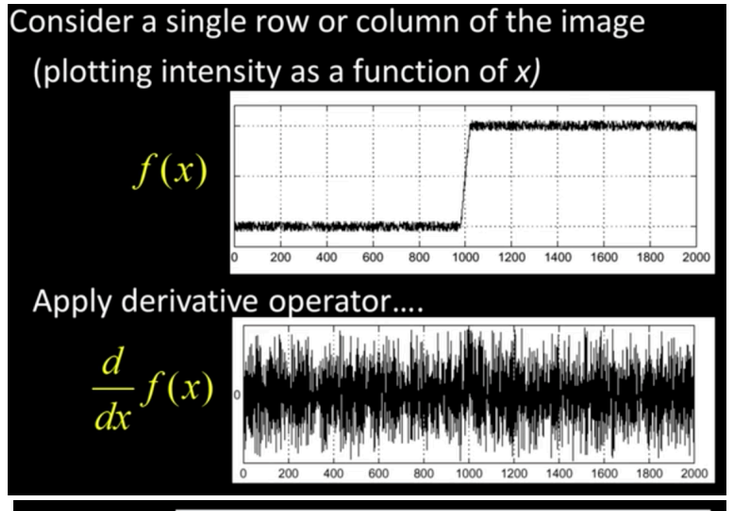








- «Reduced» images tells a lot
- Changes in intensity, Edge detection, derivatives (finite differences) and gradients.
- Noise (Smooth first)
- Derivative of Gaussian, sigma, scale
- Canny edge detector
- 2nd order derivatives, laplacian



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

1	-2	1

Filter mask

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

0	1	0
1	-4	1
0	1	0