CSC476 Introduction to Computer Vision
Lecture 4

More on Histogram Equalization, Border Effect, Image Derivatives, Edge Detection

Bei Xiao

Last lecture

- Linear Algebra Review
- Gaussian Kernel
- Histogram Equalization

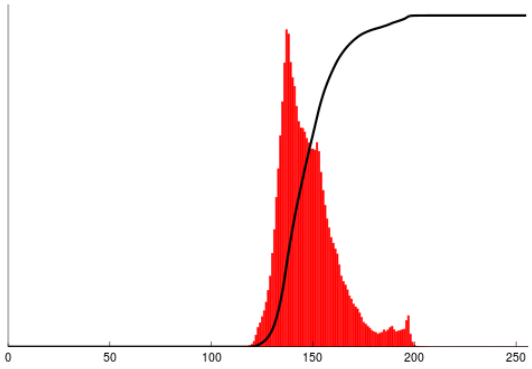
Today's Lecture

- More on image histograms
- Border effect and padding
- Image Derivatives/gradients
- Canny edge detection

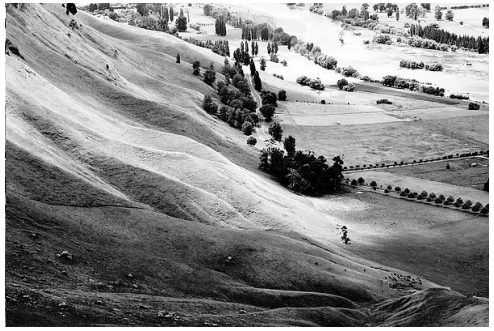
Learning objective check

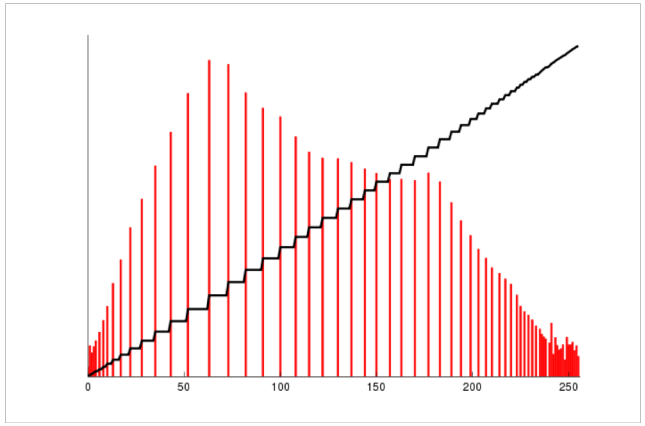
- You should be so far:
- 1) fluent with convolving images with filter
- 2) fluent with convolving matrix with another matrix.
- 3) Know basic Numpy matrix manipulation.
- 4) Create box, 2D Gaussian and 1D Gaussian filters of all sizes.

Image Histogram Equalization



Notice the “shape” of the histogram is preserved!





Same number of pixels In each bin!

Questions

1. Why don’t we do the equalization directly on the histograms? What will happen to the image if it has a perfectly flat histograms?
2. What is CDF? Why are there steps (zig-zags) on the CDF?
3. How to make sure the “shape” of the PDF is preserved in the transformation?

Let’s look at in more details

We start with a gray-scale jpeg image of 8 by 8

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

https://en.wikipedia.org/wiki/Histogram_equalization

Let’s look at in more details

The histograms for this image (8 by 8 pixels) is shown in the following table (intensity that has zero pixels are skipped:

Value	Count	Value	Count	Value	Count	Value	Count	Value	Count
52	1	64	2	72	1	85	2	113	1
55	3	65	3	73	2	87	1	122	1
58	2	66	2	75	1	88	1	126	1
59	3	67	1	76	1	90	1	144	1
60	1	68	5	77	1	94	1	154	1
61	4	69	3	78	1	104	2		
62	1	70	4	79	2	106	1		
63	2	71	2	83	1	109	1		

Let's look at in more details

The cumulative distribution function (CDF) is shown below

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	
…	…	
154	64	255

Number of pixels

This cdf shows that the min value in the subimage is 52 and max is 154. The cdf of value 154 corresponding to the total number of pixels (64)

Let's look at in more details

How do we compute the normalized CDF?

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Cdf(v): original cdf of pixel v
Cdfmin: minimum non-zero vale of the cdf
M×N: number of pixels, e.g. 64 (8x8)
L: 256

Quiz:

How do we compute normalized CDF of pixel 62?

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	?
…	…	
154	64	255

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Quiz:

How do we compute normalized CDF of pixel 62?

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	?
…	…	
154	64	255

H(62) = round((15-1)/63*255)
= 57

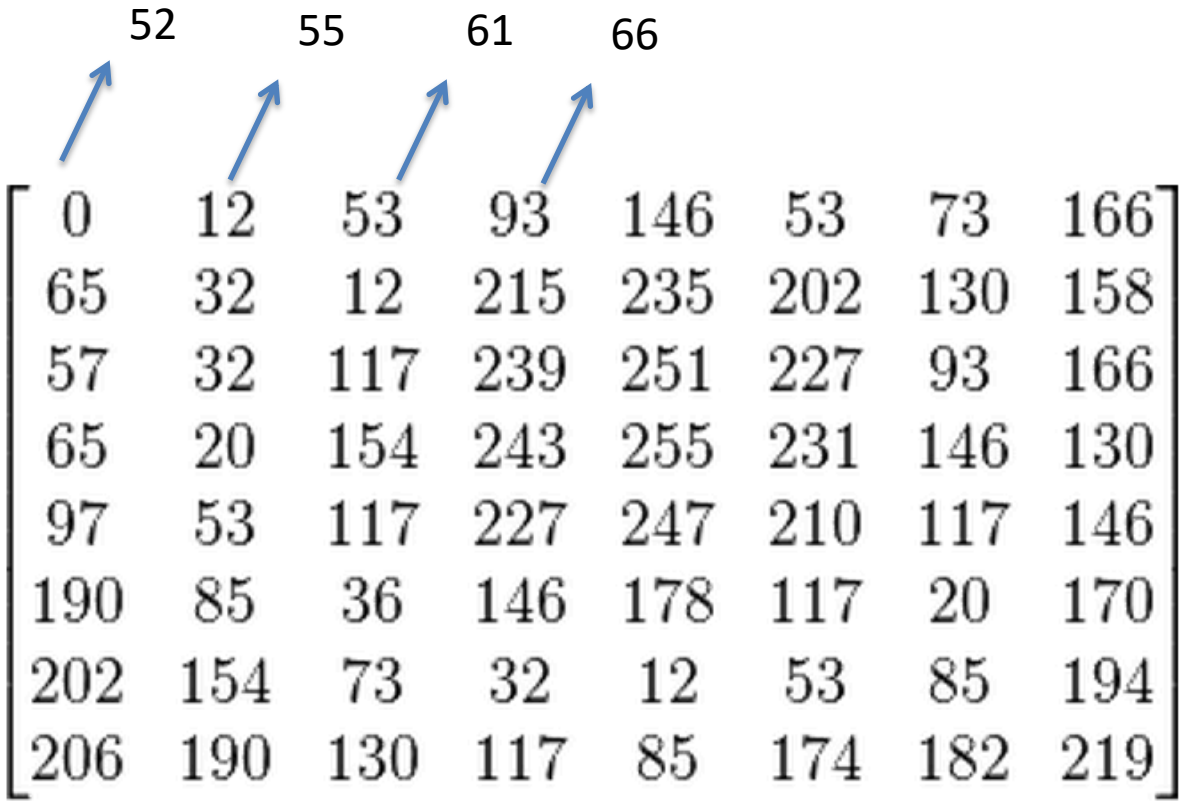
Let’s look at in more details

The cumulative distribution function (CDF) is shown below

Value	cdf	scaled cdf
52	1	0
55	4	12
58	6	22
59	9	32
60	10	36
61	14	53
62	15	57
…	…	…
154	64	255

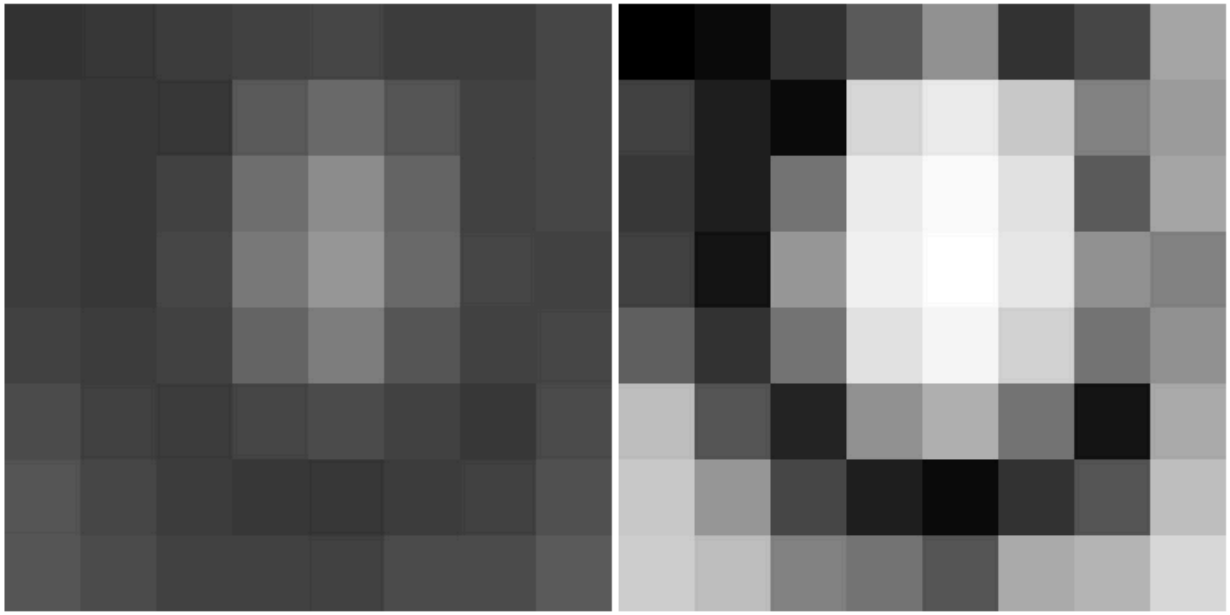
Let's look at in more details

Now we can directly map the scaled cdf back to pixel values using the look up table we derived above



Let’s look at in more details

Notice that the minimum value (52) is now 0 and the maximum value (154) is now 255.

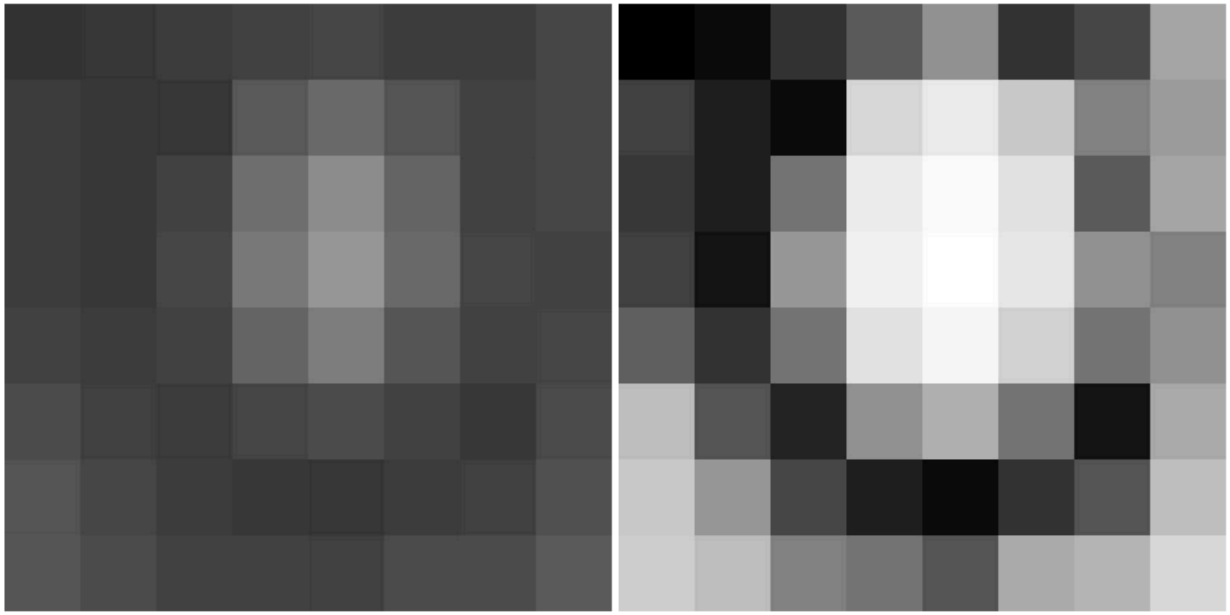


Original

Equalized

Let’s look at in more details

Notice that the minimum value (52) is now 0 and the maximum value (154) is now 255.



Original

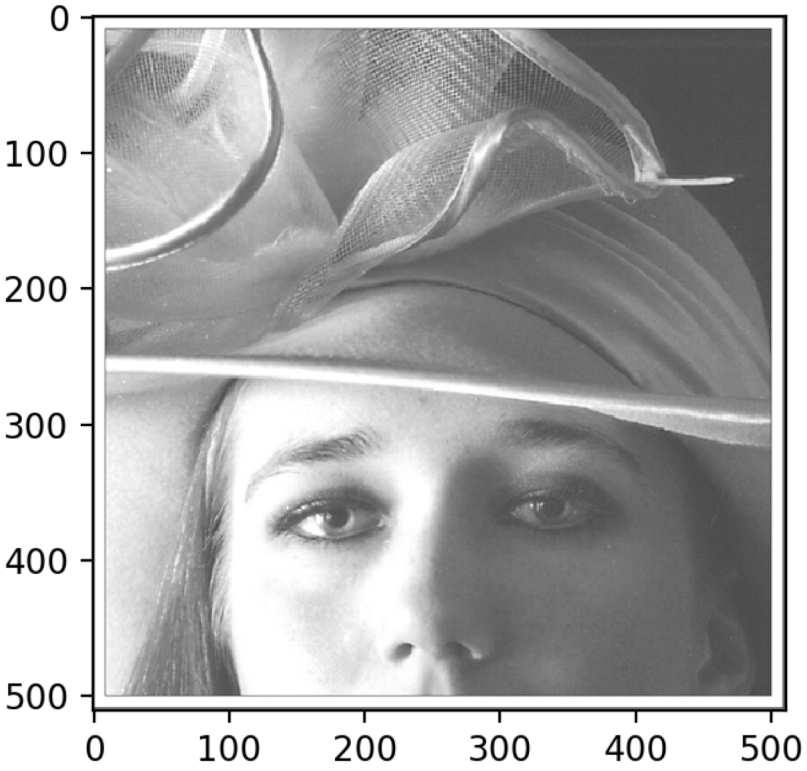
Equalized

Homework 2: histogram equalization

Hint:

```
# read in the image
im = misc.imread('lowcontrast.jpg',flatten=1)

#Compute cdf in Python:

np.histogram(im.flatten(),nbr_bins,normed=True)

cdf = imhist.cumsum()

# normalize
cdf = 255 * cdf / cdf[-1]

# Using linear interpretation of cdf to find new pixels.
im2 = np.interp(im.flatten(),bins[:-1],cdf)
```

Homework 2: histogram equalization



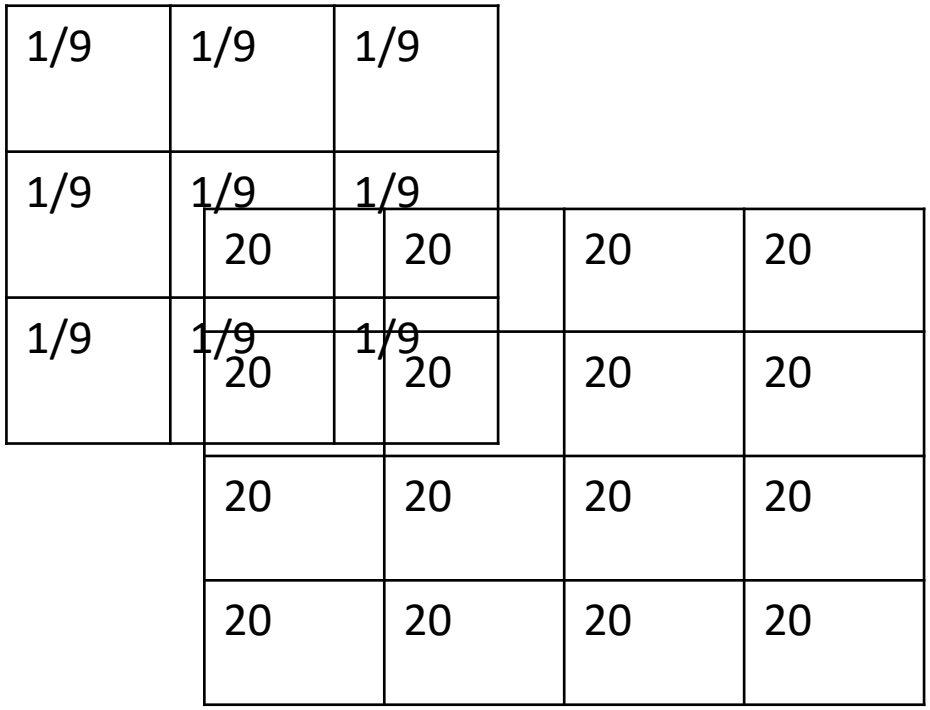
Back to Convolution

- Let’s blur a flat gray image:

How should it look like?

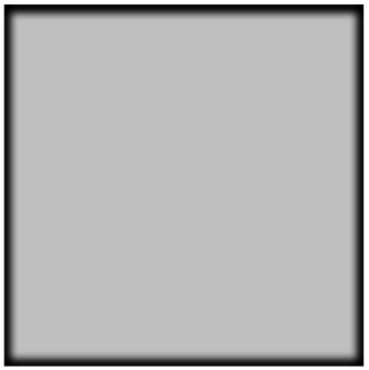
Back to Convolution

- Convolve with a box kernel



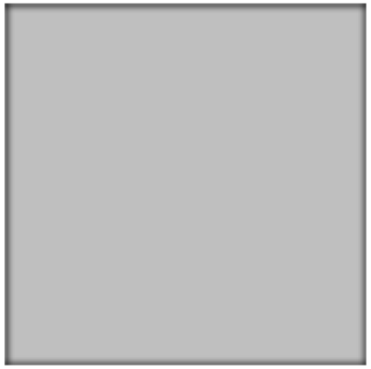
Border Handling

Depending on how you do the convolution, you could end up with 3 different images.



266x266 Image

full



256x256 Image

same

246x246 Image

valid

Back to Convolution

- Zero padding:

Filled in borders with zeros, computed everywhere the kernel touches.

Numpy.convolve(a,v,mode =‘full’ or ‘same’)

This returns the convolution at each point of overlap, with an output shape of (N+M-1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen.

Back to Convolution

- Convolve with a box kernel, suppose there are zeros outside the image matrix.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

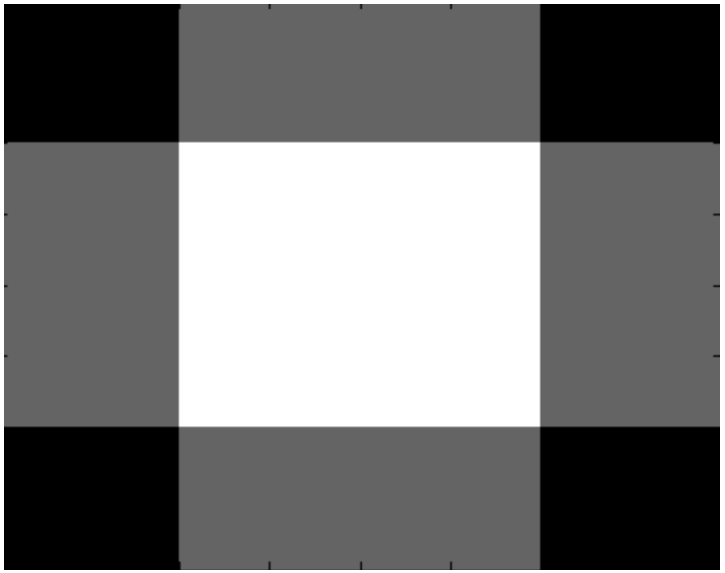
20	20	20	20
20	20	20	20
20	20	20	20
20	20	20	20

8.9	13.3	13.3	8.9
13.3	20.0	20.0	13.3
13.3	20.0	20.0	13.3
8.9	13.3	13.3	8.9

Back to Convolution

- This is called “same” in MATLAB/Numpy

4 ×4



4 ×4

Back to Convolution

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

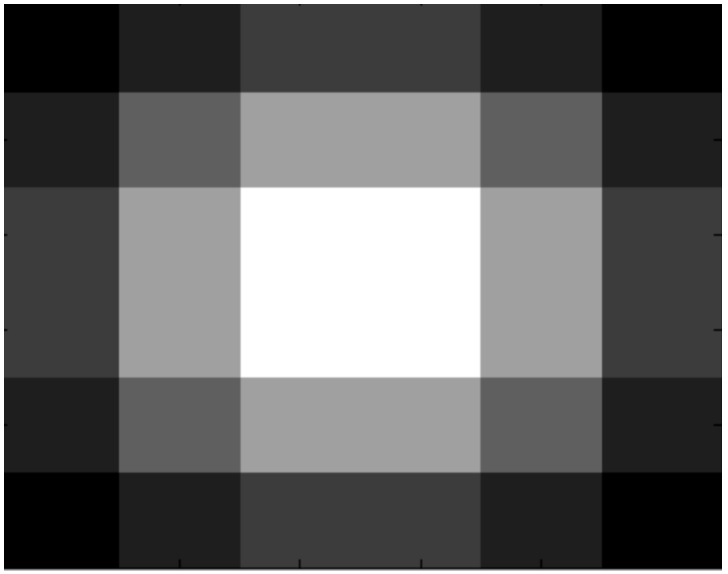
20	20	20	20
20	20	20	20
20	20	20	20
20	20	20	20

2.2	4.4	6.6	6.6	4.4	2.2
4.4	8.9	13.3	13.3	8.9	4.4
6.6	13.3	20.0	20.0	13.3	6.7
6.6	13.3	20.0	20.0	13.3	6.7
4.4	8.9	13.3	13.3	8.9	4.4
2.2	4.4	6.6	6.6	4.4	2.2

Back to Convolution

- This is called “full” in MATLAB/Numpy.

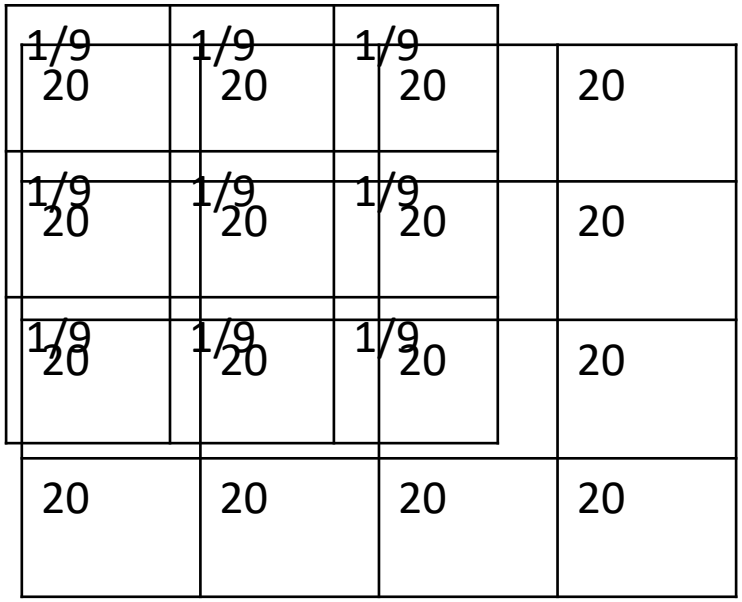
4 ×4



6 ×6

Back to Convolution

- Only compute at places where the kernel fits the image



20	20
20	20

Back to Convolution

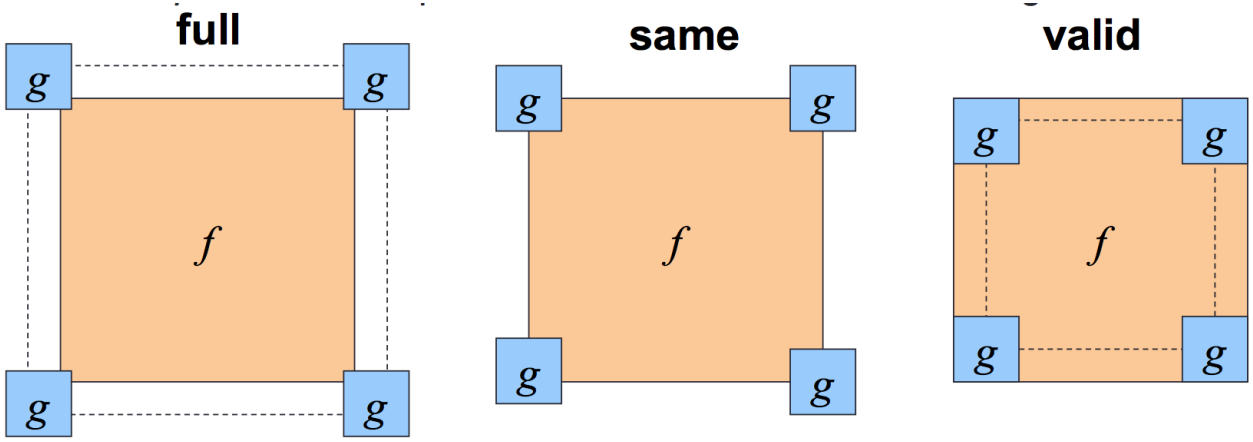
- This is called “valid” in MATLAB/Numpy.

4 ×4

2 ×2

To summarize

- Python:
  - Numpy.convolve(a,v,mode="full")
  - Scipy.signal.convole2d(a,v,mode, boundary, value)



There are other methods

- The first two methods that I described fill missing values in by substituting zero
- Can fill in values with different methods
  - Reflect image along border
  - Pull values from other side
- Read Chapter 3.2 “padding”.

- scipy.ndimage.convolve(input, weights, output-None, mode =“reflect”)

There are other methods

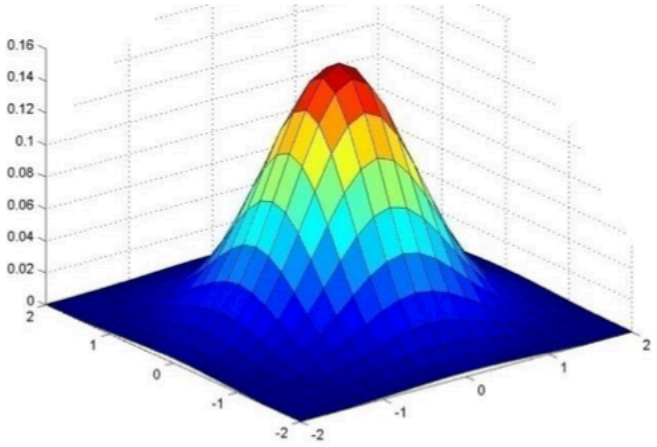
zero

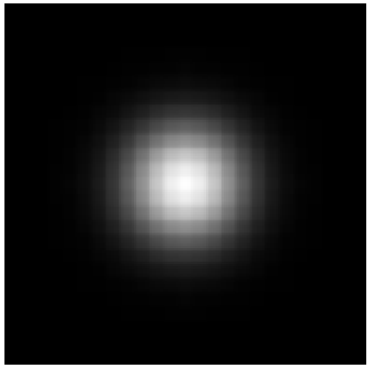
valid

Nearest neighbor

Gaussian Filter

- Gaussian = normal distribution function





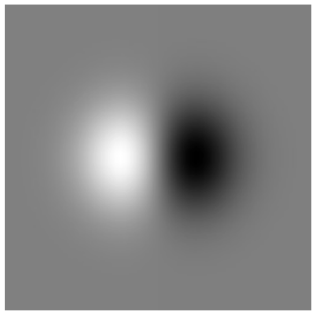
$$K(i,j) = \frac{1}{Z}\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

Derivative of Gaussian

- Take the derivative of the filter with respect to i:

$$\frac{\partial K(i,j)}{\partial i} = \frac{-i}{\sigma^2 Z} \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

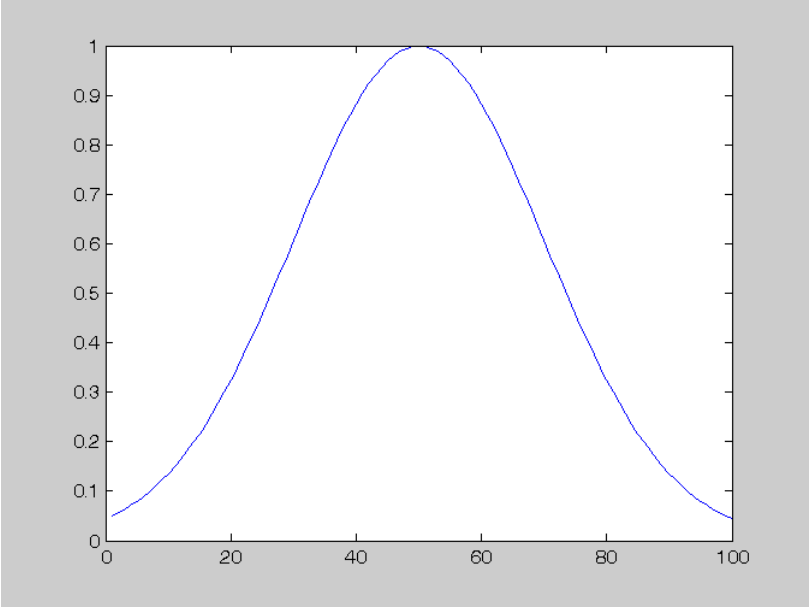
- Filter looks like:

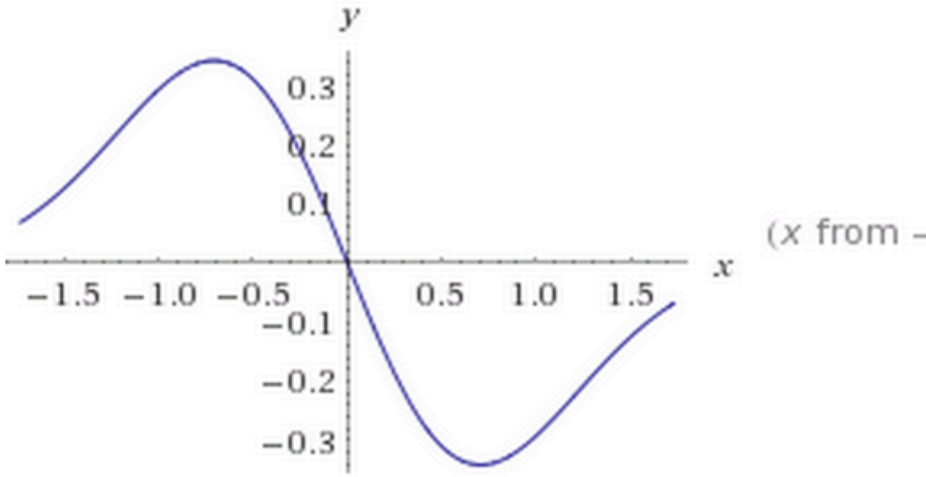


- Basically blur then take the derivative

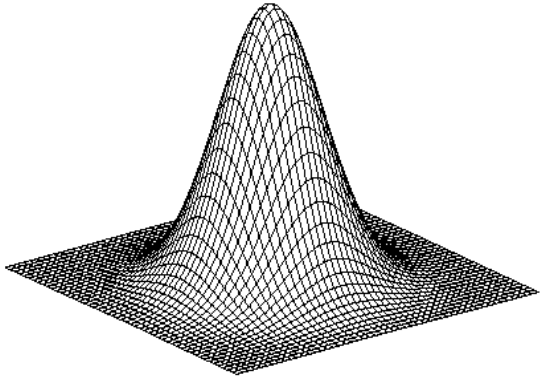
Source: M. Tappen

Derivative of Gaussian



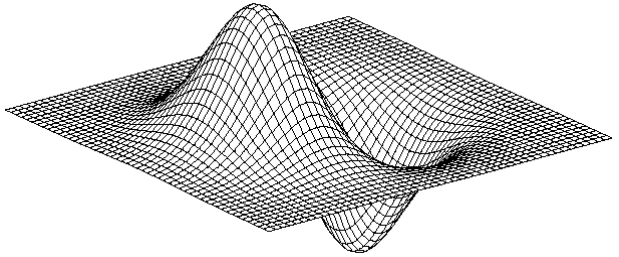


2D edge detection filters



Gaussian

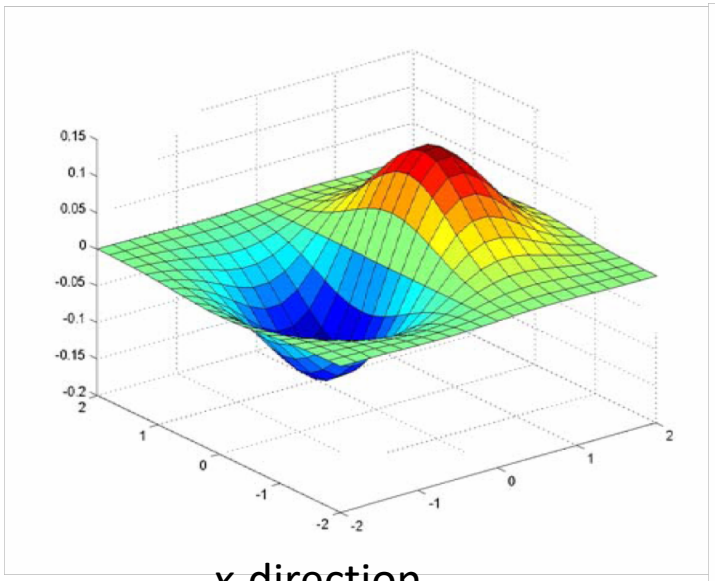
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



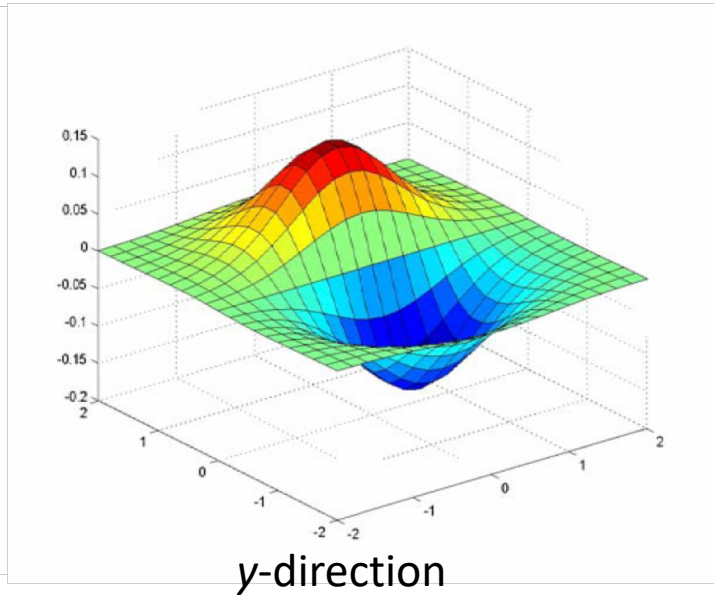
derivative of Gaussian (x)

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

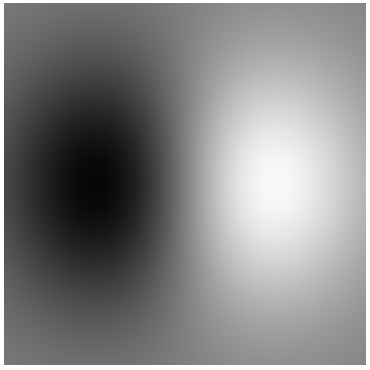
Derivative of Gaussian filter



x-direction



y-direction



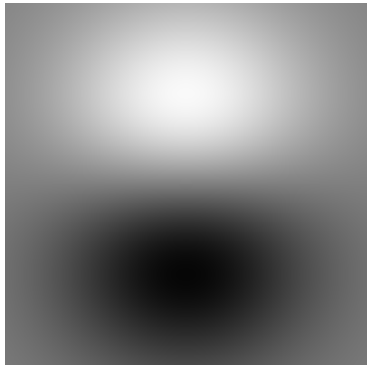
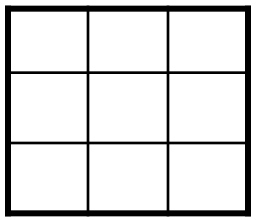


Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f,* then compute the derivative
  - Option 2: take discrete derivative (finite difference)

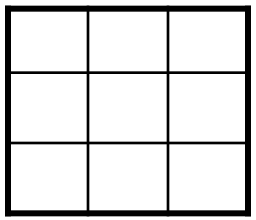
How would you implement this as a linear filter?

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



H_x

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



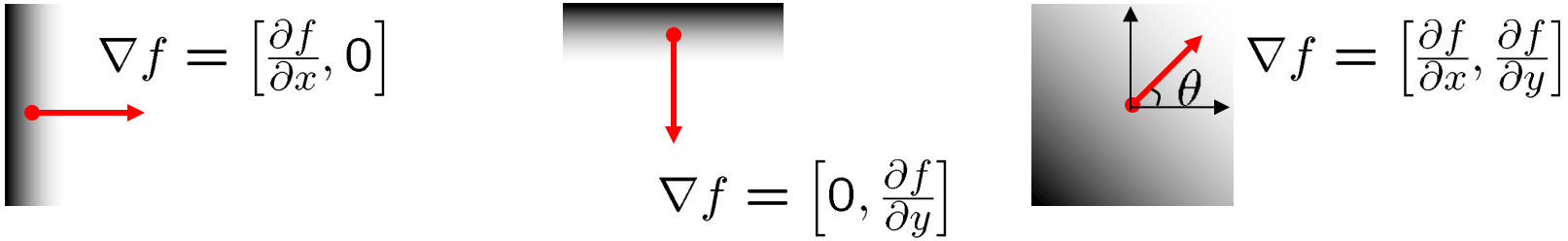
H_y

Source: S. Seitz

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 increase in intensity



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

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

The gradient direction is given by:

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

- how does this relate to the direction of the edge?

Source: Steve Seitz

The Sobel operator

- Common approximation of derivative of Gaussian

$\frac{1}{8}$

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

s_x

$\frac{1}{8}$

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

s_y

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value

Example of Sobel filtered image

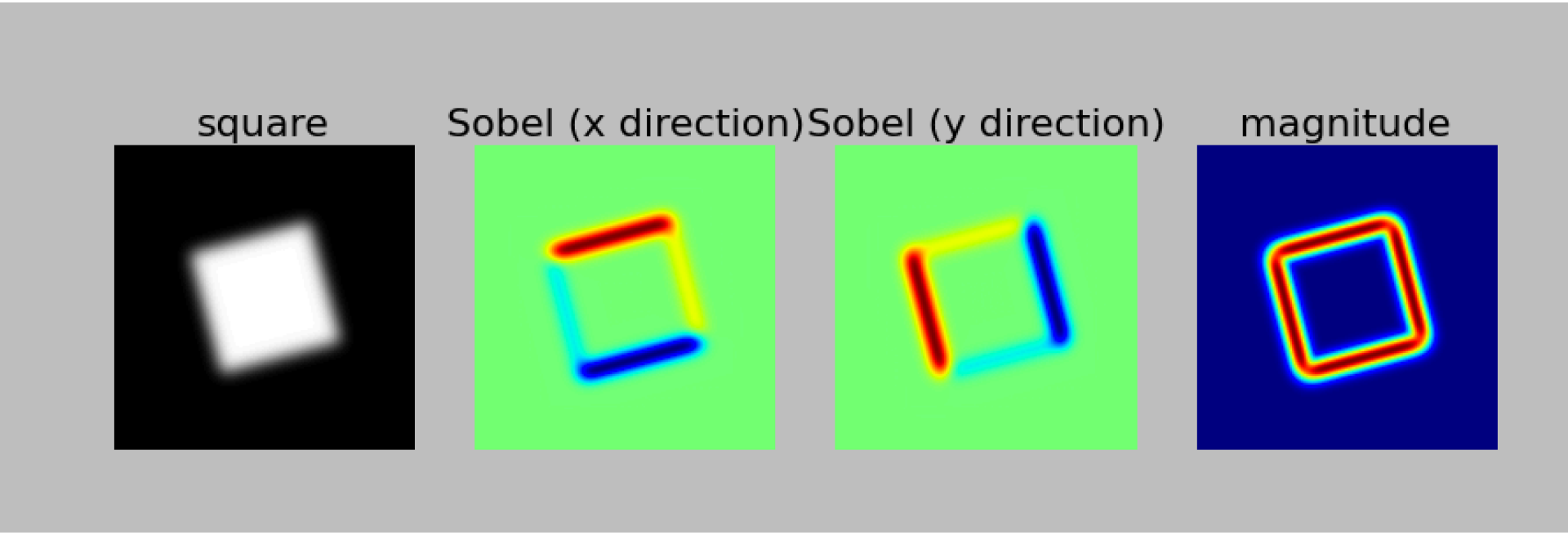
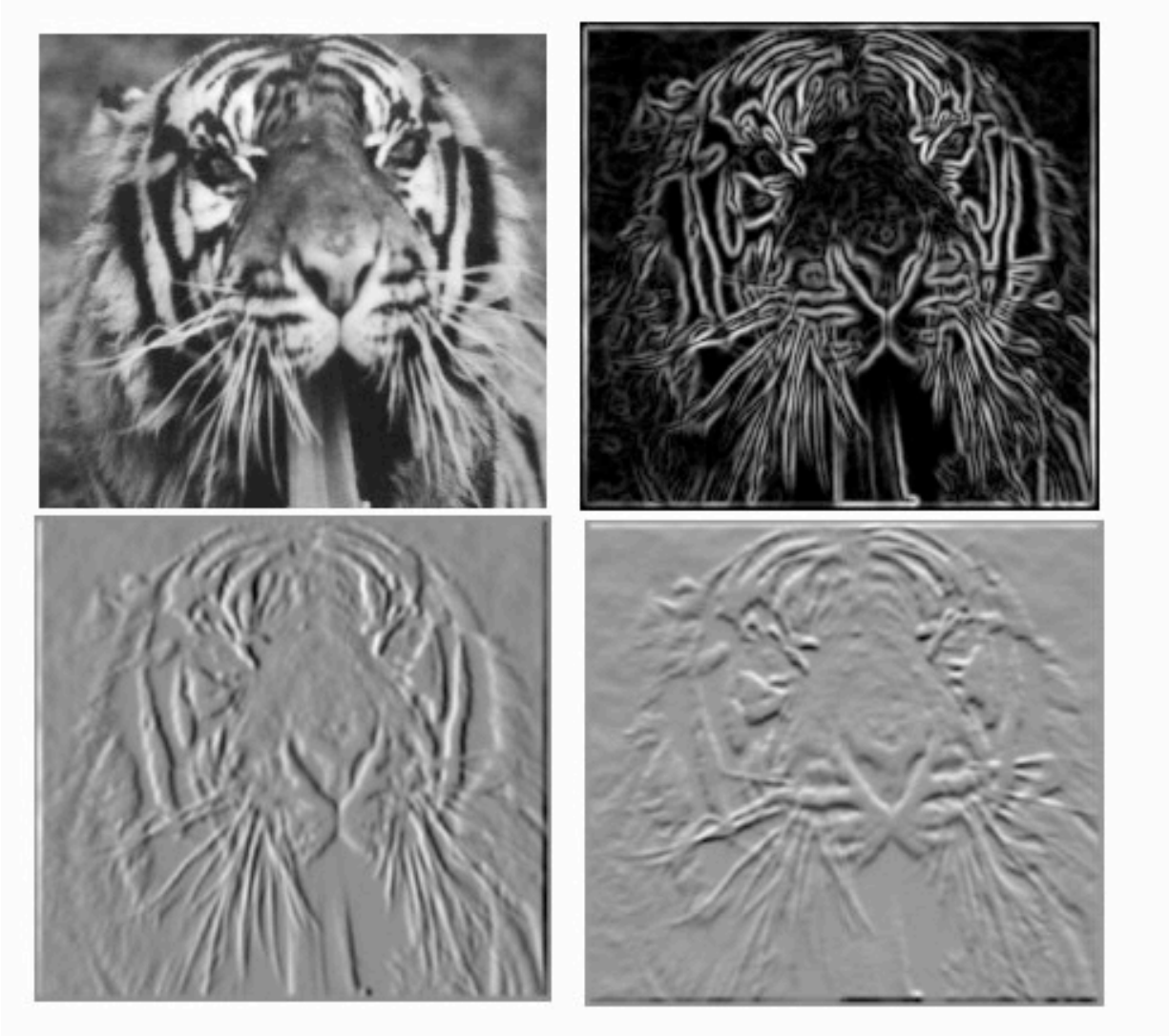
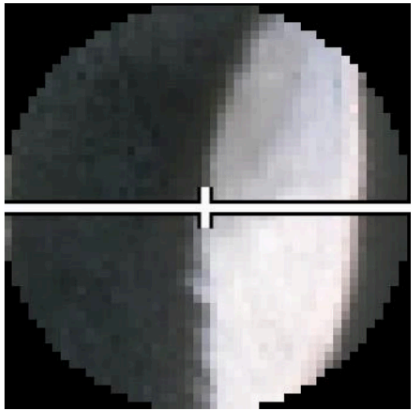


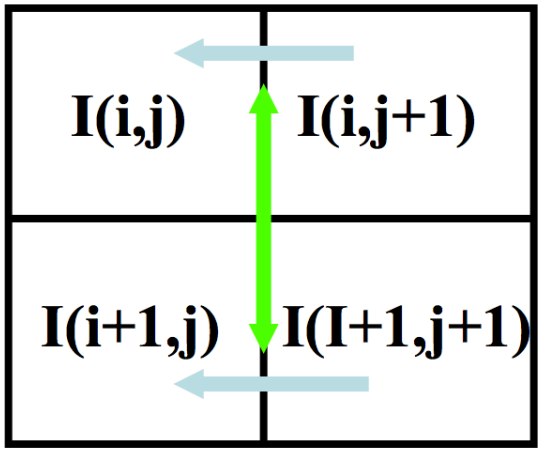
Image gradient



Source: L. Lazebnik

Compute gradient: first order derivatives





Compute gradient in the X-direction:

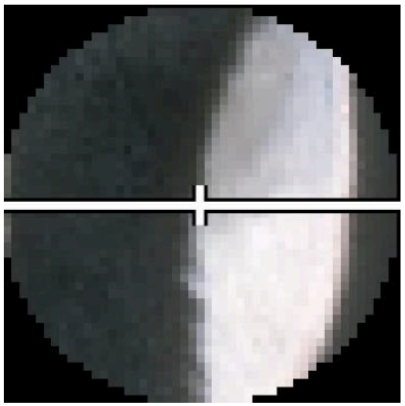
1) **Take the image intensity difference in the X-direction**
2) **Average the difference in the Y-direction(smoothing)**

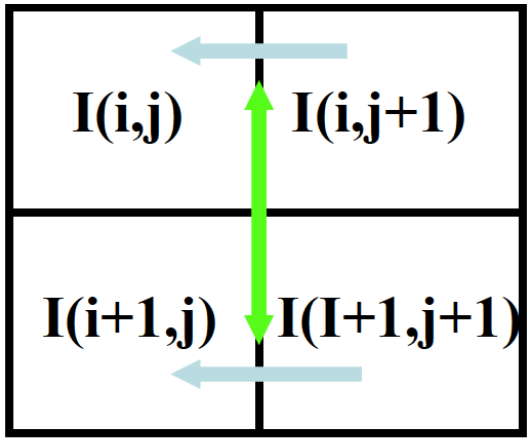
$$\frac{\delta I}{\delta x}(i,j) = \frac{1}{2}(\boxed{(I(i,j+1) - I(i,j))} + (\boxed{I(i+1,j+1) - I(i+1,j)}))$$

Slide source: Jianbo Shi

Compute gradient: first order derivatives

$$\frac{\delta I}{\delta x}(i,j) = \frac{1}{2}((I(i,j+1) - I(i,j)) + (I(i+1,j+1) - I(i+1,j)))$$

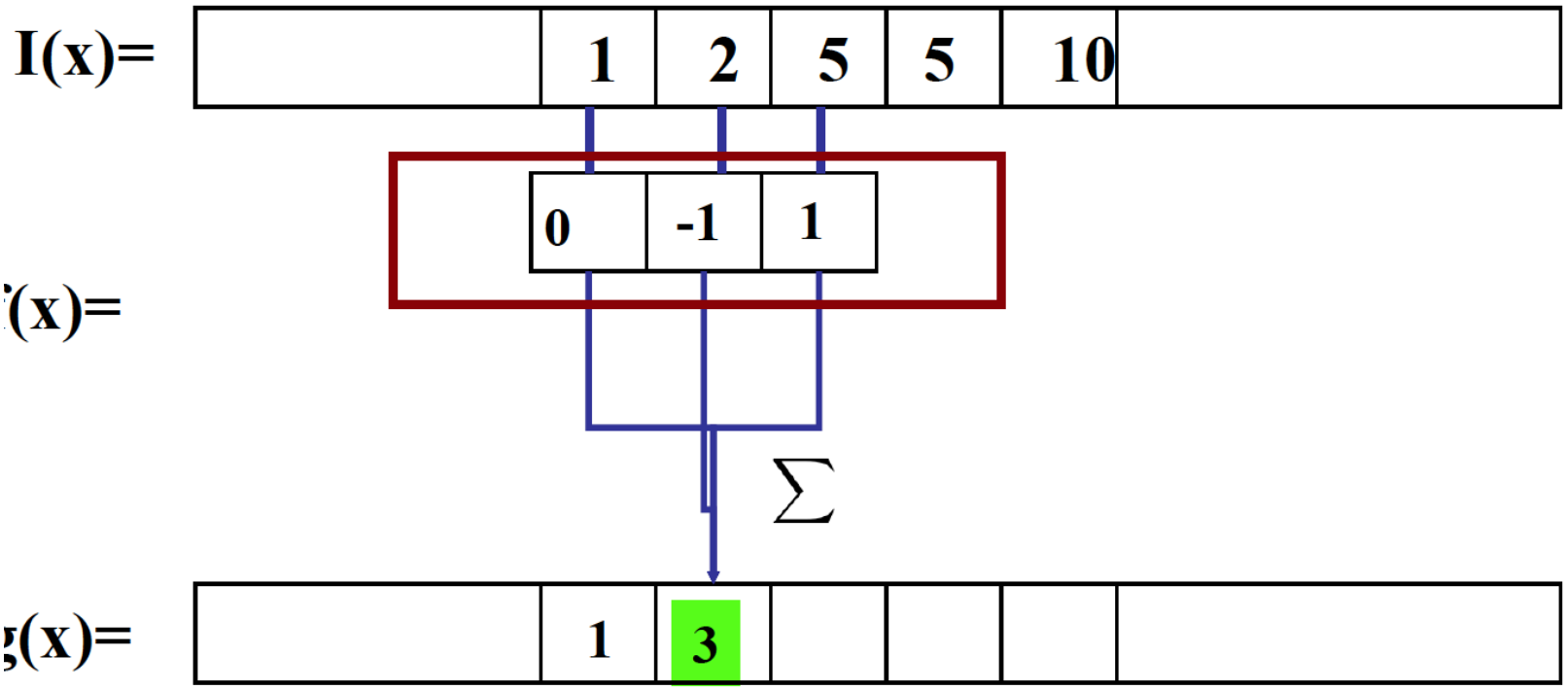




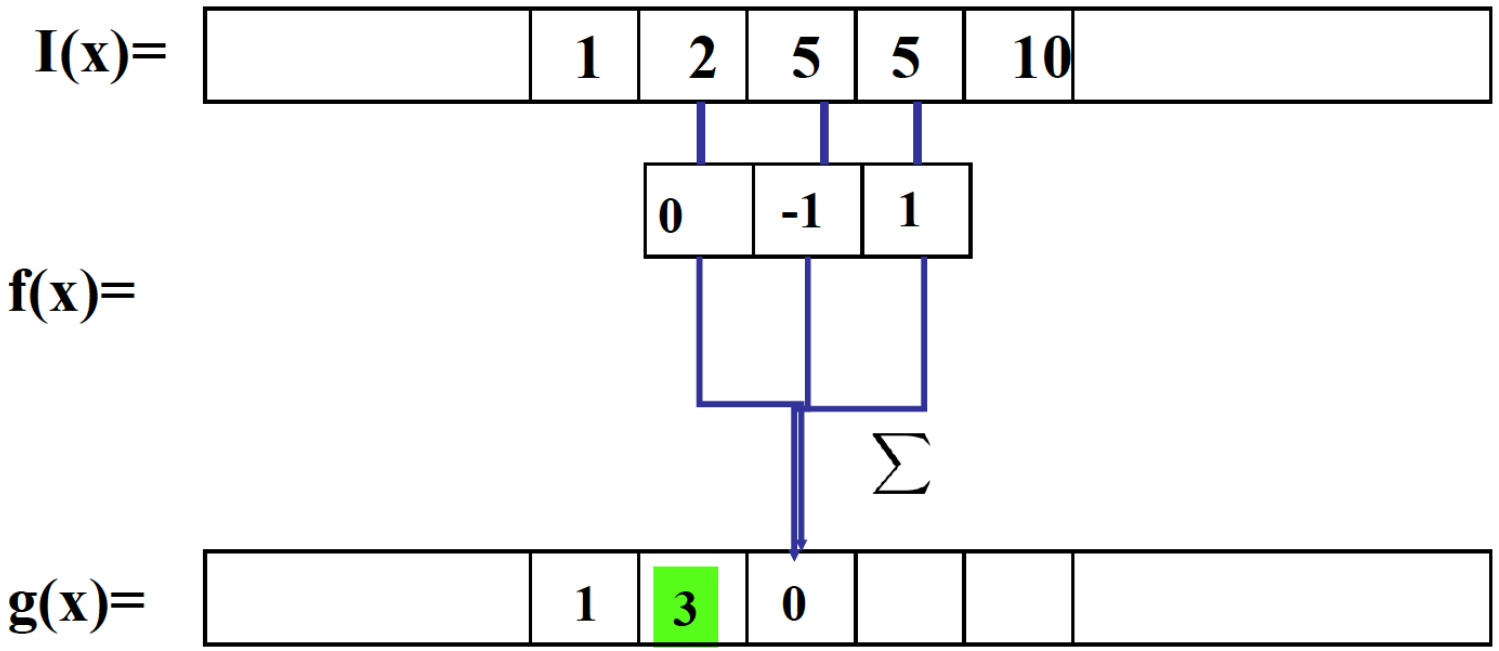

```
[nr,nc] = size(Is);
Ix = zeros(nr,nc);  % generate a empty matrix of size nr by nc
for i=1:nr-1,
    for j=1:nc-1,
        Ix(i,j) = 0.5*( (Is(i,j+1) - Is(i,j)) + (Is(i+1,j+1) - Is(i+1,j)));
    end
```

Slide source: Jianbo Shi

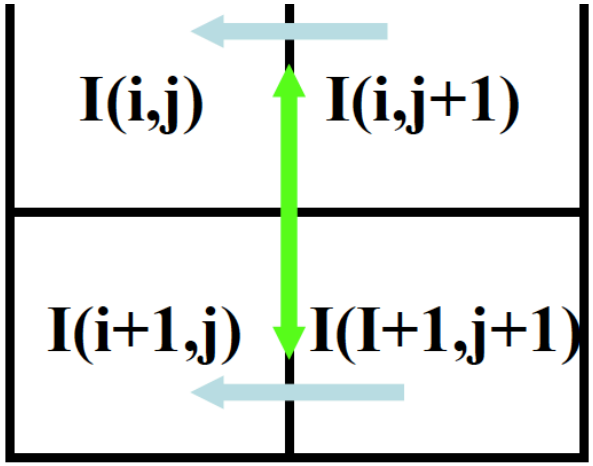
Compute gradient as convolution operation!



Compute gradient as convolution operation!



Compute gradient: first order derivatives

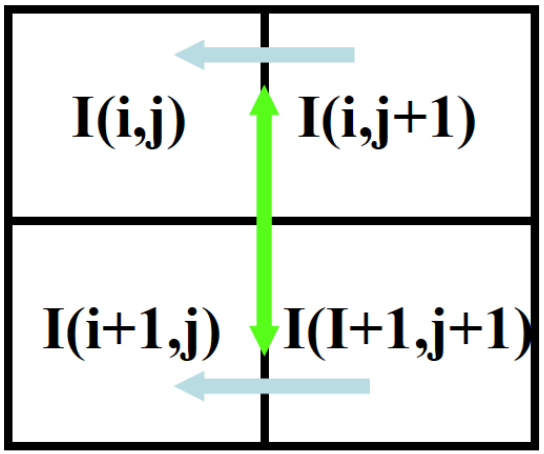


$$\frac{\delta}{\delta x} = \boxed{1 \quad -1}$$

$$\frac{\delta I}{\delta x}(i,j) = (I(i,j+1) - I(i,j));$$

$$= I \otimes (\frac{\delta}{\delta x})$$

Compute gradient: first order derivatives



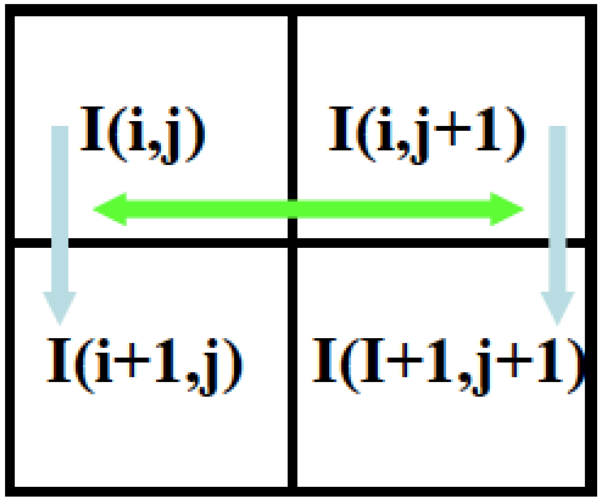
$$\frac{\partial}{\delta x} = \boxed{1 \mid -1}$$

$$\mathbf{S} = \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array}$$

$$\frac{\delta I}{\delta x}(i,j) = \frac{1}{2}((I(i,j+1) - I(i,j)) + (I(i+1,j+1) - I(i+1,j)))$$

$$= (I \otimes \frac{\delta}{\delta x}) \otimes S$$

Compute gradient: first order derivatives



$$\frac{\delta}{\delta y} = \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}$$

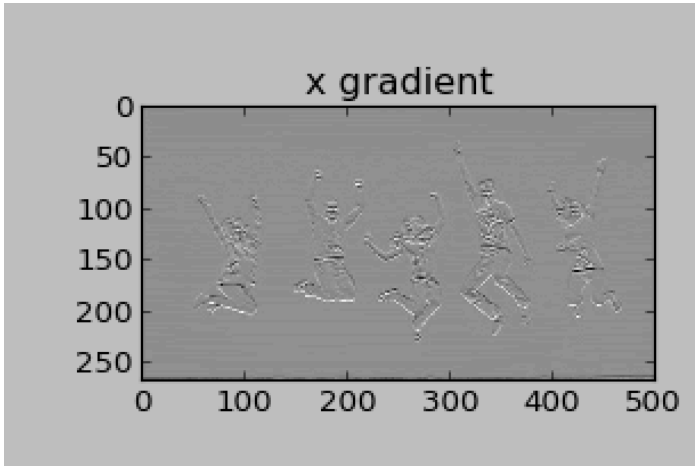
$$\frac{\delta I}{\delta y}(i,j) = \frac{1}{2}((I(i+1,j) - I(i,j)) + (I(i+1,j+1) - I(i,j+1)))$$

$$I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'$$

Example

Exercise in Python

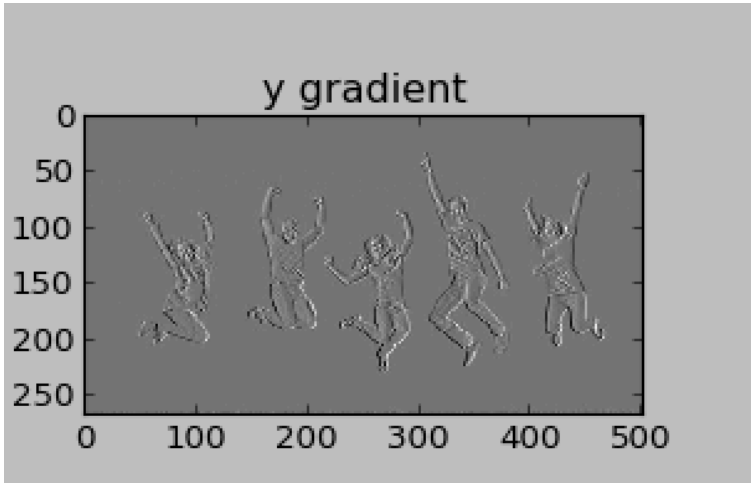
- s1 = np.array([1,1])
- dx = np.array([1,-1])
- dy = np.array([1,-1])
- x = ndimage.convolve1d(I,dx,axis= 0)
- gx_I = ndimage.convolve(x,s)



$$I_x = (I \otimes \frac{\delta}{\delta x}) \otimes S$$

Usage in Python

- s1 = np.array([1,1])
- dx = np.array([1,-1])
- dy = np.array([1,-1])
- y = ndimage.convolve1d(I,dx,axis= 1)
- gy_I = ndimage.convolve(y,s)



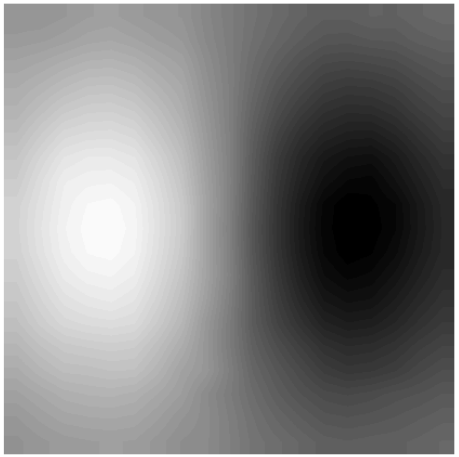
Or : gx_I,gy_I = np.gradient(I)[:2]

$$I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'$$

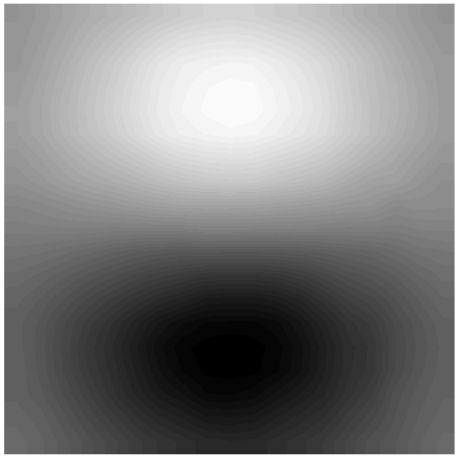
Smoothed derivative filter

$$\frac{\delta}{\delta x} \otimes G = \frac{\delta G}{\delta x} \longrightarrow \frac{\delta G}{\delta x} = -\frac{2x}{\sigma_x^2} G(x, y)$$

Gx



Gy



Sobel Filter

- Product of averaging and gradient.
- An cross product of two 1 d filter, Gaussian and gradient

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

Exercise: Create image derivatives using Sobel filter

1. Generate an image of a rotated rectangle

```
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

im = np.zeros((256, 256))
im[64:-64, 64:-64] = 1
im = ndimage.rotate(im, 15, mode='constant')
```

2. Blur the image using a Gaussian filter

```
im = ndimage.gaussian_filter(im, 8)
```

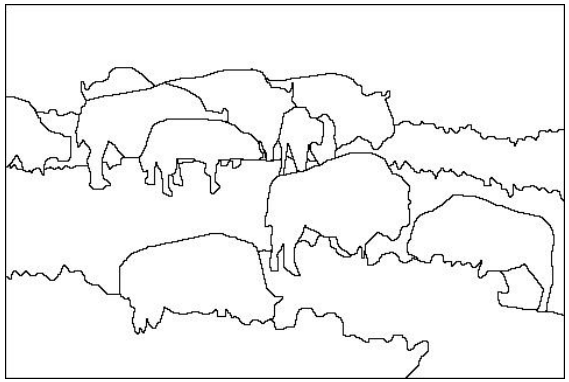
3. Apply Sobel filter to both x and y direction.

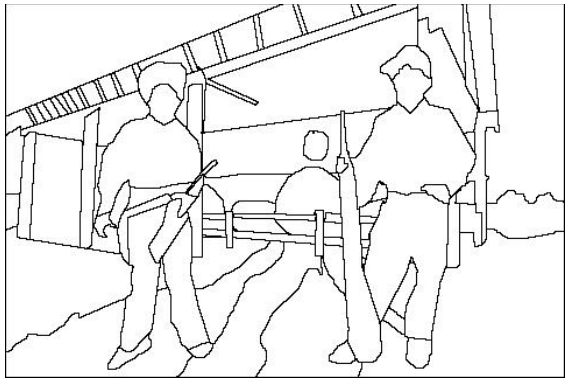
```
sx =  ndimage.sobel(im, axis=0, mode='constant')
```

4. Display the original image, x-derivatives, y-derivatives, and the gradient magnitude. You can use np.hypot to compute magnitude.
See here: http://docs.scipy.org/doc/numpy/reference/generated/numpy.hypot.html)

Where do humans see boundaries?

image | human segmentation | gradient magnitude




- Berkeley segmentation database:
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Edge detection

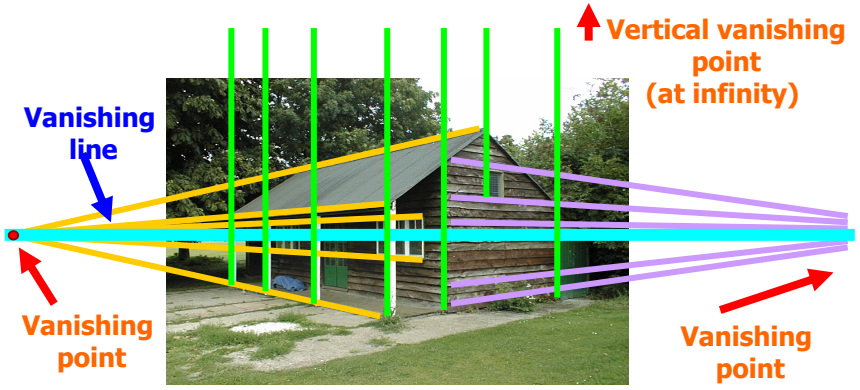
- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels
- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Source: D. Lowe

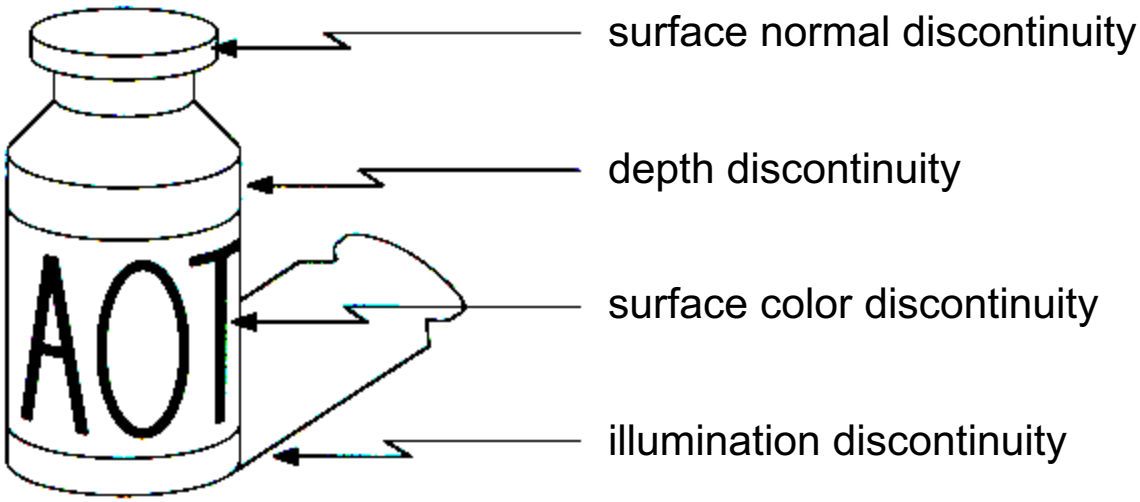
Why do we care about edges?

- Extract information, recognize objects

- Recover geometry and viewpoint



Origin of Edges



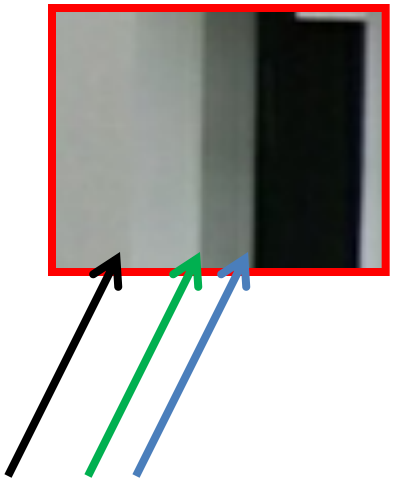
- Edges are caused by a variety of factors

Source: Steve Seitz

Closeup of edges

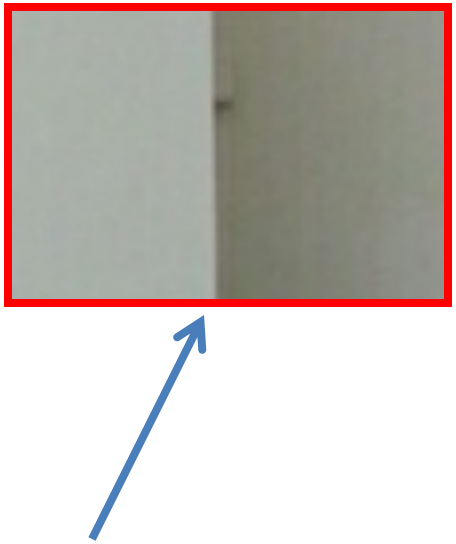
Source: D. Hoiem

Closeup of edges



Source: D. Hoiem

Closeup of edges



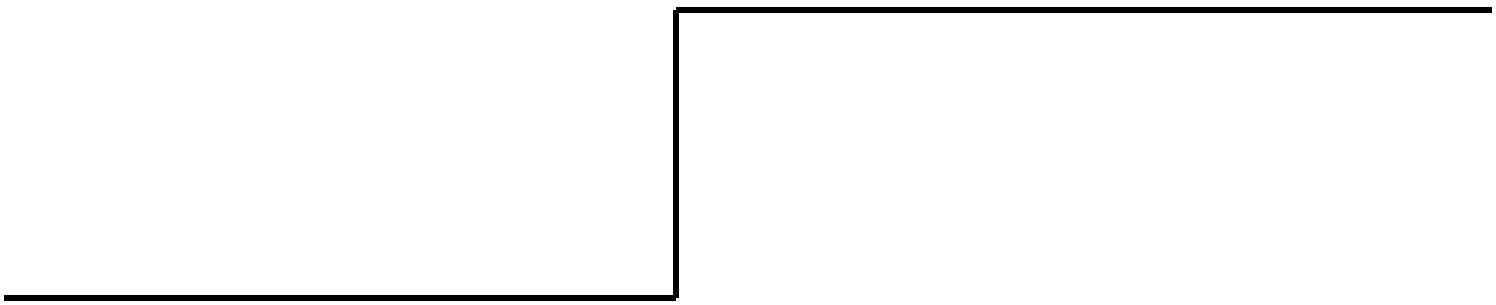
Source: D. Hoiem

Closeup of edges

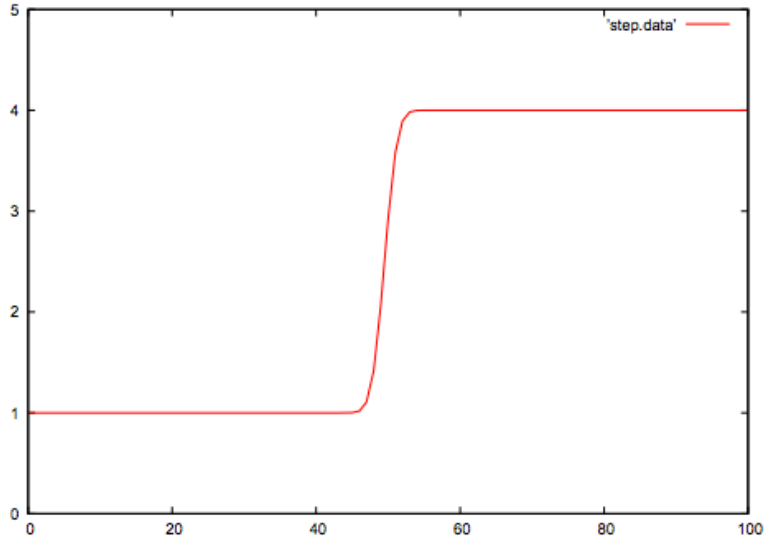
Source: D. Hoiem

Edge is Where Change Occurs

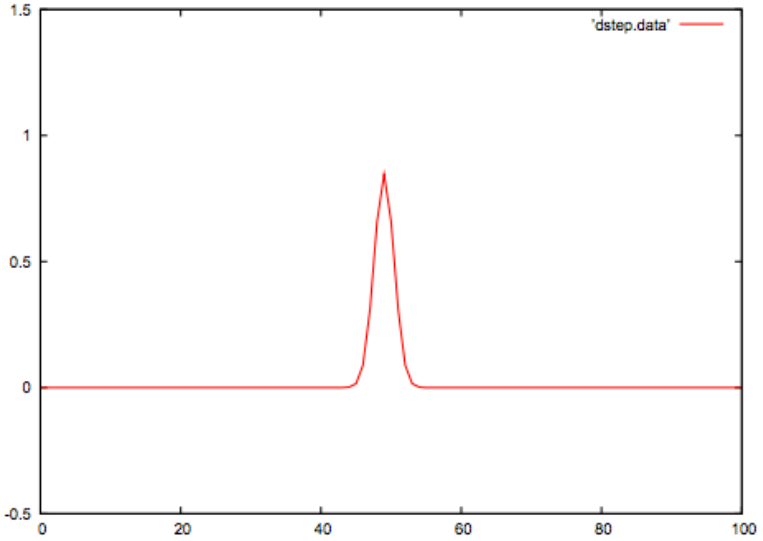
- Change is measured by derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.



Edge is Where Change Occurs



(a)



(b)

Characterizing edges

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

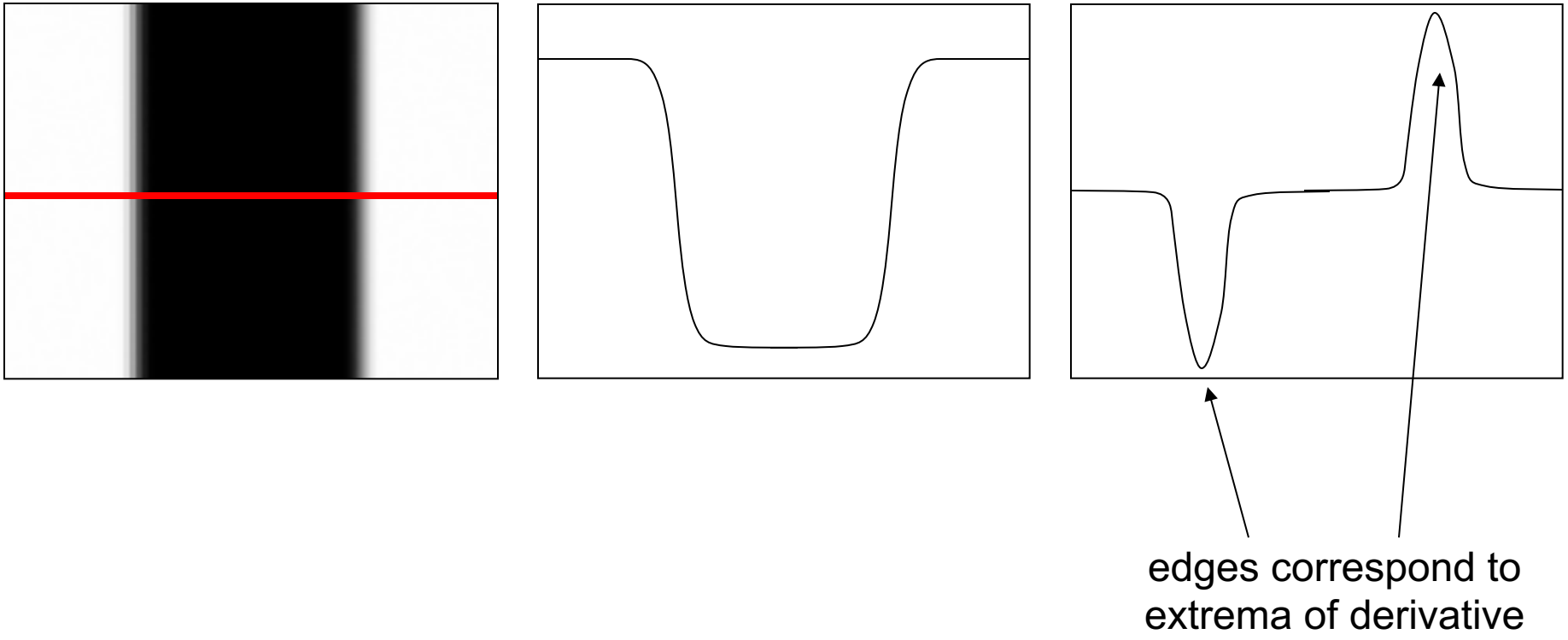
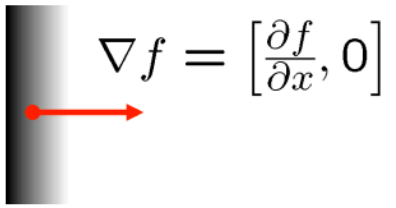
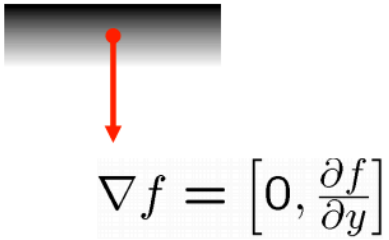


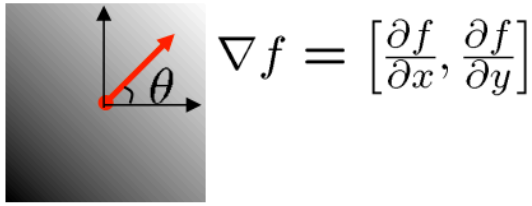
Image gradient

- 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 increase in intensity







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

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

The gradient direction is given by:

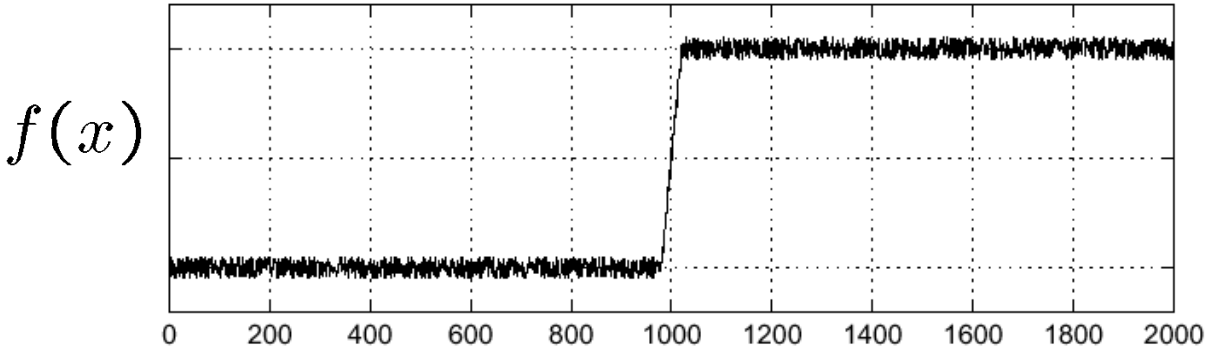
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

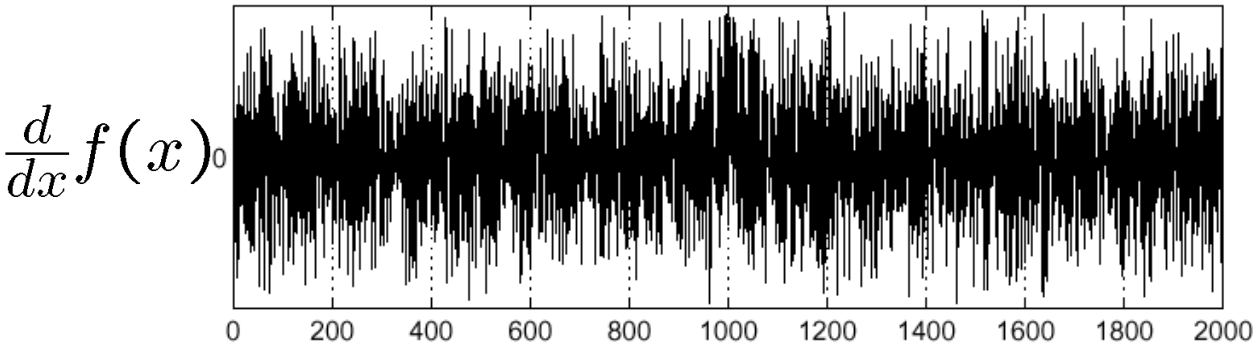
- how does this relate to the direction of the edge?

Source: 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?

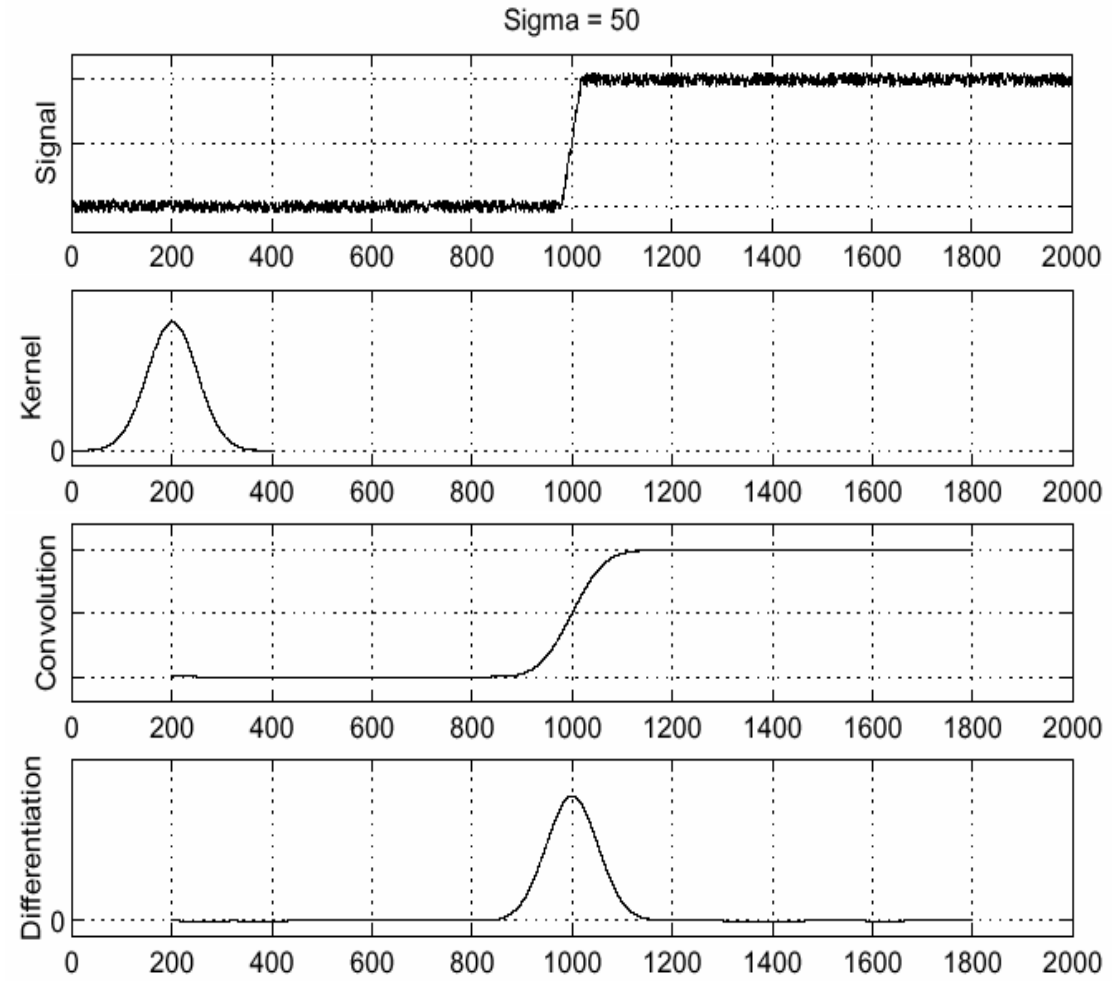
Source: S. Seitz

Effects of noise

- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What can we do about it?

Source: D. Forsyth

Solution: smooth first



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

Source: S. Seitz

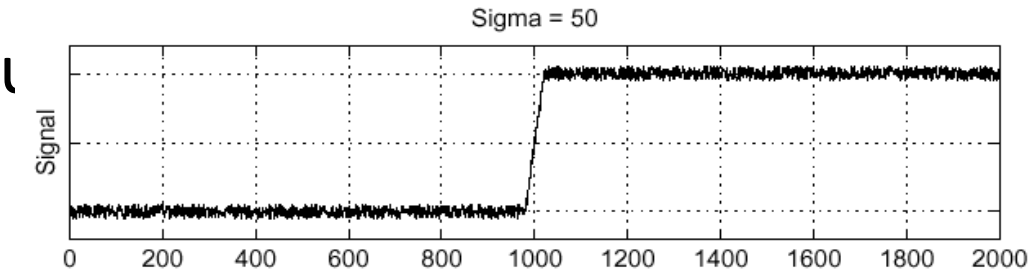
Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative:

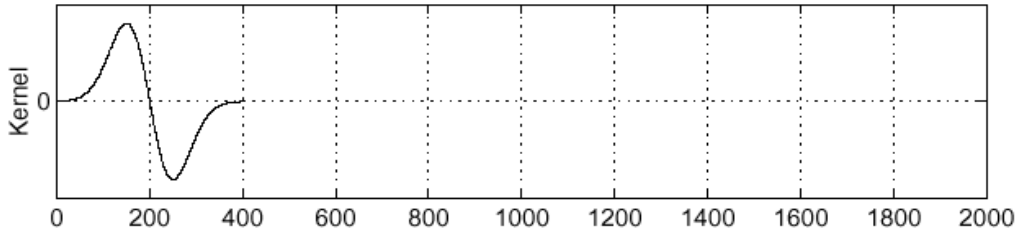
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves u

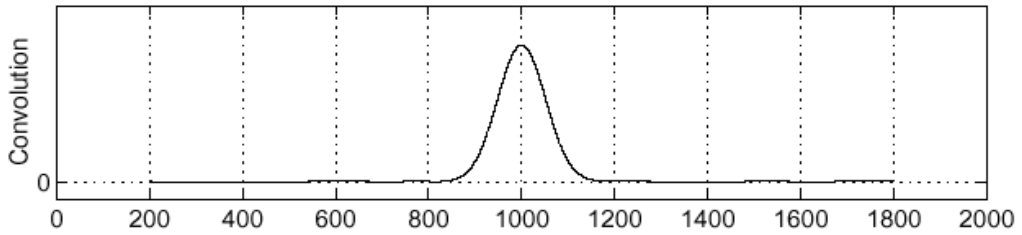
f



$\frac{d}{dx} g$

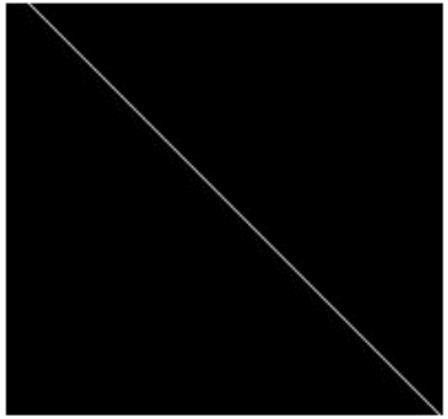


$f * \frac{d}{dx} g$



Source: S. Seitz

Image with Edge



Edge Location

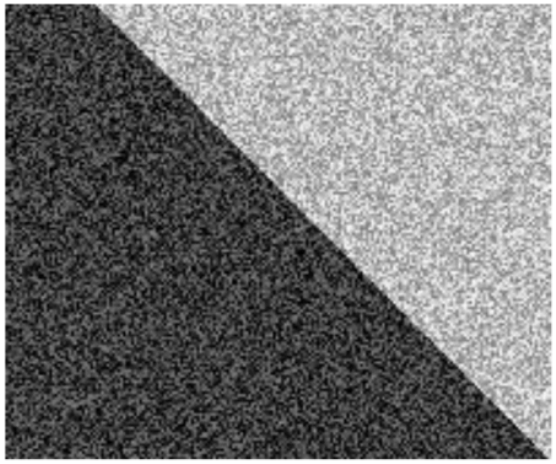
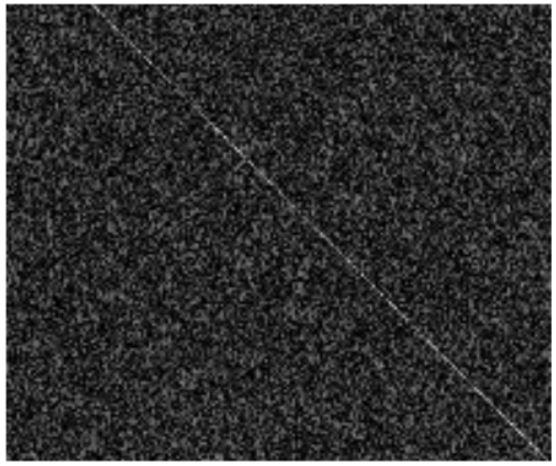
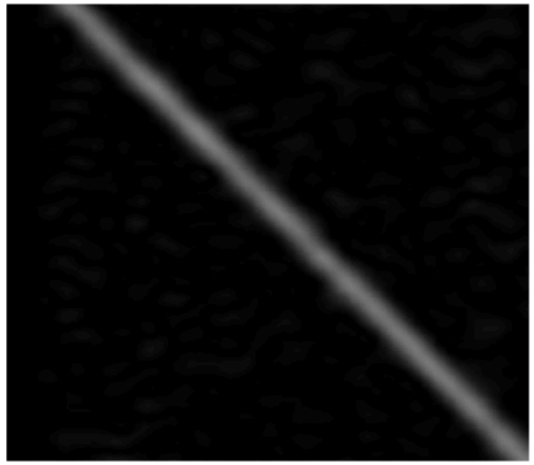


Image + Noise

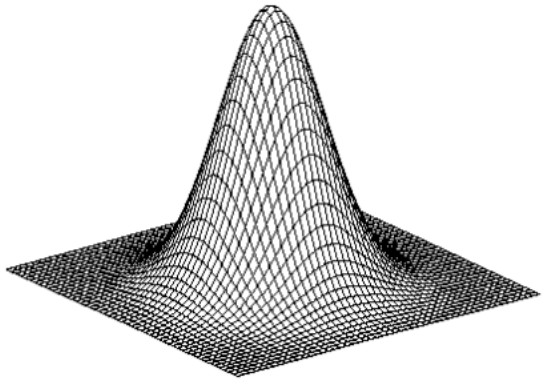


Derivatives detect
edge *and* noise

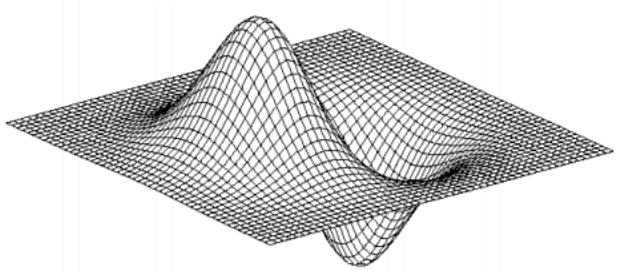


Smoothed derivative removes
noise, but blurs edge

2D edge detection filters



* [1 -1] =



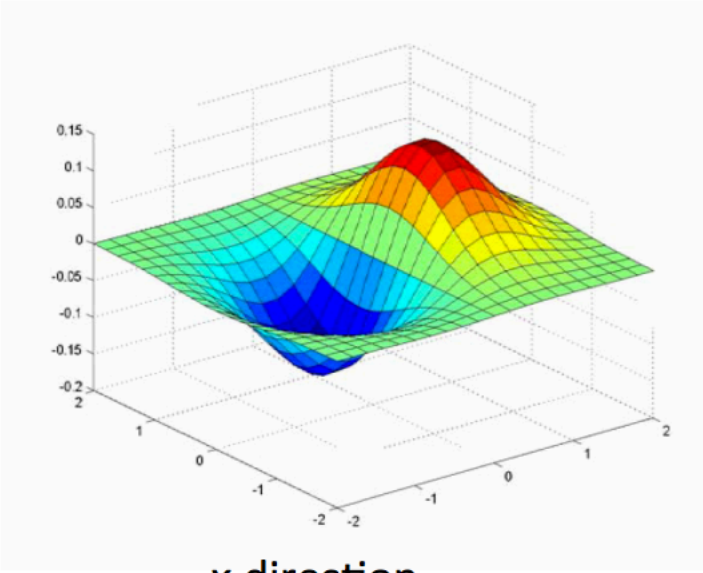
Gaussian

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

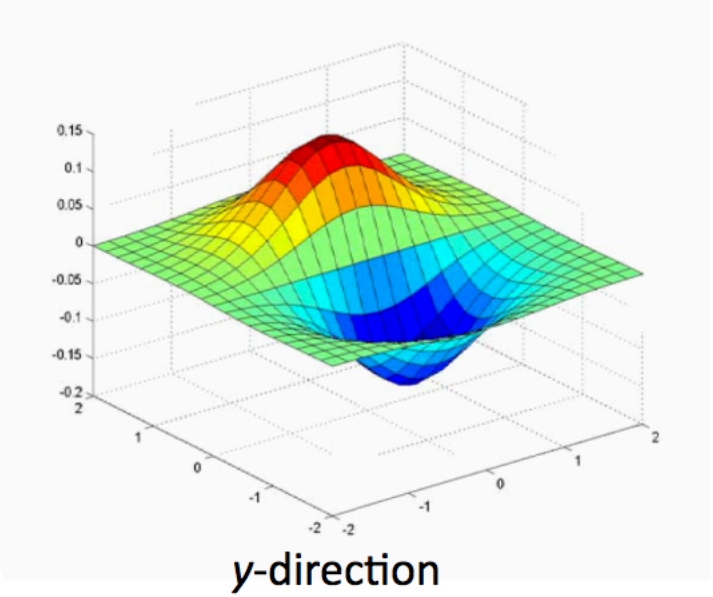
derivative of Gaussian (x)

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

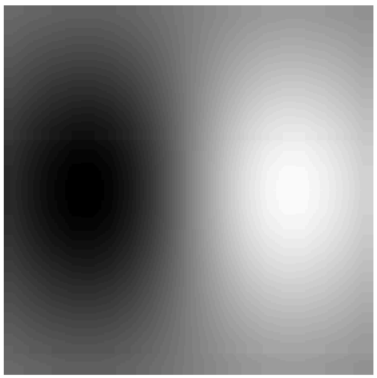
Derivative of Gaussian filter

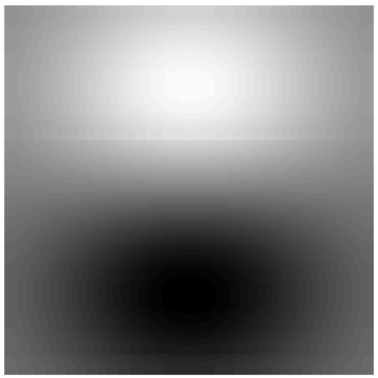


x-direction



y-direction





The Sobel operator

- Common approximation of derivative of Gaussian
  - A mask (not a convolution kernel)

$\frac{1}{8}$

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

s_x

$\frac{1}{8}$

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

s_y

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient magnitude

What kind of filters are those?

Sobel filter:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

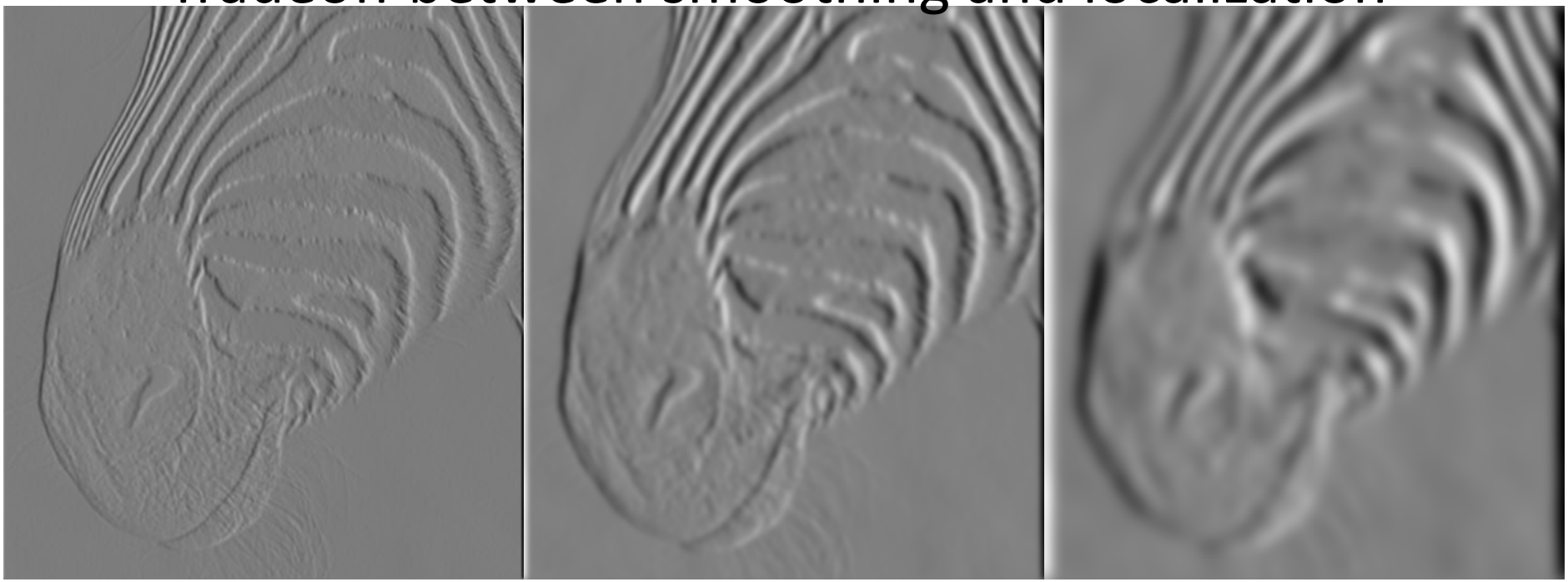
Magnitude:

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

Direction:

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

Tradeoff between smoothing and localization



1 pixel 3 pixels 7 pixels

FIGURE 5.3: The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (**left** to **right**). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.

Source: D. Forsyth

Original

σ= 1 pixel

σ= 2 pixel

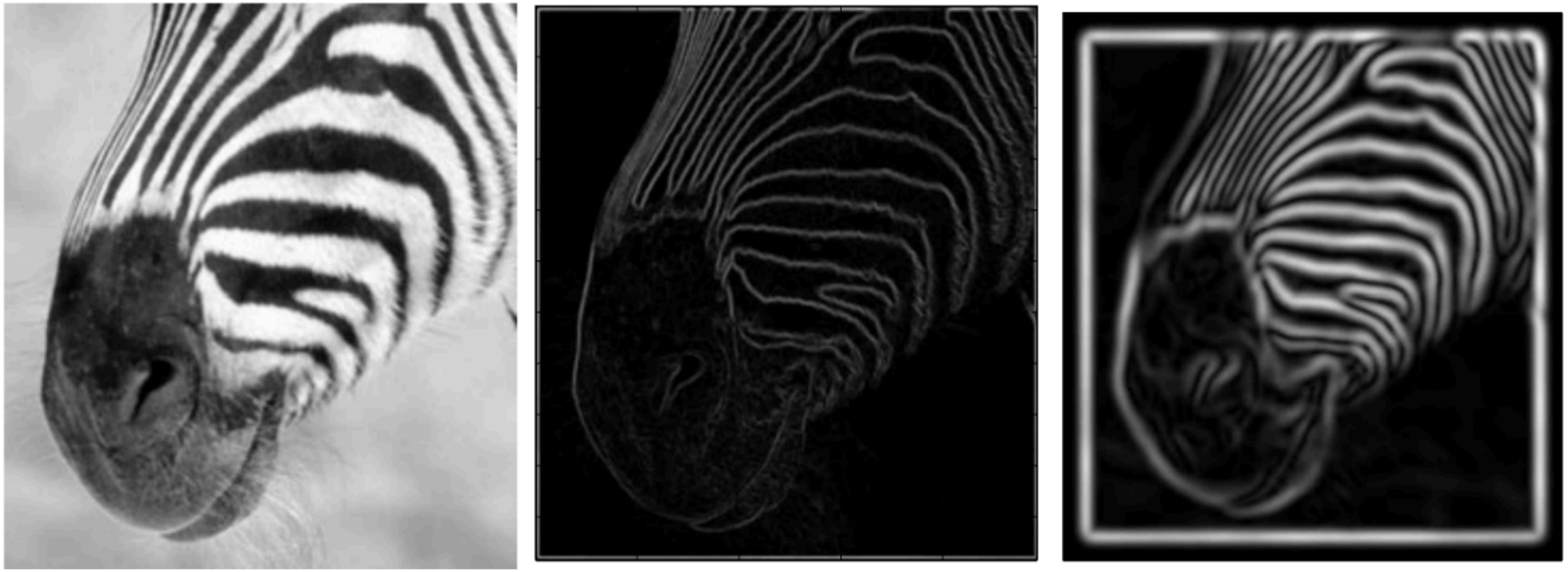


FIGURE 5.4: The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. At the **center**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 1$ pixel; and on the **right**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 2$ pixel. Notice that large values of the gradient magnitude form thick trails.

Source: D. Forsyth

Implementation issues

- The gradient magnitude is large along a thick “trail” or “ridge,” so how do we identify the actual edge points?
- How do we link the edge points to form curves?

Source: D. Forsyth

Designing an edge detector

- Criteria for a good edge detector:
  - **Good detection:** the optimal detector should find all real edges, ignoring noise or other artifacts
  - **Good localization**
    - the edges detected must be as close as possible to the true edges
    - the detector must return one point only for each true edge point
- Cues of edge detection
  - Differences in color, intensity, or texture across the boundary
  - Continuity and closure
  - High-level knowledge

Source: L. Fei-Fei

Canny edge detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

Canny edge detector

- Minimizes the probability of multiply detecting an edge
- Minimizes the probability of failing to detect an edge
- Minimizes the distance of the reported edge to the true edge.

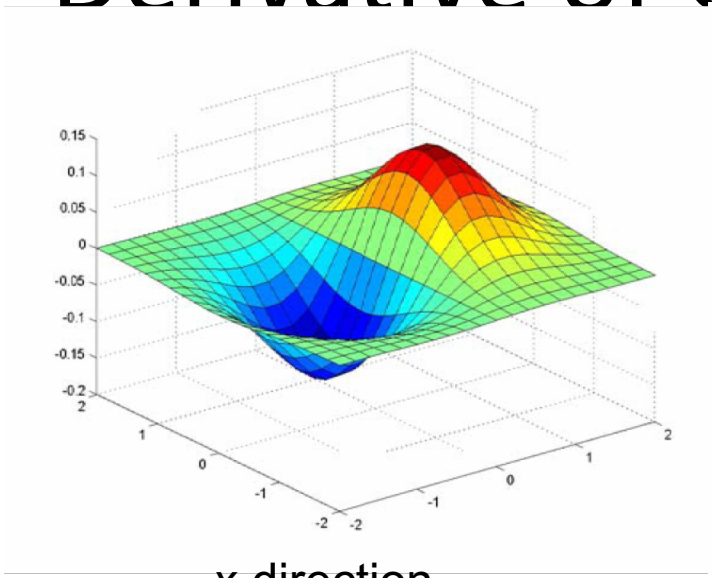
J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

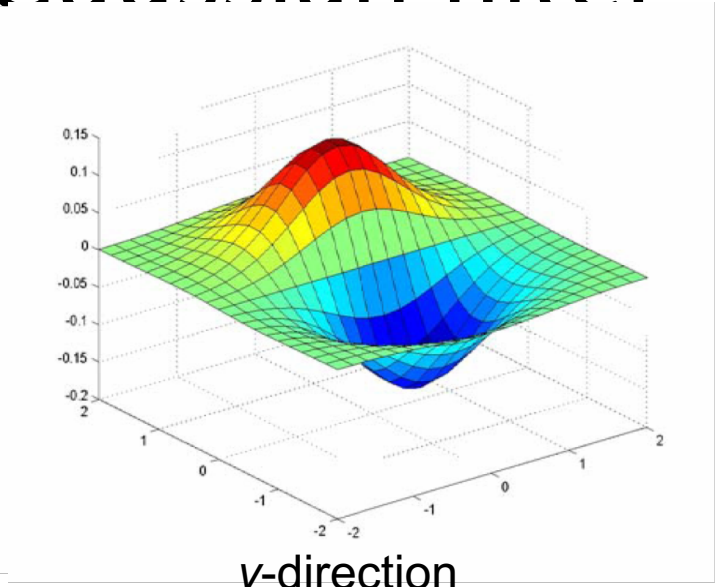
Example

original image (Lena)

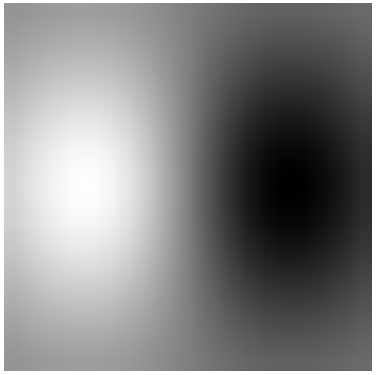
Derivative of Gaussian filter

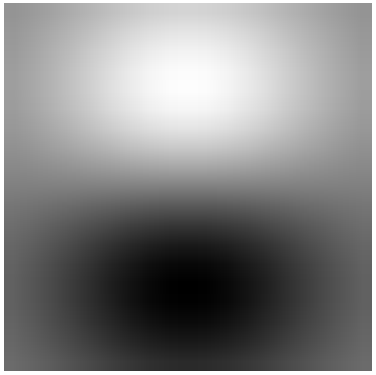


x-direction

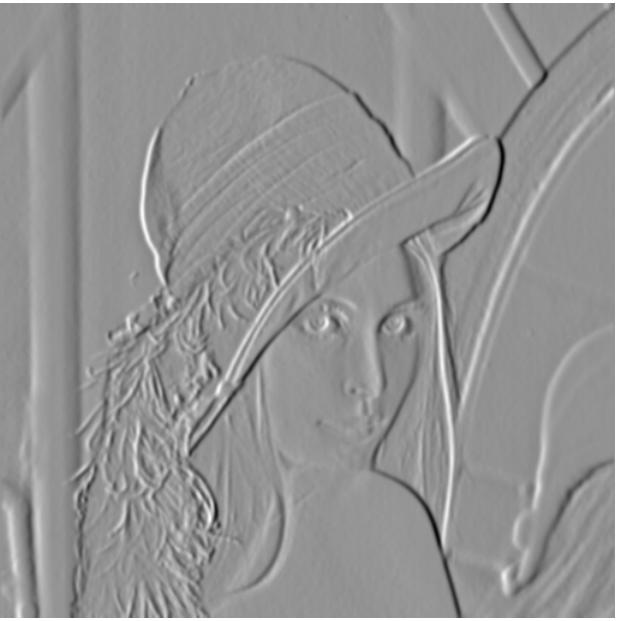


y-direction





Compute Gradients (DoG)



X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

The Canny edge detector

- original image (Lena)

The Canny edge detector

norm of the gradient

The Canny edge detector

thresholding

The Canny edge detector

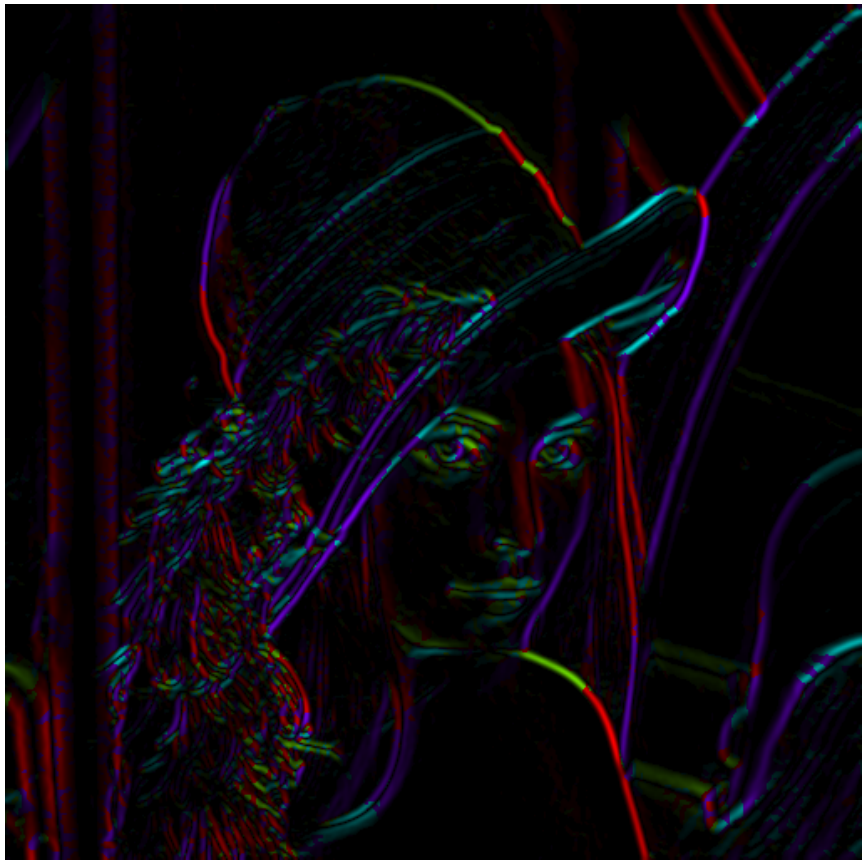
thinning

(non-maximum suppression)

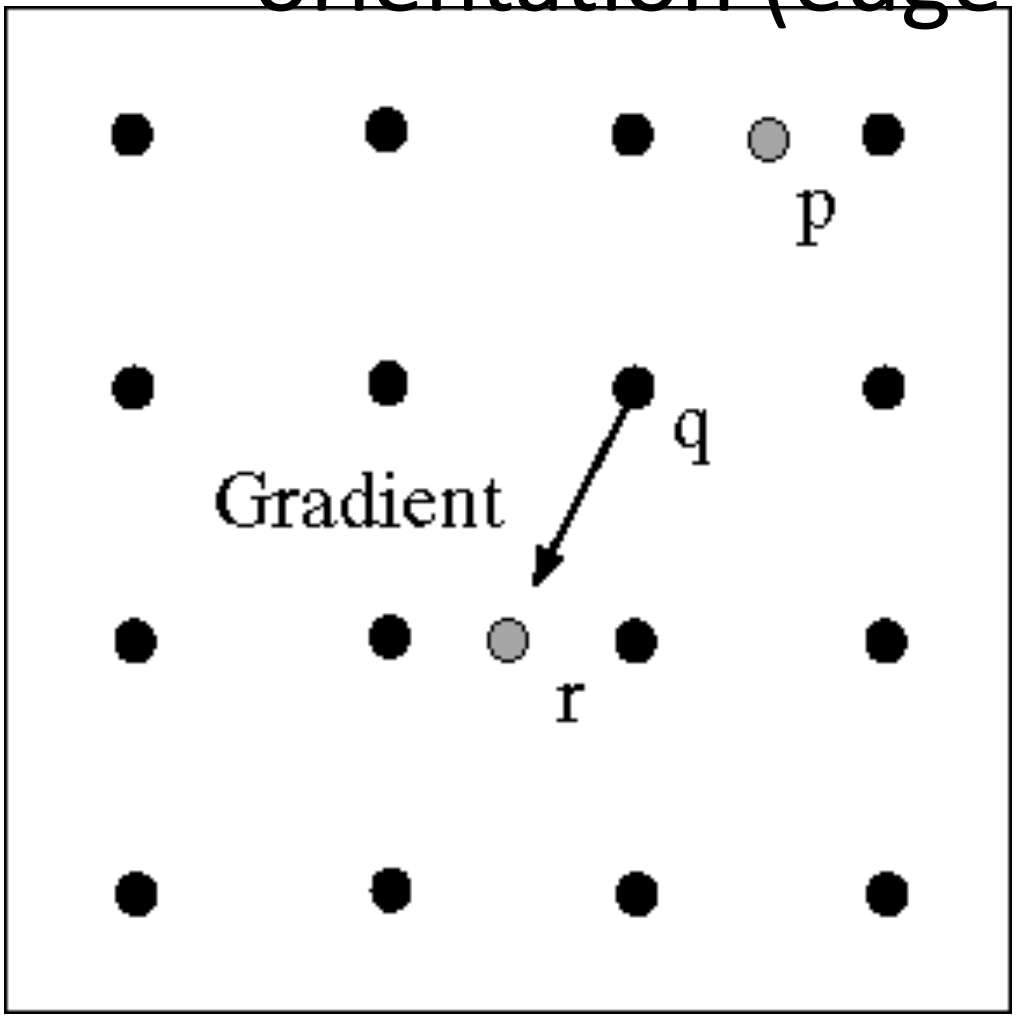
Get Orientation at Each Pixel

- Threshold at minimum level
- Get orientation

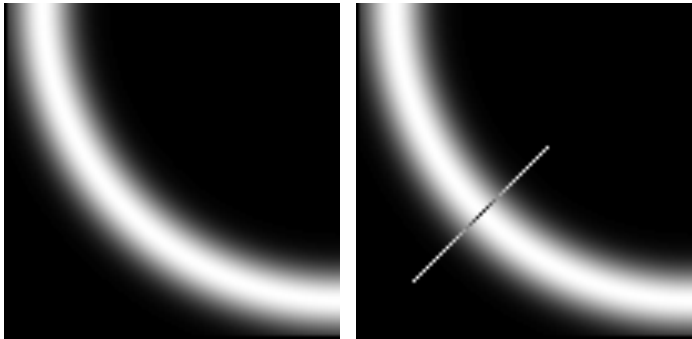
theta = atan2(gy, gx)



Non-maximum suppression for each orientation (edge thinning)

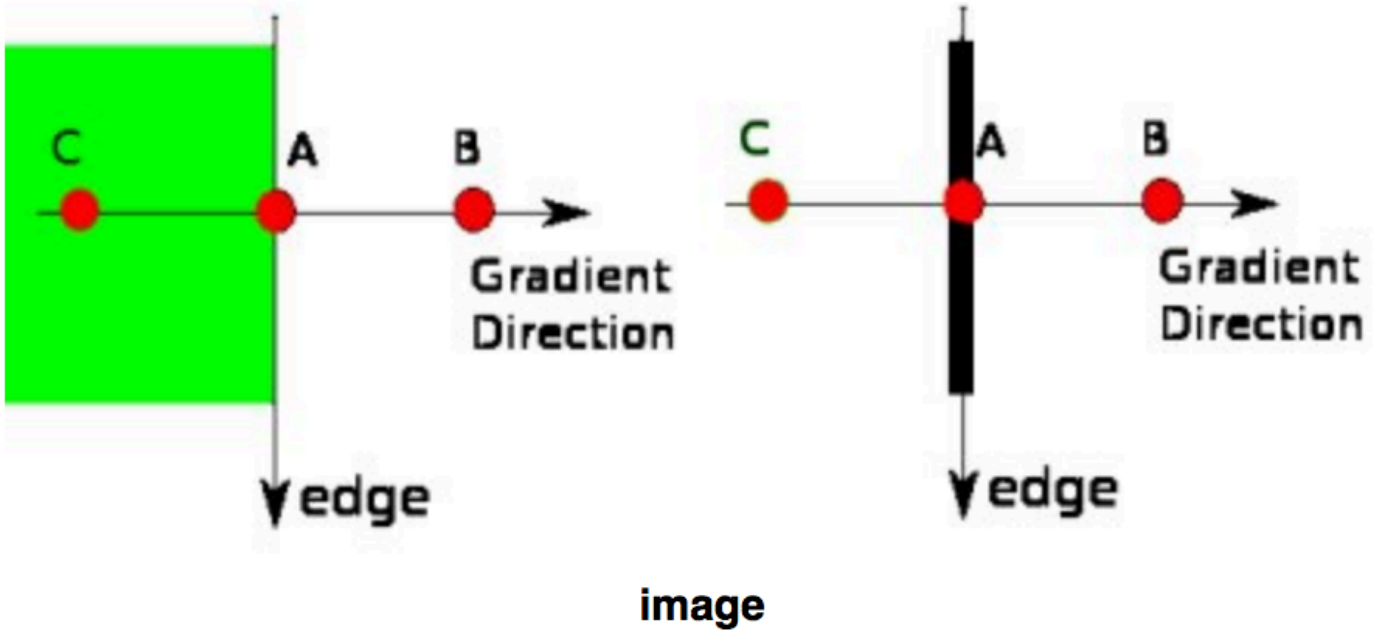


At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.



Source: D. Forsyth

Non-maximum suppression for each orientation



Non-maximum suppression

- m(x,y) is the local peak
- Canny calls local peak detection non-maximum suppression

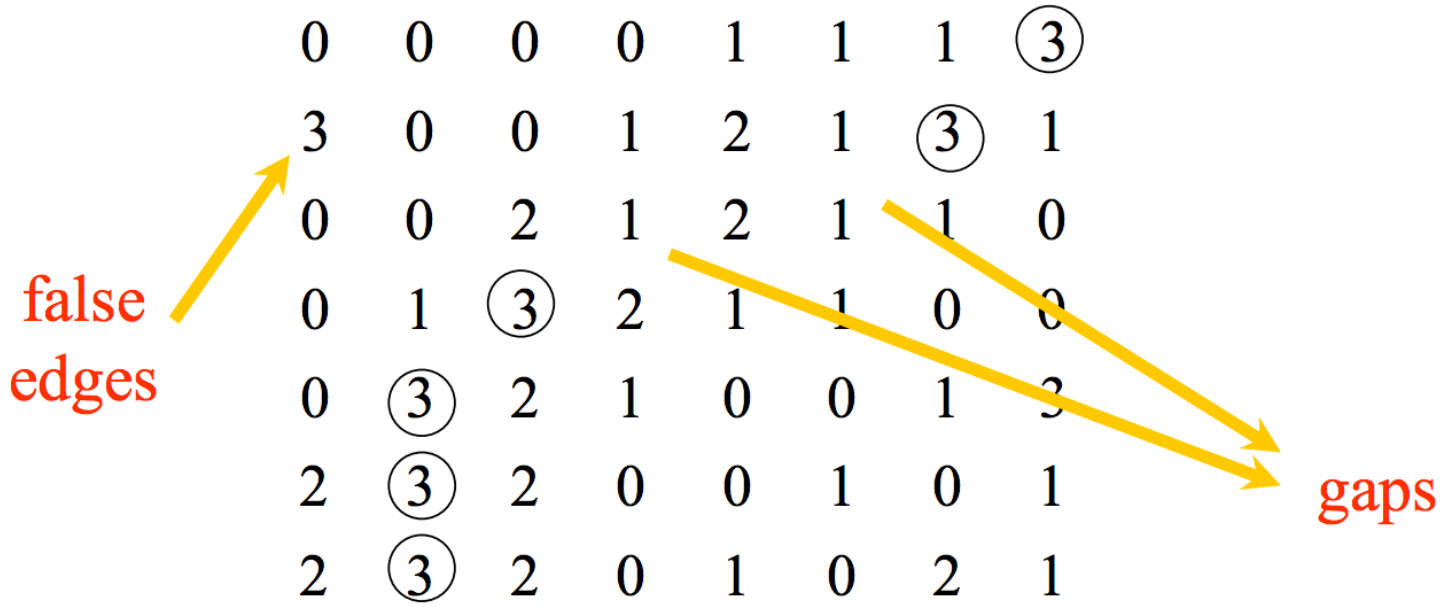
$$m(x, y) > m(x + \delta x, y + \delta y),$$

$$m(x, y) > m(x - \delta x, y - \delta y).$$

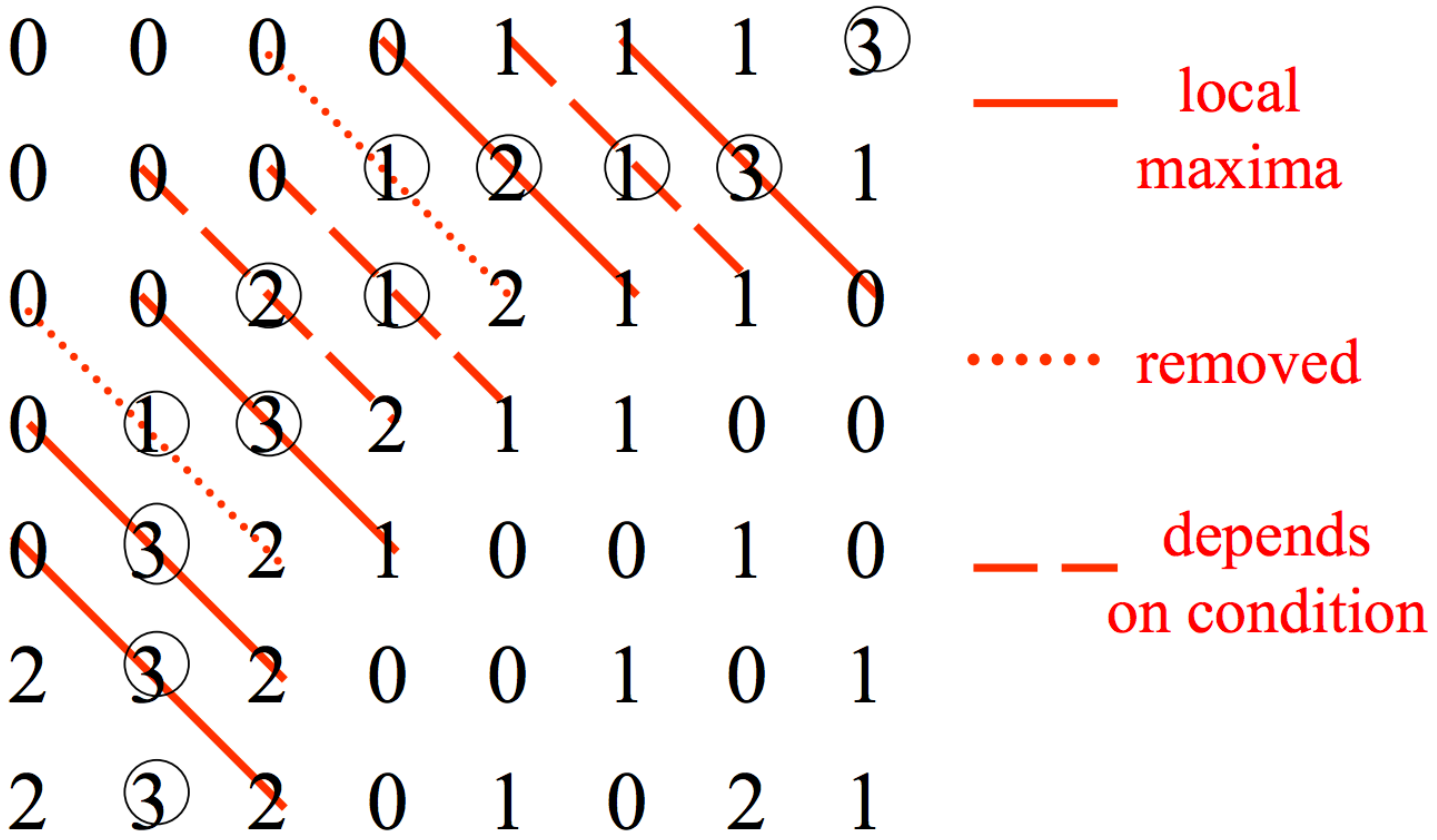
M(x,y) is a local peak whenever it is greater than the values in the gradient direction and the opposite of the gradient direction.

Non-maximum suppression

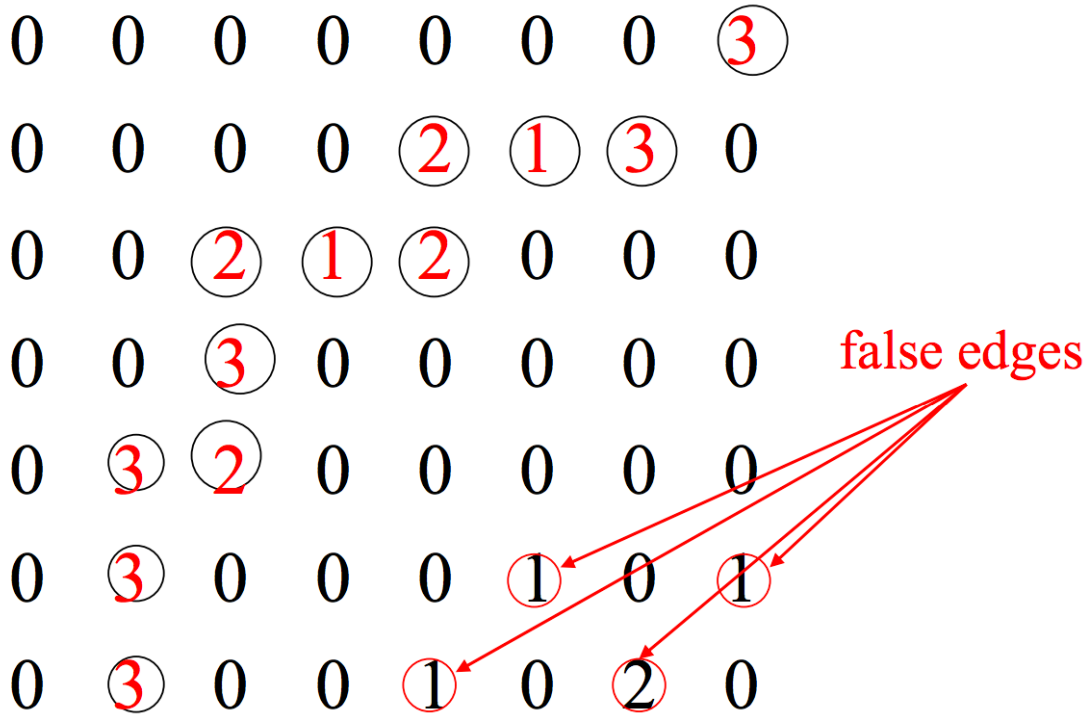
- Thin the broad ridges in M[i,j] into ridges that are only one pixel wide
- Find local maxima in M[i,j] by suppressing all values along the line of the Gradient that are not peak values of the ridge



Non-maximum suppression



Non-maximum suppression



- The suppressed magnitude image will contain many false edges caused by noise or fine texture

Before Non-max Suppression

After non-max suppression

Double thresholding

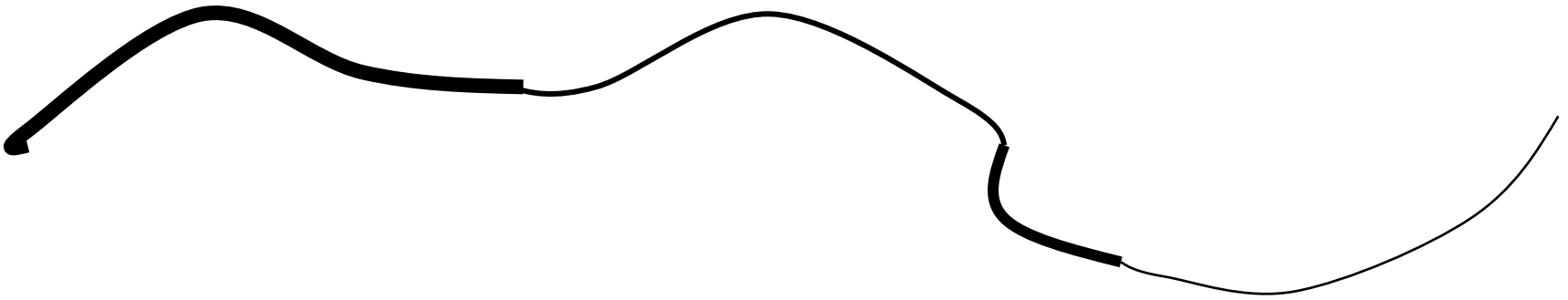
- The remaining pixels are then categorized into strong, weak and non-edges.
- Above a high threshold, mark as strong.
- Between high and lo, mark as weak.
- Below lo threshold, mark as non-edges

Hysteresis thresholding

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

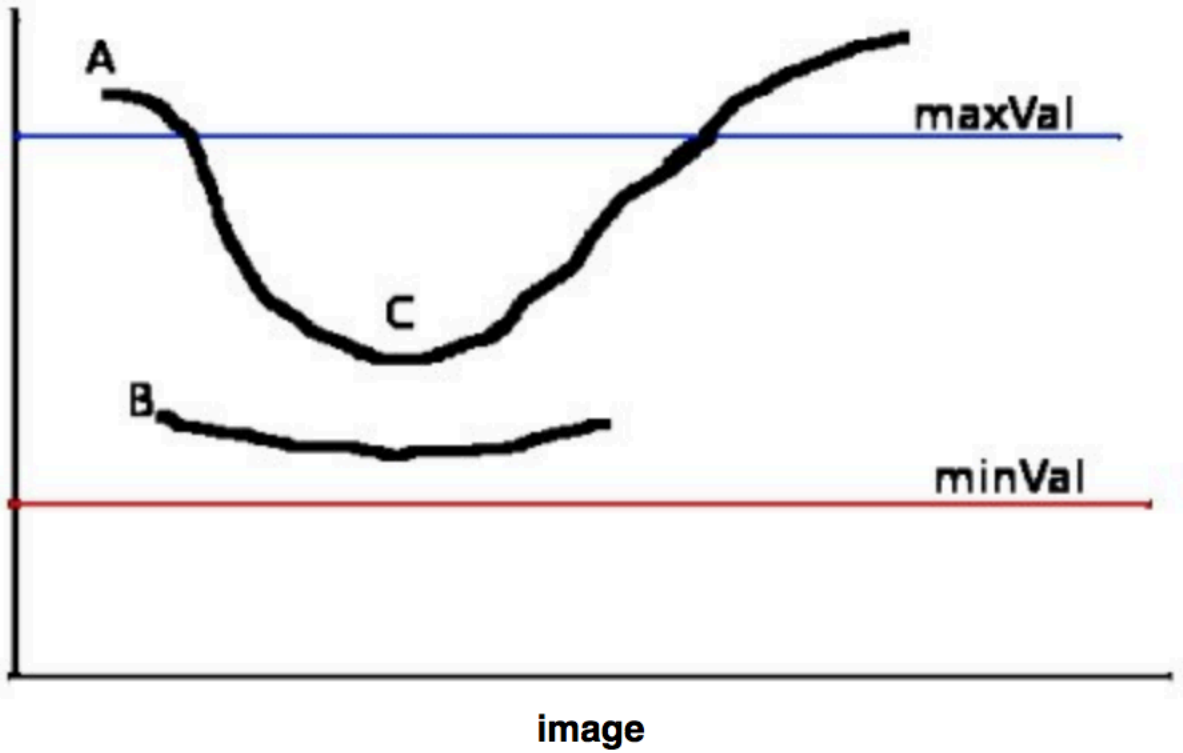
Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

Hysteresis thresholding



Final Canny Edges

Canny Edge Operator

- Smooth image I with 2D Gaussian: $G * I$
- Find local edge normal directions for each pixel

$$\overline{\mathbf{n}} = \frac{\nabla(G * I)}{\left|\nabla(G * I)\right|}$$

- Compute edge magnitudes $\left|\nabla(G * I)\right|$
- Find the location of the edges by finding zero-crossings along the edge normal directions (**non-maximum suppression**)

$$\frac{\partial^2(G * I)}{\partial \overline{\mathbf{n}}^2} = 0$$

- Threshold edges in the image with **hysteresis** to eliminate spurious responses

Read Canny's original paper for further details

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Source: S. Seitz

Canny detection with Python

1. Read in an image and filter with a Gaussian filter.
2. Apply sobel filter to x and y direction.
3. Computer gradients magnitudes and orientation.
4. Performing non-maximum suppression upper threshold
5. Perform threading ad labeling edge maps.

1. Compare your result with :

   cv2.Canny2()

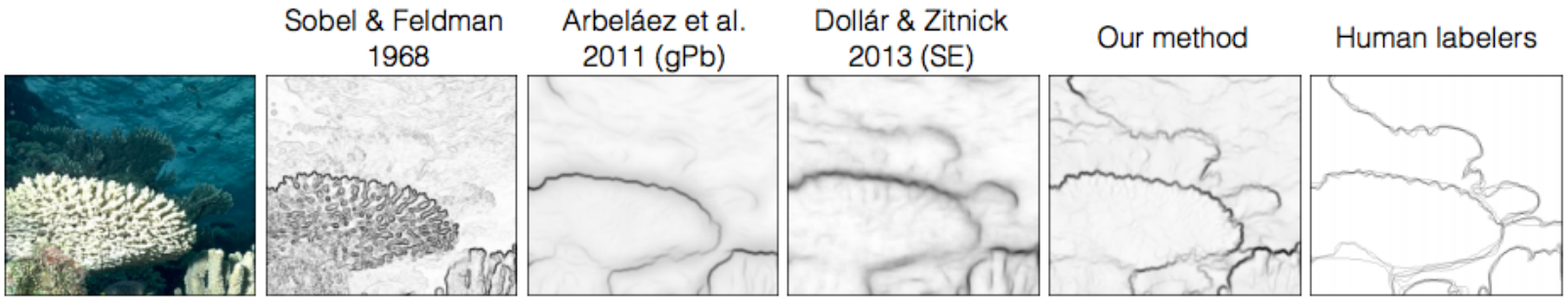
http://dasl.unlv.edu/daslDrexel/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html

Take-home reading

- Szeliski Chapter 4.2 Edges
- A very good introduction of Canny edge detection:
- http://www.classes.cs.uchicago.edu/archive/2005/fall/35040-1/edges.pdf

- A youtube video is also helpful:
- https://www.youtube.com/watch?v=-Z3kr26Eci4

Edge detection vs. boundary detection

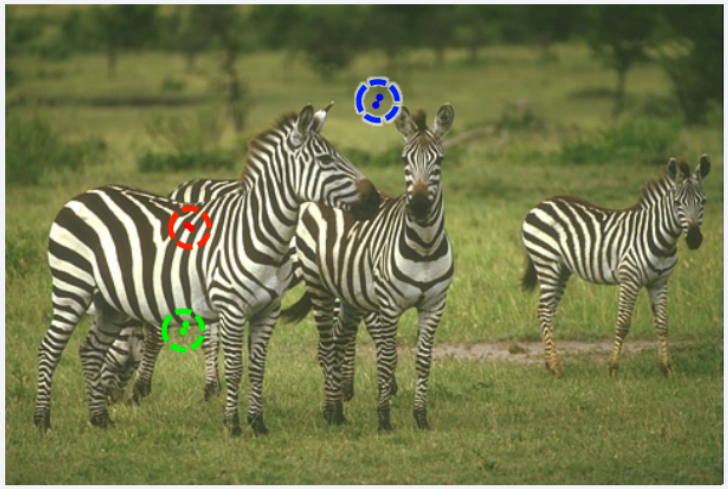


1. Classical methods use local derivative filters with fixed scales and only a few orientations. Tend to emphasize small and unimportant edges.
2. Contemporary methods uses multiple scales, multiple feature from image patches (color, textures, intensity). Using statistical methods (give each pixel a probability of being a boundary) to learn boundaries.
3. Isola et al. (2014) uses mutual information between pixels to detect boundary.

Key observation: *Pixels belonging to the same object have higher statistical association than pixels belonging to different objects.*

Slide courtesy from Philip Isola

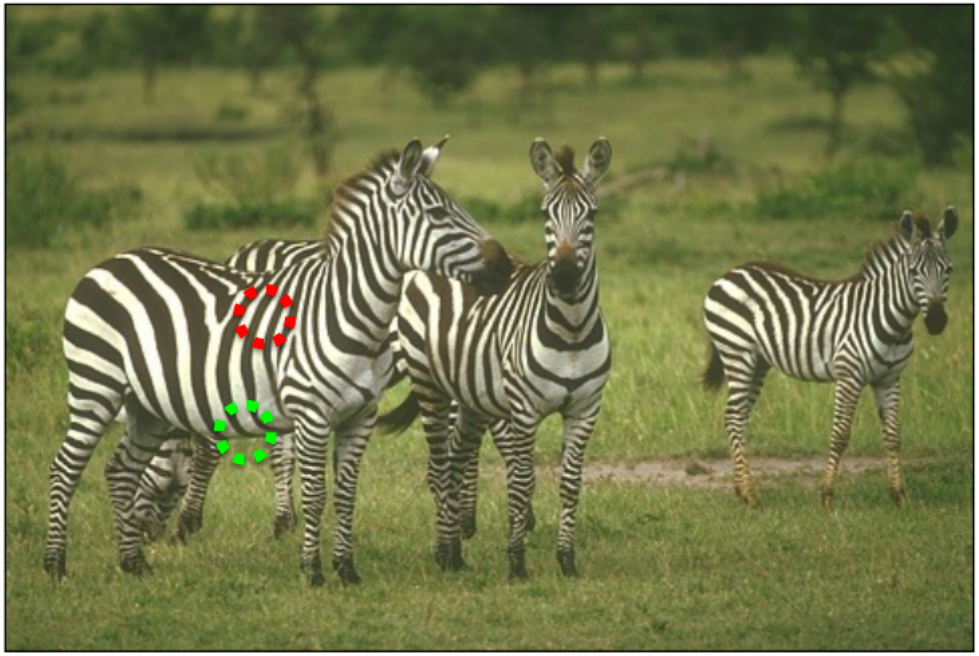
Point-wise mutual information reveals object structure

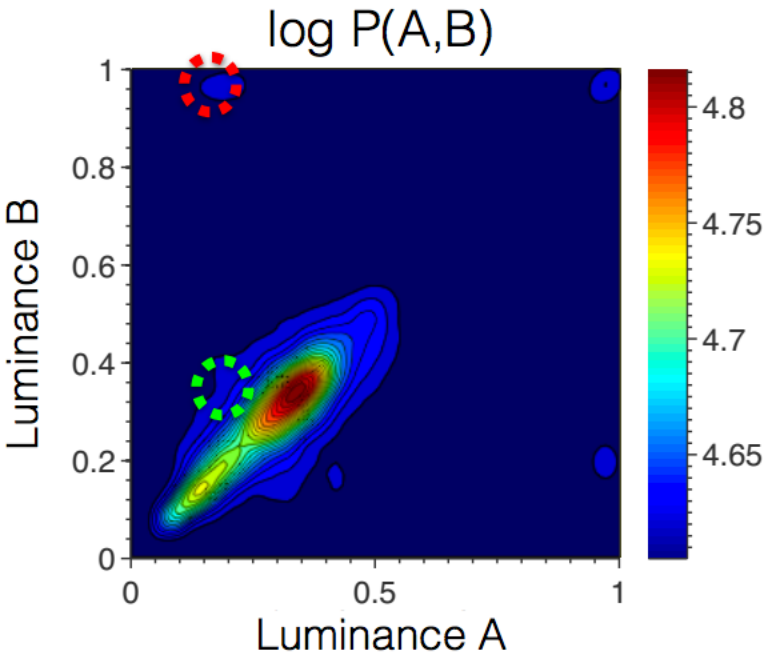


Above, black-next-to-white occurs over and over again. This pattern shows up in the image's statistics as a *suspicious coincidence* — these colors must be part of the same object!

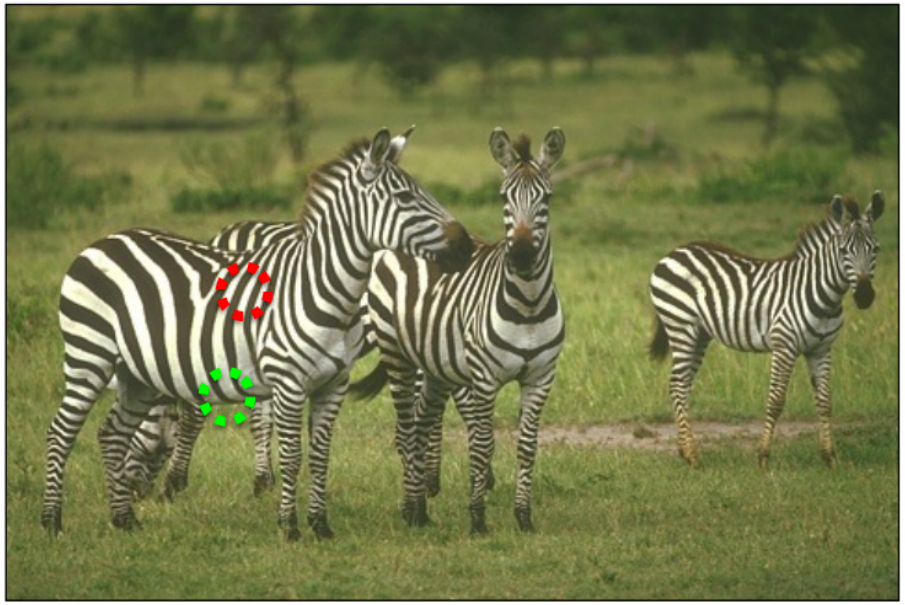
How do we distinguish the red and the green patches?

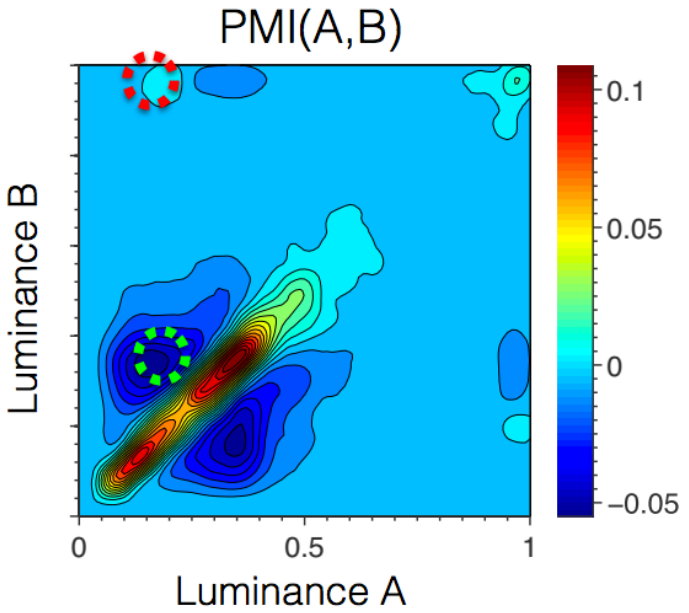
Slide courtesy from Philip Isola





$P(A, B) =$ how often each color A occurs next to each color B *within this image*.





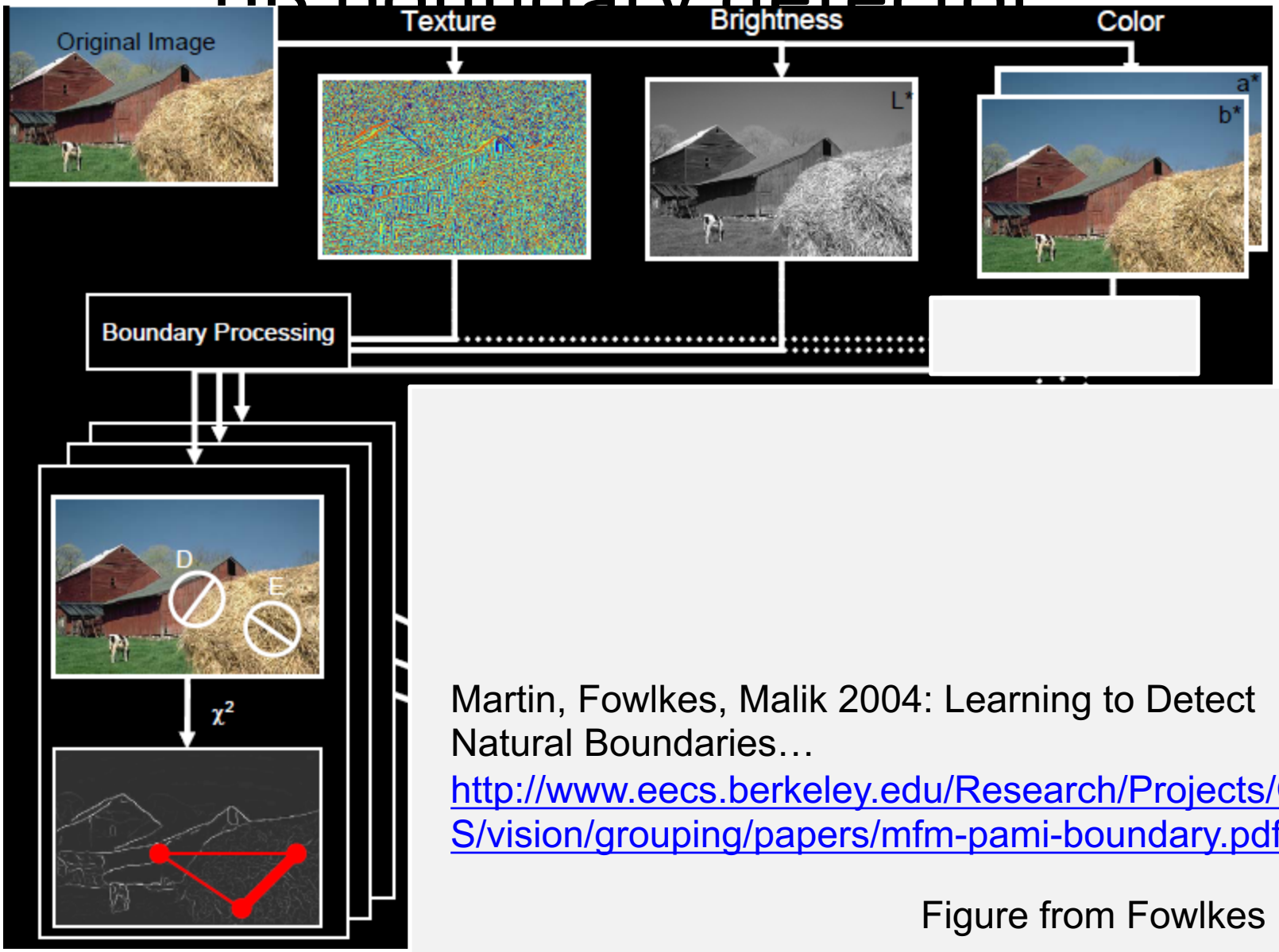
Pointwise mutual information (PMI)

$$\mathrm{PMI}_\rho(A,B) = \log \frac{P(A,B)^\rho}{P(A)P(B)}$$

Use PMI as affinity measure for affinity-based pixel grouping.

How much more likely is observing A given that we saw B in the same local region, compared to the base rate of observing A in the image.

pB boundary detector



Martin, Fowlkes, Malik 2004: Learning to Detect Natural Boundaries…
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/mfm-pami-boundary.pdf

Figure from Fowlkes

pB Boundary Detector

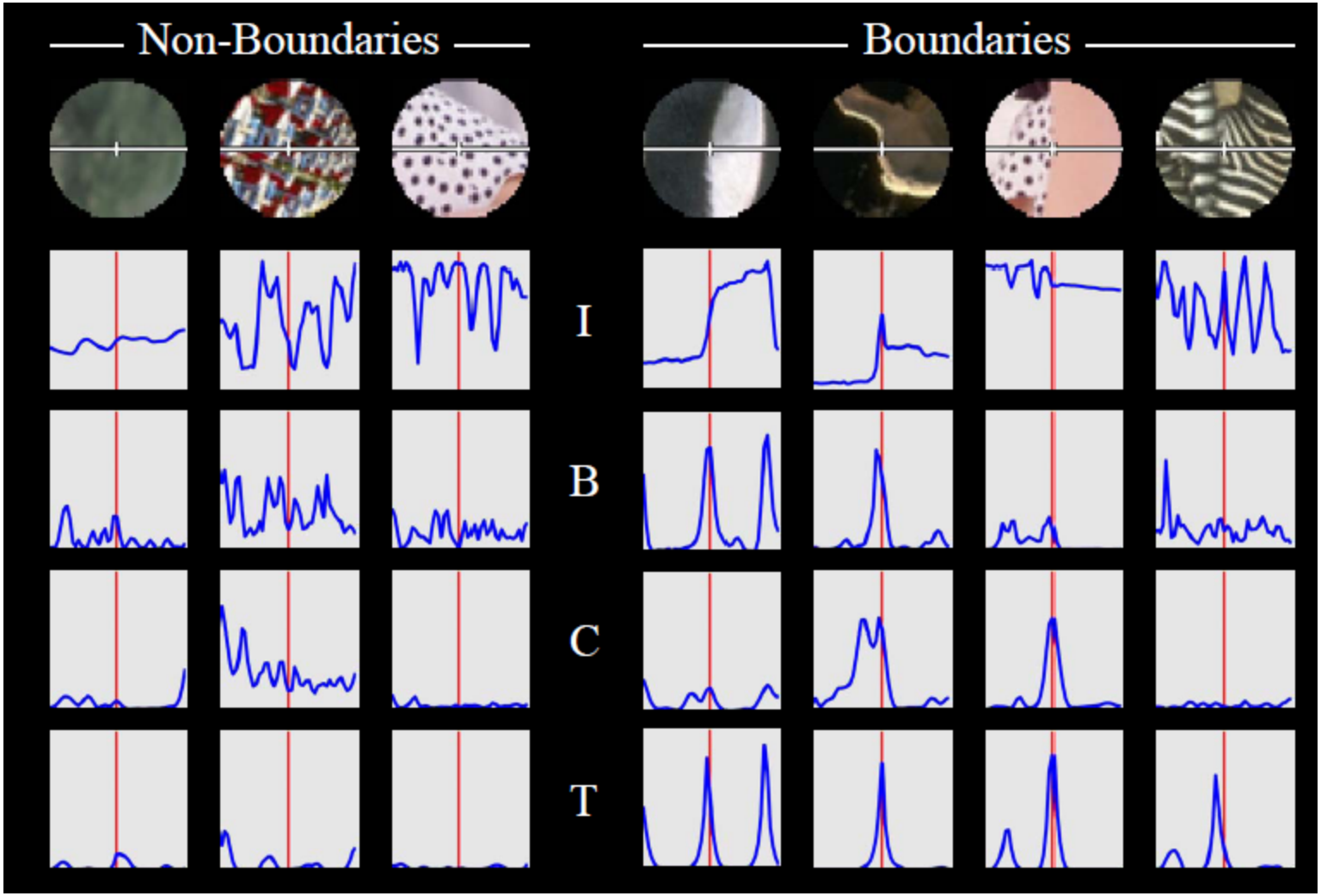


Figure from Fowlkes

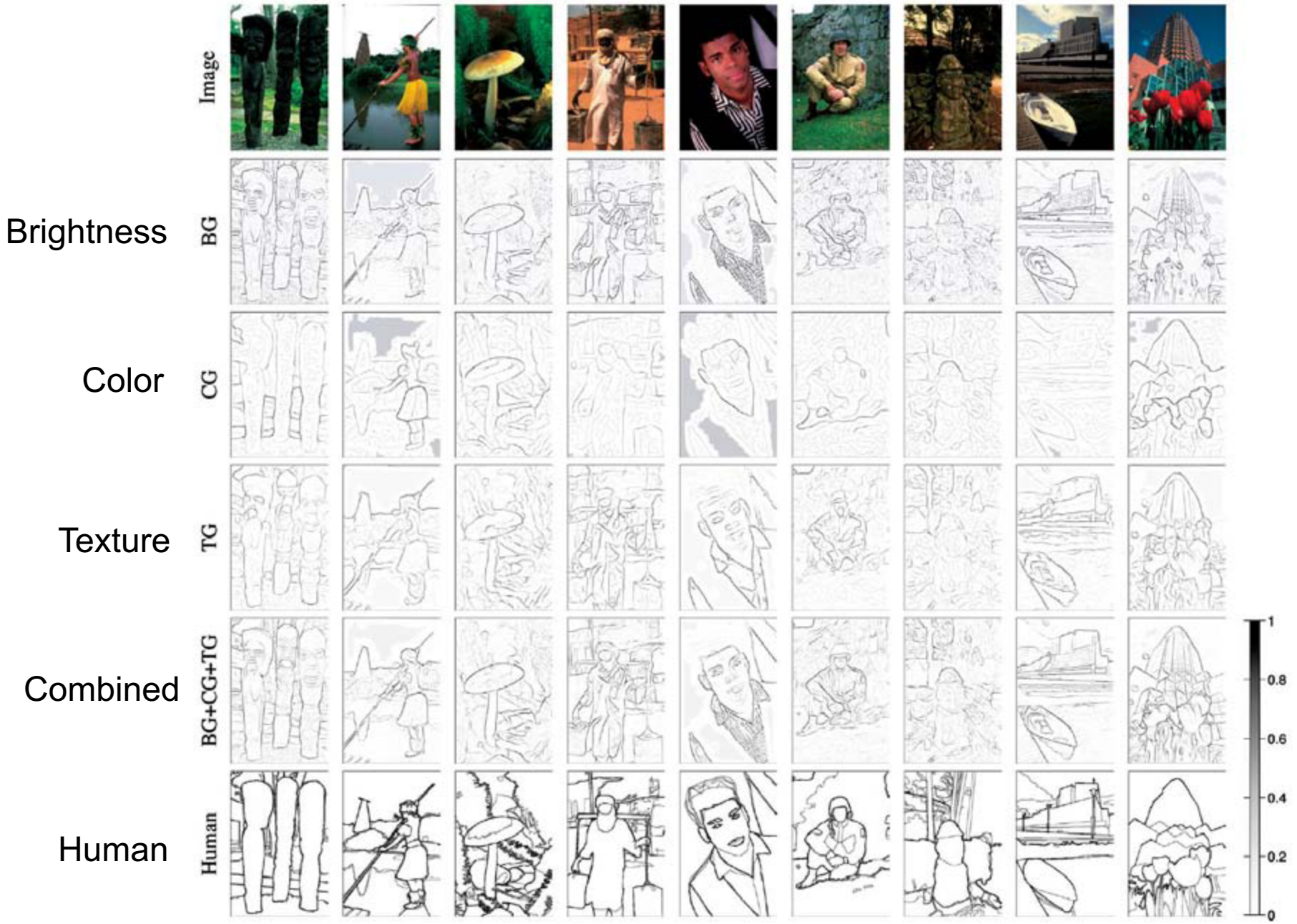
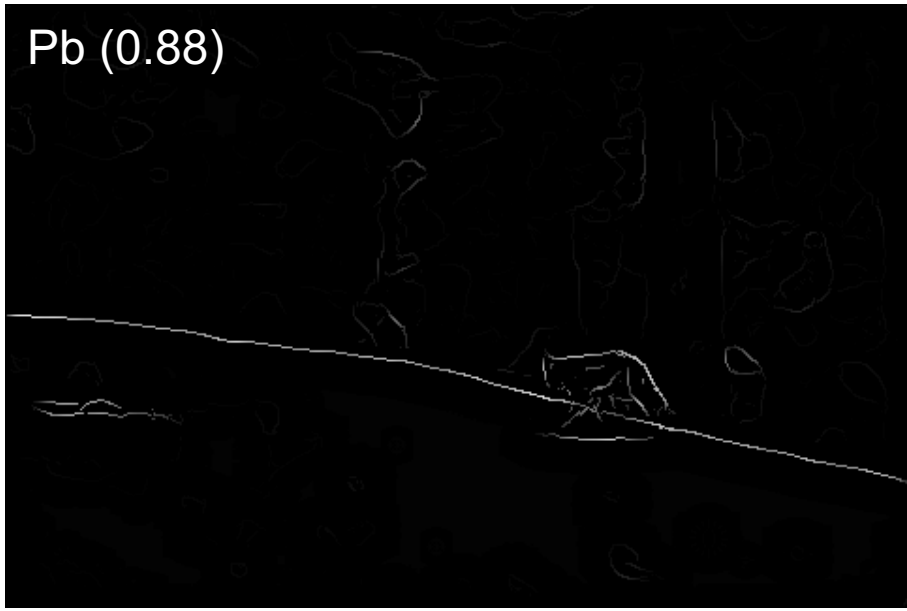
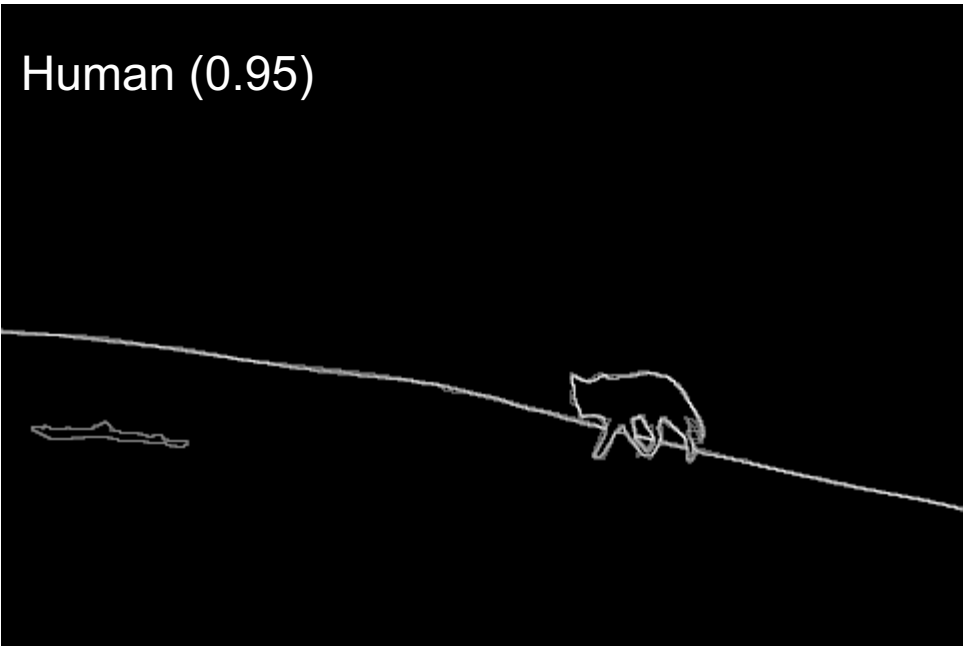


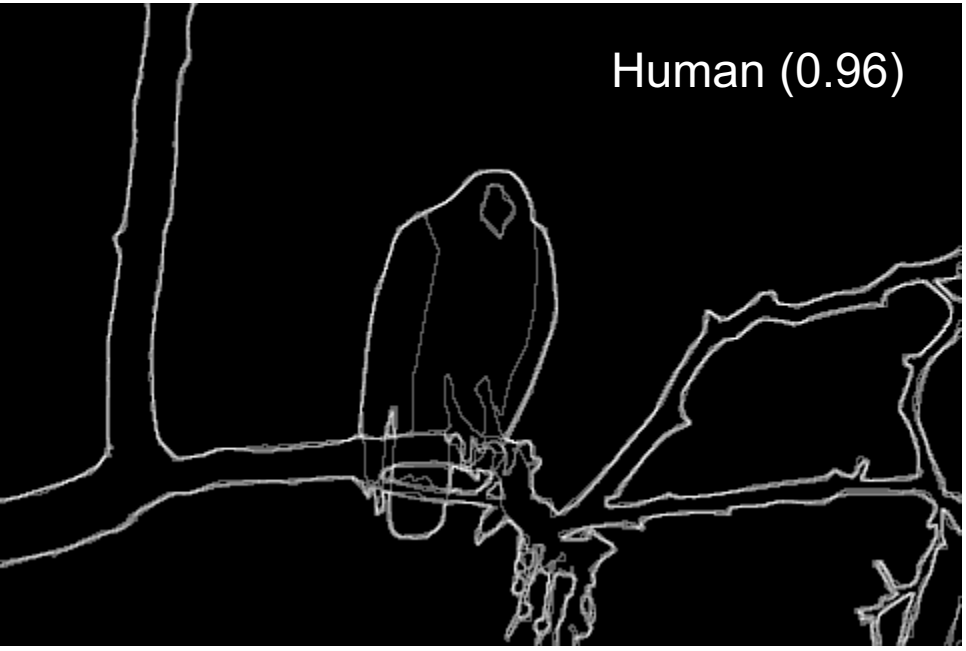
Image
BG
CG
TG
BG+CG+TG
Human
Brightness
Color
Texture
Combined
Human
1
0.8
0.6
0.4
0.2
0

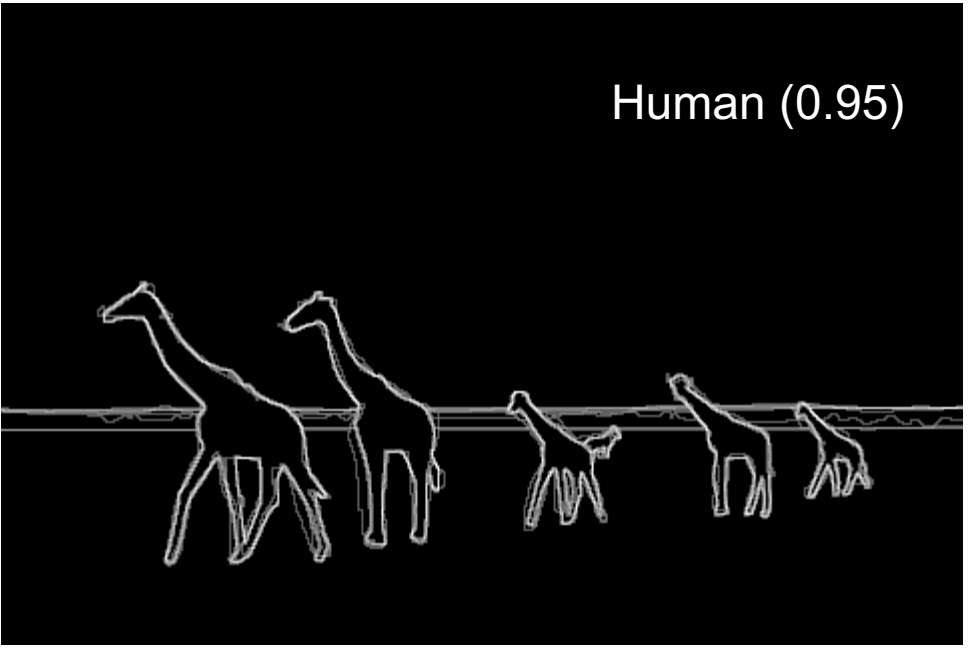
Results



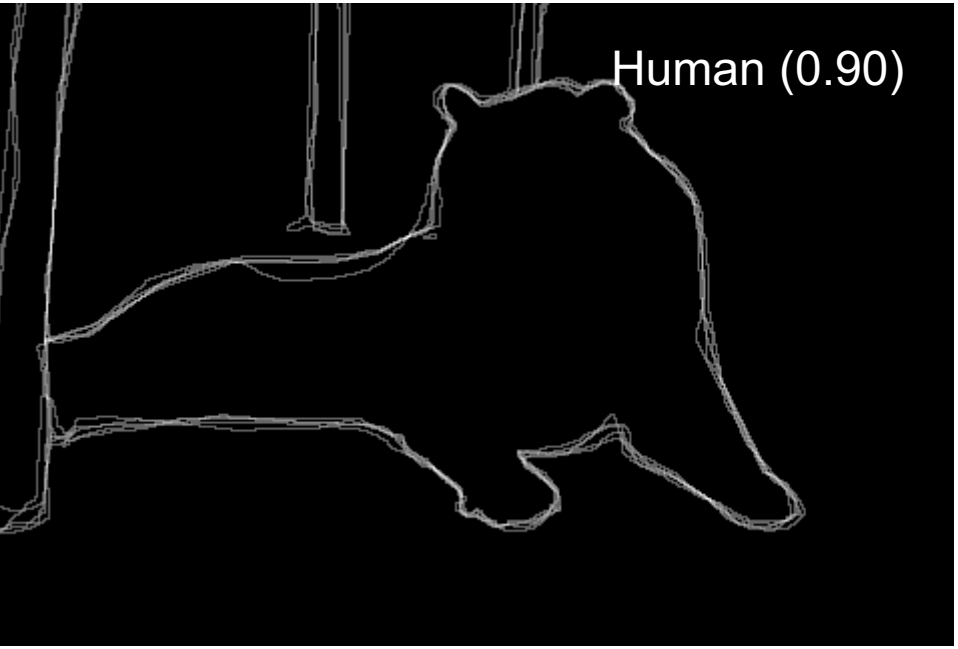


Results



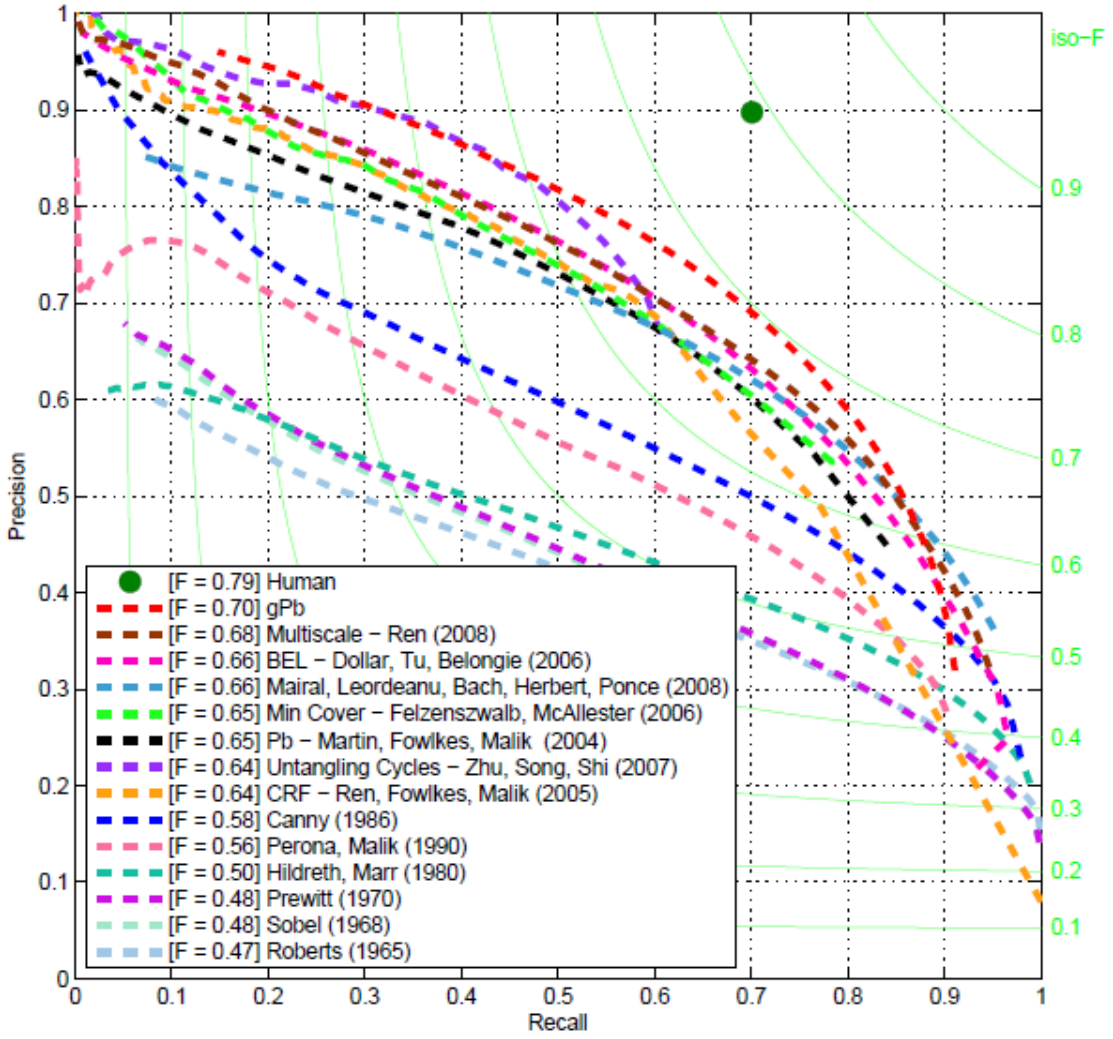

Human (0.95)


Pb (0.63)



For more:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/bench/html/108082-color.html

45 years of boundary detection

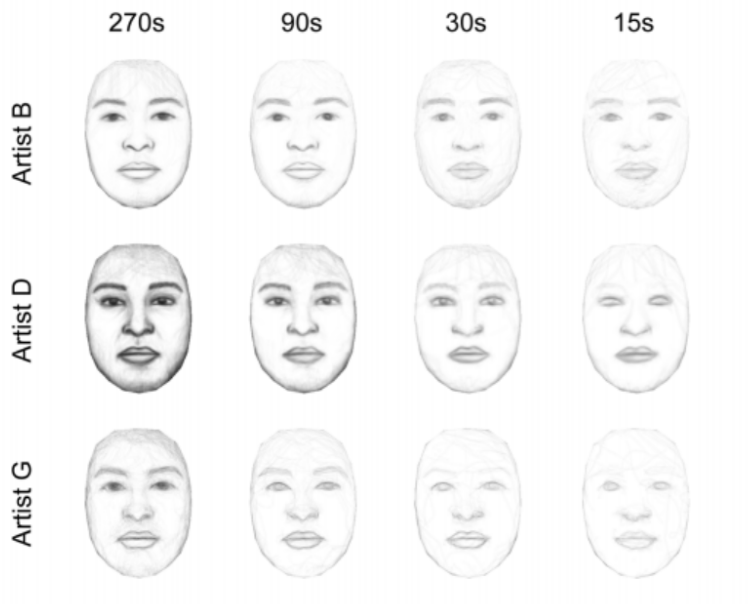


Source: Arbelaez, Maire, Fowlkes, and Malik. TPAMI 2011 (pdf)

State of edge detection

- Local edge detection works well
  - But many false positives from illumination and texture edges
- Some methods to take into account longer contours, but could probably do better
- Few methods that actually “learn” from data. Your project 5, Sketch Tokens, will do so.
- Poor use of object and high-level information

Style and abstraction in portrait sketching, Berger et al. SIGGRAPH 2013



- Learn from artist's strokes so that edges are more likely in certain parts of the face.

Next class

- Laplacian of Gaussian
- Steerable filter
- Fourier transform

- Reading: Chapter 3.1-3.3! Very important to keep up the reading.

- You should have read Chapter 1. Skip Chapter 2 since we haven’t covered it.