Image Filtering
Part II

Week - 3

Announcements

- ~~Remember to enroll yourself in ED through the following link.~~

~~https://edstem.org/au/join/yHXkDY (Students have been enrolled manually.)~~

- Course Representatives have been announced on wattle and ED.
  - Arjun Raj (u7526852@anu.edu.au) [COMP4528]
  - Ruxuan Li (u7763307@anu.edu.au) [ENGN4528]
  - Jiabao Han (u7765516@anu.edu.au) [COMP6528]
  - Xingchen Zhang (u7670173@anu.edu.au) [COMP6528]

Announcements

- Lab sessions after week 1
  - We will have tutorials related to lectures.
  - The questions are taken from past exam papers
  - Please attend the lab session to get your questions answered about the Clabs and tutorial questions.
- Some Past exam papers will be released during the teaching break.
- Reports for CLAB-1 will be due in less than 2 weeks. (23:59, Friday, Mar. 15th).

Feedbacks from Tutors

- Python background. (only a few students have issues, struggled with setting up the environment…)
- We require that student can write code in python and also be familiar with object-oriented programming.
- Feedbacks can be provided to course Convenors, Tutors, and Course representatives.

Outline

- Review
- Linear filter
  - Edge detection
- Non-linear filter
  - median filter
  - bilateral filter

Review: Common types of noise

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels, sometimes, it is also referred to as Salt and pepper noise.
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original

Salt and pepper noise

random black & white pixels

random white pixels

Impulse noise

Gaussian noise

Source: S. Seitz

Review: Correlation filtering

Say the averaging window size is (2k+1) x (2k+1):

$$G[x,y] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[x+u, y+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[x,y]}$$

Attribute uniform weight to each pixel

Loop over all pixels in neighborhood around image pixel F[x,y]

Now we can generalize to allow **different weights** depending on neighboring pixel's relative position:

$$G[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[x+u, y+v]$$

Non-uniform weights

Slide credit: Kristen Grauman

Review: Correlation filtering

new img kerel/filter original img

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

This is called cross-correlation, denoted $\boxed{G = H \otimes F}$

- Filtering an image: replace each pixel with a linear combination of its neighbors.

- The filter "kernel" or "mask" *H* [*u*,*v*] defines the weights in the linear combination.

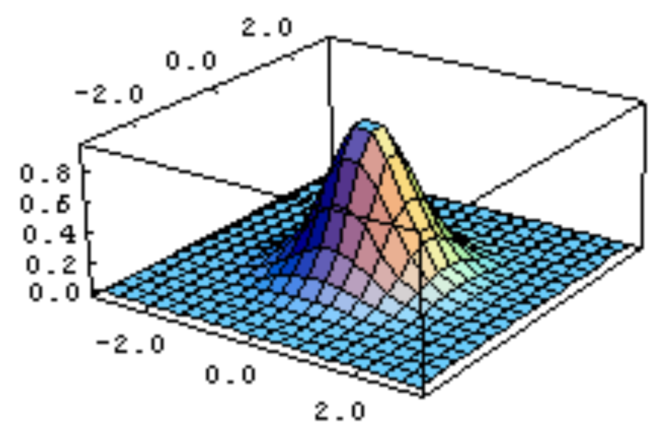
Slide credit: Kristen Grauman

Review: Gaussian Filter

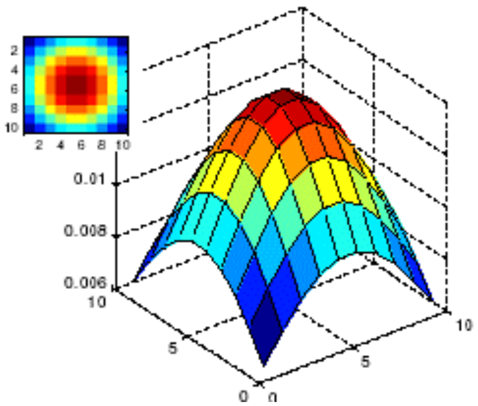
- To let nearest neighboring pixels have the most influence on the output.

- Kernel: approximation of a 2d Gaussian function:

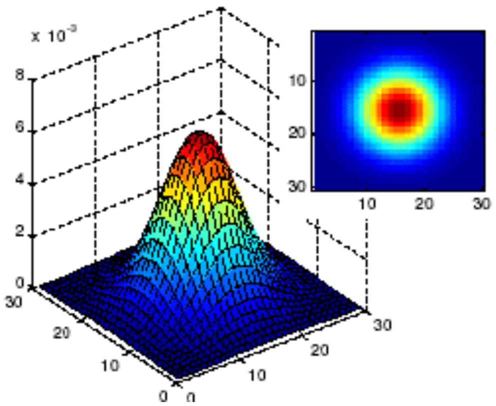
$$h(u,v) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{u^2+v^2}{2\sigma^2}}$$



- Two factors for Kernel: size and variance



σ = 5 with 10x 10 kernel



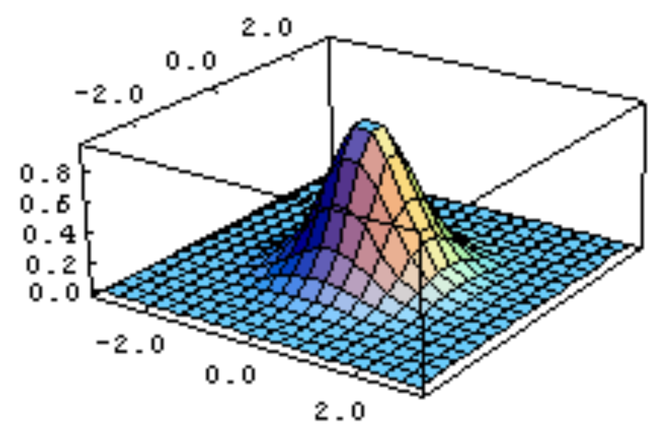
σ = 5 with 30 x 30 kernel

Review: Gaussian Filter

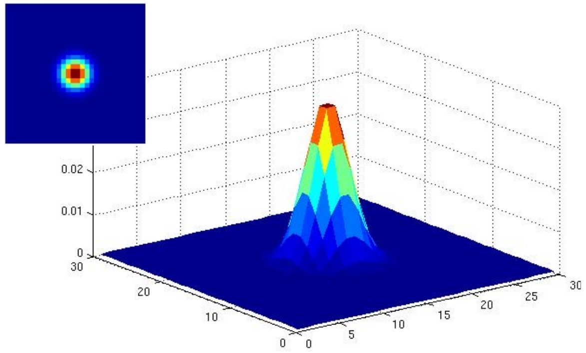
- To let nearest neighboring pixels have the most influence on the output.

- Kernel: approximation of a 2d Gaussian function:

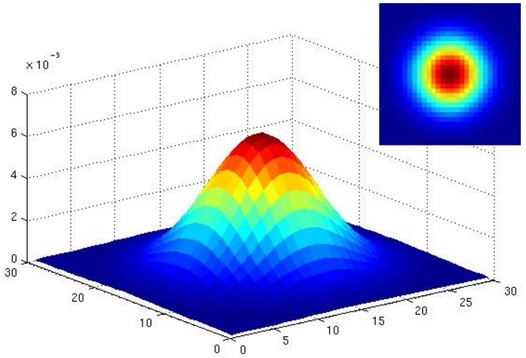
$$h(u,v) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{u^2+v^2}{2\sigma^2}}$$



- Two factors for Kernel: size and variance



σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Review: Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

linear combination

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

$$G = H \star F$$

Notation for convolution operator

Comparisons

Correlation:

$$G[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] \boxed{F[x+u, y+v]}$$

$$G = H \otimes F$$

Convolution:

$$G[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] \boxed{F[x-u, y-v]}$$

$$\boxed{G = H \star F}$$

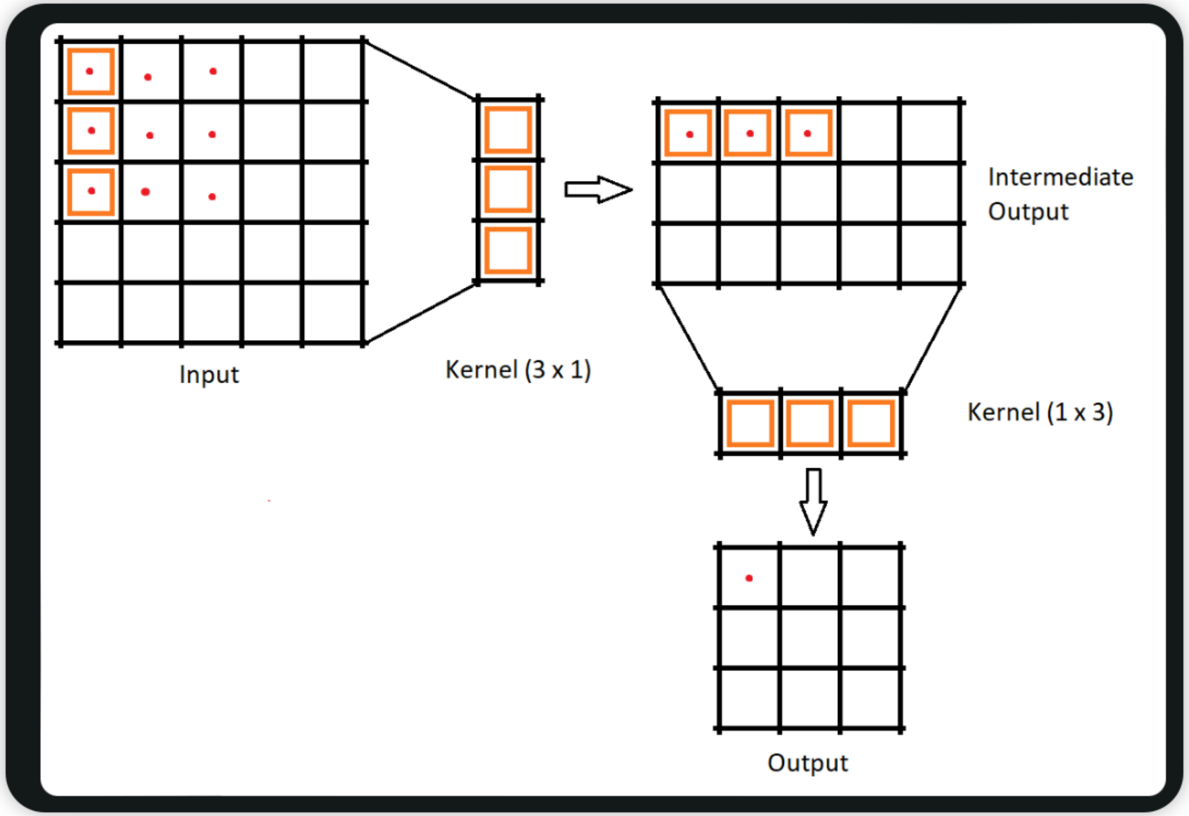
Recall: Separability

- In some cases, the filter is separable, and we can factor the 2D convolution into two steps:
  - Convolve all rows with a 1D filter
  - Convolve all columns with a 1D filter

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \end{bmatrix}$$

- What is the complexity of filtering an n×n image with an m×m kernel?
  - $O(n^2 m^2)$
- What if the kernel is separable?
  - $O(n^2 m)$

A **separable filter** in image processing can be written as product of two more simple filters. Typically a 2-dimensional convolution operation is separated into two 1-dimensional filters. This reduces the computational costs on an $N \times M$ image with a $m \times n$ filter from $\mathcal{O}(M \cdot N \cdot m \cdot n)$ down to $\mathcal{O}(M \cdot N \cdot (m+n))$. [1]



https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-separable-convolutions/#

Separable Kernel

- How can we know whether a kernel is separable?
- Solution:
  - Treat a kernel as a matrix K.
  - Take the singular value decomposition (SVD) of K

$$K = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

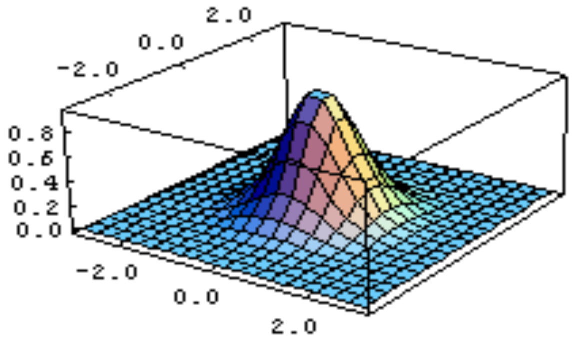
- If only the first singular value σ_0 is non-zero, the kernel is called separable and and provide the vertical and horizontal kernels as:

$\sqrt{\sigma_0}\mathbf{u}_0$ and $\sqrt{\sigma_0}\mathbf{v}_0^T$, respectively.

Reference: Chapter 3.2.1 in the Computer Vision: Algorithms and Applications

This kernel is an approximation of a 2d Gaussian function:

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



Properties of Gaussian filter (proof provided in the supp reading)

- Rotational symmetry treats features of all orientations equally (isotropy).
- Convolution with self gives another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into the product of two 1D Gaussians

Source: K. Grauman

Separability of the Gaussian filter

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

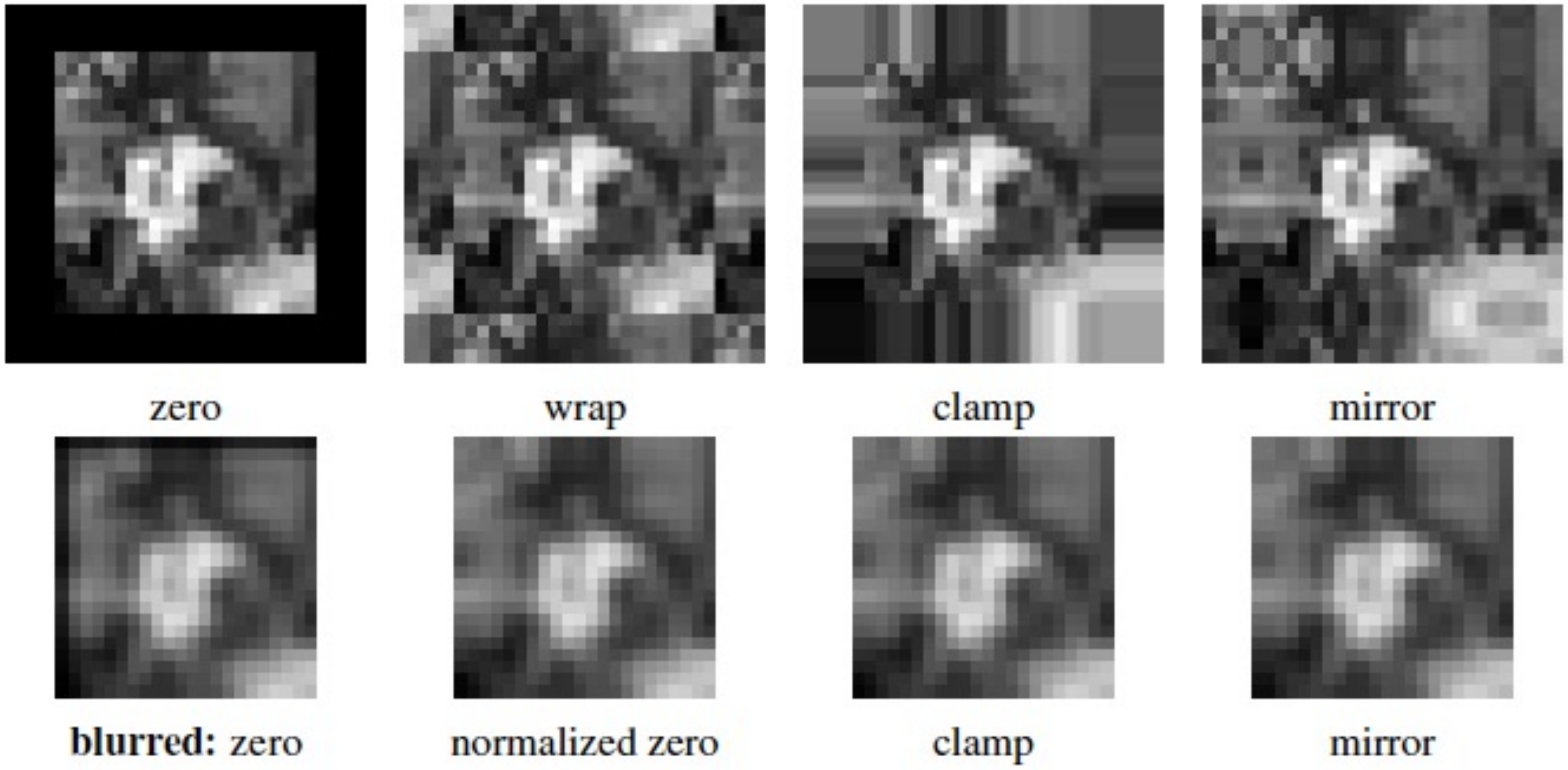
$$= \boxed{\left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}}\right)} \boxed{\left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}}\right)}$$

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

How to handle borders



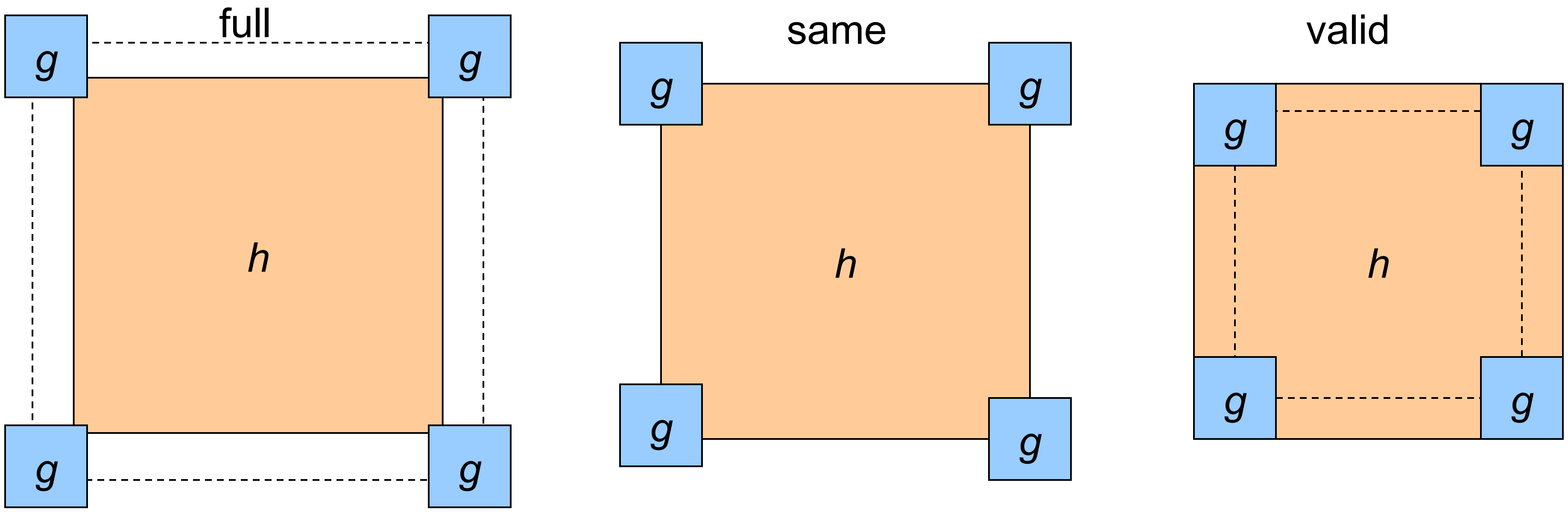
From Szeliski, Computer Vision, 2010

Example: Border handling

Source: S. Marschner

Matlab’s border handling

- What is the size of the output?
- Cv2.filter2d(srt, dst, -1, *bordertypes*)
  - output size is sum of sizes of h and g
  - *shape* = ‘same’: output size is same as h



Source: S. Lazebnik

Matlab's border handling

- What is the size of the output?
- Cv2.filter2d(srt, dst, *bordertypes*)
  - output size is sum of sizes of h and g
  - *shape* = 'same': output size is same as h

Enumerator	
BORDER_CONSTANT Python: cv.BORDER_CONSTANT	`iiiiii|abcdefgh|iiiiiii` with some specified `i`
BORDER_REPLICATE Python: cv.BORDER_REPLICATE	`aaaaaa|abcdefgh|hhhhhhh`
BORDER_REFLECT Python: cv.BORDER_REFLECT	`fedcba|abcdefgh|hgfedcb`
BORDER_WRAP Python: cv.BORDER_WRAP	`cdefgh|abcdefgh|abcdefg`

Source: S. Lazebnik

Discrete Filtering

- Linear filter: Weighted sum of pixels over rectangular neighborhood—*kernel* defines weights

- Think of kernel as template being matched by correlation (`Python:` *cv2.filter2D*)

- Convolution: Correlation with kernel rotated 180° (~~Matlab: `conv2`~~)

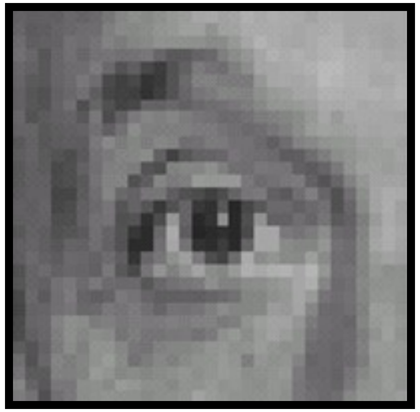
1	-1	-1
1	2	-1
1	1	1

Practice with linear filters

A taxonomy of useful filters

- Impulse, Shifts,
- Blur
  - Box filter
  - Gaussian filter
  - Bilateral filter
  - Asymmetrical filter: motion blur
- Edges
  - [-1 1]
  - Derivative filter
  - Derivative of a gaussian
  - *Oriented filters*
  - *Gabor filter*
  - *Quadrature filters: phase and magnitude.*
  - *Elongated edges: filling gaps…*

Practice with linear filters



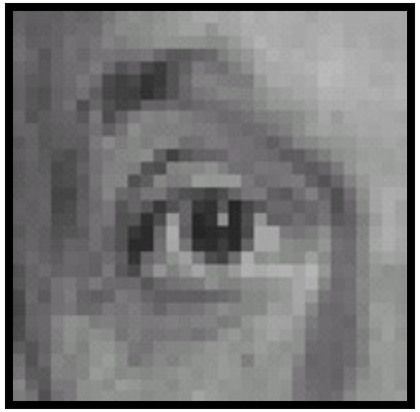
Original

0	0	0
0	1	0
0	0	0

?

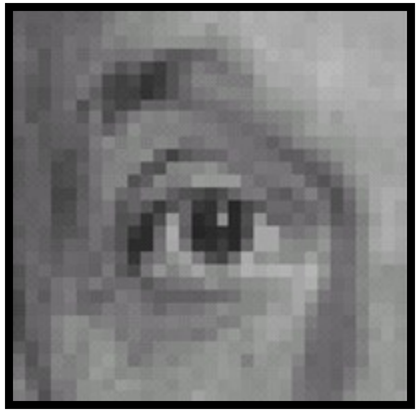
Source: D. Lowe

Practice with linear filters



Original

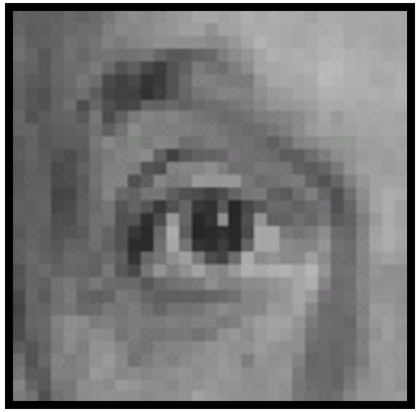
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters



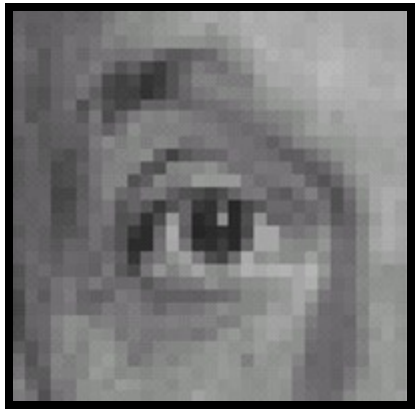
Original

0	0	0
0	0	1
0	0	0

?

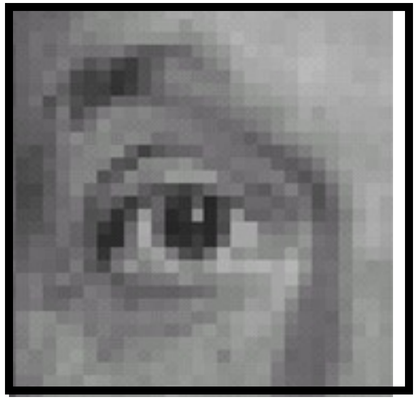
Source: D. Lowe

Practice with linear filters



Original

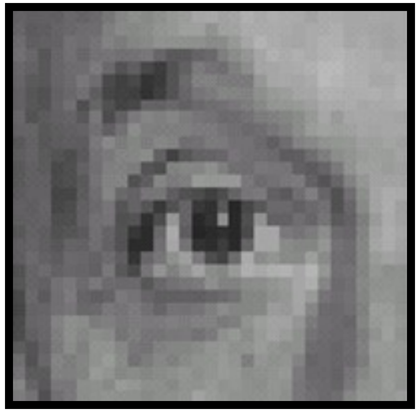
0	0	0
0	0	1
0	0	0



Shifted left by 1 pixel

Source: D. Lowe

Practice with linear filters



Original

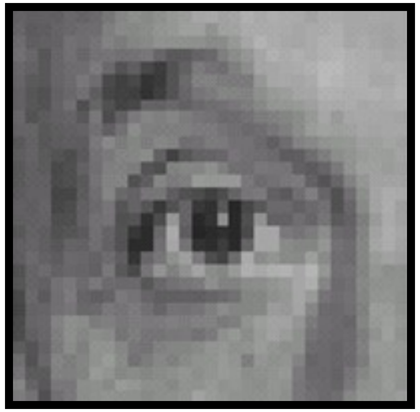
$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

?

(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters



Original

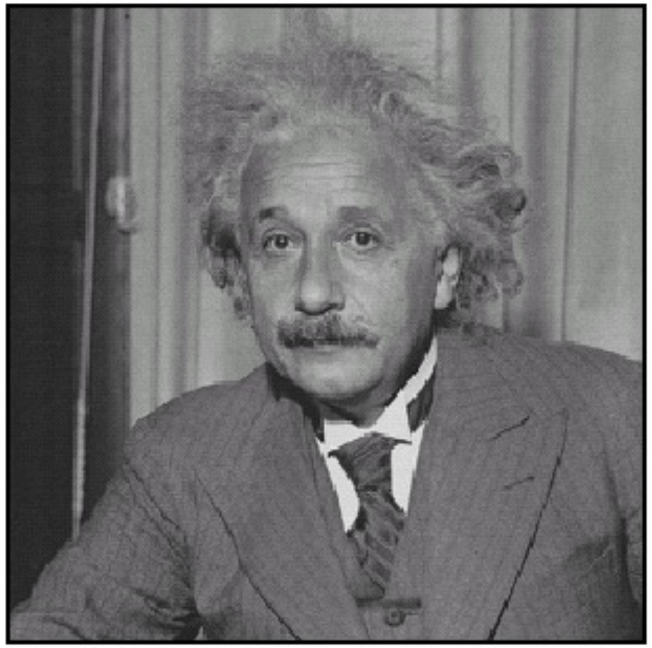
$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Sharpening filter

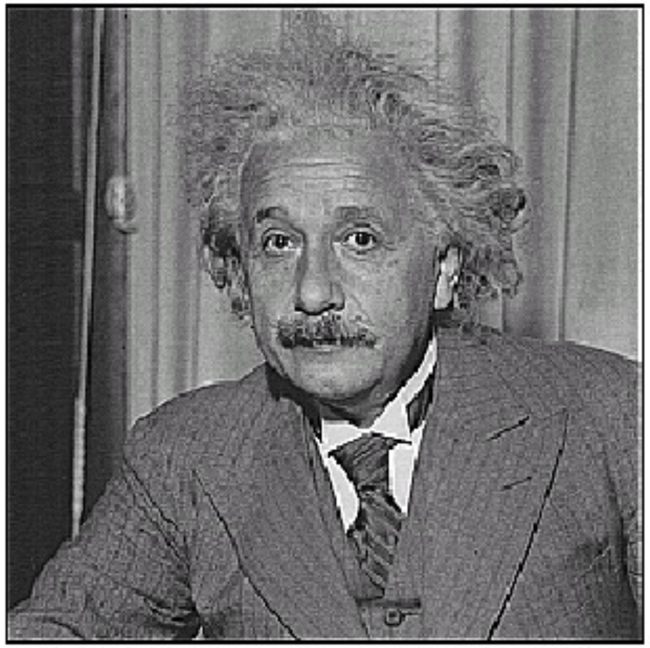
- Accentuates differences with local average

Source: D. Lowe

Sharpening



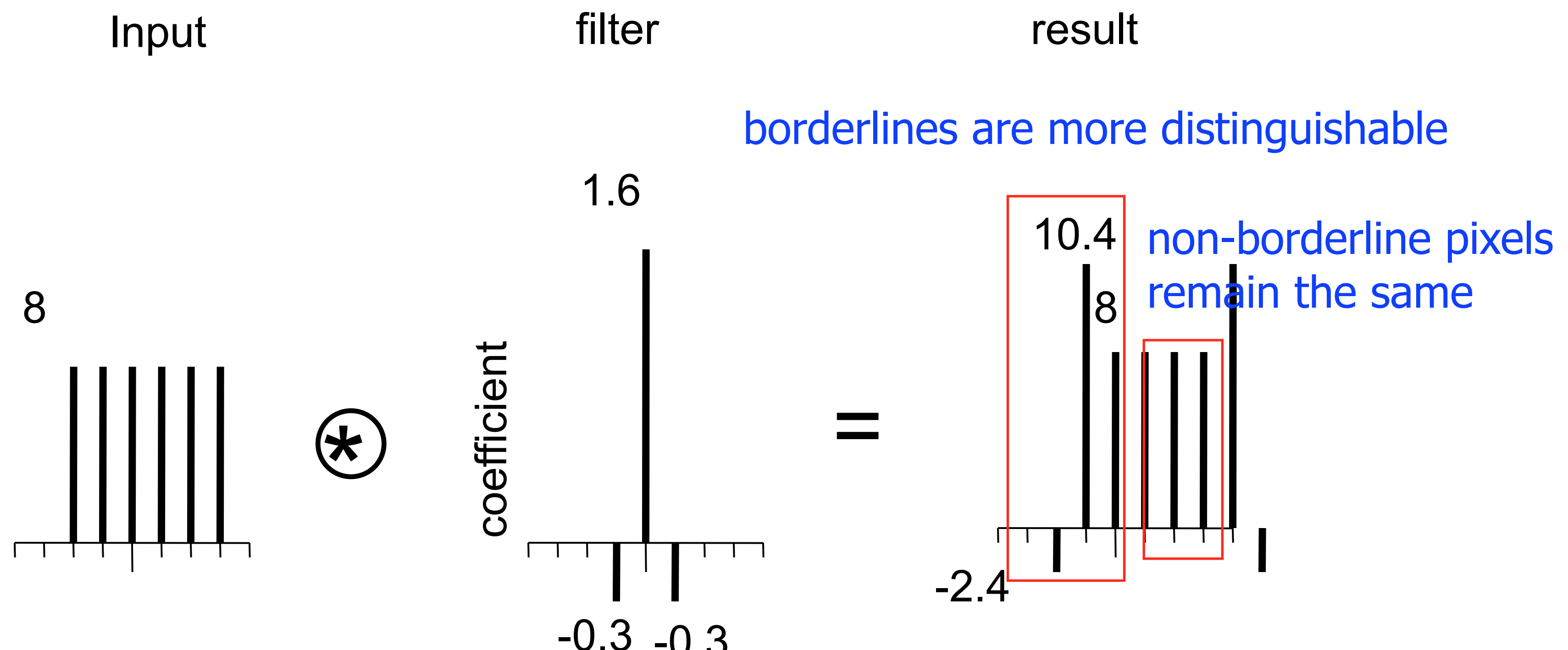
before



after

Source: D. Lowe

1-D sharpening example



Sharpened
(differences are accentuated; constant areas are left untouched).

[-1 1]

$$\frac{\partial \mathbf{I}}{\partial x} \simeq \mathbf{I}(x,y) - \mathbf{I}(x-1,y)$$

g[m,n]

$\otimes$

[-1, 1]

h[m,n]

=

f[m,n]

$[-1\ 1]^T$

$$\frac{\partial \mathbf{I}}{\partial y} \approx \mathbf{I}(x,y) - \mathbf{I}(x,y-1)$$

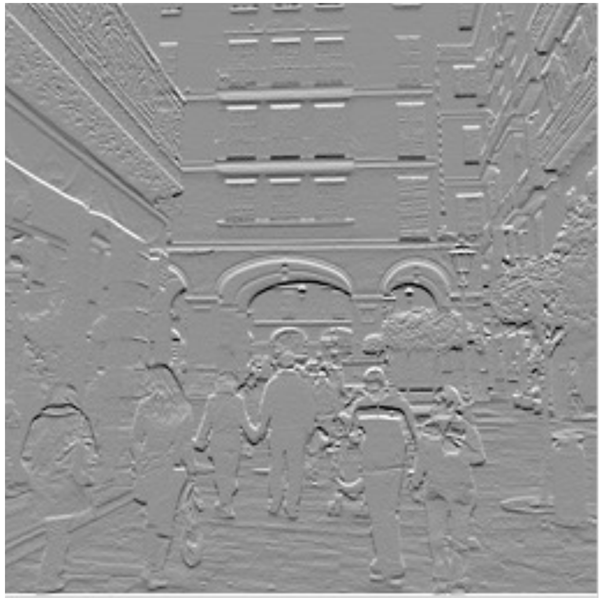
g[m,n]

$\otimes$

$[-1, 1]^T$

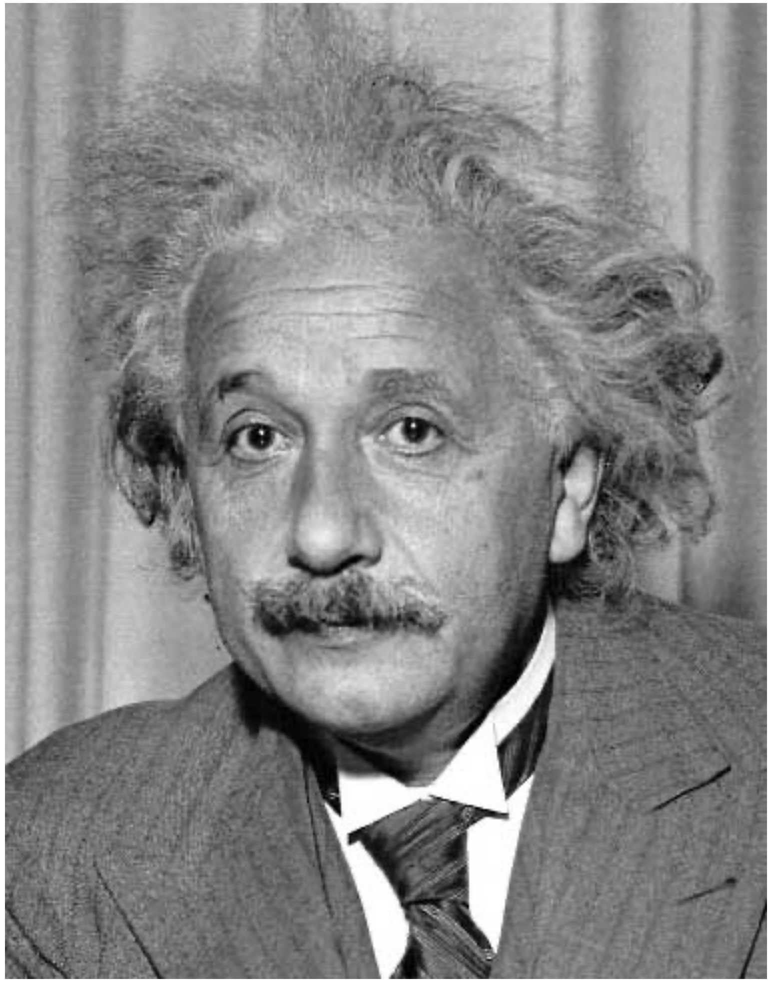
h[m,n]

=



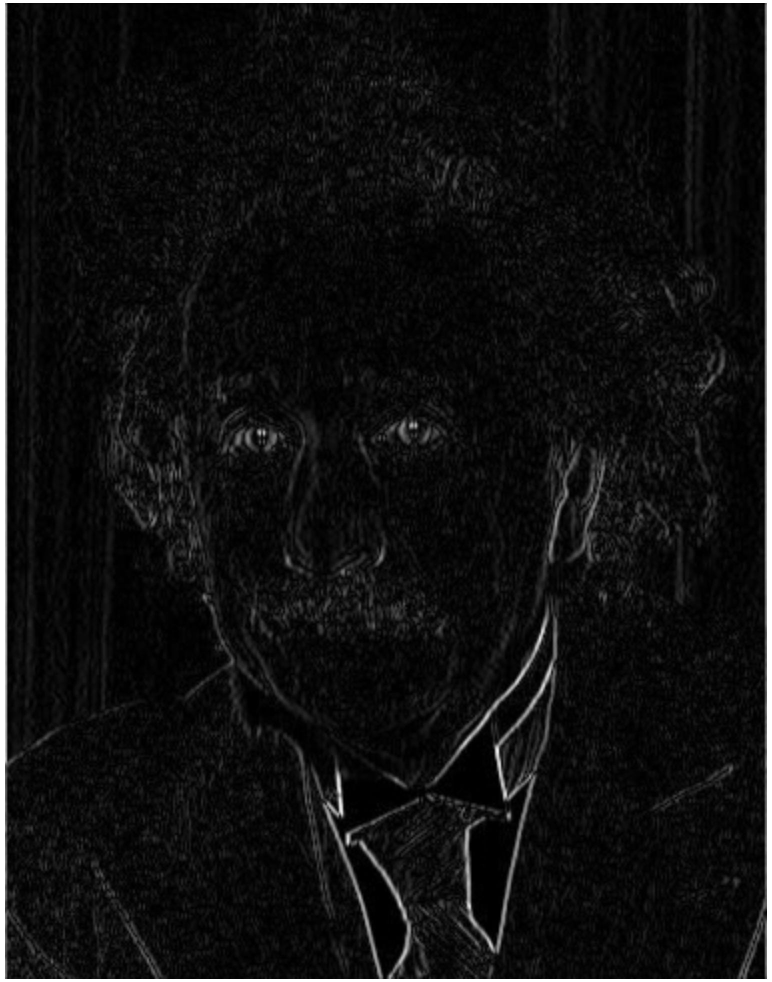
f[m,n]

Derivative filters



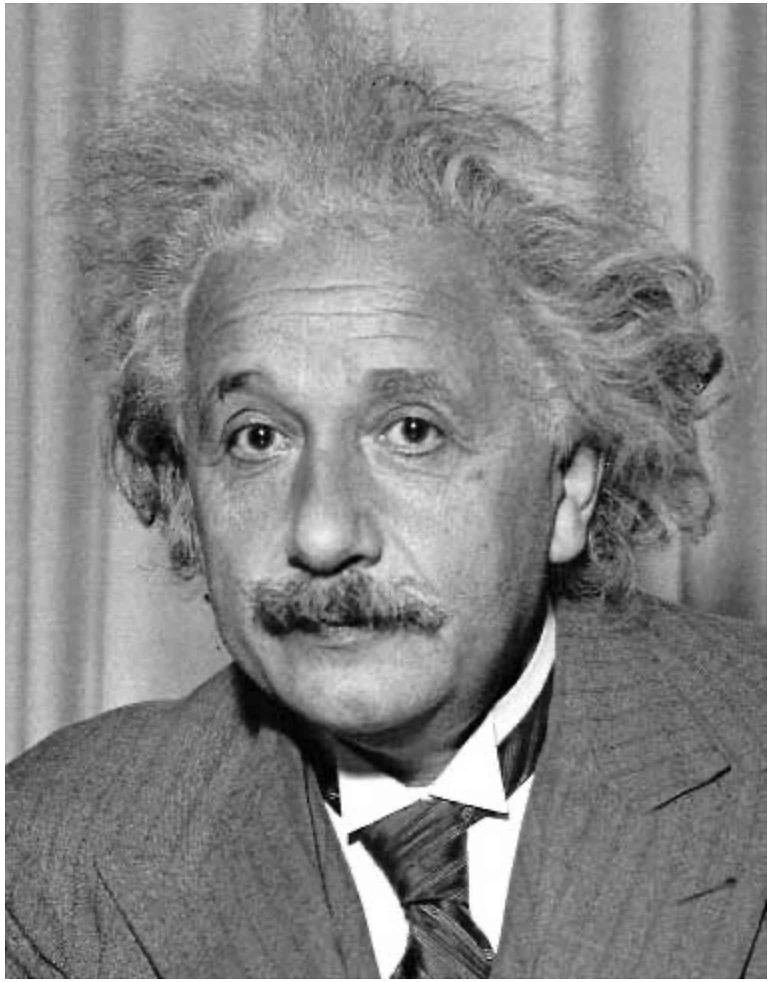
-1	0	1
-2	0	2
-1	0	1

Sobel



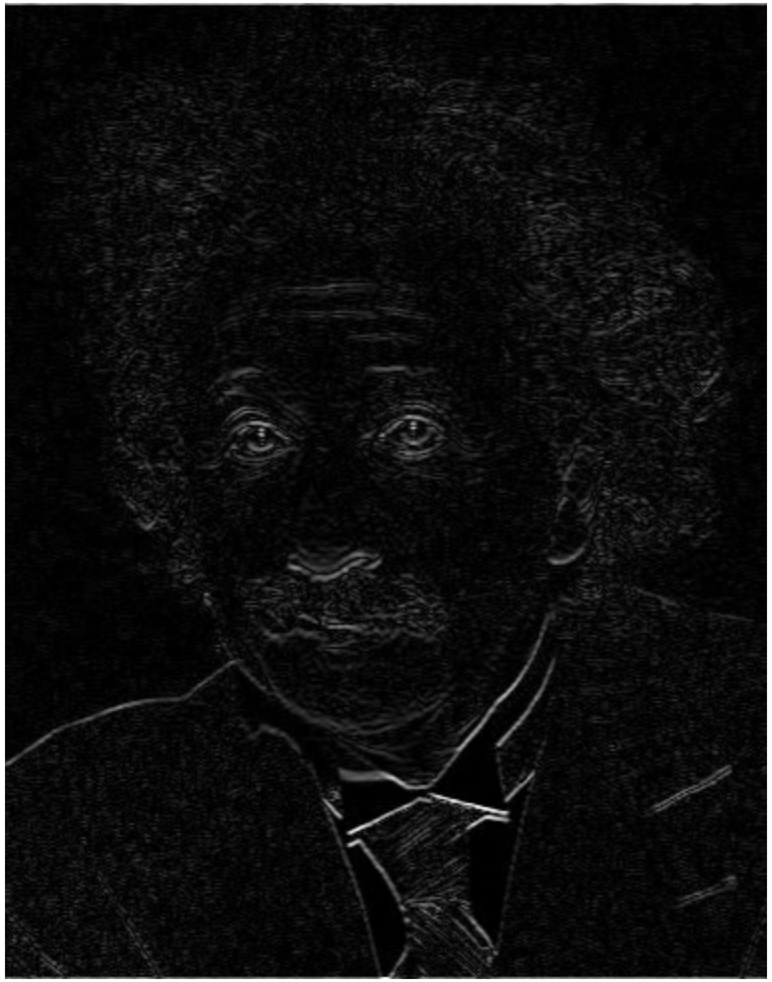
Vertical Edge
(absolute value)

Derivative filters



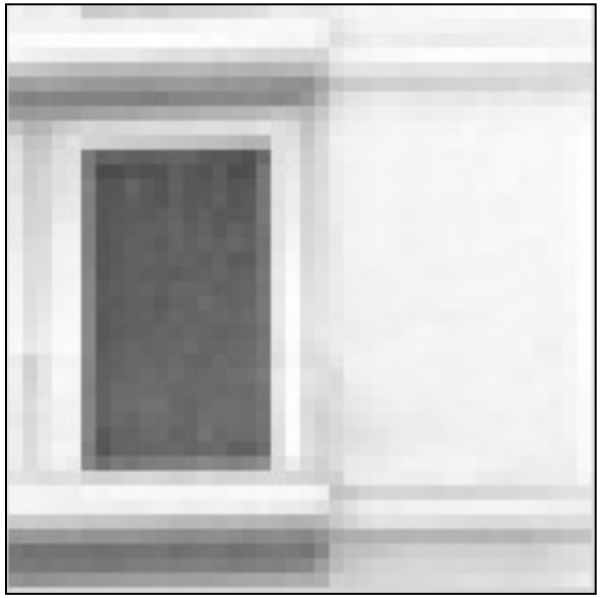
1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

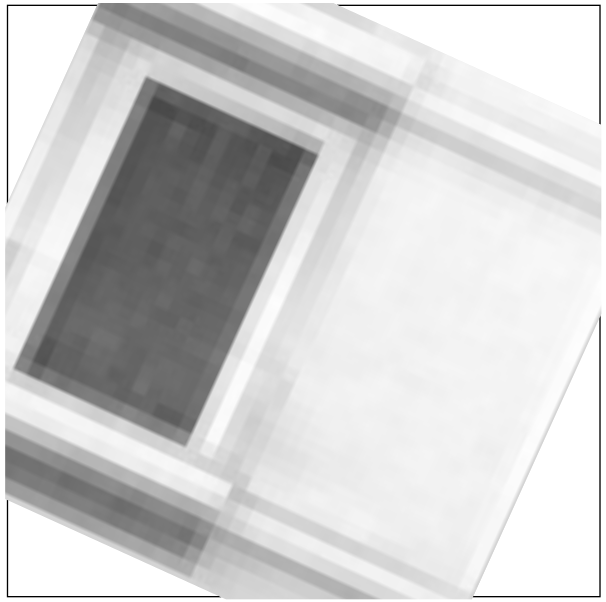
Image rotation



g[m,n]

$\otimes$? =

h[m,n]



f[m,n]

It is linear, but not a spatially invariant operation. There is not convolution.

How could we synthesize motion blur?

Motion blur filter

g[m,n]

$\otimes$

h[m,n]

=

f[m,n]

Motion blur filter

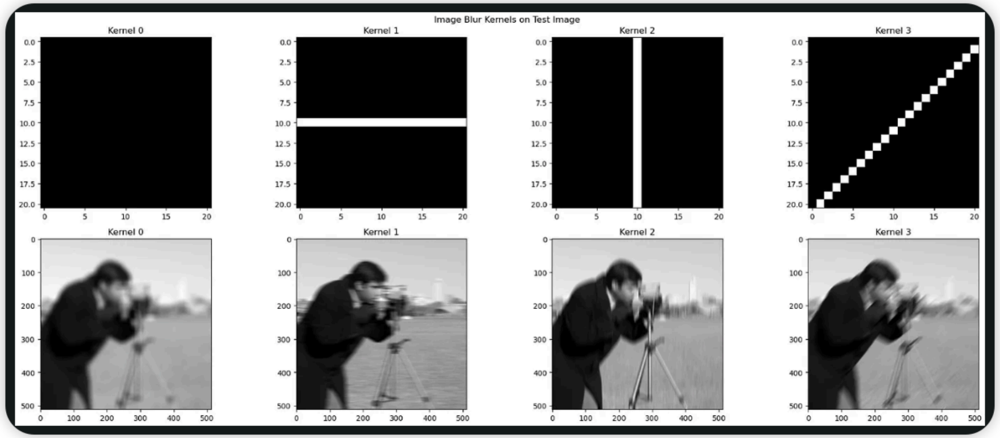
$\otimes$

h[m,n]

=

g[m,n]

f[m,n]



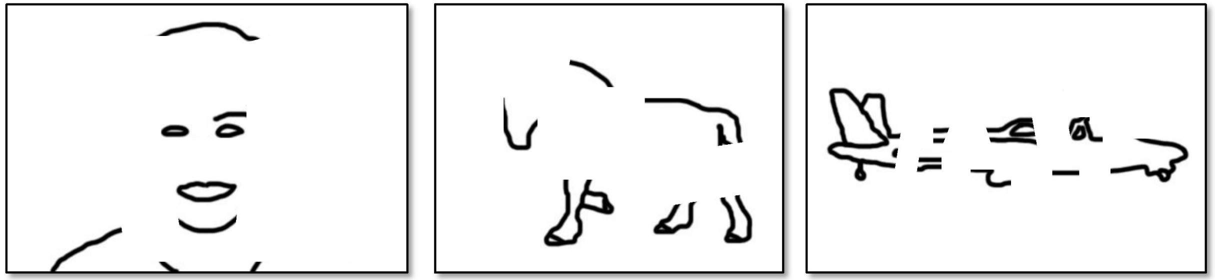
Summary

- Linear filters and convolution useful for
  - Enhancing images (smoothing, removing noise)
    - Box filter
    - Gaussian filter
- Separable filters more efficient

Edge Detection

Edge detection

- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**

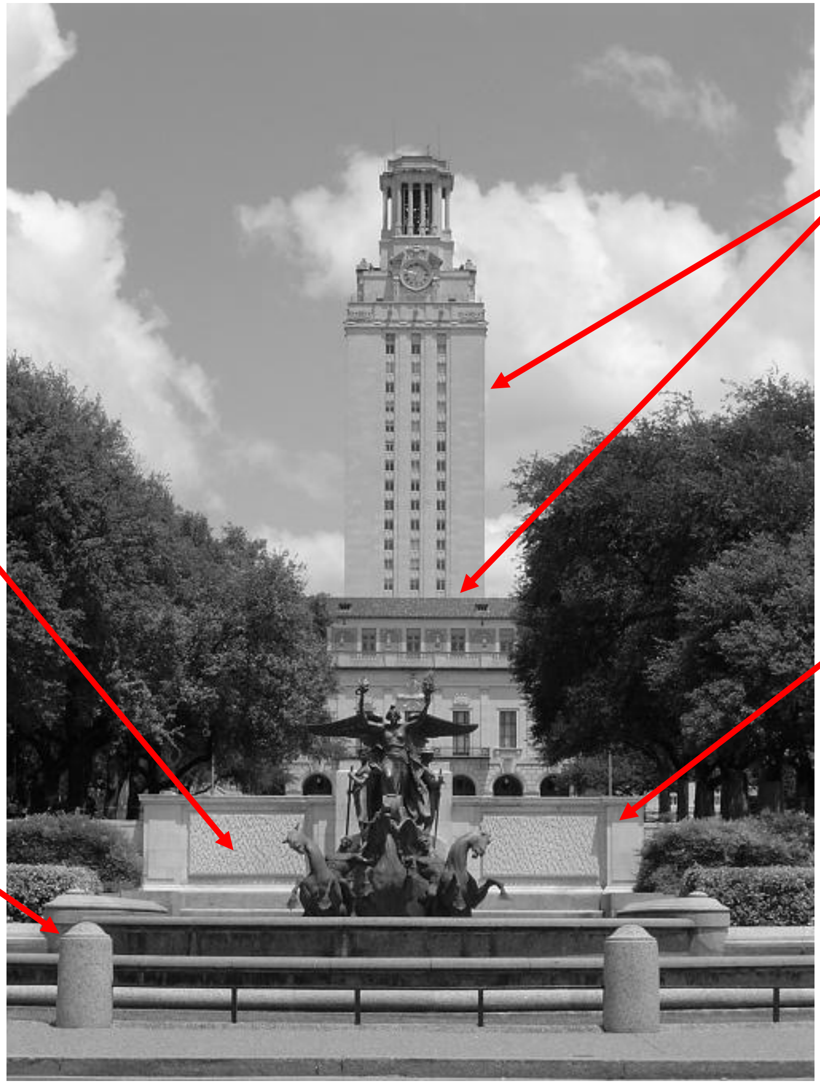


•Figure from J. Shotton et al., PAMI 2007

- **Main idea**: look for strong gradients, post-process

What causes an edge?

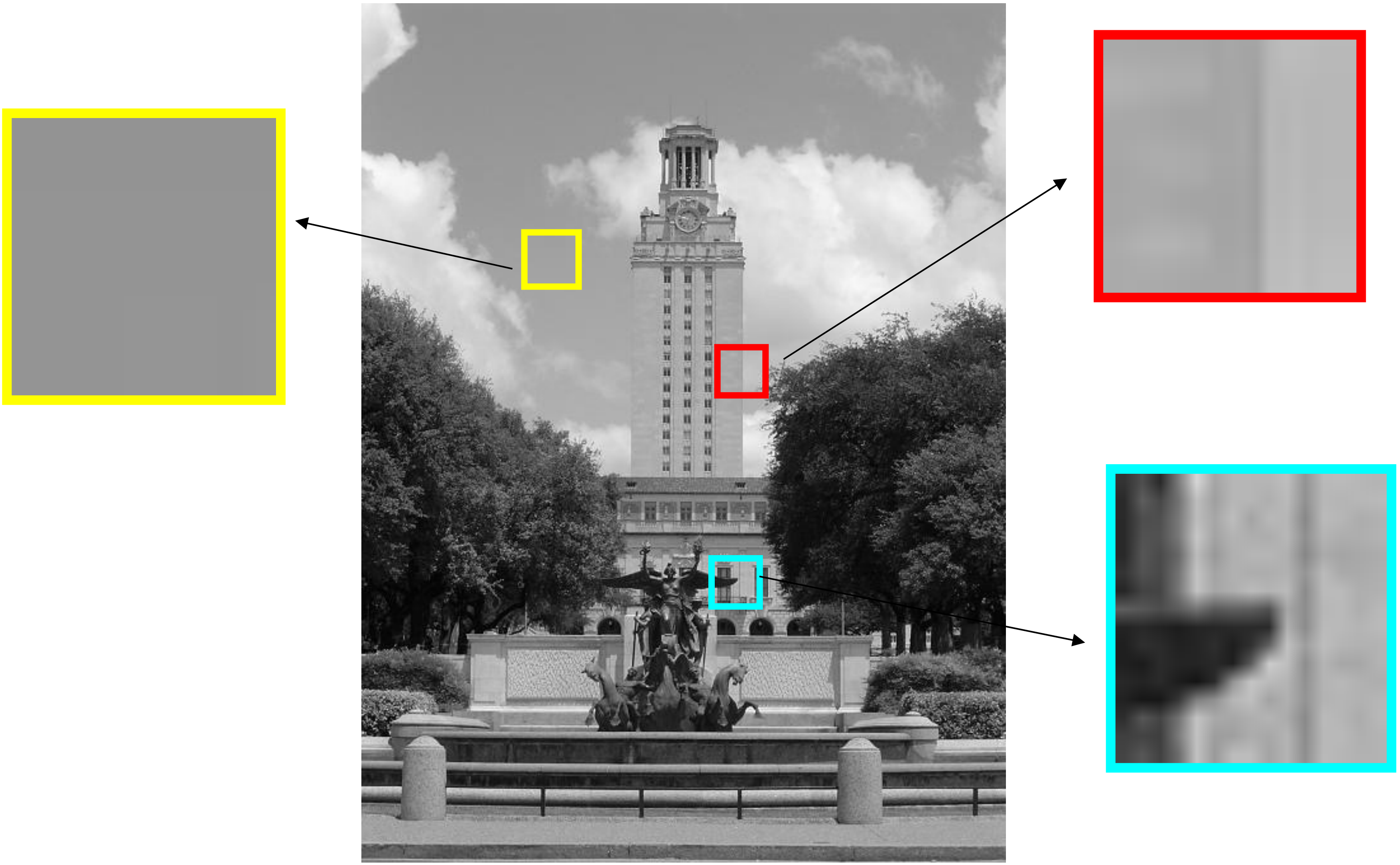
- Reflectance change: appearance information, texture
- Change in surface orientation: shape



- Depth discontinuity: object boundary
- Cast shadows

•Slide credit: Kristen Grauman

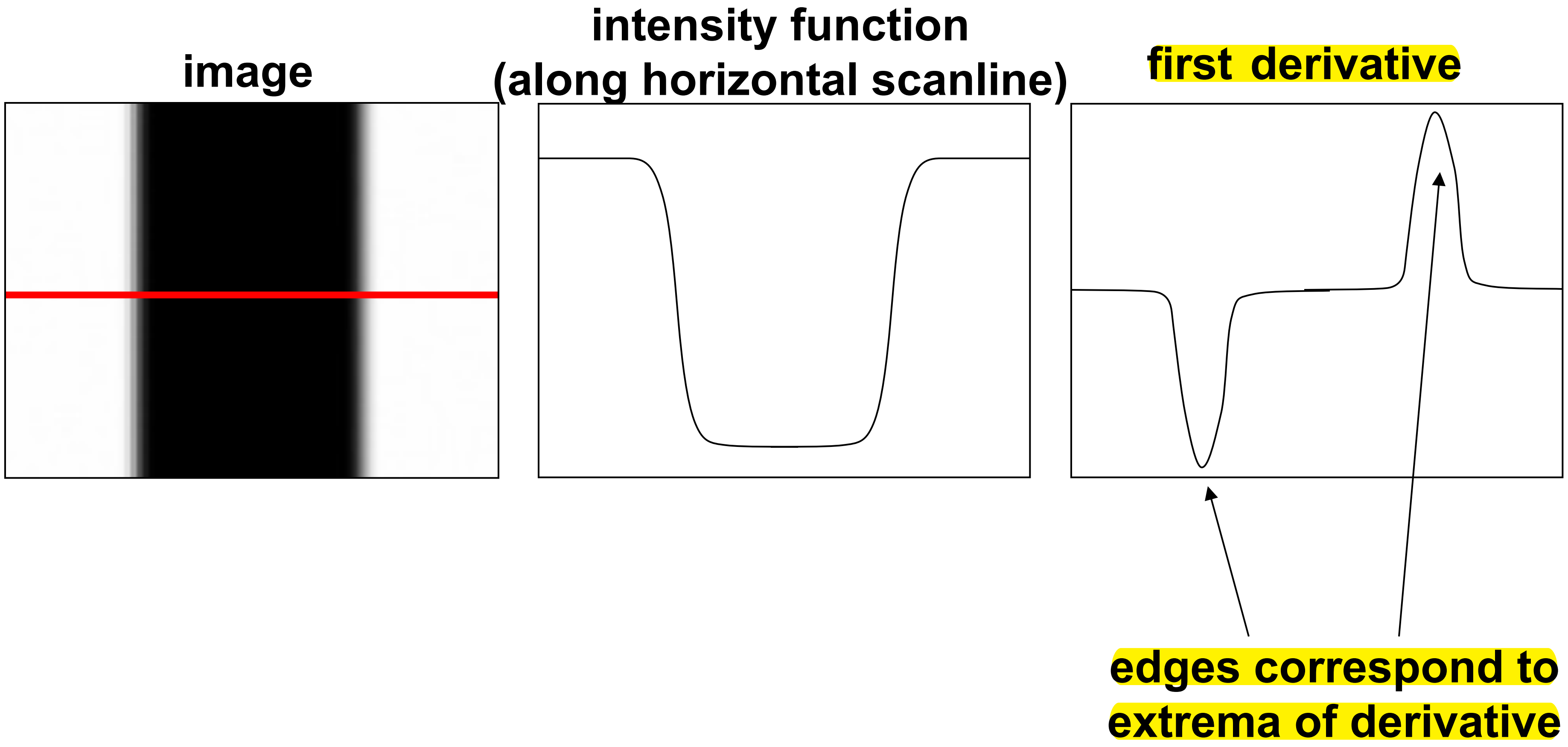
Edges/gradients and invariance



•Slide credit:
Kristen Grauman

Derivatives and edges

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



- Source: L. Lazebnik

Methods

- Using first order derivative: *Look for extrema*
  - Sobel operator computes an approximation of the gradient of the image intensity function
  - Prewitt, Roberts,...
  - Derivative of Gaussian
- Using second order derivative: *Look for zero-crossings*
  - Laplacian : isotropic
  - Second directional derivative
  - LOG/DOG (Laplace of Gaussian/Difference of Gaussians)

$$\nabla^2 \mathbf{I}$$

Common edge detectors

- Basic gradient edge detectors
  - **Sobel**, Prewitt, Gaussian derivative …
- **LoG/Marr** edge detector
- **Canny** edge detector

Gradient Edge Detectors

approximation for gradients of image intensity function

$$\textbf{Prewitt:} \quad M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \; ; \quad M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\textbf{Sobel:} \quad M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \; ; \quad \boxed{M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}$$

$$\textbf{Roberts:} \quad M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \; ; \quad M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

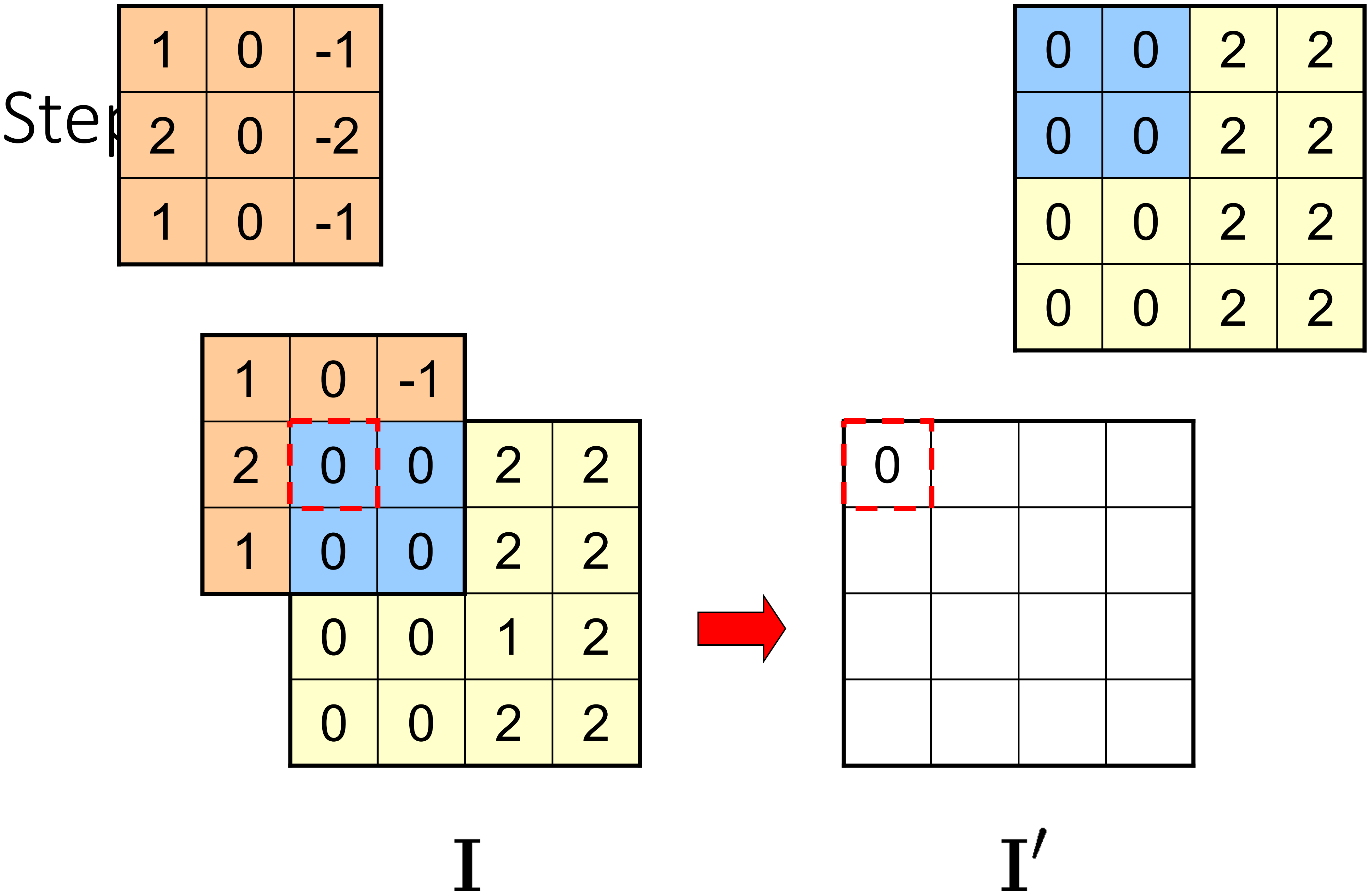
Eg: Convolve an image with a Sobel kernel

$$\mathbf{K} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

• Rotate

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

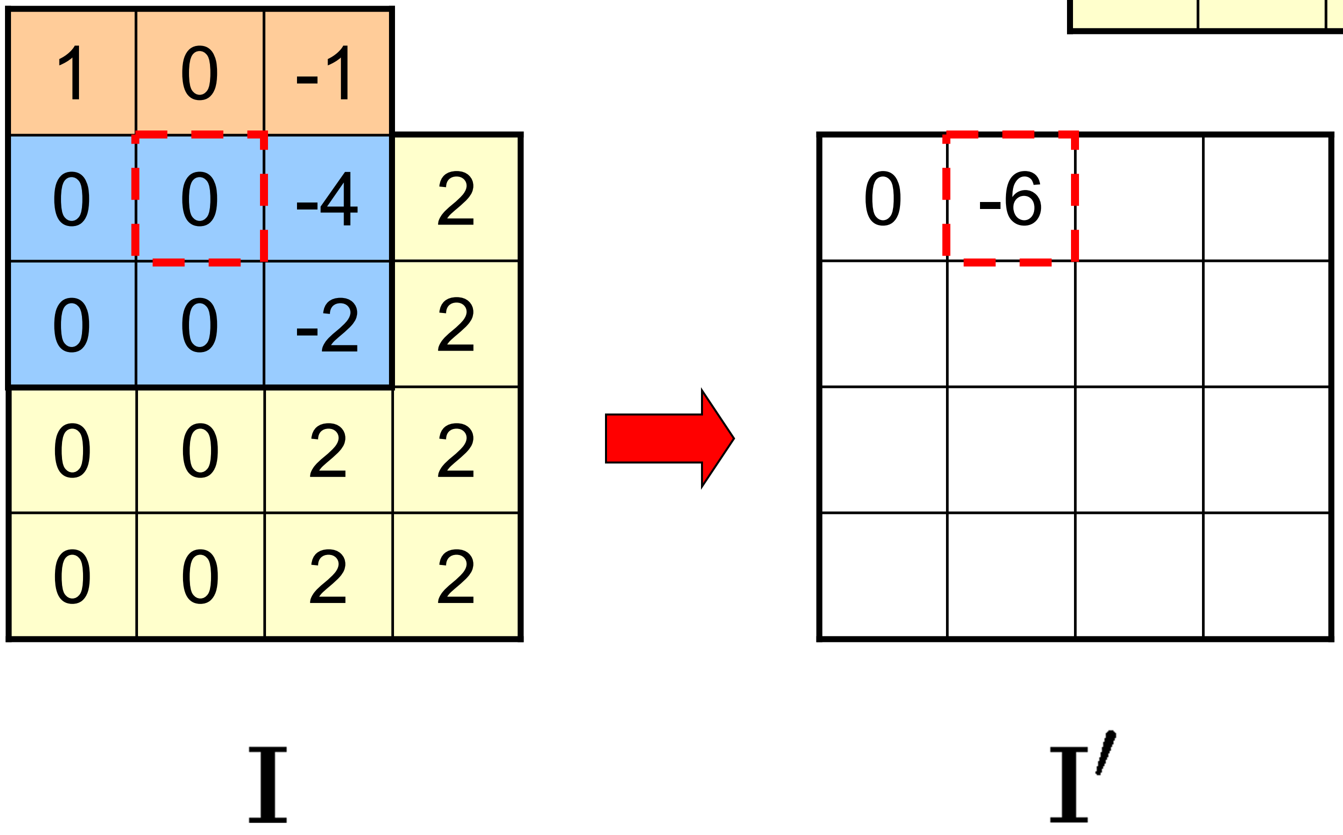
$$\mathbf{I} \quad \begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{bmatrix}$$


Step
1 0 -1
2 0 -2
1 0 -1
0 0 2 2
0 0 2 2
0 0 2 2
0 0 2 2
1 0 -1
2 0 0 2 2
1 0 0 2 2
0 0 1 2
0 0 2 2
0
$\mathbf{I}$
$\mathbf{I}'$

Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2



Step

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

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

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

$\mathbf{I}$

$\Rightarrow$

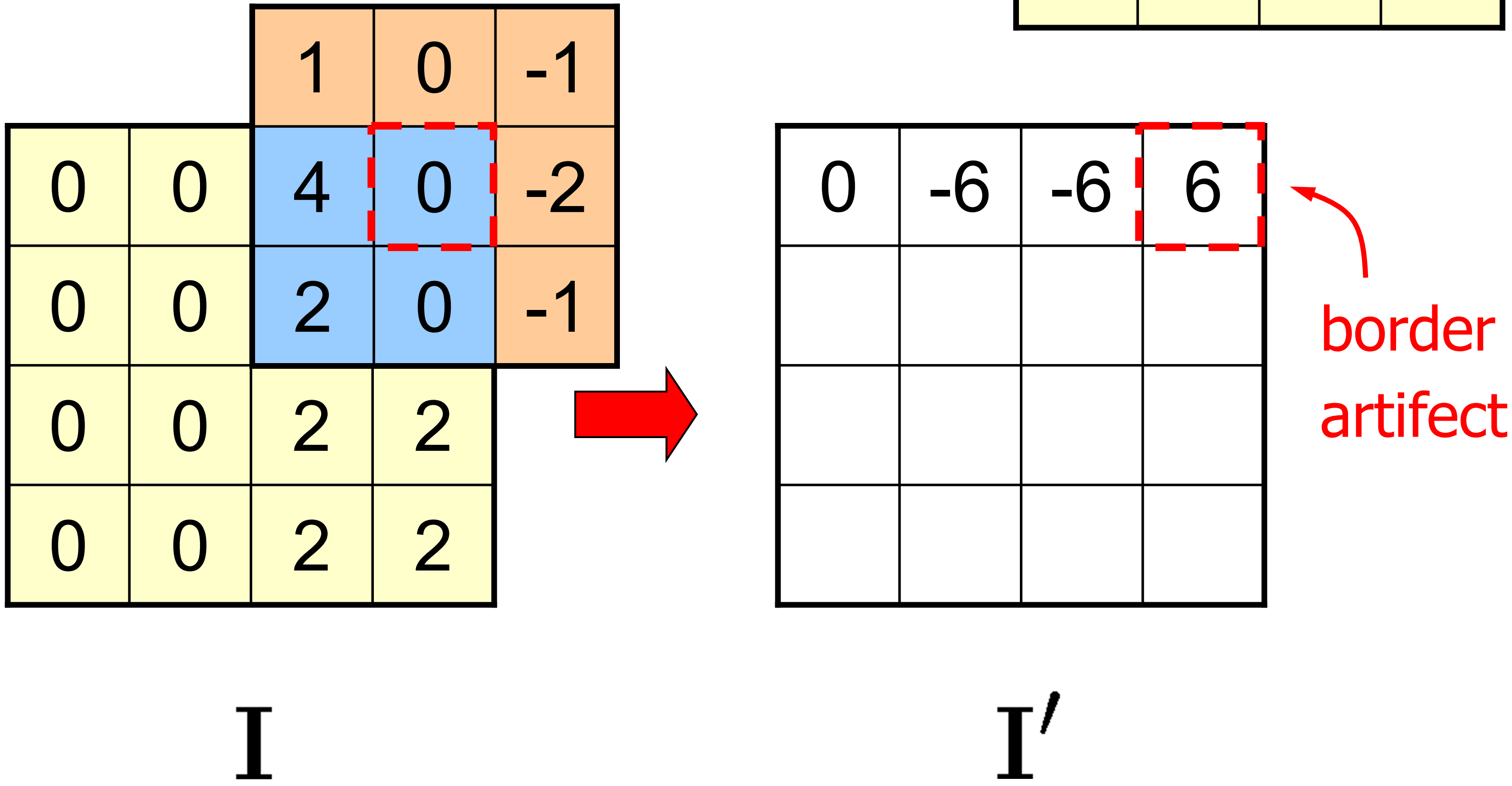
0	-6	-6	

$\mathbf{I}'$

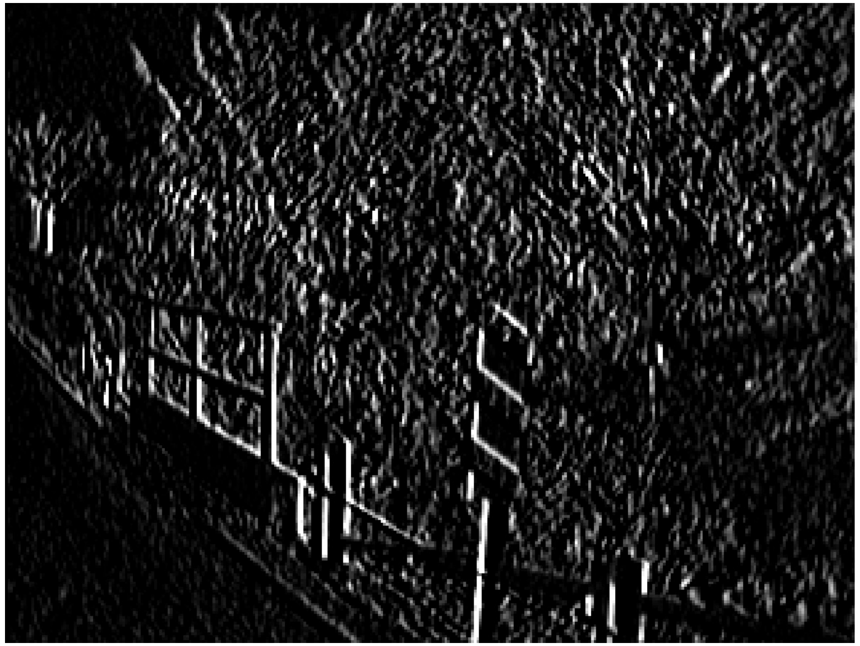
Ste

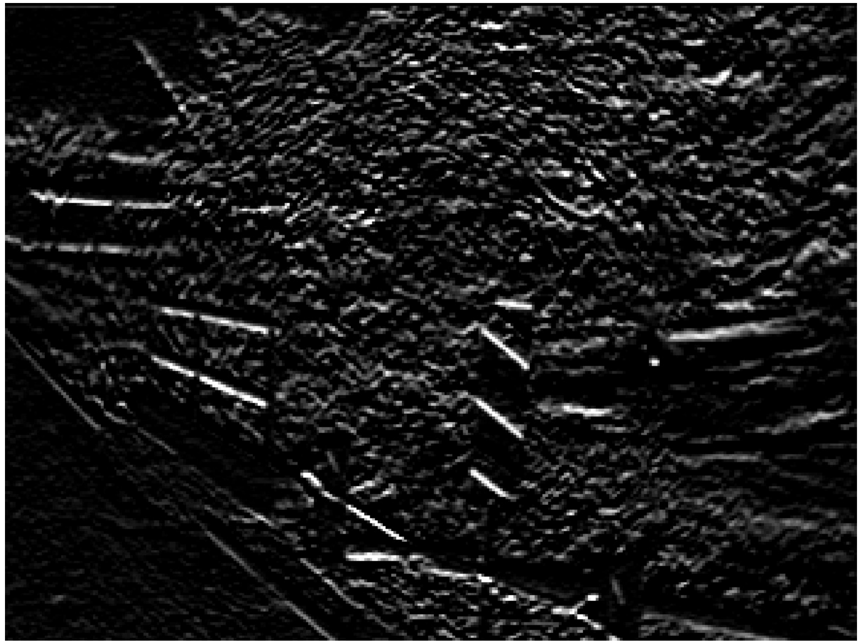
1	0	-1
2	0	-2
1	0	-1

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2



Sobel edge detection: example





•Vertical edges •Horizontal edges

Derivatives with convolution

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

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

For discrete data, we can approximate it using finite differences:

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

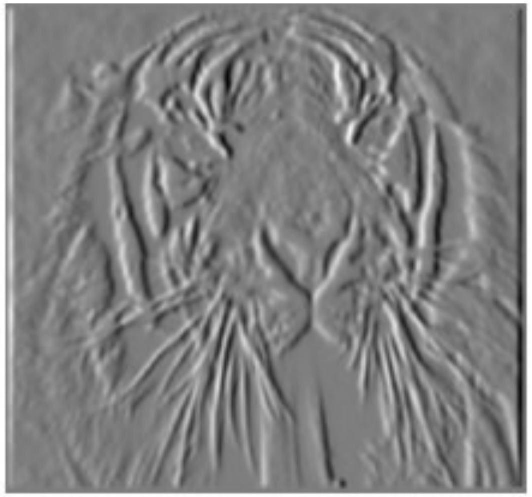
To implement above as convolution, what would be the associated filter?

•Slide credit: Kristen Grauman

Partial derivatives of an image

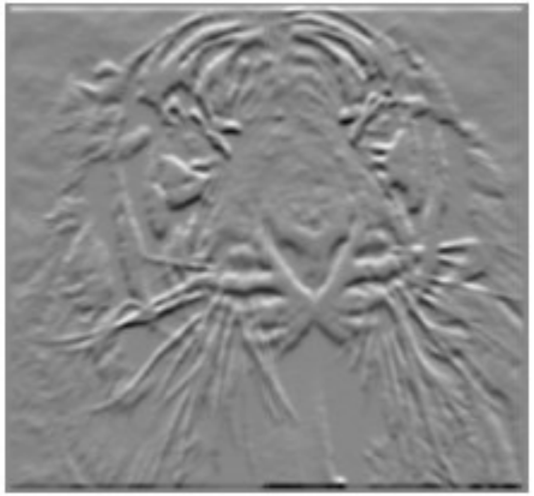
$$\frac{\partial f(x,y)}{\partial x}$$

horizontal



1	-1

vertical



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

-1
1

or

1
-1

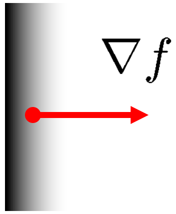
- Which shows changes with respect to x?
  - (showing filters for correlation)

Image gradient

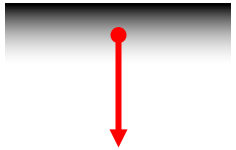
The gradient of an image:

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

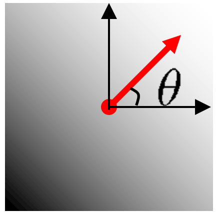
The gradient points in the direction of most rapid change in intensity



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



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



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

- The **gradient direction** (orientation of edge normal) is given by:

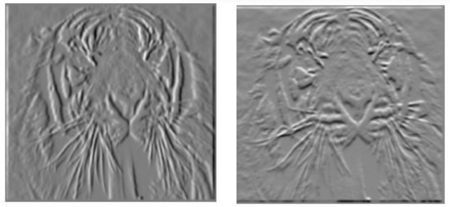
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

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

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

$$\text{Gradient Magnitude} = \sqrt{(G_x)^2 + (G_y)^2}$$

$$\text{Gradient Direction} = \text{arctan2}(G_y, G_x)$$

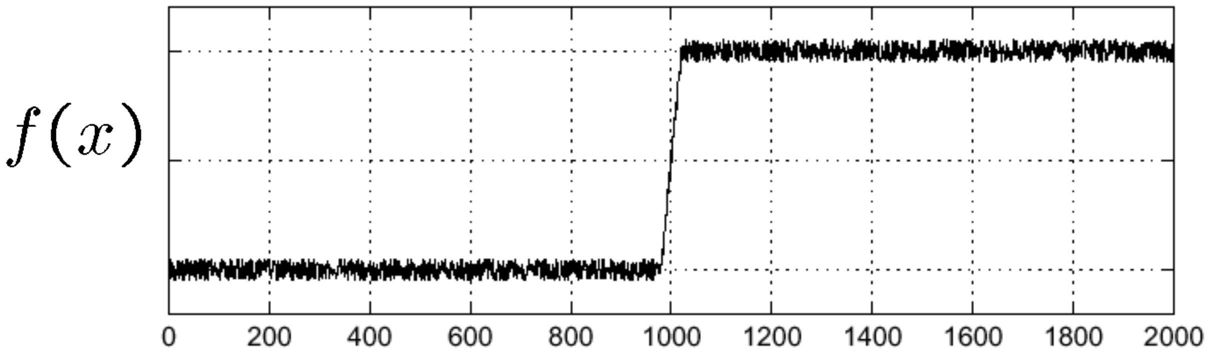


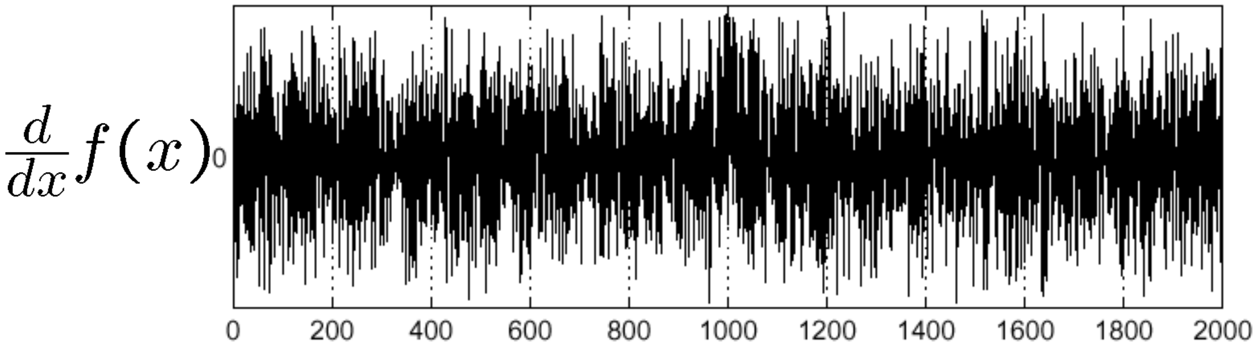
•Slide credit Steve Seitz

Effects of noise

Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal





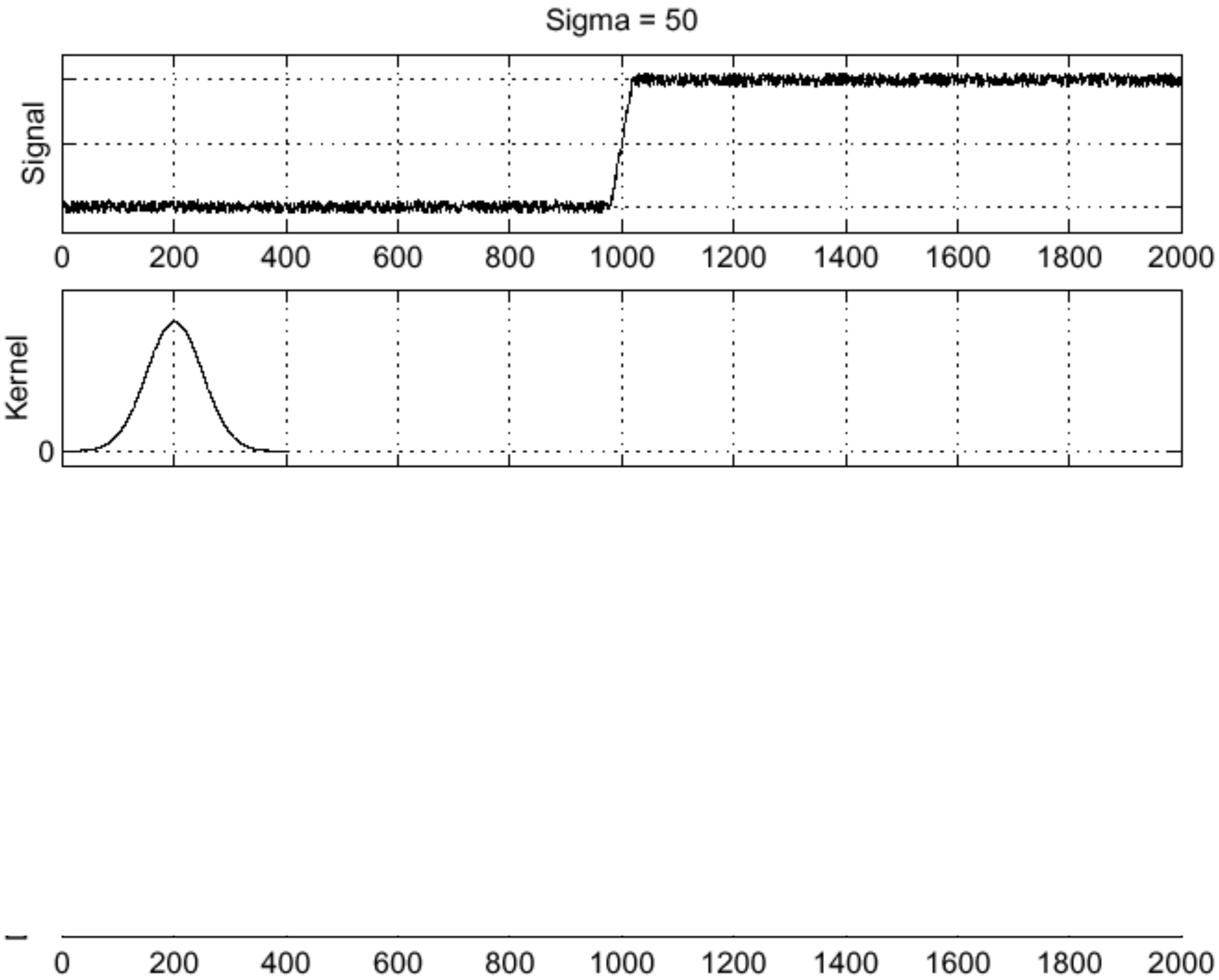
- Where is the edge?

•Slide credit Steve Seitz

Solution: smooth first

f

smooth kernel h



- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

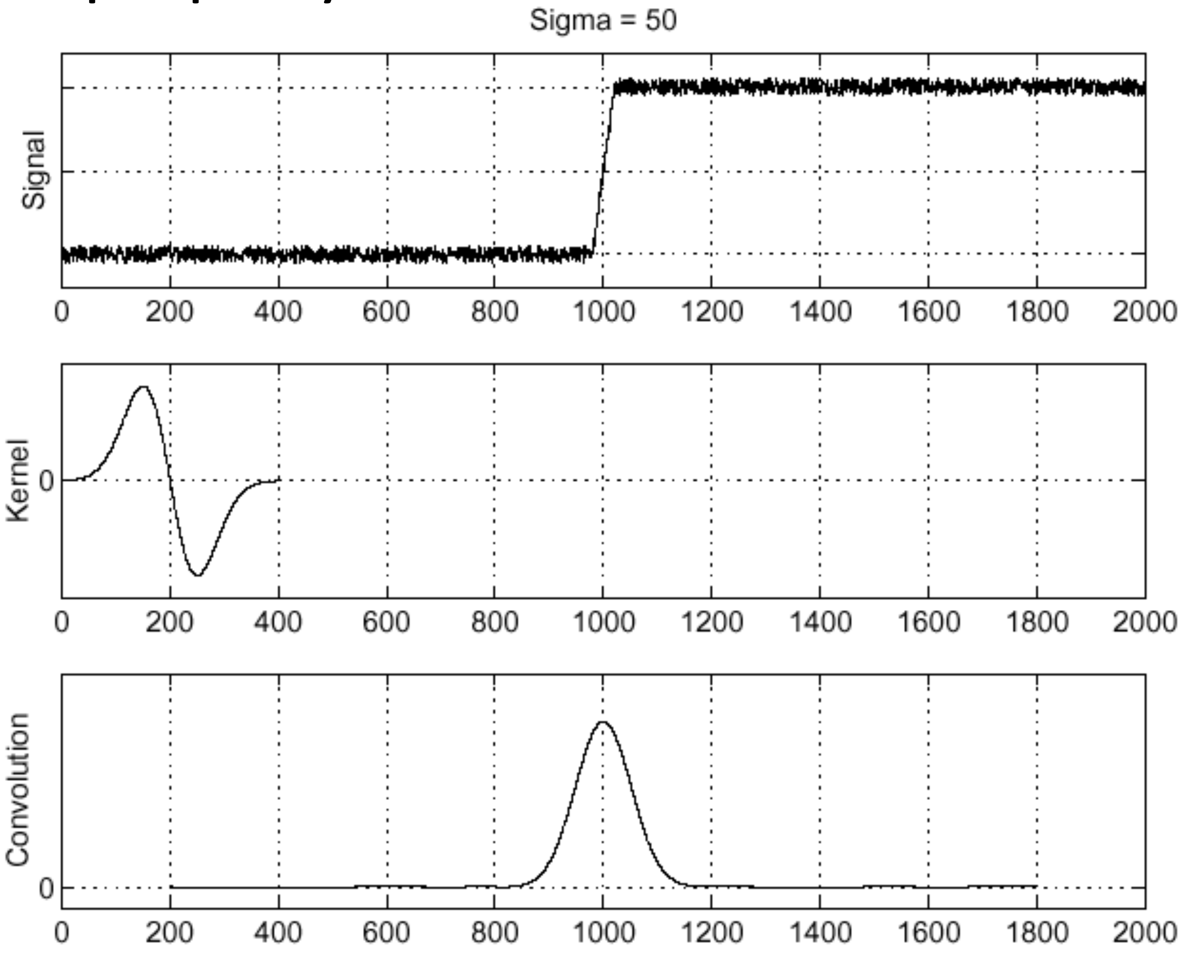
$$\boxed{\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f}$$

Differentiation property of convolution.

f

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

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



first-order derivative

•Slide credit Steve Seitz

Derivative of Gaussian filters

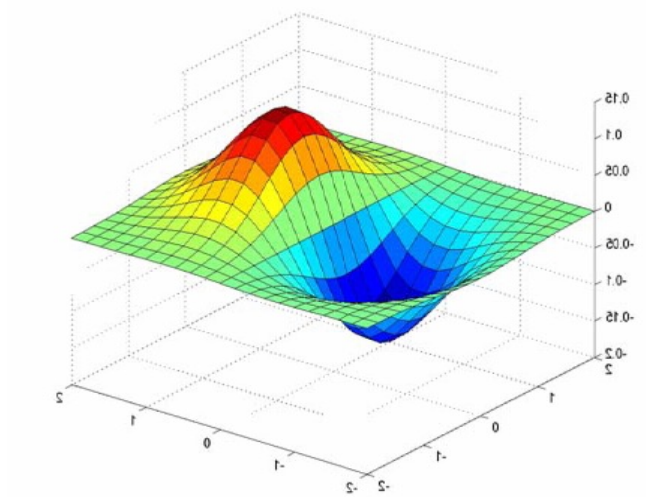
g: Gaussian smooth filter
h: vertical edge detector (appxi. of vertical derivatives)

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

according to the property of correlation operation

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

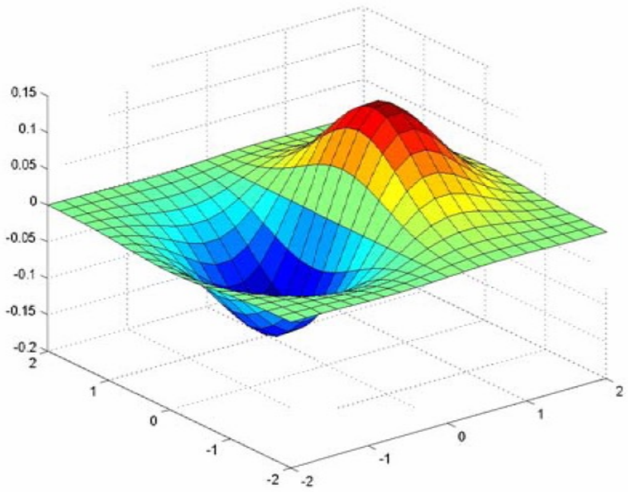
$$\boxed{\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f}$$

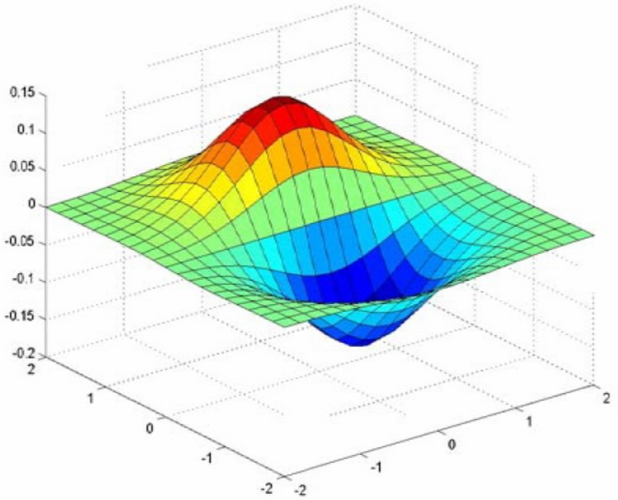


partial der. of x = h, the vertical edge detector
h = g, the Gaussain smooth filter
f = I, the original image

Derivative of Gaussian filters

[1, -1].T

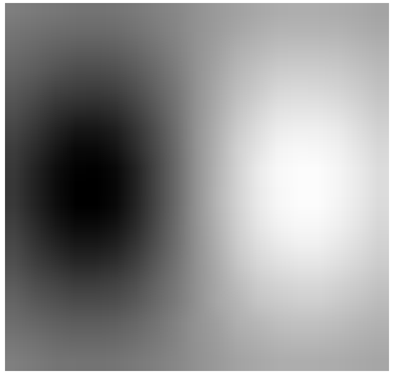


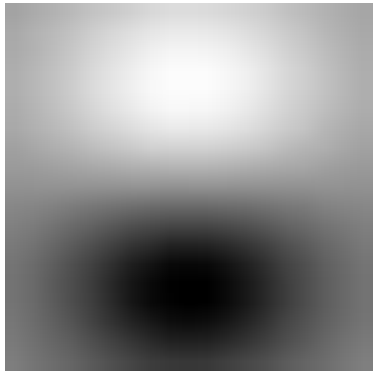


[1, -1]

•*x*-direction

•*y*-direction





•Source: L. Lazebnik

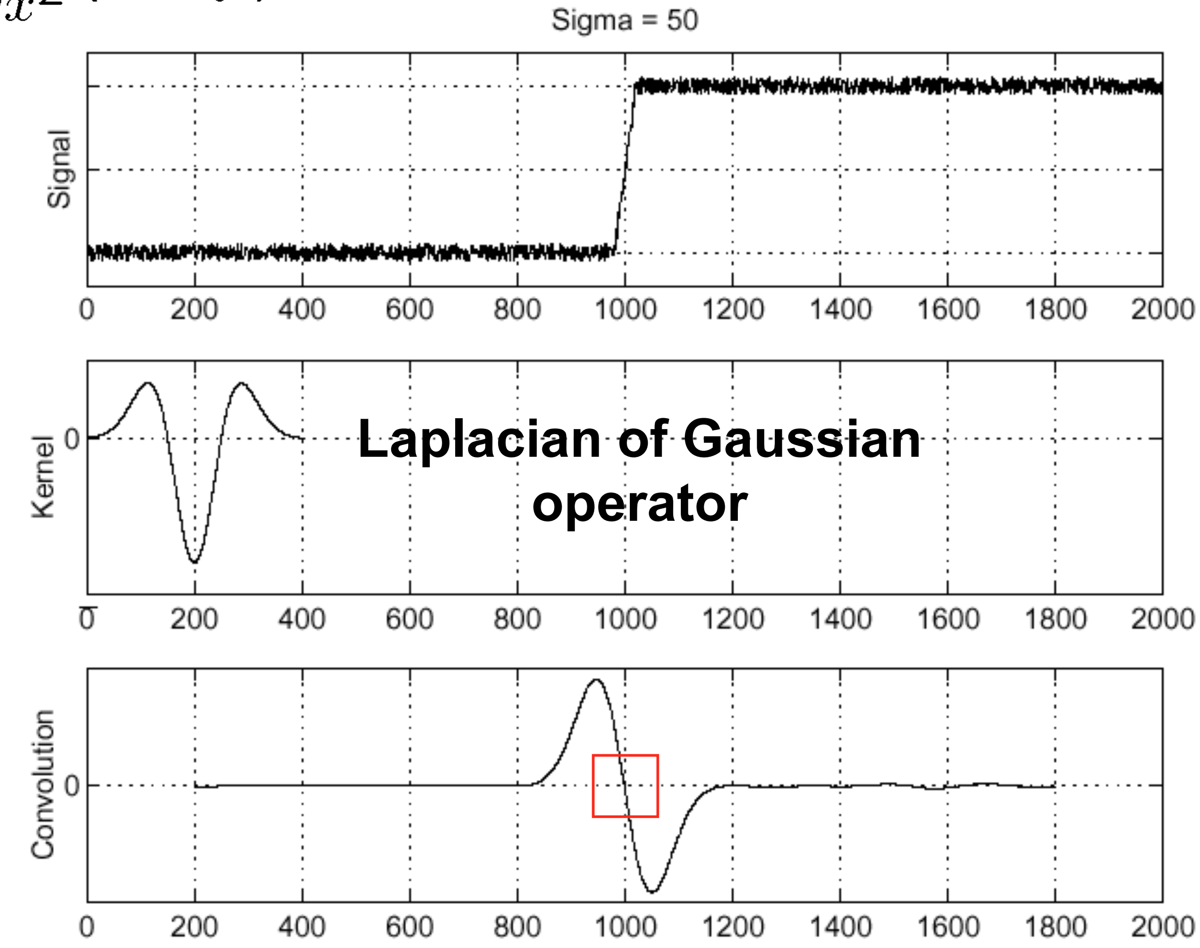
Laplacian of Gaussian

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

$$f$$

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

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



second-order derivative

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

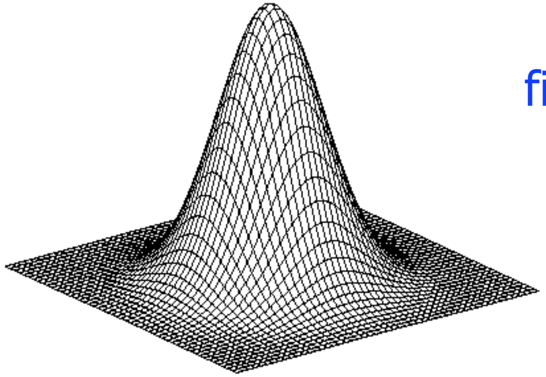
•Slide credit: Steve Seitz

2D edge detection filters

first-order derivative of 2D Gaussian

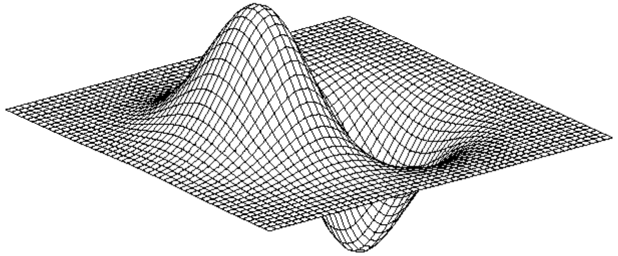
second-order of ...

Laplacian of Gaussian



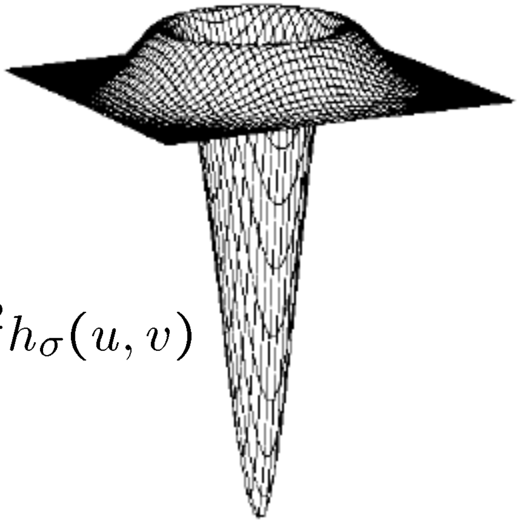
Gaussian

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



derivative of Gaussian

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



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

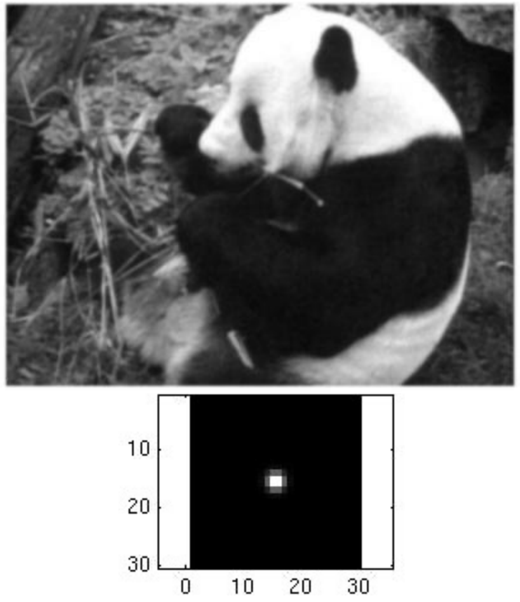
- ∇^2 is the Laplacian operator:

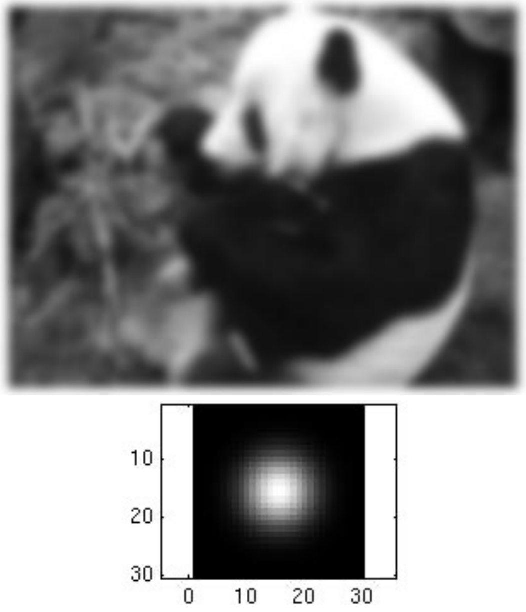
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

•Slide credit: Steve Seitz

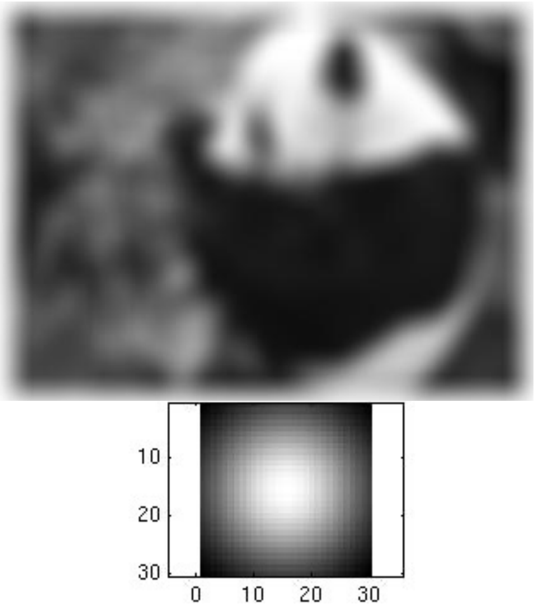
Smoothing with a Gaussian

- Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



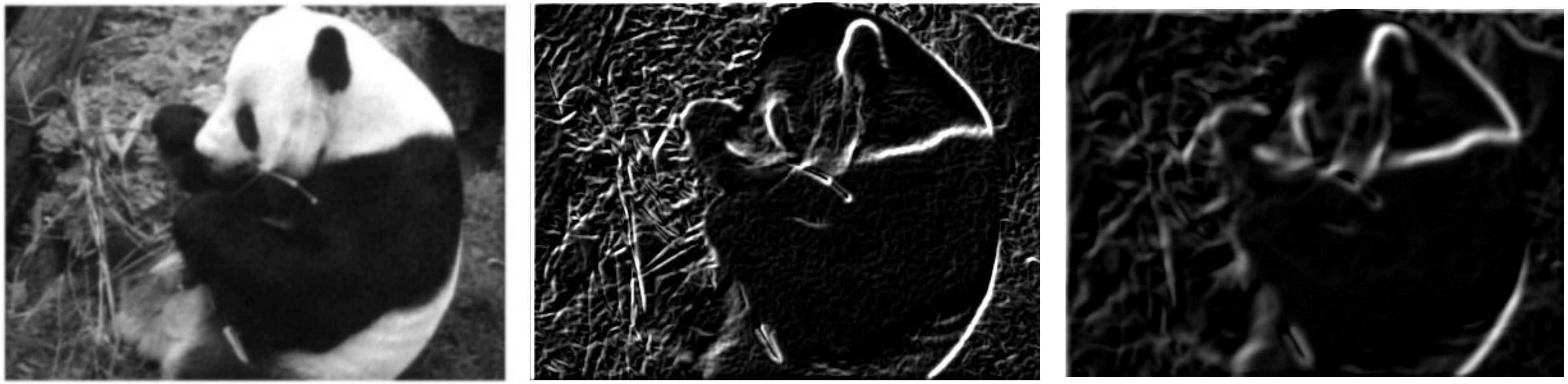


•…



•Slide credit: Kristen Grauman

Effect of σ on derivatives



σ = 1 pixel

σ = 3 pixels

- The apparent structures differ depending on Gaussian's scale parameter.

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

•Slide credit:
Kristen Grauman

Gradients -> edges

Primary edge detection steps:

1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization

   Determine which local maxima from filter output are actually edges vs. noise

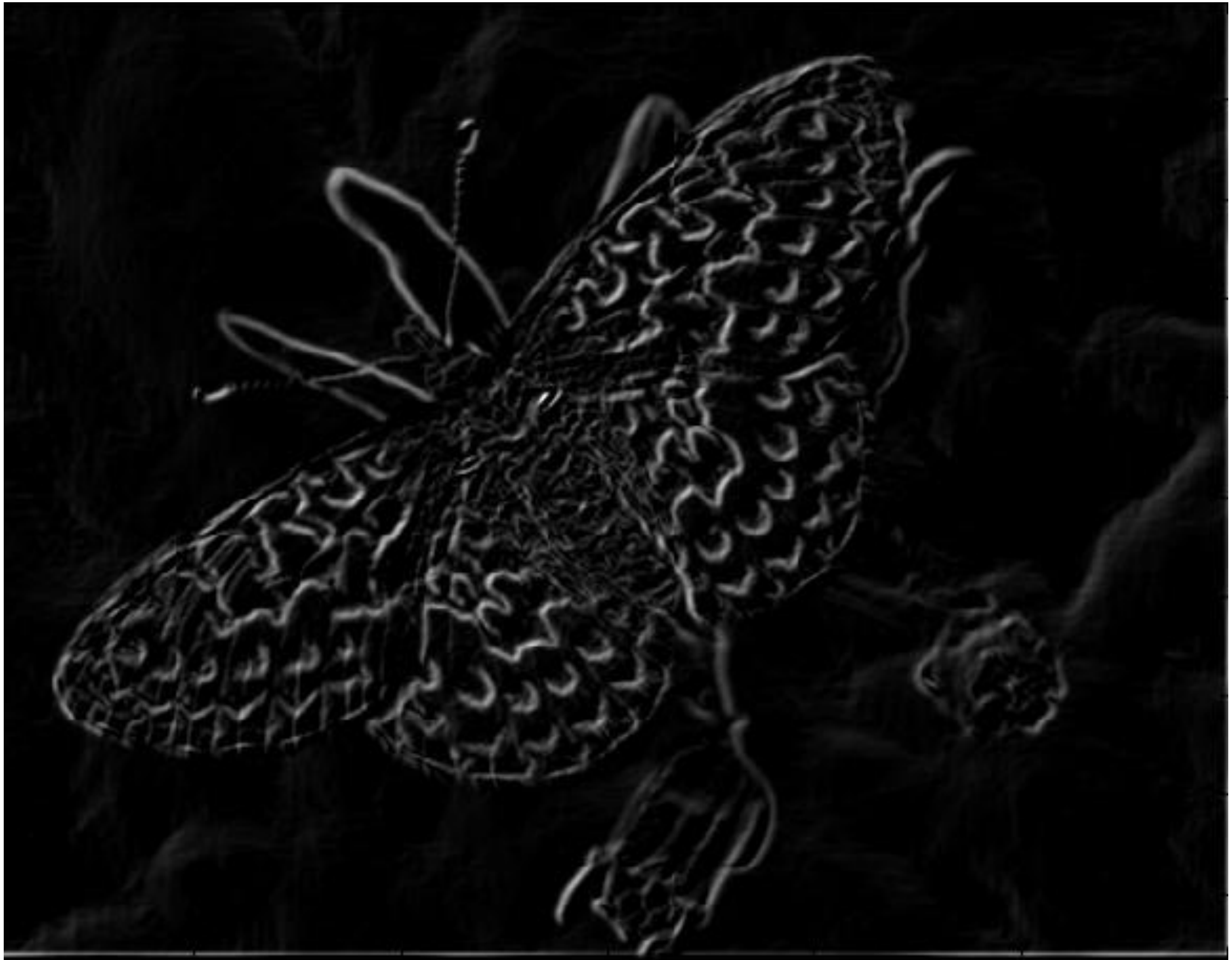
   - Threshold, Thin

Thresholding

- Choose a threshold value t
- Set any pixels less than t to zero (off)
- Set any pixels greater than or equal to t to one (on)

Original image

Gradient magnitude image



Thresholding gradient with a lower threshold

more edges/noises

Thresholding gradient with a higher threshold

less edges/noises

Canny edge detector

- Filter image with derivative of Gaussian [filter image with Gaussian filter first followed by finding the derivative of an image]
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin wide “ridges” down to single pixel width
- **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

- Python Opencv *cv2.Canny(image, T_lower, T_upper)*

1. Smoothing: Gaussian filter to reduce noise and remove small-scale details
2. Gaudient Calculation: Sobel/Prewitt operators for magnitude and direction
3. Non-maximum suppression: suppress all gradient values except for the local maxima. help to produce thin, continuous edges
4. Edge tracking by Hysteresis: thresholding hte gradient magnitude. Above the high threshold - strong edges, between low and high thresholds - weak edges. help to reduce noise and produce continuous edges

The Canny edge detector

original image (Lena)

Slide credit: Steve Seitz

The Canny edge detector

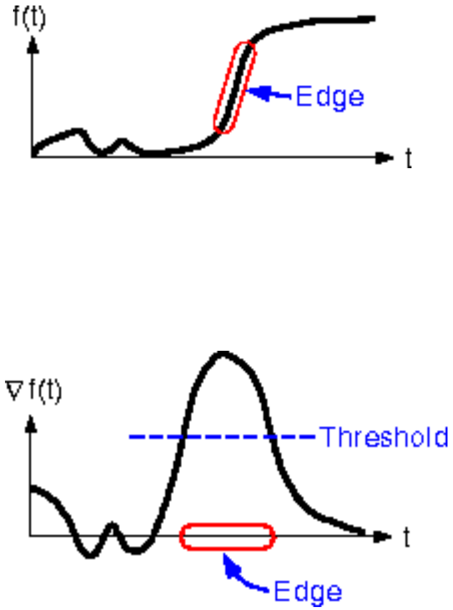
- norm of the gradient

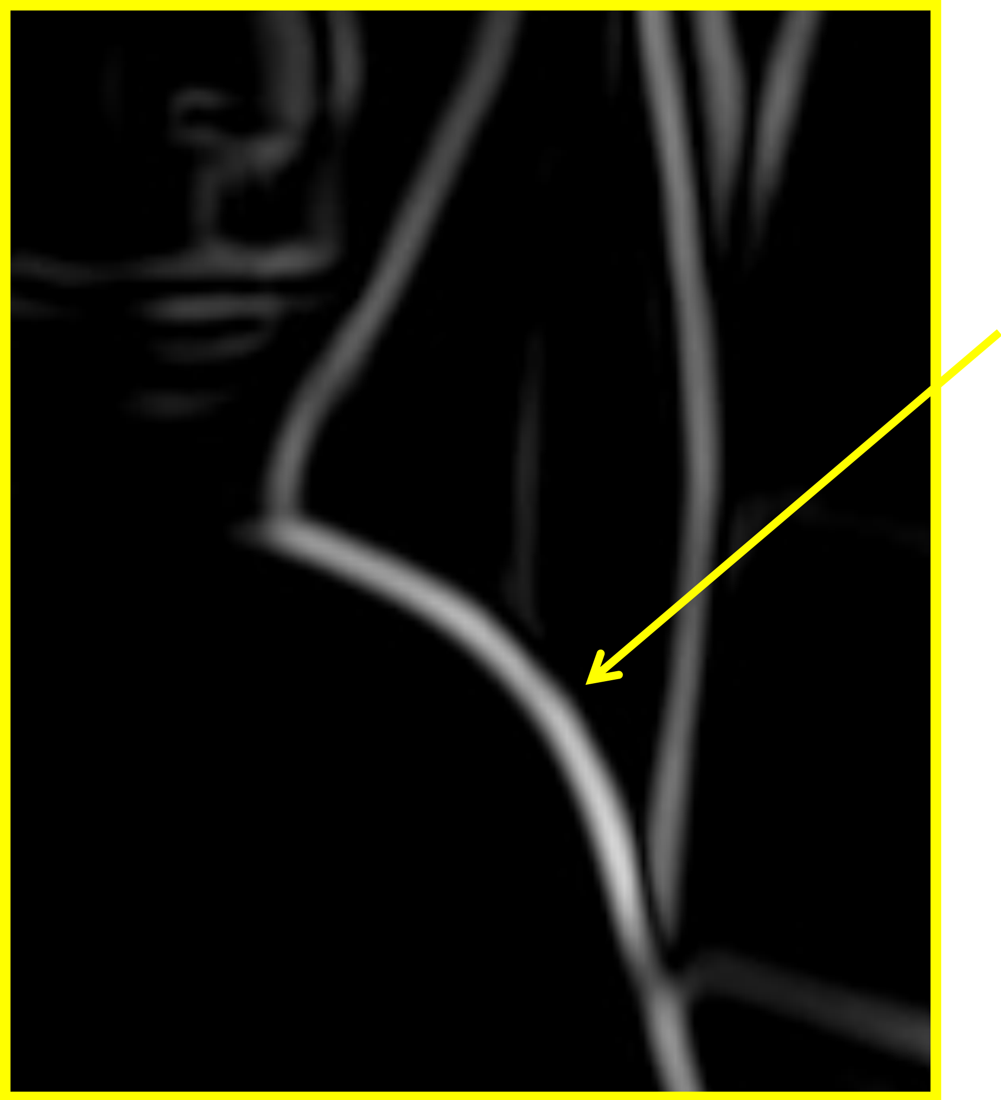
magnitude

The Canny edge detector

- thresholding

The Canny edge detector

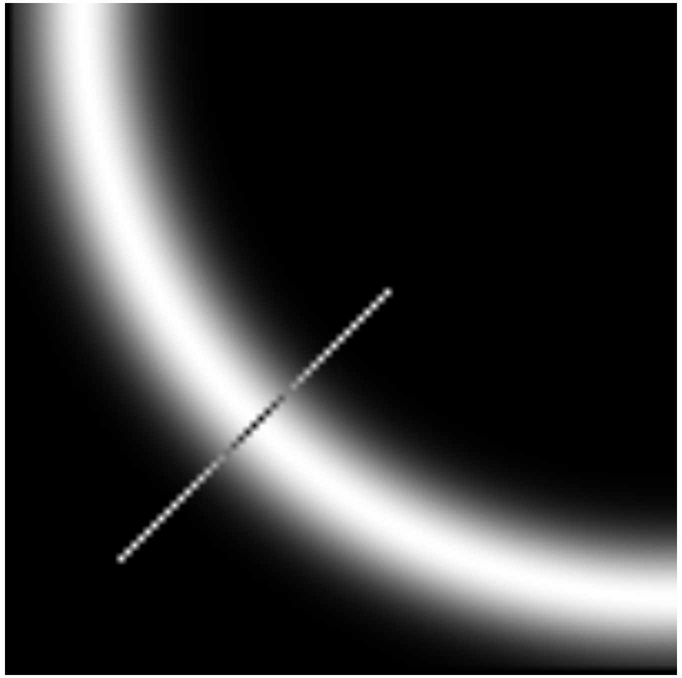


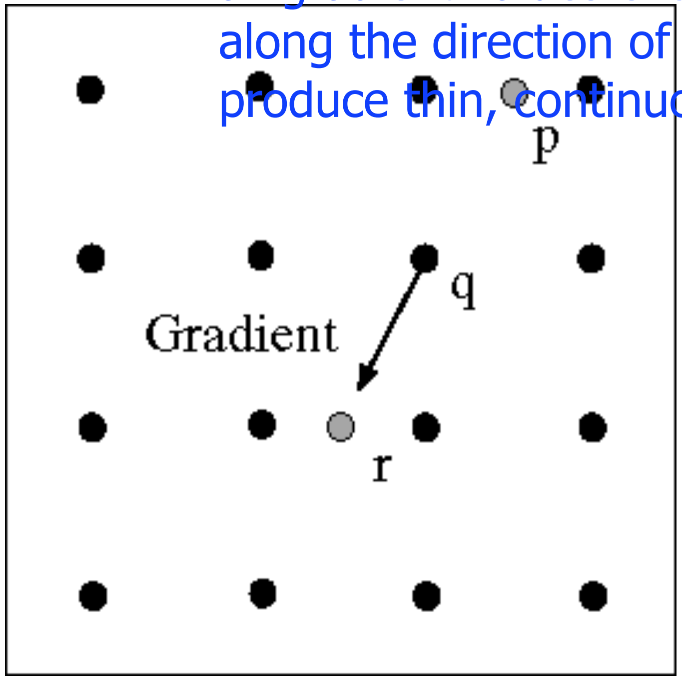


- How to turn these thick regions of the gradient into curves?

Non-maximum suppression

After calculating the gradient magnitude and direction, non-maximum suppression is applied to thin the edges. This step involves suppressing all gradient values except for the local maxima along the direction of the gradient. It helps to produce thin, continuous edges in the output.





Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

The Canny edge detector

- Problem: pixels along this edge didn't survive the thresholding

- thinning
- (non-maximum suppression)

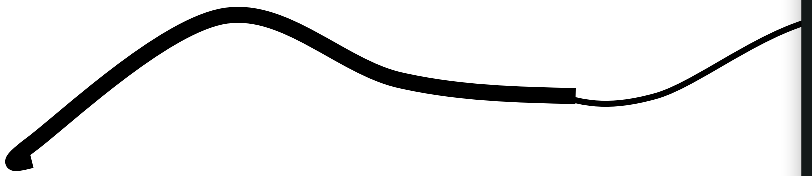
Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

•Credit: James Hays

Hysteresis thresholding

- Use a high threshold to start edge curves, and a low threshold to continue them.



1. **Thresholding:**
Apply two thresholds to the gradient magnitude image:
   - High threshold (T_{high}): Pixels with gradient magnitudes above this threshold are considered strong edges.
   - Low threshold (T_{low}): Pixels with gradient magnitudes between the low and high thresholds are considered weak edges.
2. **Edge Connectivity:**
Identify pixels in the gradient magnitude image that are connected to strong edges. This can be done using depth-first search (DFS), breadth-first search (BFS), or any other connectivity analysis method.
3. **Output Image:**
Create a binary edge map where strong edges and weak edges connected to strong edges are set to 1 (white), and all other pixels are set to 0 (black).

- Source: Steve Seitz

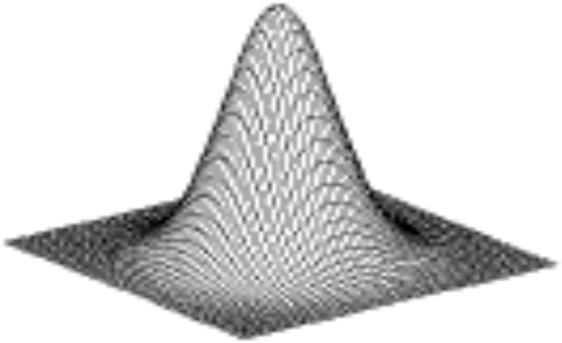
Final Canny Edges

•Credit: James Hays

Recap: Canny edge detector

- Filter image with **derivative of Gaussian**
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin wide “ridges” down to single pixel width
- **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



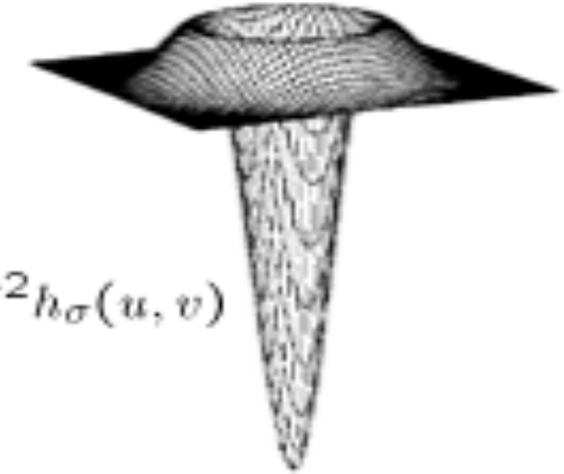
Gaussian

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

derivative of Gaussian

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

Laplacian of Gaussian

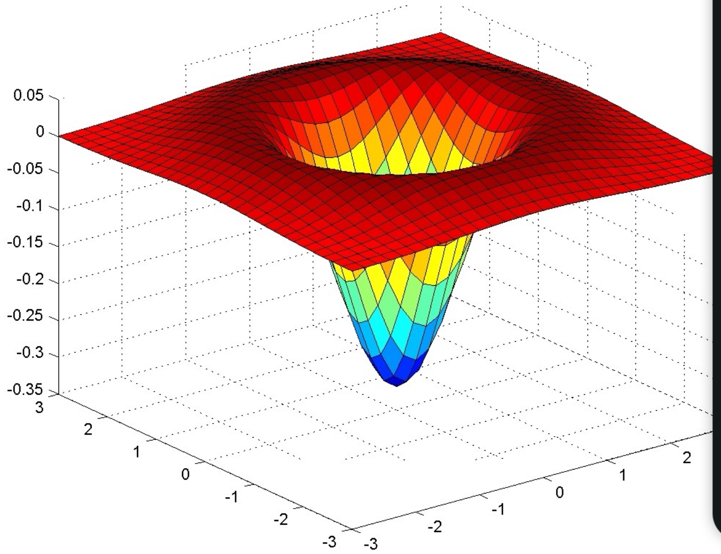


$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 **is the Laplacian operator:**

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

Laplacian of Gaussian (LoG):
Marr edge detector



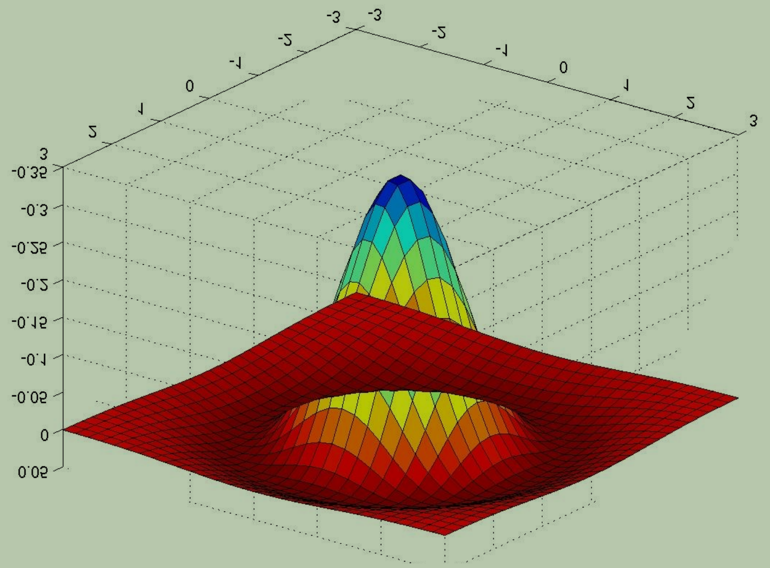
1. **Apply Gaussian Smoothing:**
Before applying the Laplacian operator, it's common to smooth the image using Gaussian blur to reduce noise and suppress high-frequency components. This helps to improve the accuracy of edge detection and reduces false detections.
2. **Compute Laplacian of Gaussian (LoG) Kernel:**
Construct a 2D Laplacian of Gaussian kernel. This involves multiplying the Laplacian operator with a 2D Gaussian function to create a filter that combines edge detection with smoothing.
3. **Convolution:**
Convolve the smoothed image with the LoG kernel. This is typically done using convolution, where each pixel in the image is replaced by a weighted sum of its neighbors according to the LoG kernel.
4. **Thresholding:**
After convolving the image with the LoG kernel, apply a threshold to the resulting image to identify edge pixels. Pixels with values above a certain threshold are considered part of an edge, while pixels below the threshold are discarded.
5. **Optional: Post-processing**
Depending on the application, you may perform additional post-processing steps such as edge thinning, edge linking, or noise reduction to refine the detected edges.

$$\boxed{LoG_\sigma = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{(x^2+y^2)}{2\sigma^2}}}$$

LoG kernel — Mexican hat

An example of an 11×11 mask approximating the Laplacian of Guassian with $\sigma^2 = 2$ is given below:

0	0	0	−1	−1	−2	−1	−1	0	0	0
0	0	−2	−4	−8	−9	−8	−4	−2	0	0
0	−2	−7	−15	−22	−23	−22	−15	−7	−2	0
−1	−4	−15	−24	−14	−1	−14	−24	−15	−4	−1
−1	−8	−22	−14	52	103	52	−14	−22	−8	−1
−2	−9	−23	−1	103	180	103	−1	−23	−9	−2
−1	−8	−22	−14	52	103	52	−14	−22	−8	−1
−1	−4	−15	−24	−14	−1	−14	−24	−15	−4	−1
0	−2	−7	−15	−22	−23	−22	−15	−7	−2	
0	0	−2	−4	−8	−9	−8	−4	−2	0	
0	0	0	−1	−1	−2	−1	−1	0	0	



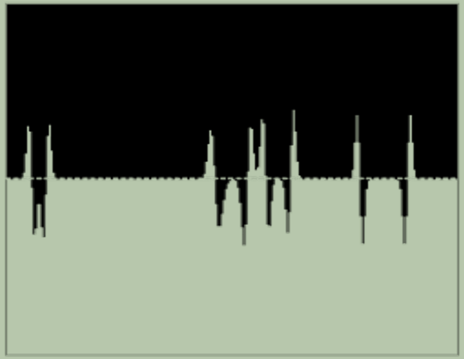
Detecting zero-crossings

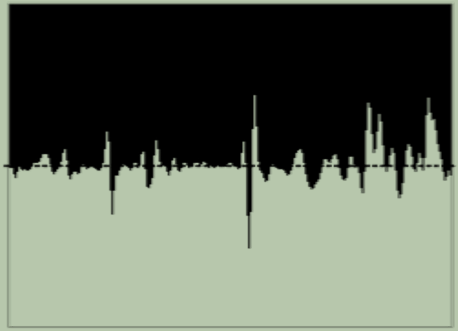
- The edge locations occur at points in the LoG-filtered image where the value passes through zero. i.e., where there is a sign change.

- The number of zero crossings is influenced by the size of the Gaussian; the greater the smoothing, the smaller the number of zero crossings.

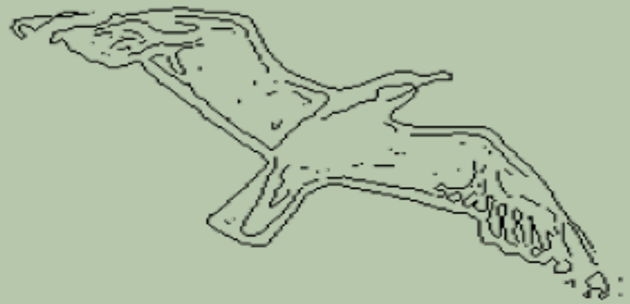
- A simple algorithm is to declare a pixel p to be zero crossing if any one of its 8-neighbors is of opposite sign. To restrict the detected zero crossings to the more significant ones, we can impose a threshold T.

Example: 2-D LoG filtering

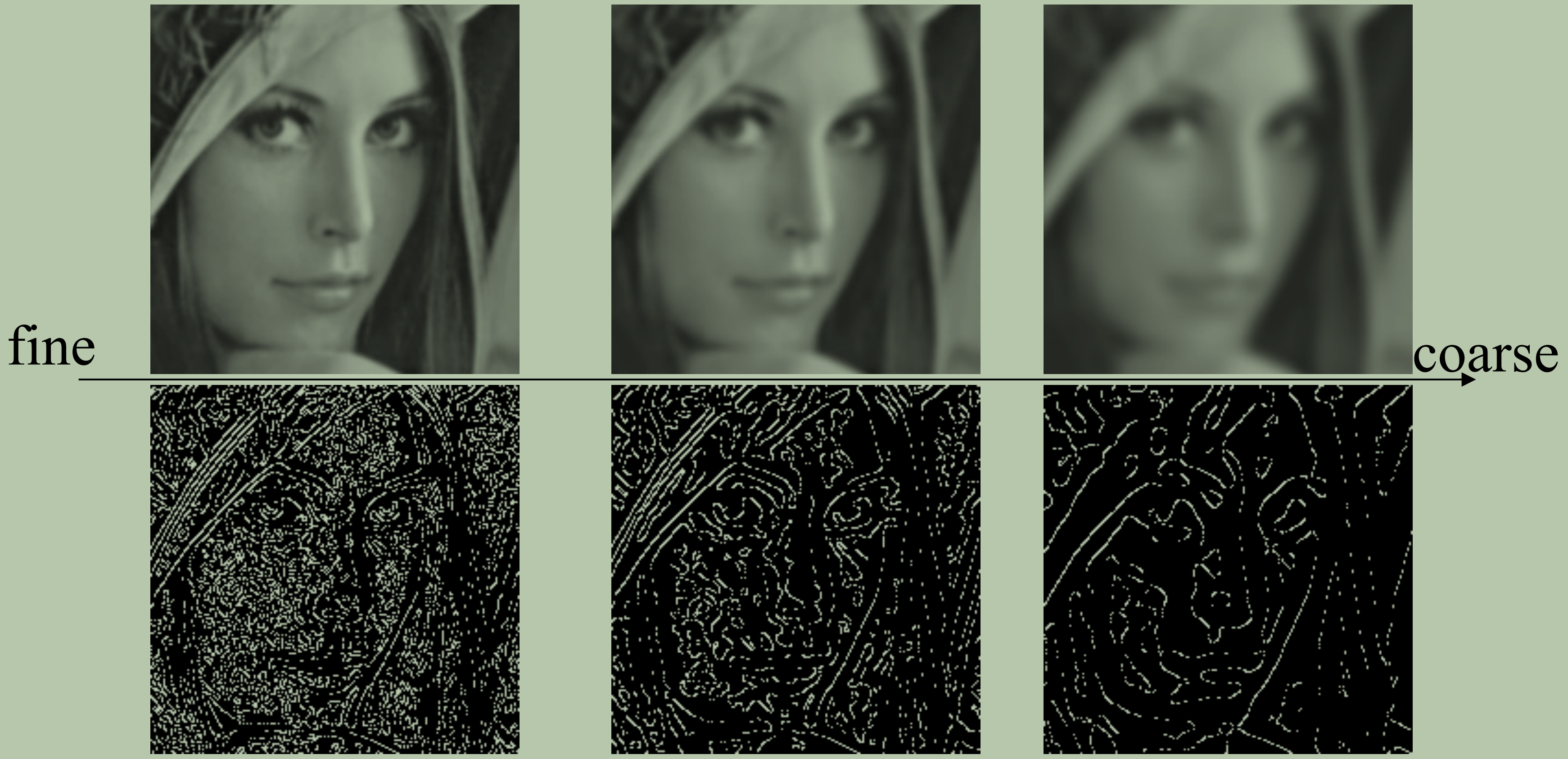




Example: LoG/Marr edge detector



Change of Scale (sigma in Gaussian/LoG)



the larger the sigma value is, mroe blurred the image will be, the less edges will be detected

96

Sobel vs. LoG vs. Canny Edge Detector

Sobel

LoG

Canny Edge Detector

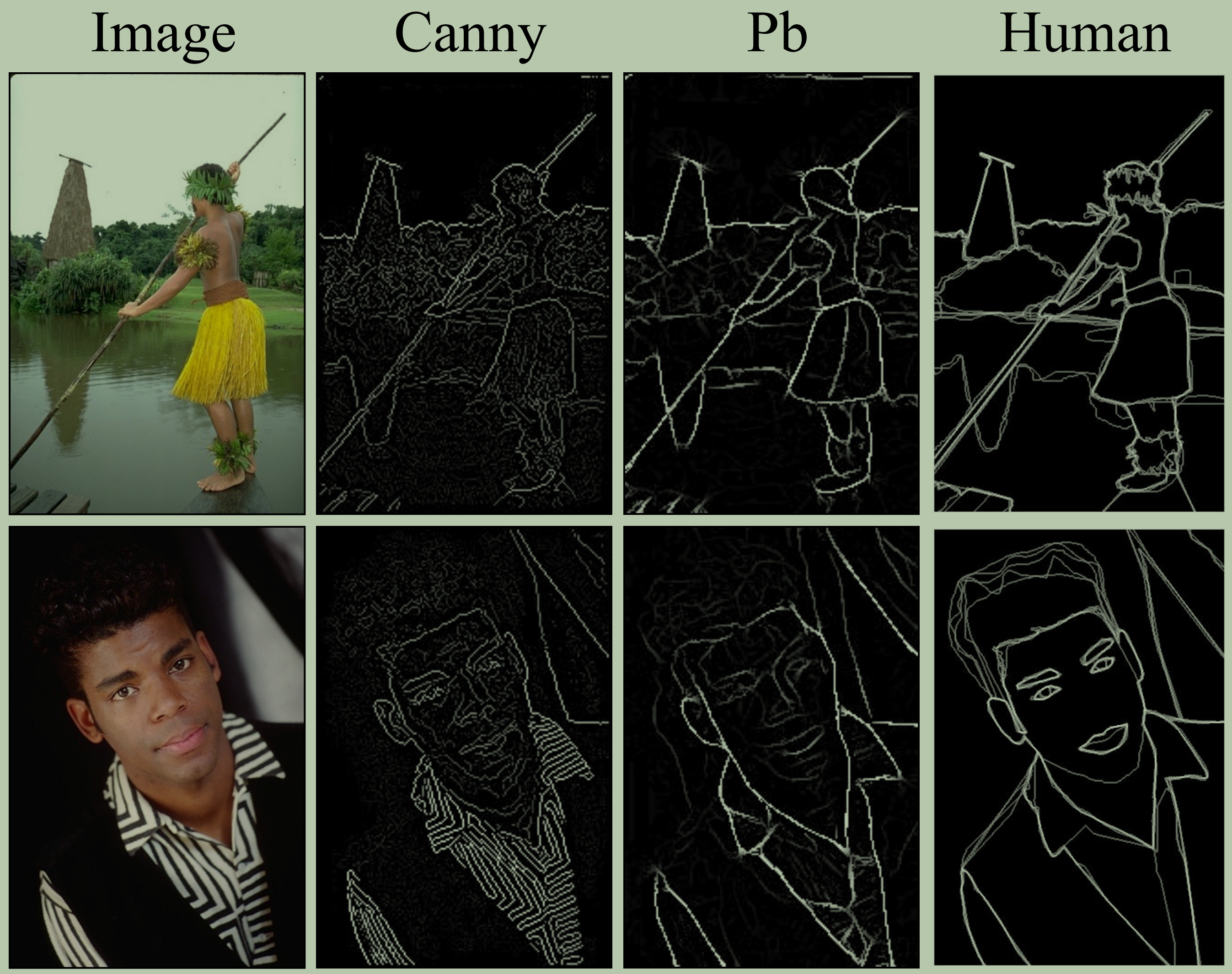
Other Edge Detectors

- Machine learning based edge detector.
- To learn how human label edges of natural images.

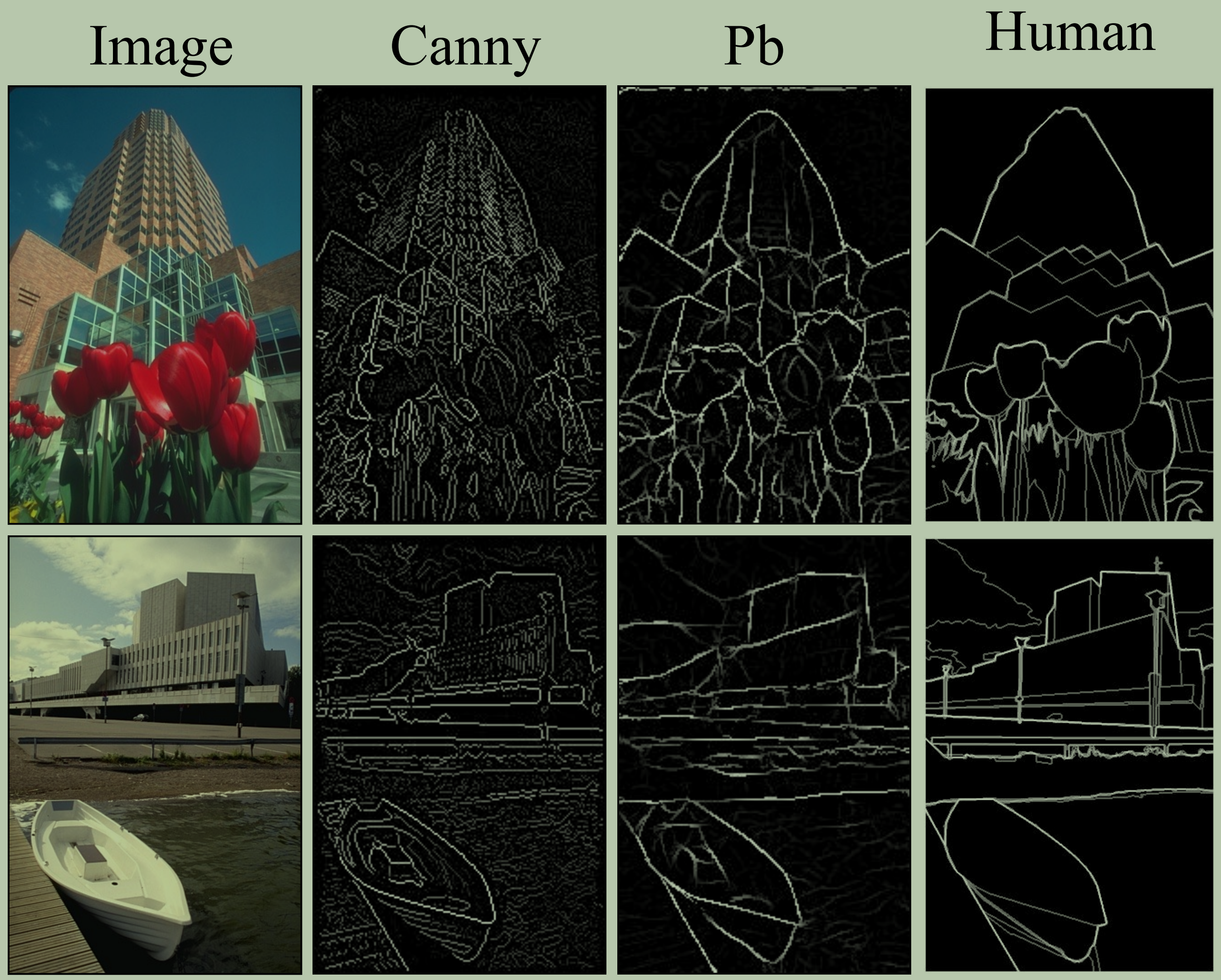
Reference:
Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues,
David R. Martin, Charless C. Fowlkes, and Jitendra Malik, TPAMI, 2004

Sample human-labeled training images and their corresponding edge maps

Learned edge detector: Pb edge I

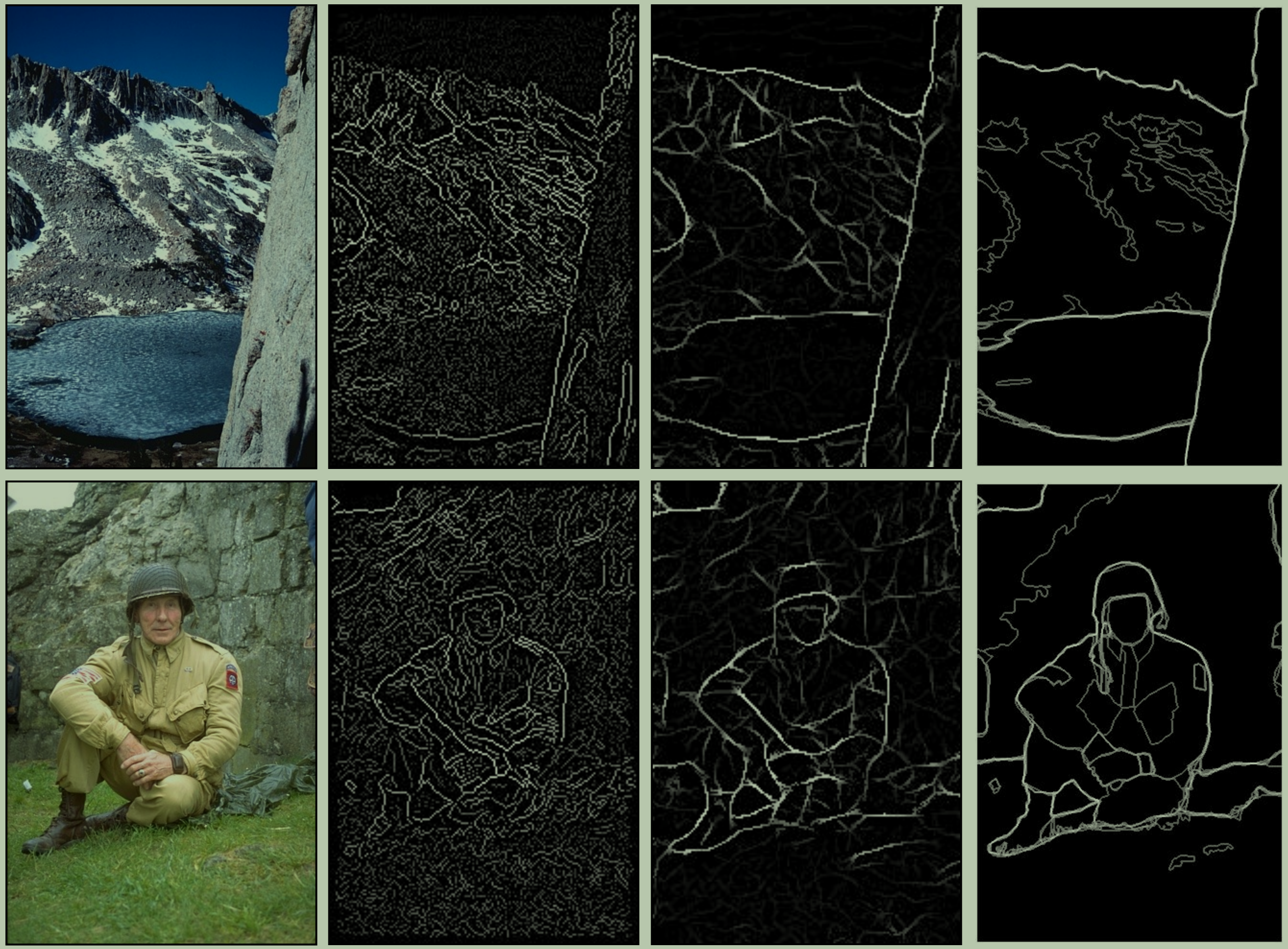


Pb result II



Pb result III

Image Canny Pb Human



Mask/filter properties

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove "high-frequency" components; "low-pass" filter
- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast

•Slide credit:
Kristen Grauman

Non-linear Filter

- Median Filter

- Bilinear Filter

Nonlinear Image Filtering

Median Filter (a nonlinear filter)

Median filter

- The median filter is a sliding-window based spatial filter.
- It replaces the value of the center pixel with the median of the intensity values in the neighborhood of that pixel.
- Median filtering is a nonlinear operation often used in image processing to reduce "salt and pepper" noise. A median filter is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges.
- **Median filters** are particularly effective in the presence of *impulse noise*, also called '***salt – and – pepper***' noise because of its appearance as white and black dots superimposed on an image.
- For every pixel, a 3x3 neighborhood with the pixel as center is considered. In **median filtering**, the value of the pixel is replaced by the median of the pixel values in the 3x3 neighborhood.

Questions

What is the value of the yellow box after 3x3 median filtering?

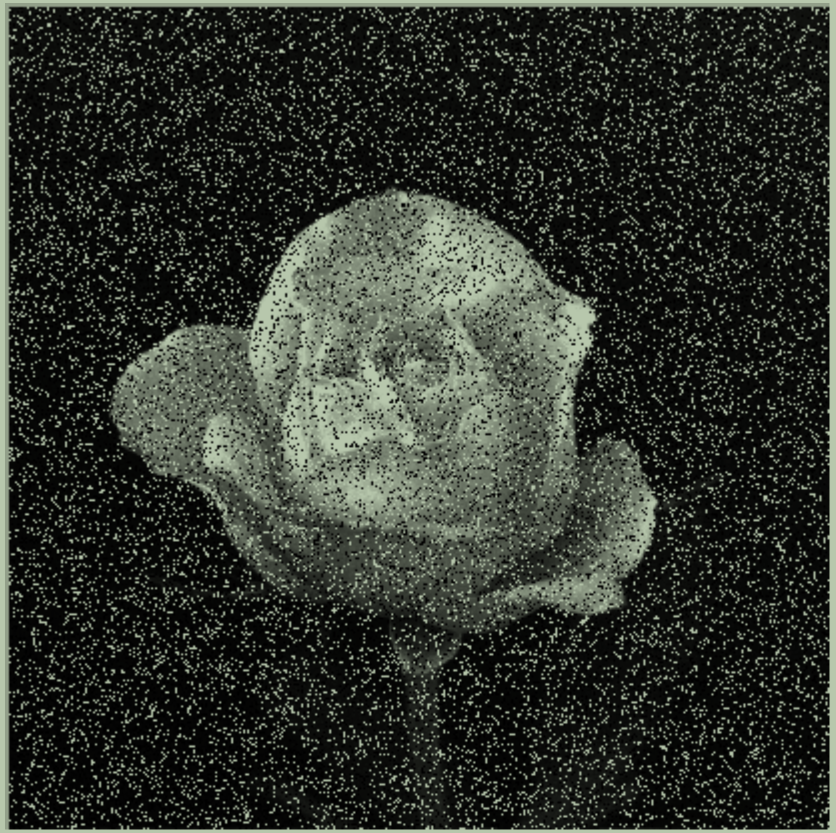
1	2	2	2	3
2	5	2	4	3
2	2	2	2	3
4	3	7	5	3
3	3	3	1	1

Questions

What is the value of the yellow box after 3x3 median filtering?

1	2	2	2	3
2	5	2	4	3
2	2	2 3	2	3
4	3	7	5	3
3	3	3	1	1

Example: Median filter result



Why median filter is non-linear

- Median(A(x)+B(x))!= Median(A(x)) + Median(B(x))

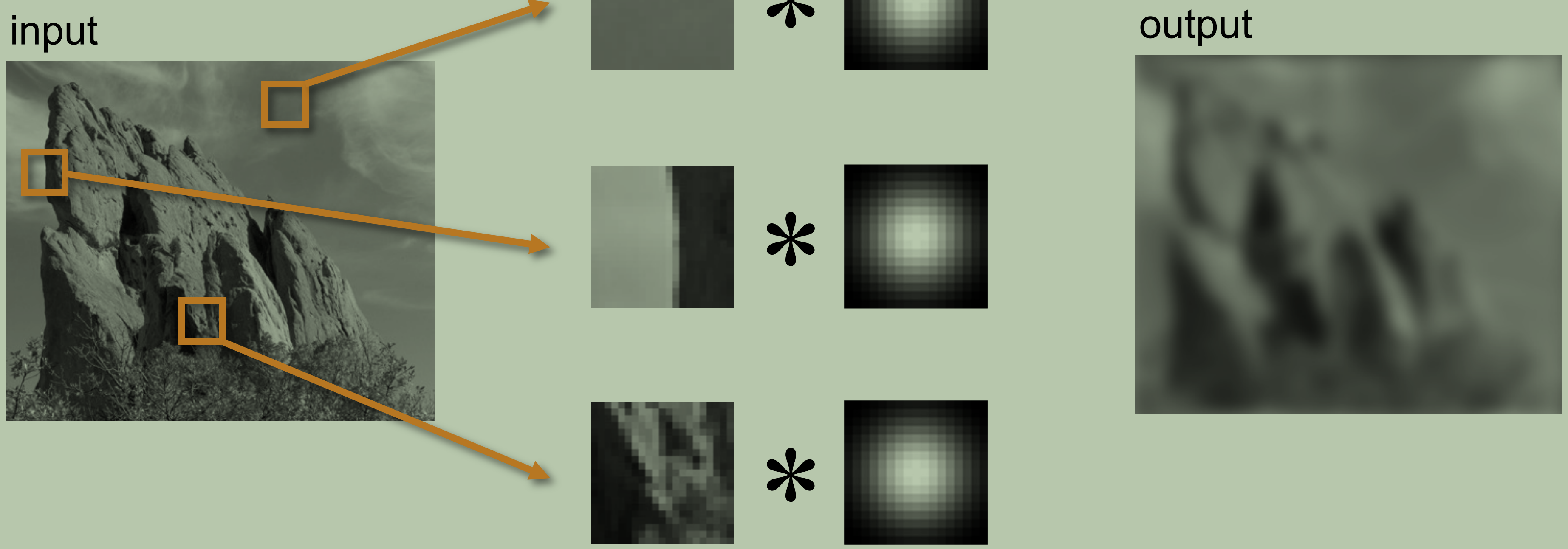
Bilateral filter (a nonlinear filter)

How to smooth an image without causing excessive edge-blur ?

Bilateral Filter (Tomasi et al.1998)

- It smooths regions while preserving edges

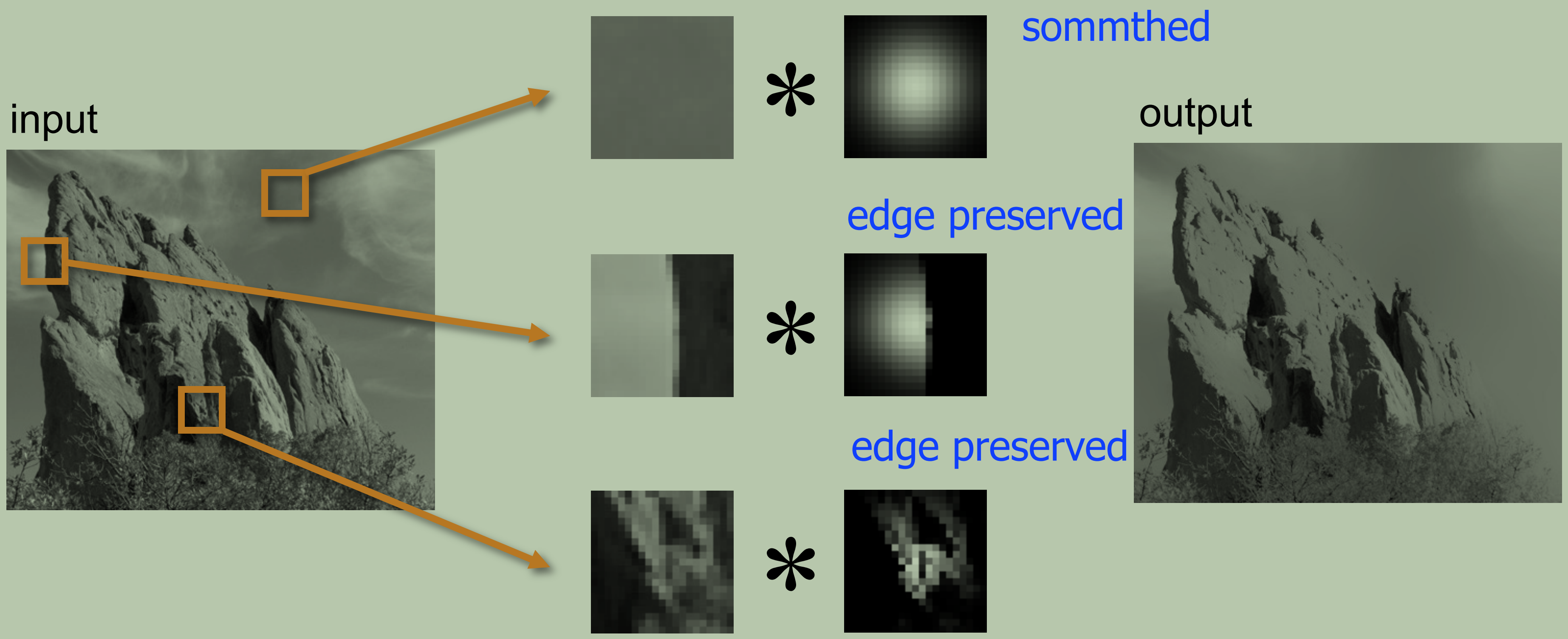
Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

Bilateral Filter
No Averaging across Edges

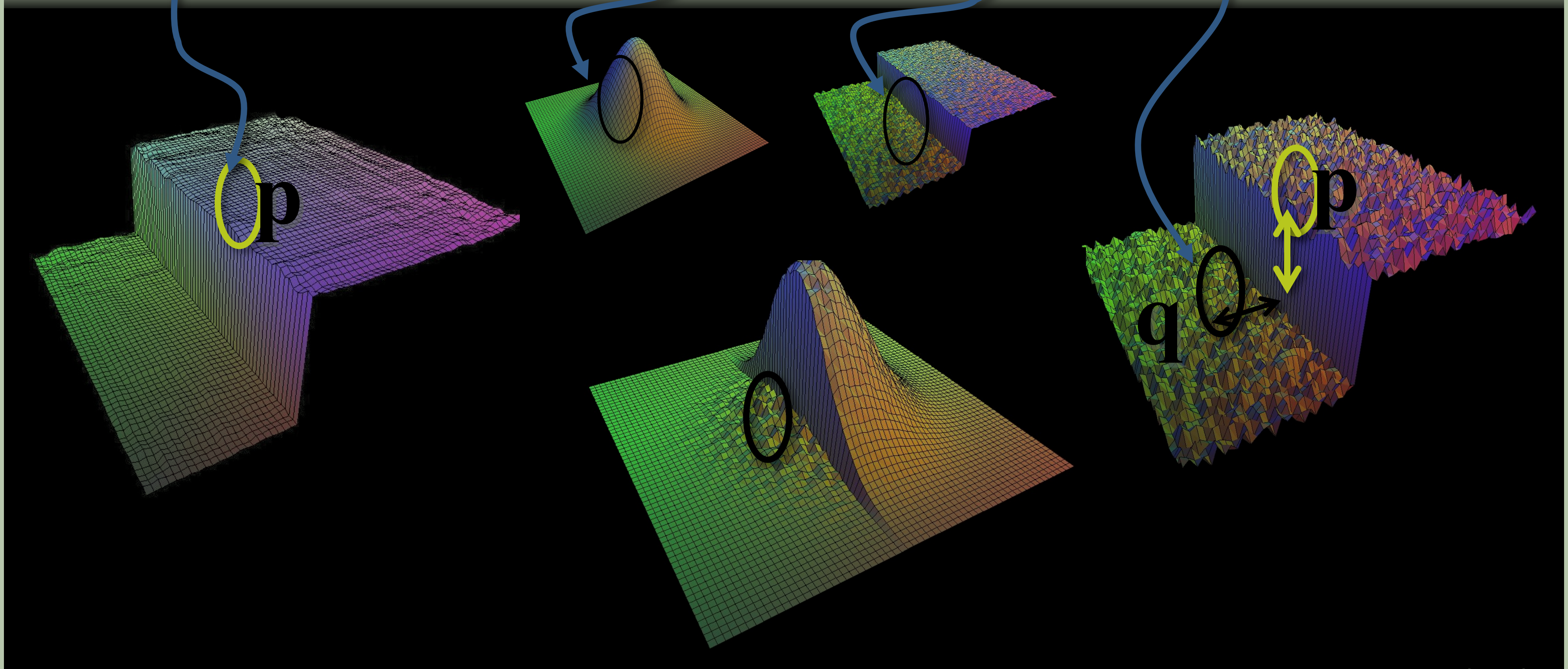
[Aurich 95, Smith 97, Tomasi 98]



The kernel shape depends on the image content.

Bilateral Filter on a Height Field

$$\underbrace{\overset{\text{smoothed img}}{BF[I]_{\mathbf{p}}}}_{g(i,\,j)} = \underset{\text{normaliser}}{\frac{1}{W_{\mathbf{p}}}} \sum_{\mathbf{q}\in S} \underbrace{\underbrace{\overset{\text{domain kernel}}{G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)}} \; \underbrace{\overset{\text{range kernel}}{G_{\sigma_r}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}}}_{w(i,\,j,\,k,\,l)} \; \underset{f(k,\,l)}{\overset{\text{original img}}{I_{\mathbf{q}}}}$$



reproduced from [Durand 02]

In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

smoothed img original img

$$\mathbf{g}(i,j) = \frac{\sum_{k,l} \mathbf{f}(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}. \tag{3.34}$$

The weighting coefficient $w(i,j,k,l)$ depends on the product of a *domain kernel* (Figure 3.19c),

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right), \tag{3.35}$$

and a data-dependent *range kernel* (Figure 3.19d),

$$r(i,j,k,l) = \exp\left(-\frac{\|\mathbf{f}(i,j) - \mathbf{f}(k,l)\|^2}{2\sigma_r^2}\right). \tag{3.36}$$

When multiplied together, these yield the data-dependent *bilateral weight function*

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|\mathbf{f}(i,j) - \mathbf{f}(k,l)\|^2}{2\sigma_r^2}\right). \tag{3.37}$$

Page 131, Section 3.3.2

Bilateral Filter on a Height Field

- Domain filter: G_{σ_s}

1. **Domain Filter:**
   - The domain filter, also known as the spatial filter or spatial kernel, determines which neighboring pixels are considered when computing the weighted average for each pixel.
   - It is typically a Gaussian filter applied in the spatial domain, centered at the pixel being filtered.
   - The domain filter controls the spatial extent of the filtering operation. Pixels that are closer to the central pixel in terms of spatial distance have higher weights in the computation, while pixels further away have lower weights.
   - The purpose of the domain filter is to preserve edges and fine details in the image by ensuring that pixels with similar spatial coordinates contribute more to the filtered result.

- Range filter: G_{σ_r}

2. **Range Filter:**
   - The range filter, also known as the intensity filter or range kernel, determines how neighboring pixels are weighted based on their intensity (or color) similarity to the central pixel.
   - It is typically a Gaussian filter applied in the intensity domain, centered at the intensity value of the central pixel.
   - The range filter controls the intensity similarity threshold for neighboring pixels. Pixels with intensity values similar to that of the central pixel are given higher weights, while pixels with significantly different intensity values are given lower weights.
   - The purpose of the range filter is to smooth the image while preserving edges and fine details. It ensures that only pixels with similar intensity contribute significantly to the filtered result, reducing the blurring effect on edges.

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\| \mathbf{p} - \mathbf{q} \|) \; G_{\sigma_r}(| I_{\mathbf{p}} - I_{\mathbf{q}} |) \; I_{\mathbf{q}}$$

The pixel value of the filtered image is the weighted average of the neighboring pixel's intensity/color values.

Varying the Space Parameter

input

range (data-dependent)

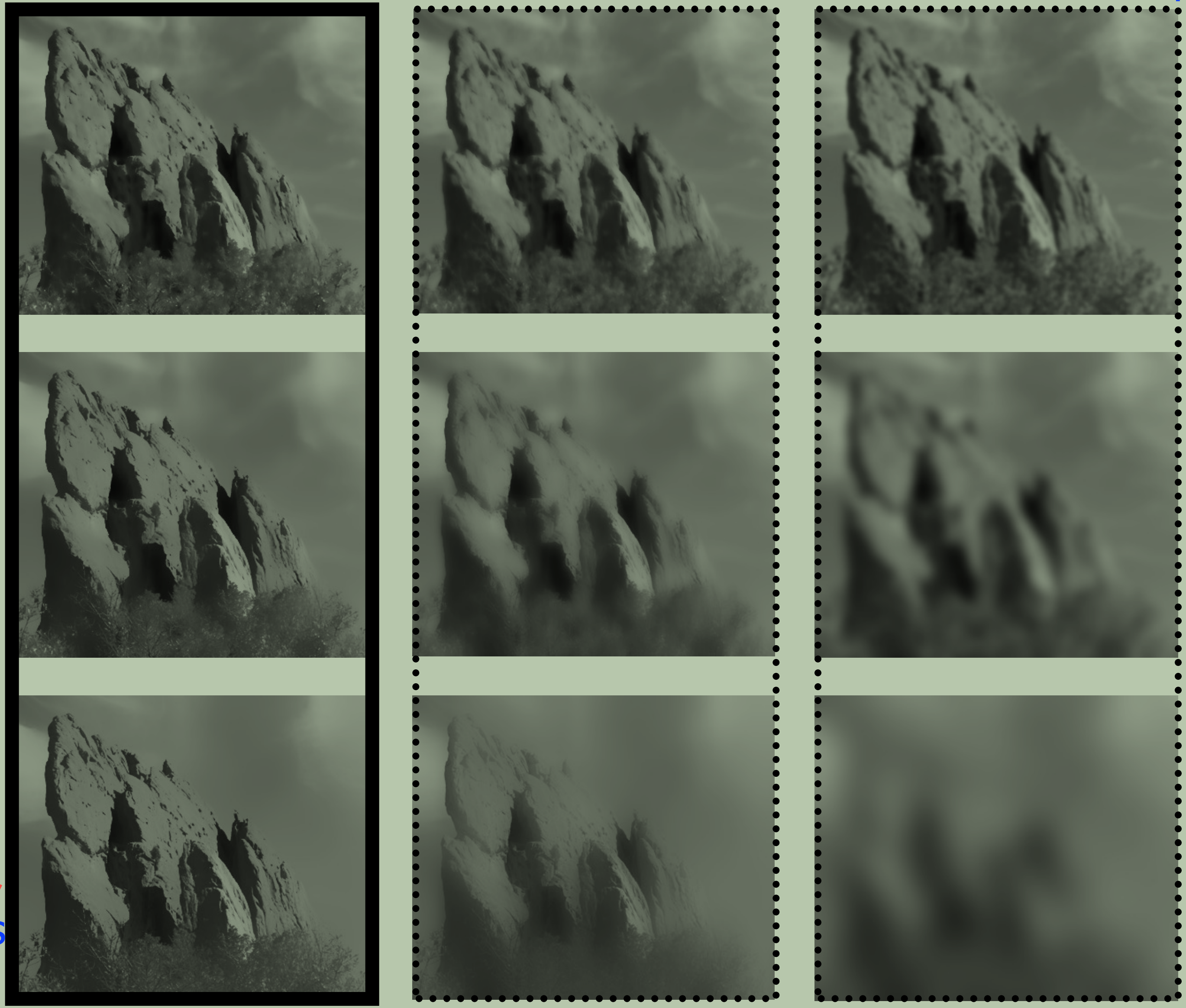
$\sigma_r = 0.1$ $\sigma_r = 0.25$ $\sigma_r = \infty$ (Gaussian blur)

the original img keeps more details

$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$



domain (2D Gaussian)

Guassian smooth filter blurs more edges


input


$\sigma_s = 2$


$\sigma_s = 6$


$\sigma_s = 18$

Readings

- Computer Vision: Algorithms and Applications, Chapter 3.1, 3.2 and 3.3