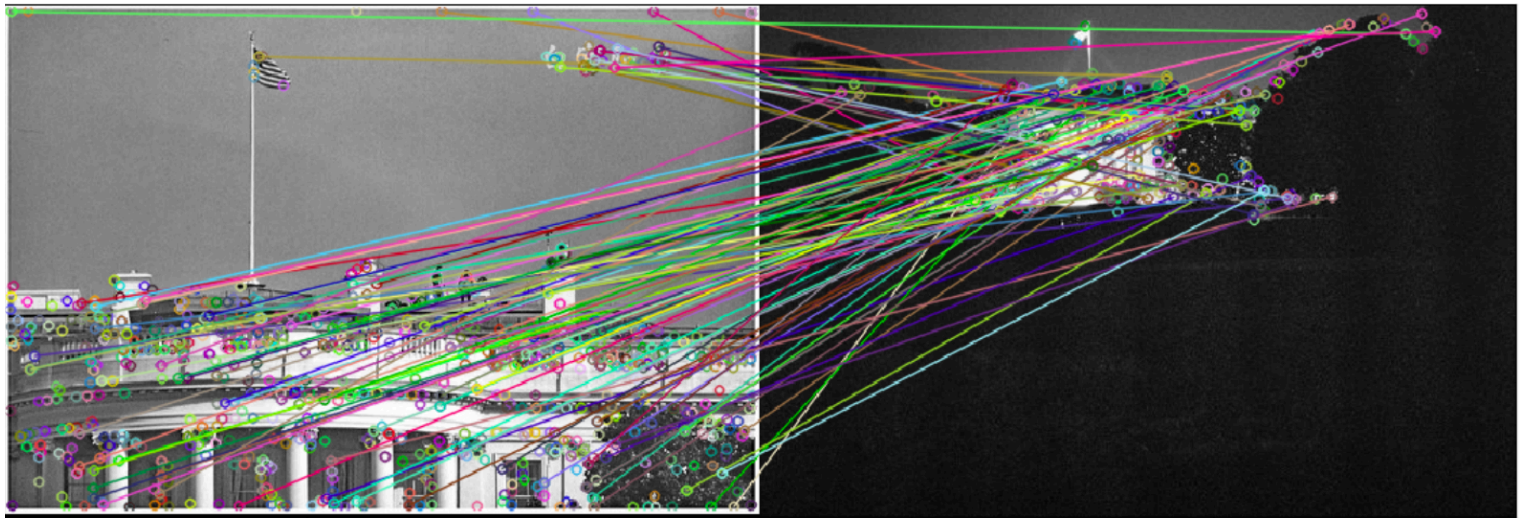
Computer Vision : Insight of object detection algorithms :

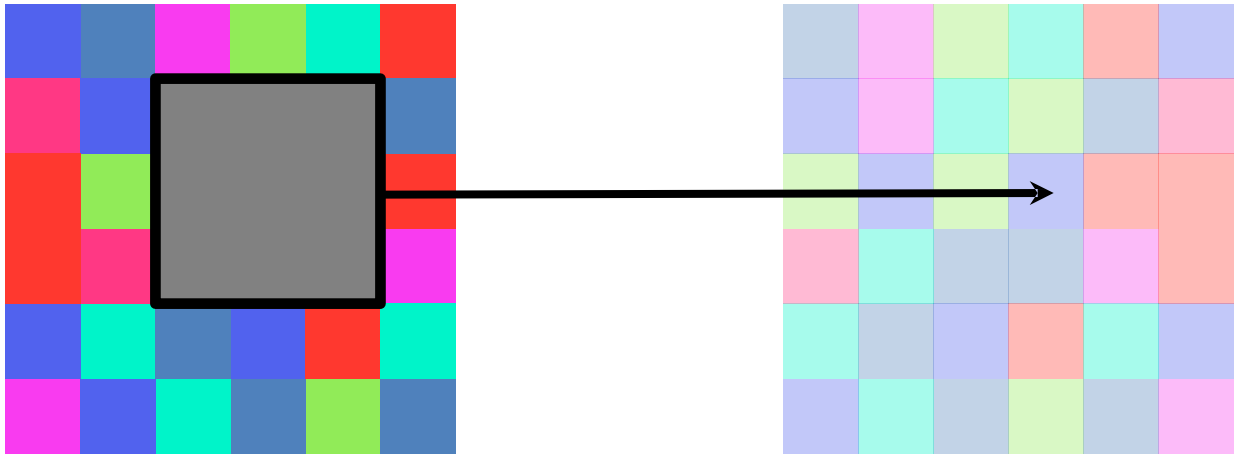
Lesson2 Image Advanced Preprocessing



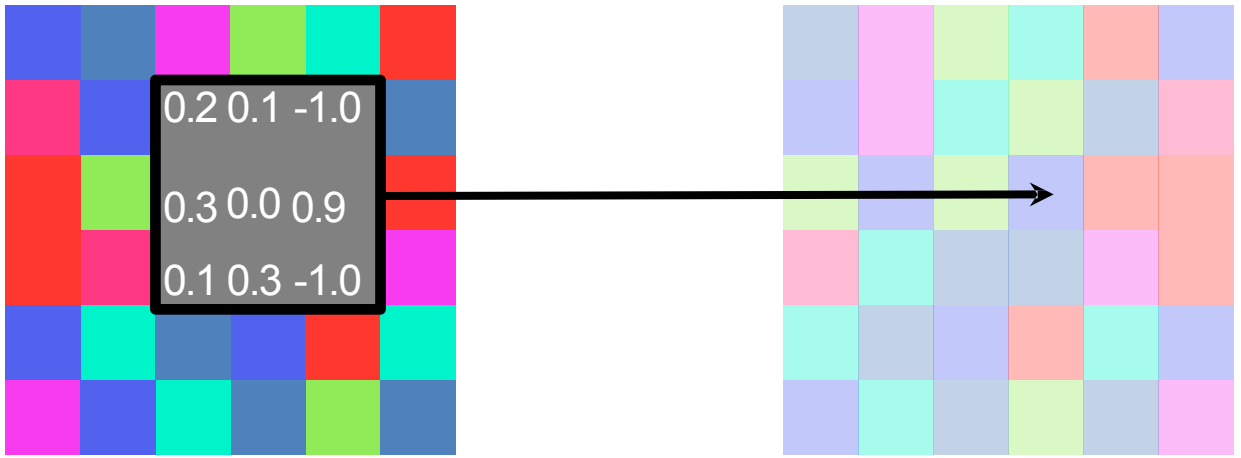
By Daniel
Research Scientist
danielshaving@gmail.com

Table of contents

Convolution: Neighborhood Operations



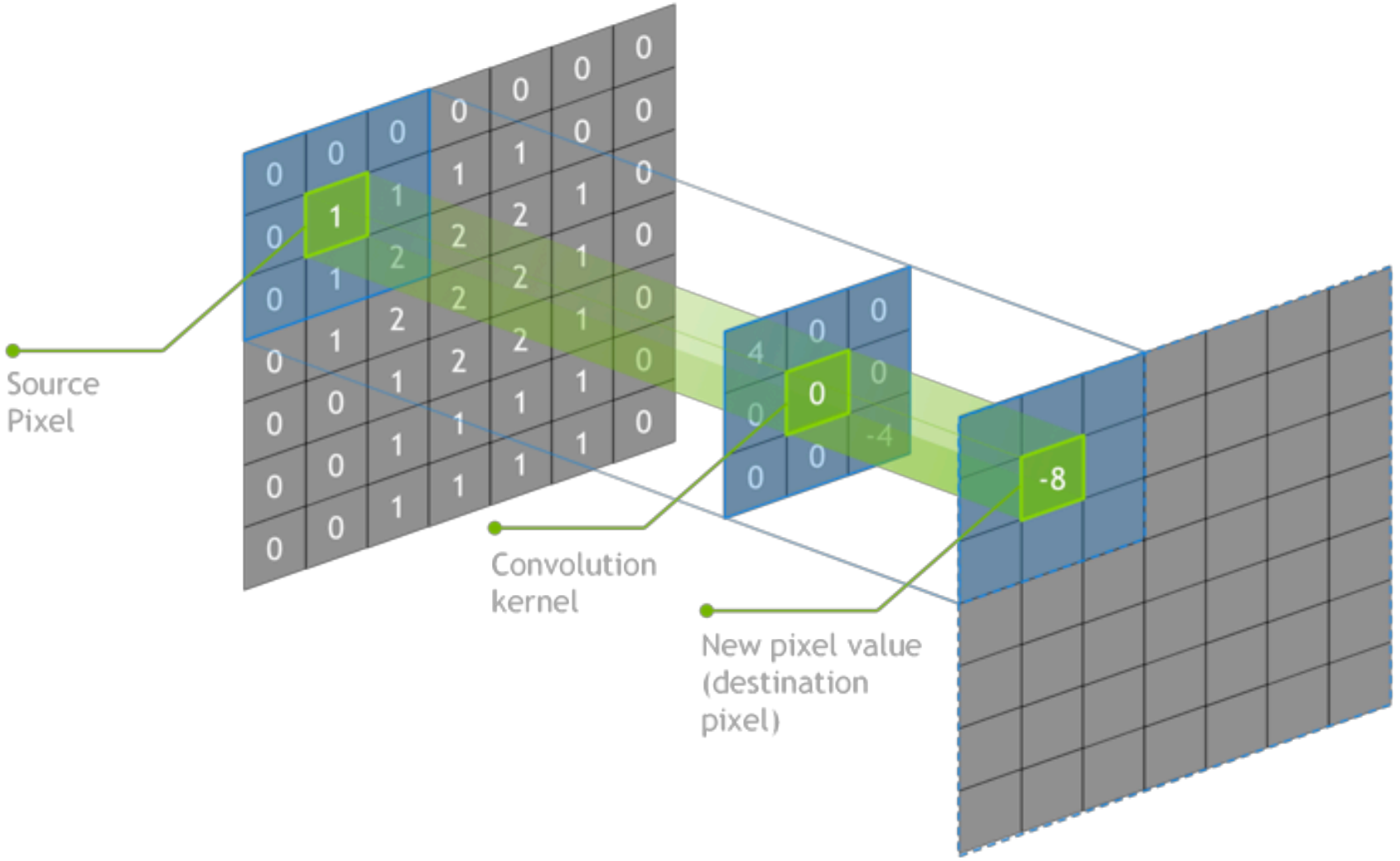
Convolution



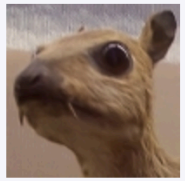
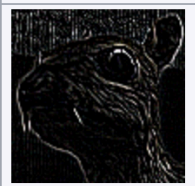
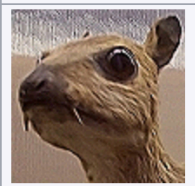
$$F = \begin{bmatrix} 0.2 & 0.1 & -1.0 \\ 0.3 & 0.0 & 0.9 \\ 0.1 & 0.3 & -1.0 \end{bmatrix}$$

$$I' = F * I$$

Convolution images



Convolution Kernels

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

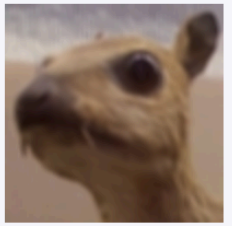
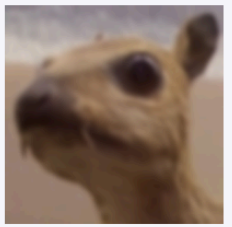
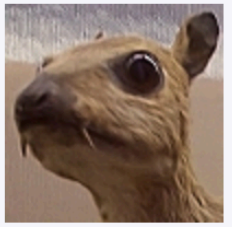
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Table of contents

Sobel Kernels and prewitt kernels

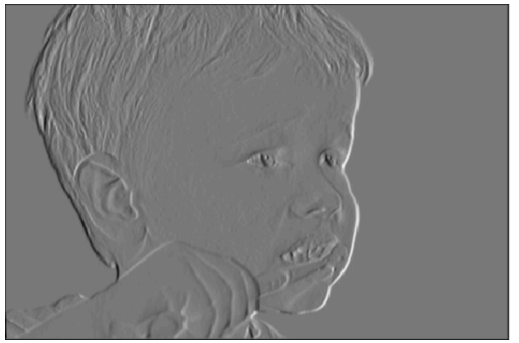
Sobel Kernel

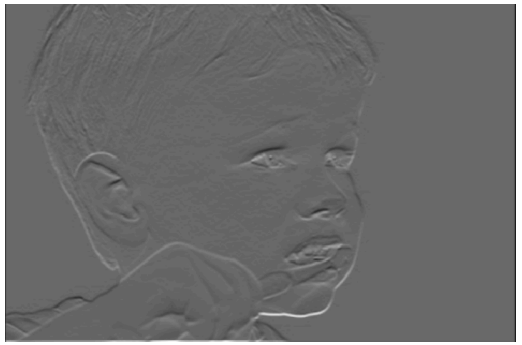
-1	0	1
-2	0	2
-1	0	1

$\partial/\partial x$

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

$\partial/\partial y$





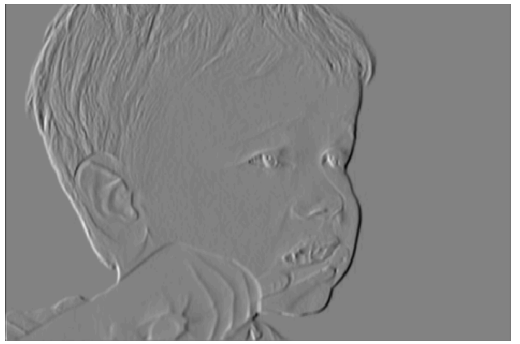
Prewitt Kernel

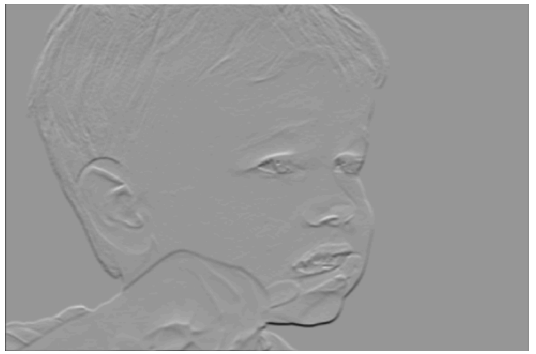
-1	0	1
-1	0	1
-1	0	1

$\partial/\partial x$

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

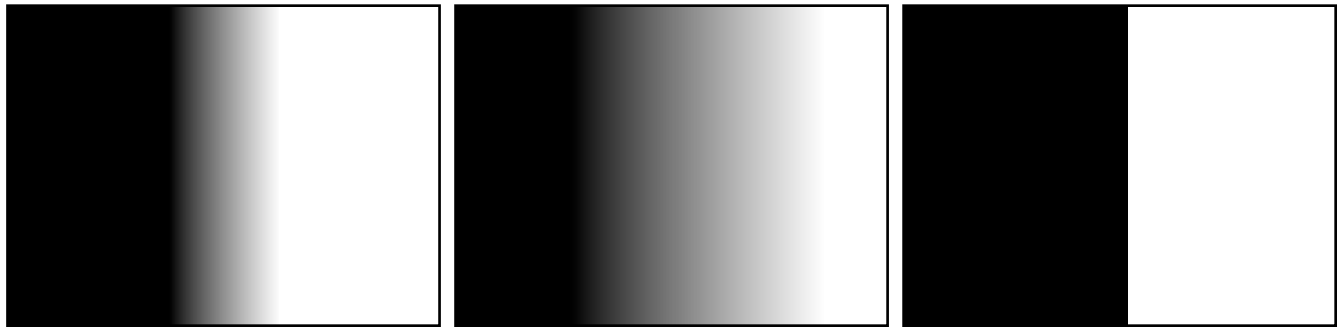
$\partial/\partial y$





Using Sobel to get magnitude

- What is an edge
  - A large change in image brightness of a short spatial distance
  - Edge strength = (I(x,y)-I(x+dx,y))/dx



$$|\nabla I| = \sqrt{I_x^2 + I_y^2} \ ,$$

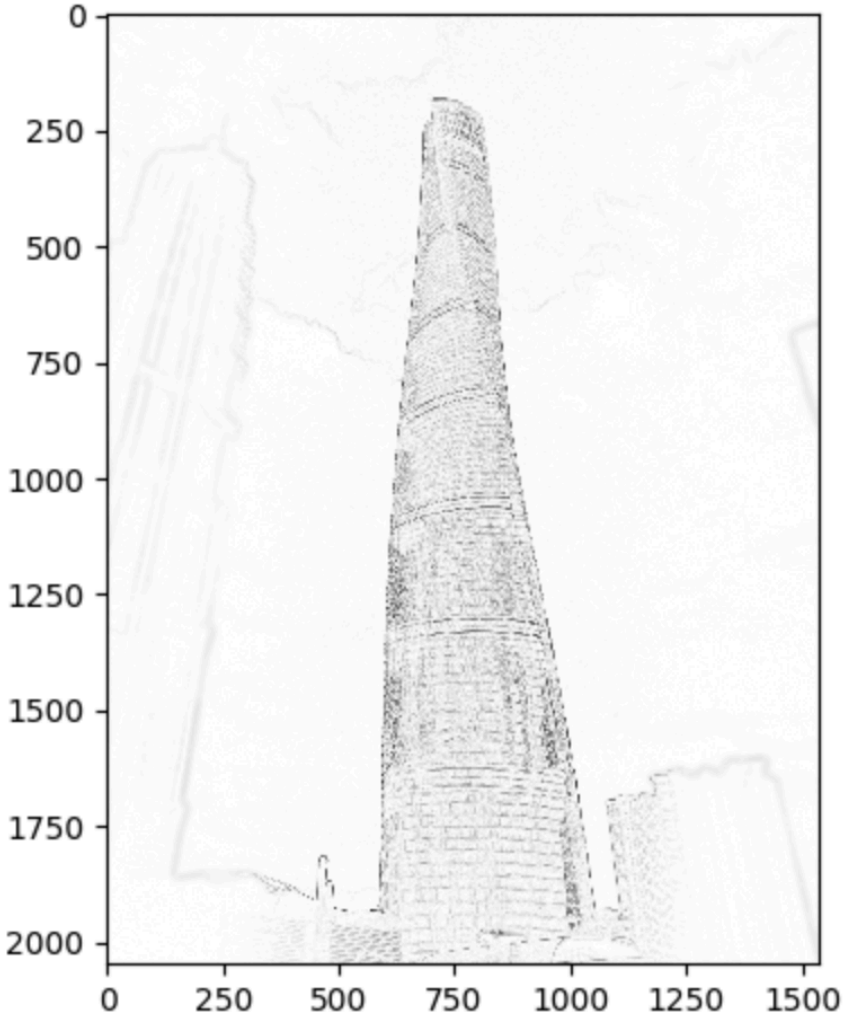
$$\alpha = \arctan2(I_y, I_x)$$

But this general definition still allows for many theories, software implementation and hardware architectures.

Using Sobel to get magnitude

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


im = np.array(Image.open('images/empire.jpg').convert('L'))
#Sobel derivative filters
imx = np.zeros(im.shape)
filters.sobel(im,1,imx)
imy = np.zeros(im.shape)
filters.sobel(im,0,imy)
magnitude = np.sqrt(imx**2+imy**2)
magnitude_inverte = 255 - magnitude
plt.imshow(magnitude_inverte,cmap='gray')
plt.show()
```

$$|\nabla I| = \sqrt{I_x^2 + I_y^2}$$

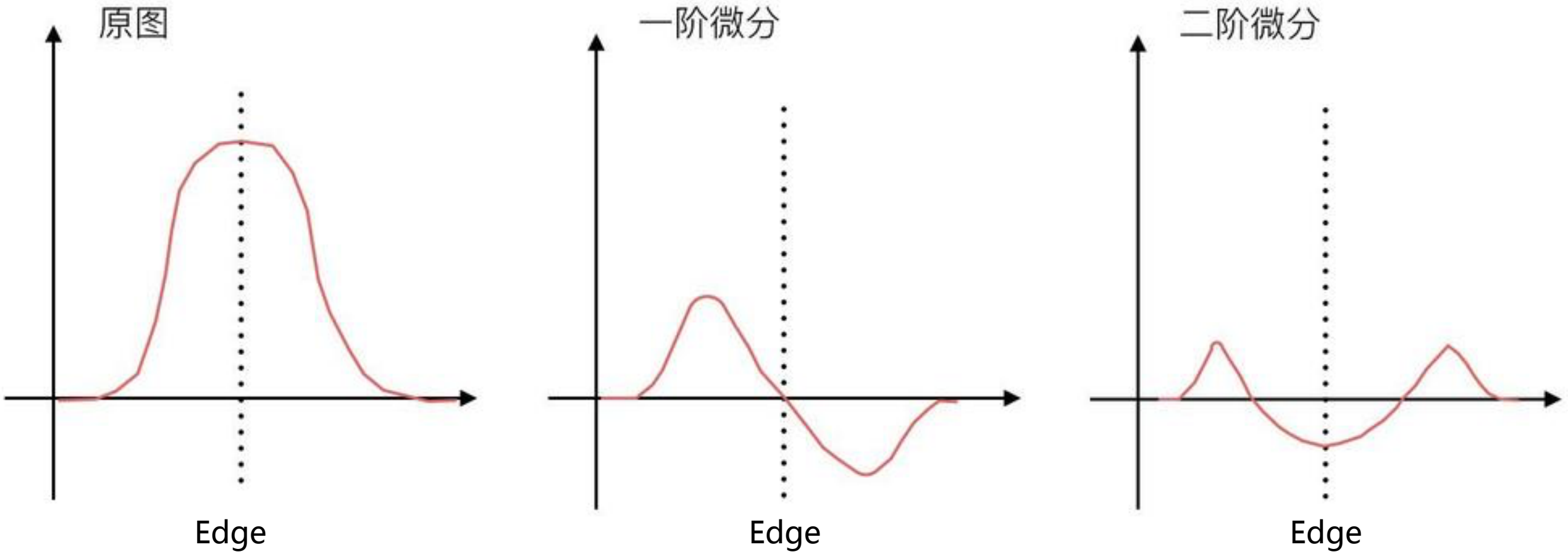


Sobel Kernels and prewire kernels

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy import signal

img = cv2.imread('ChildBrushingTeeth.jpg',cv2.IMREAD_COLOR)
(height,width,deep) = img.shape
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
dest_img = np.zeros((height,width),np.uint8)

Sobelx = [[1,2,1],[0,0,0],[-1,-2,-1]]
Sobely = [[1,0,-1],[2,0,-2],[1,0,-1]]
Prewittx = [[-1,0,1],[-1,0,1],[-1,0,1]]
Prewitty = [[-1,-1,-1],[0,0,0],[1,1,1]]

Sobelxim = signal.convolve2d(gray,Sobelx,boundary='symm', mode='same')
Sobelyim = signal.convolve2d(gray,Sobely,boundary='symm', mode='same')
Prewittxim = signal.convolve2d(gray,Prewittx,boundary='symm',
mode='same')
Prewittyim = signal.convolve2d(gray,Prewitty,boundary='symm',
mode='same')


Sobelximabs = np.abs(Sobelyim)
edge_cut = np.floor((Sobelximabs - np.min(Sobelximabs)) * 255 /
((np.max(Sobelximabs) - np.min(Sobelximabs))))

plt.imshow(edge_cut,cmap='gray')
plt.colorbar()
plt.show()
```

Table of contents

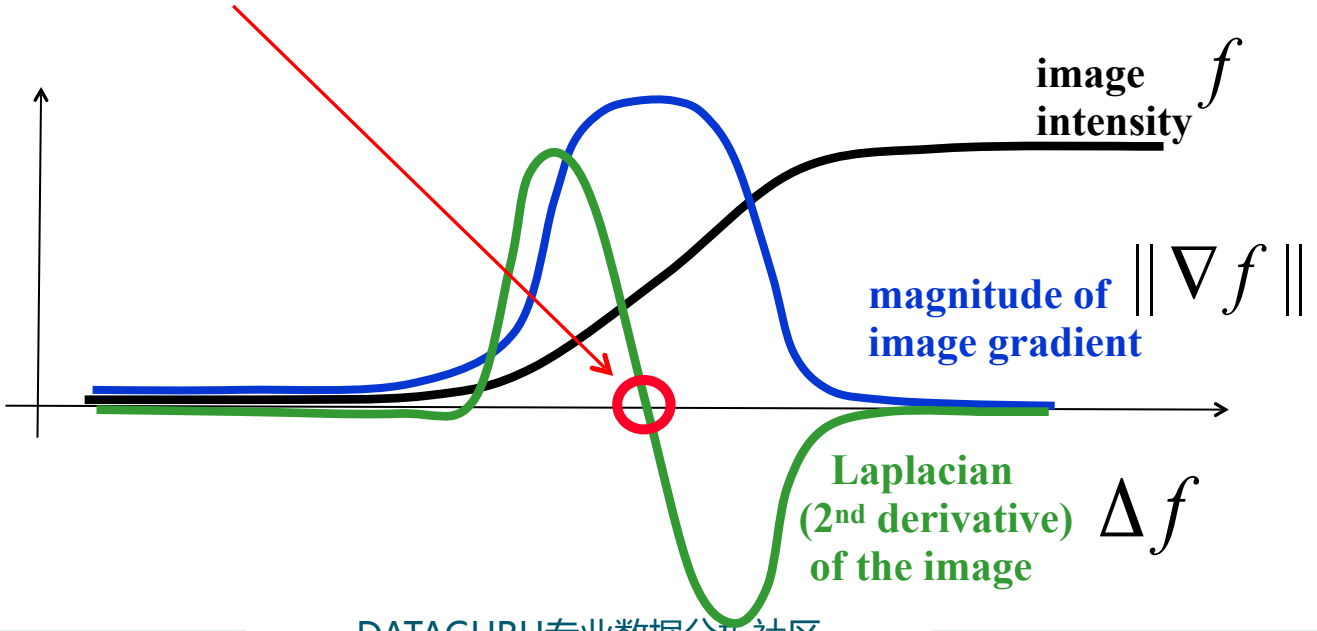
Laplacian Kernel



Laplacian kenel

- Laplacian Zero Crossing $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

- Used for edge detection
  (alternative to computing Gradient extrema)



Laplacian Kernel

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

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

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

1	1	1
1	−8	1
1	1	1

Laplacien kernels

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

img = cv2.imread("images/ChildBrushingTeeth.jpg", 0)

gray_lap = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
dst = cv2.convertScaleAbs(gray_lap)

plt.imshow(dst,cmap='gray')
plt.show()
```

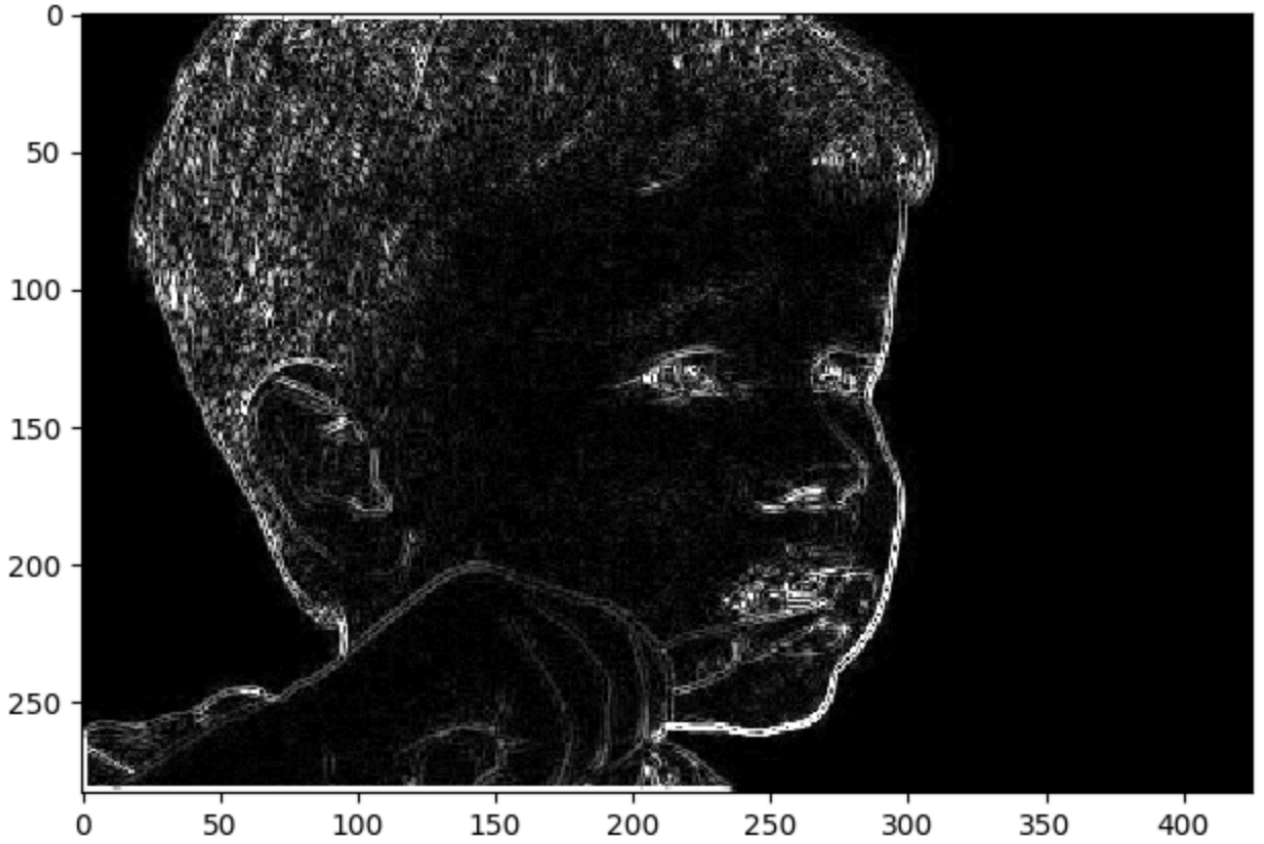
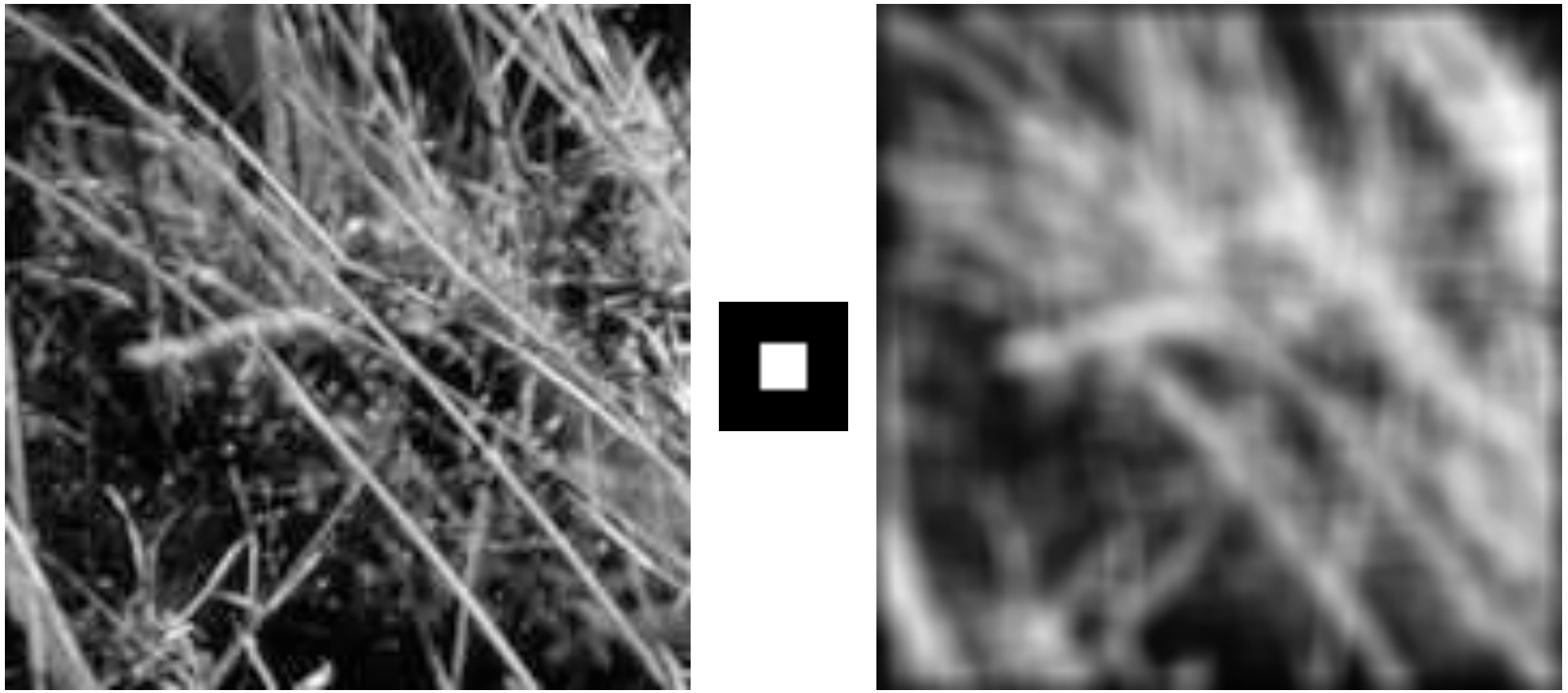


Table of contents

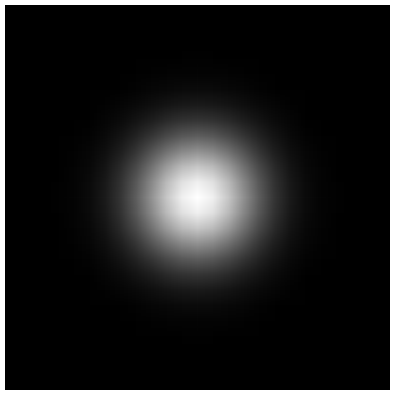
Smoothing with box filter revisited

- Smoothing with an average actually doesn't compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square



Smoothing with box filter revisited

- Smoothing with an average actually doesn't compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square
- Better idea: to eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center, like so:



"fuzzy blob"

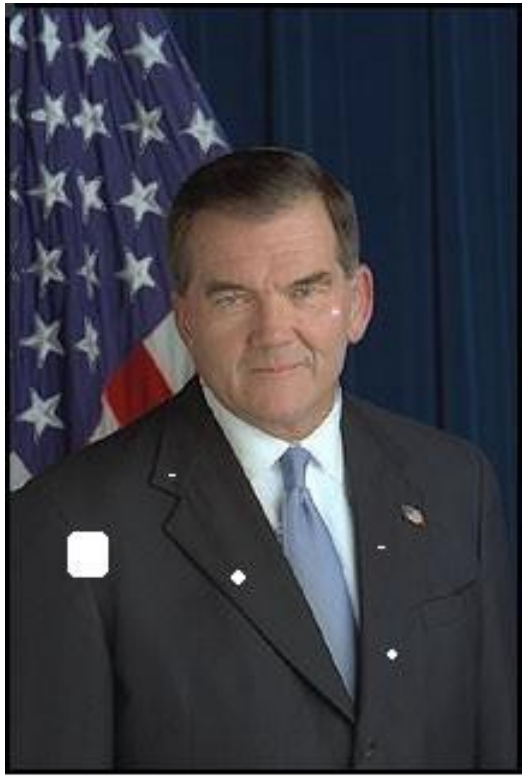
Mean Kernel Filter

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

Original image

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

Image with noise



Median filter (5x5)

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

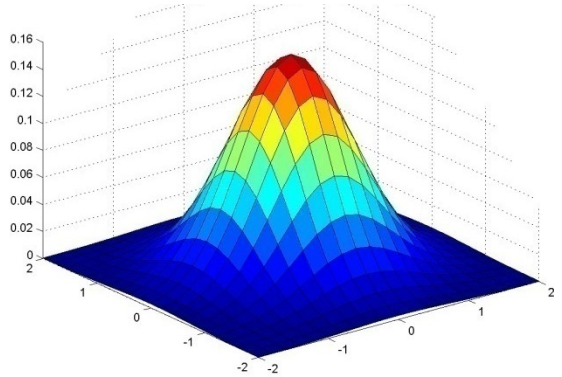
$$h[u,v]$$

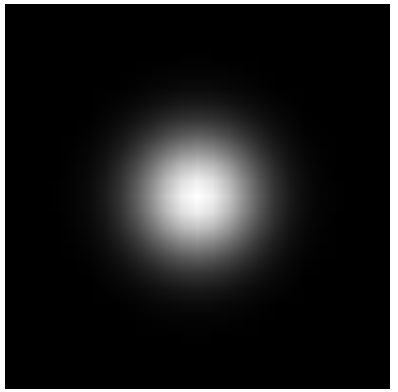
Mean Kernel

Table of contents

Gaussian Kernel

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





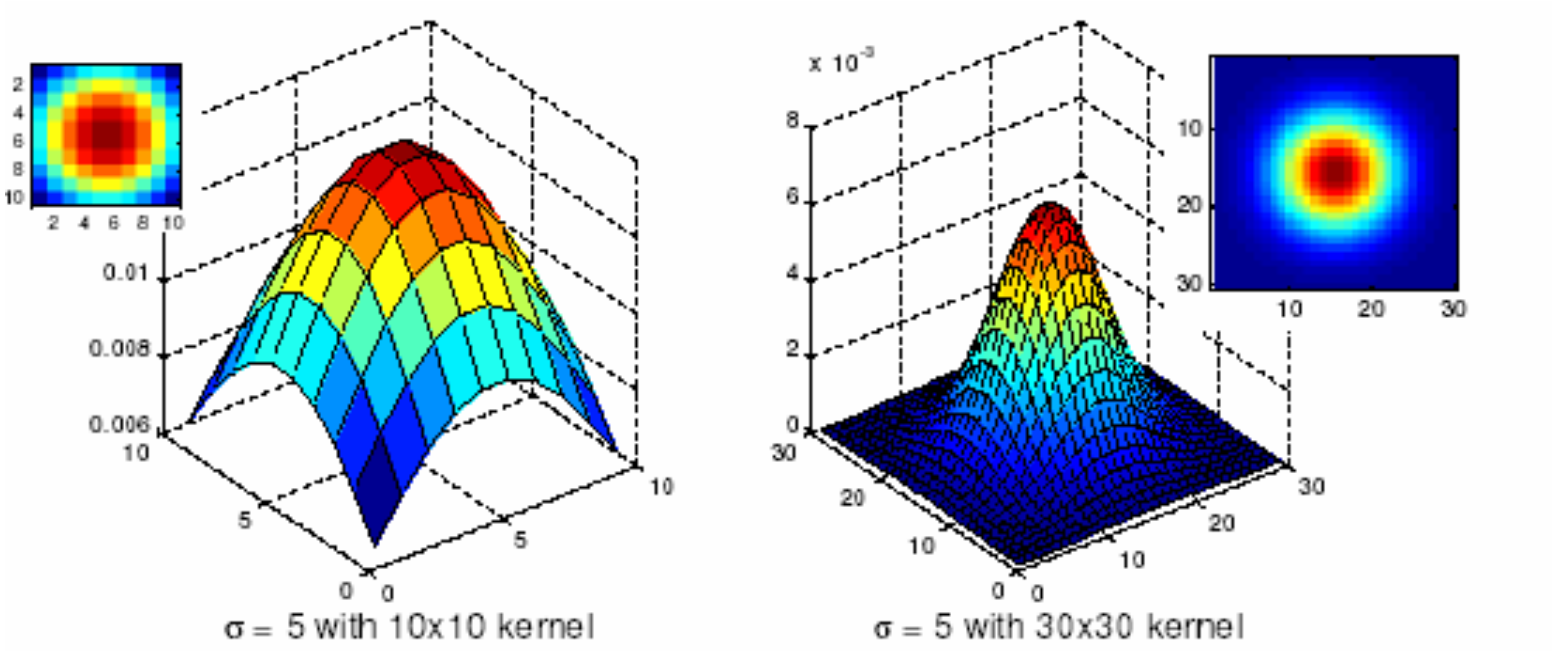
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should re-normalize weights to sum to 1 in any case)

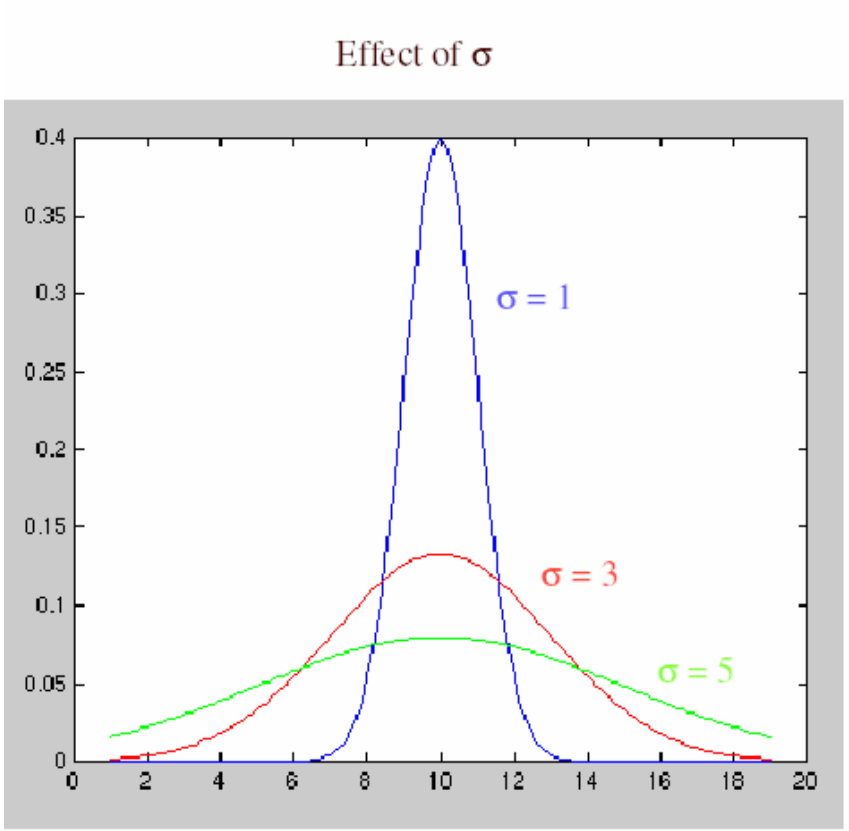
Choosing kernel width

- Gaussian filters have infinite support, but discrete filters use finite kernels



Choosing kernel width

- Rule of thumb: set filter half-width to about 3 σ



Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
- Convolution with self is 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 product of two 1D Gaussians

Separability of the Gaussian filter

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

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

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

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

Gaussian filter coding

```
from PIL import Image
from numpy import *
from scipy.ndimage import filters

im = array(Image.open('empire.jpg').convert('L'))
im2 = filters.gaussian_filter(im,5)
```

(a) (b) (c) (d)

Figure 1.9: An example of Gaussian blurring using the `scipy.ndimage.filters` module. (a) original image in grayscale, (b) Gaussian filter with $\sigma = 2$, (c) with $\sigma = 5$, (d) with $\sigma = 10$.

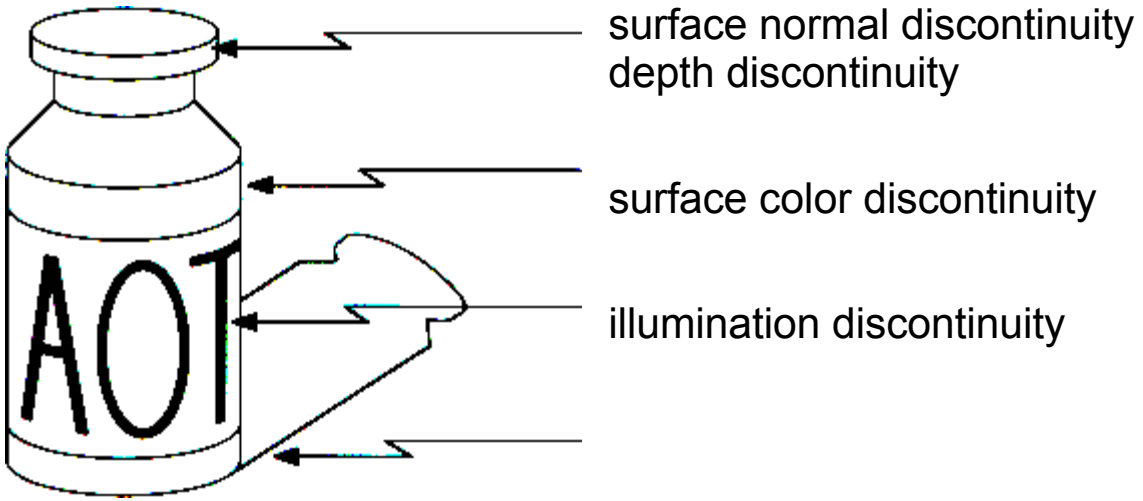
Table of contents

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)

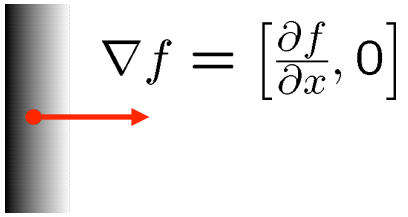
Origin of edges

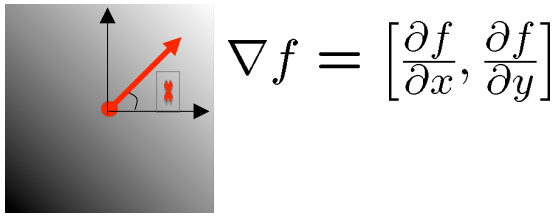
Edges are caused by a variety of factors:



Origin of edges

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

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

The gradient direction 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

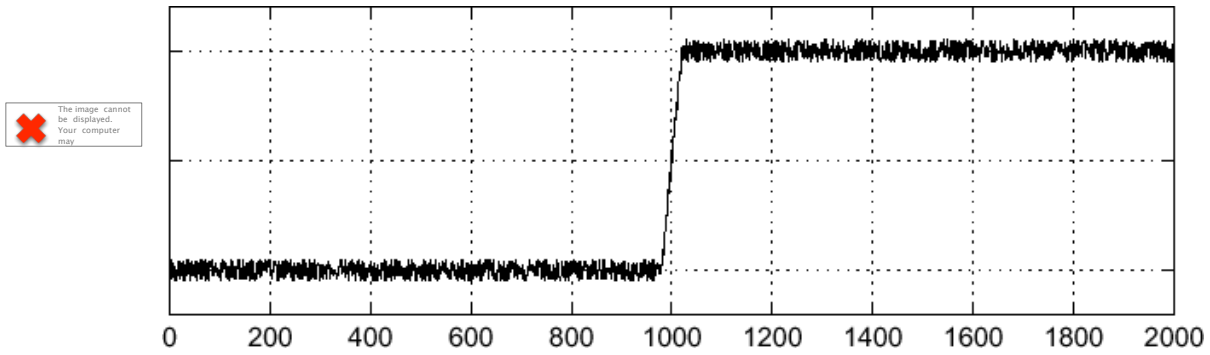
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Steve Seitz

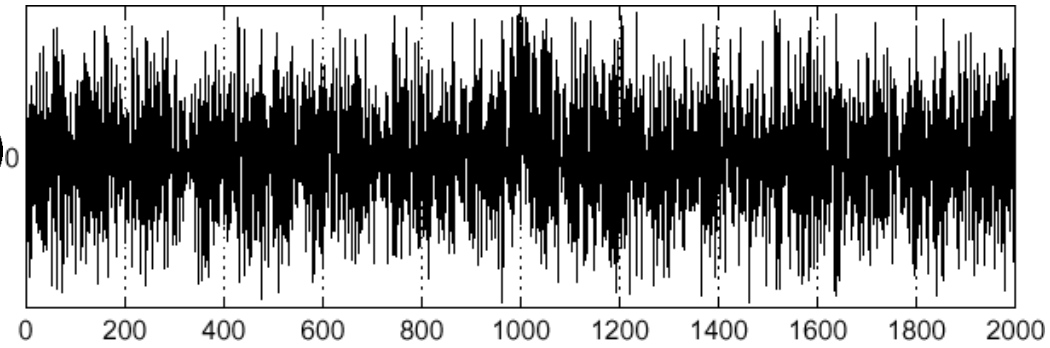
Effects of noise

Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal

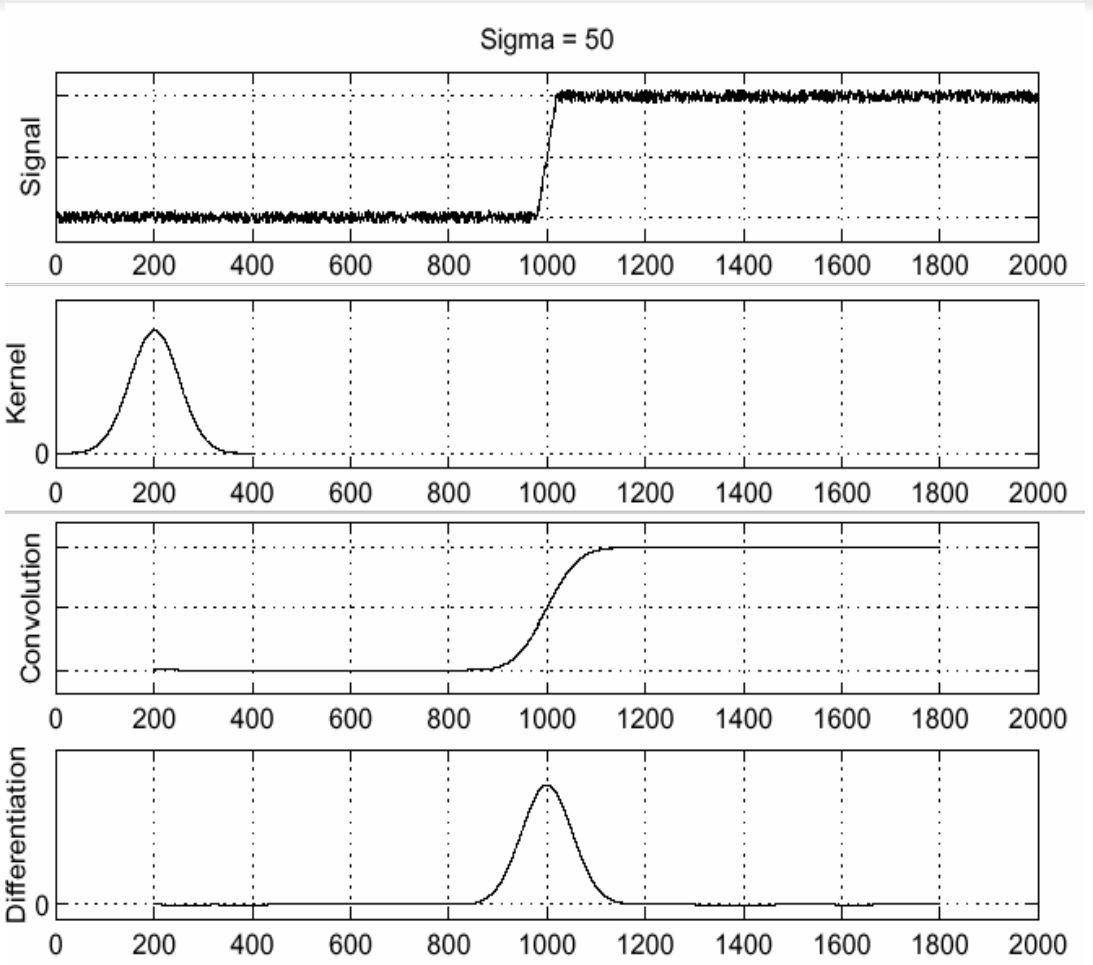


$\frac{d}{dx} f(x)$



Source: Steve Seitz

Solution: smooth first



f

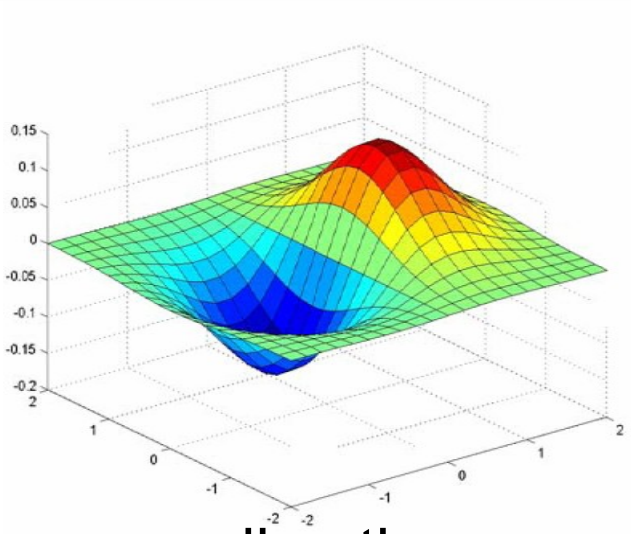
g

$f * g$

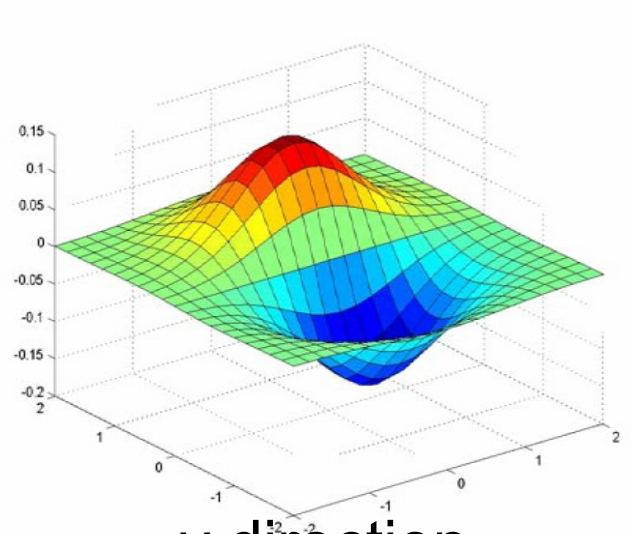
$\frac{d}{dx}(f * g)$

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

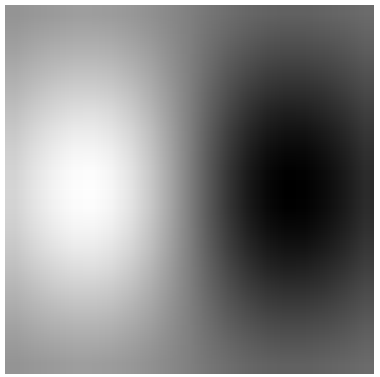
Derivative of Gaussian filter

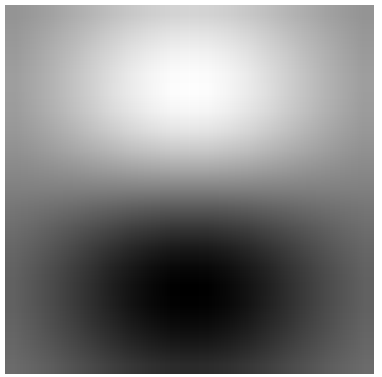


x-direction



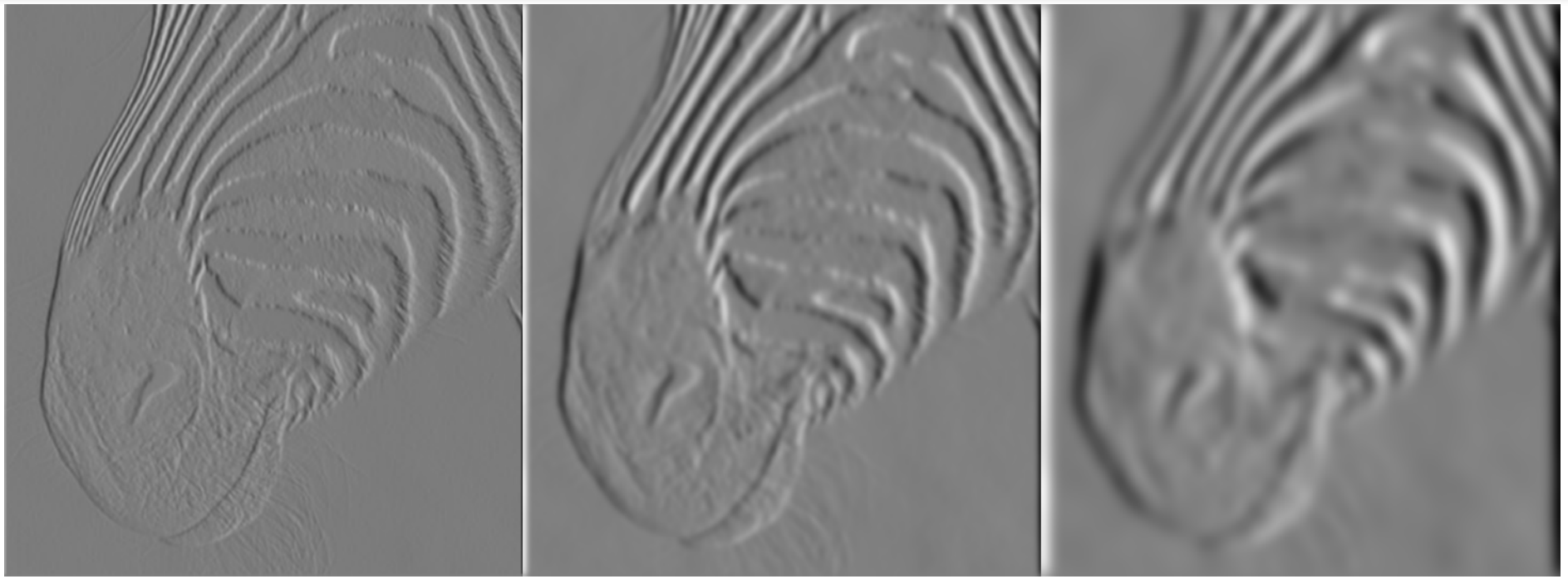
y-direction





Which one finds horizontal/vertical edges?

Scale of Gaussian derivative filter



1 pixel　　3 pixels　　7 pixels

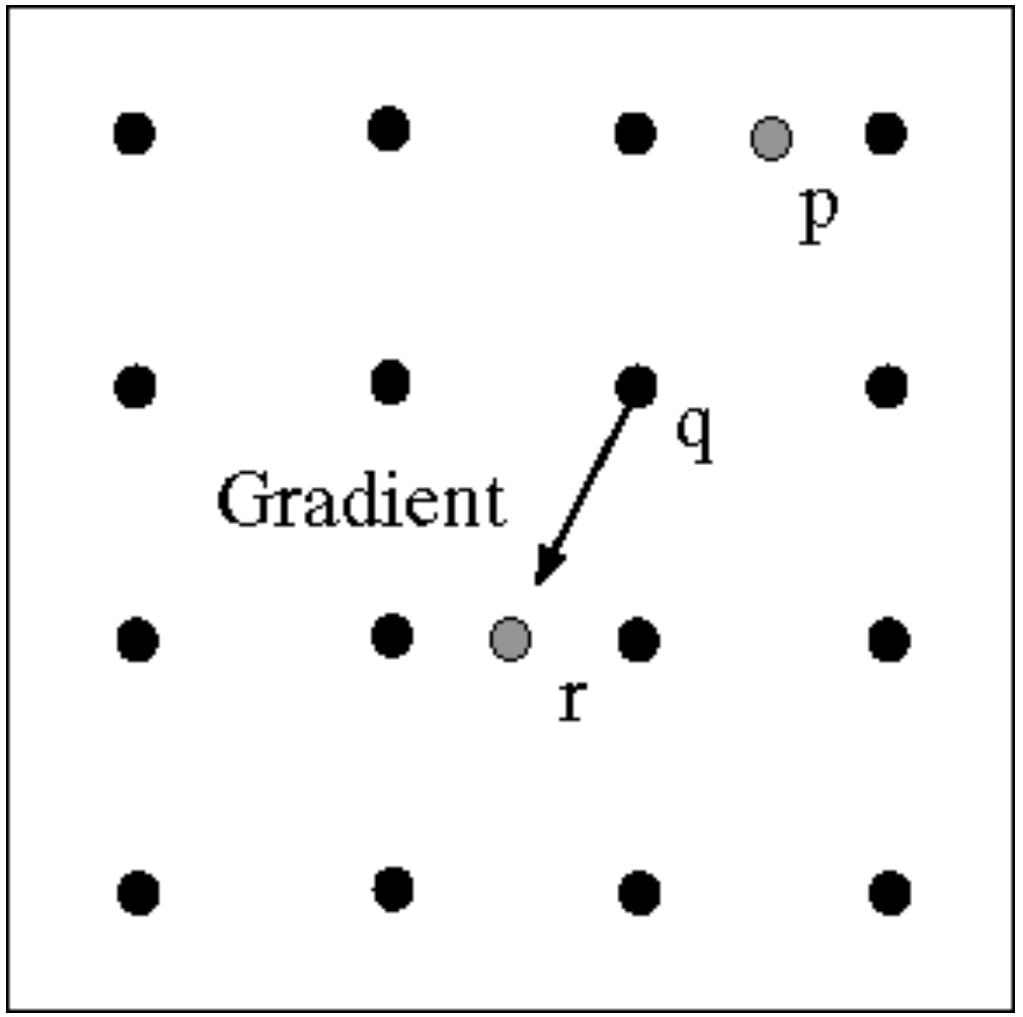
Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Canny edge detector

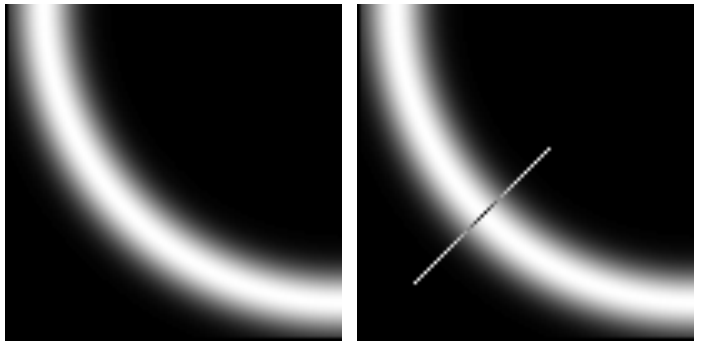
1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width

```python
import cv2
import numpy as np

img = cv2.imread("ChildBrushingTeeth.jpg", 0)

img = cv2.GaussianBlur(img, (3, 3), 0)
canny = cv2.Canny(img, 50, 150)

cv2.imshow('Canny', canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Non-maximum suppression



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



original image (Lena)

original image (Lena)

norm of the gradient

norm of the gradient

thresholding

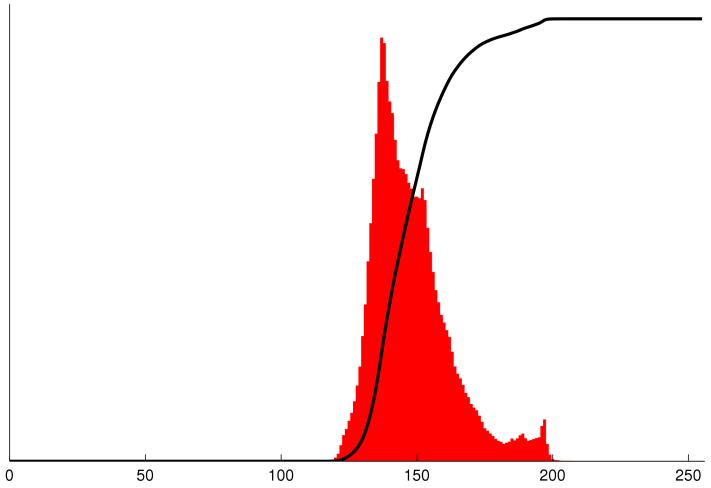
thresholding

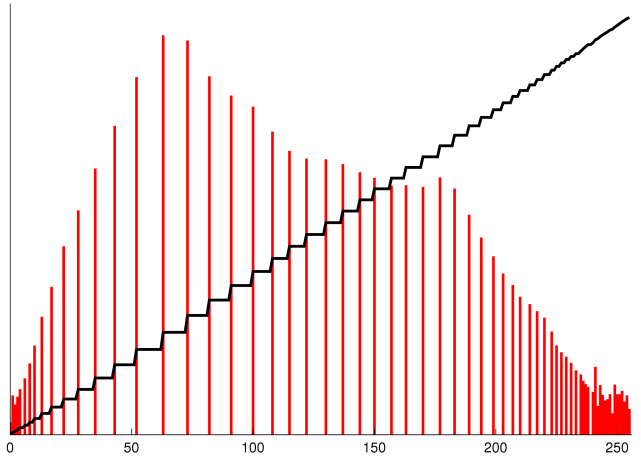
Non-maximum suppression

Non-maximum suppression

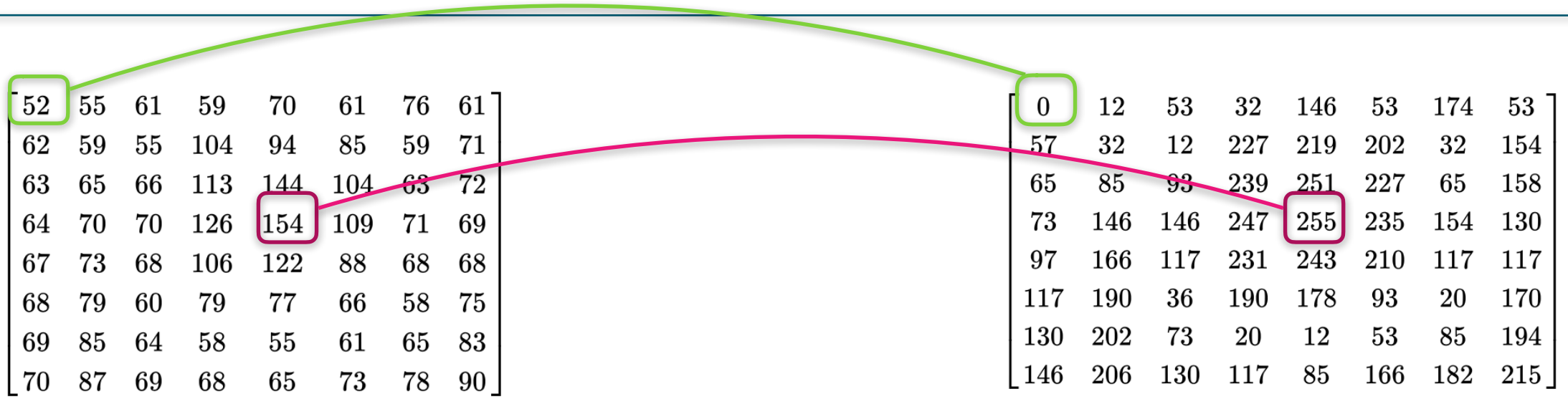
Table of contents

Histogram Equalization





Histogram Equalization



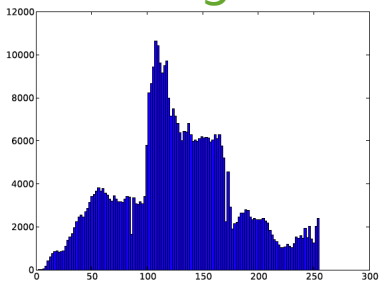
$$\begin{bmatrix} 52 & 55 & 61 & 59 & 70 & 61 & 76 & 61 \\ 62 & 59 & 55 & 104 & 94 & 85 & 59 & 71 \\ 63 & 65 & 66 & 113 & 144 & 104 & 63 & 72 \\ 64 & 70 & 70 & 126 & 154 & 109 & 71 & 69 \\ 67 & 73 & 68 & 106 & 122 & 88 & 68 & 68 \\ 68 & 79 & 60 & 79 & 77 & 66 & 58 & 75 \\ 69 & 85 & 64 & 58 & 55 & 61 & 65 & 83 \\ 70 & 87 & 69 & 68 & 65 & 73 & 78 & 90 \end{bmatrix} \quad \begin{bmatrix} 0 & 12 & 53 & 32 & 146 & 53 & 174 & 53 \\ 57 & 32 & 12 & 227 & 219 & 202 & 32 & 154 \\ 65 & 85 & 93 & 239 & 251 & 227 & 65 & 158 \\ 73 & 146 & 146 & 247 & 255 & 235 & 154 & 130 \\ 97 & 166 & 117 & 231 & 243 & 210 & 117 & 117 \\ 117 & 190 & 36 & 190 & 178 & 93 & 20 & 170 \\ 130 & 202 & 73 & 20 & 12 & 53 & 85 & 194 \\ 146 & 206 & 130 & 117 & 85 & 166 & 182 & 215 \end{bmatrix}$$

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

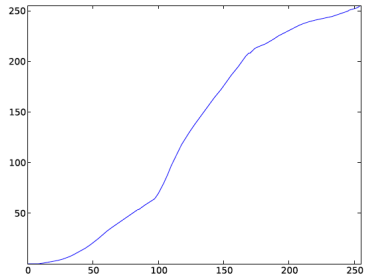
$$h(78) = \text{round}\left(\frac{46-1}{63} \times 255\right) = \text{round}\left(0.714286 \times 255\right) = 182$$

Histogram Equalization

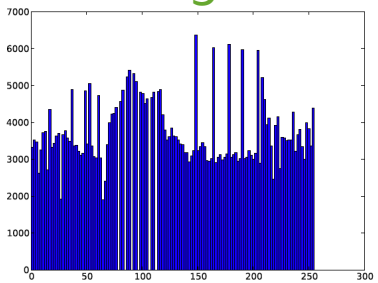
Histogram



cdf



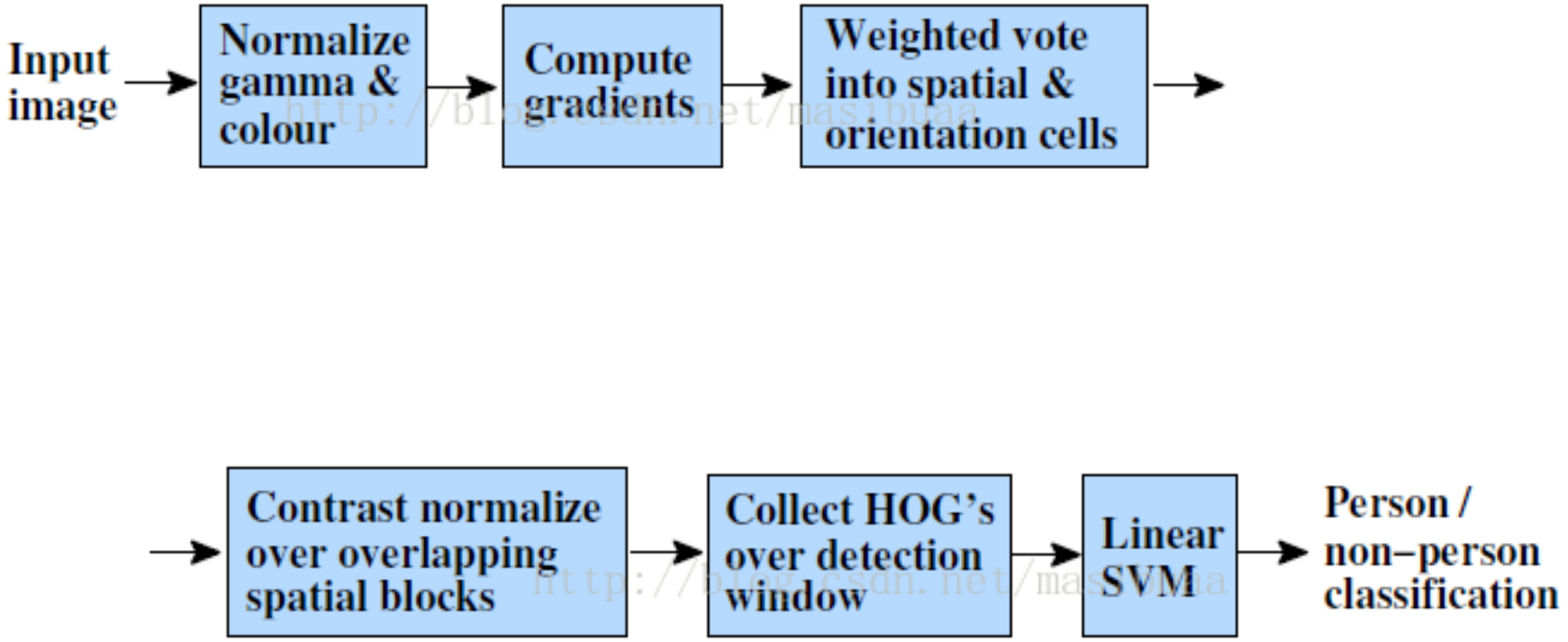
Histogram



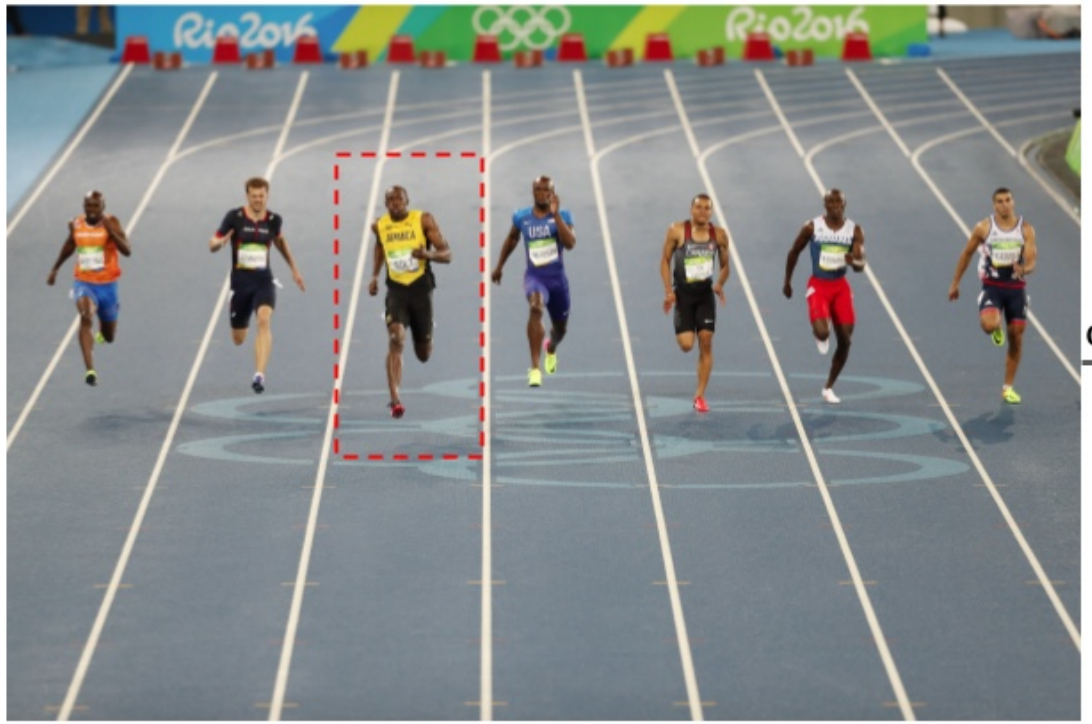
before transform after

Table of contents

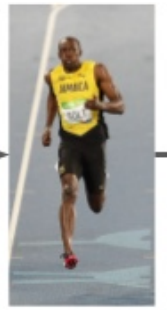
Hog (Histogram of gradient) feature



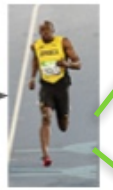
Hog (Histogram of gradient) feature



Crop



Resize

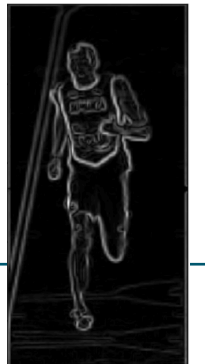


$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A}$$

Sobel X gradient

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

Sobel Y gradient

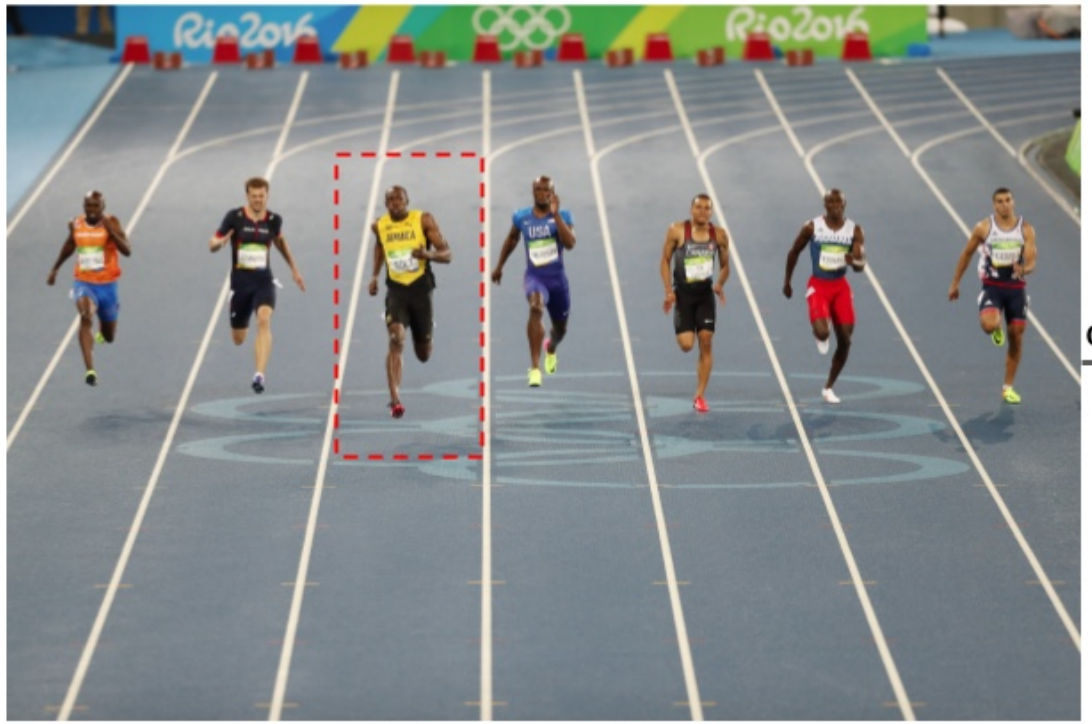


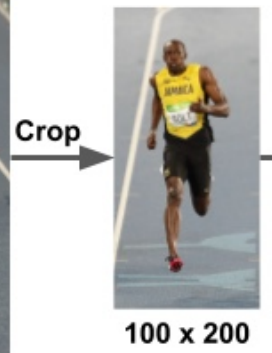
$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

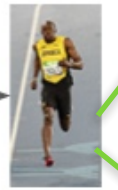
$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G_y}, \mathbf{G_x})$$

Magnitude

Hog (Histogram of gradient) feature





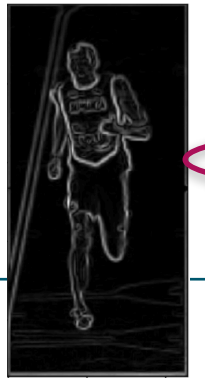


$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A}$$

Sobel X gradient

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

Sobel Y gradient

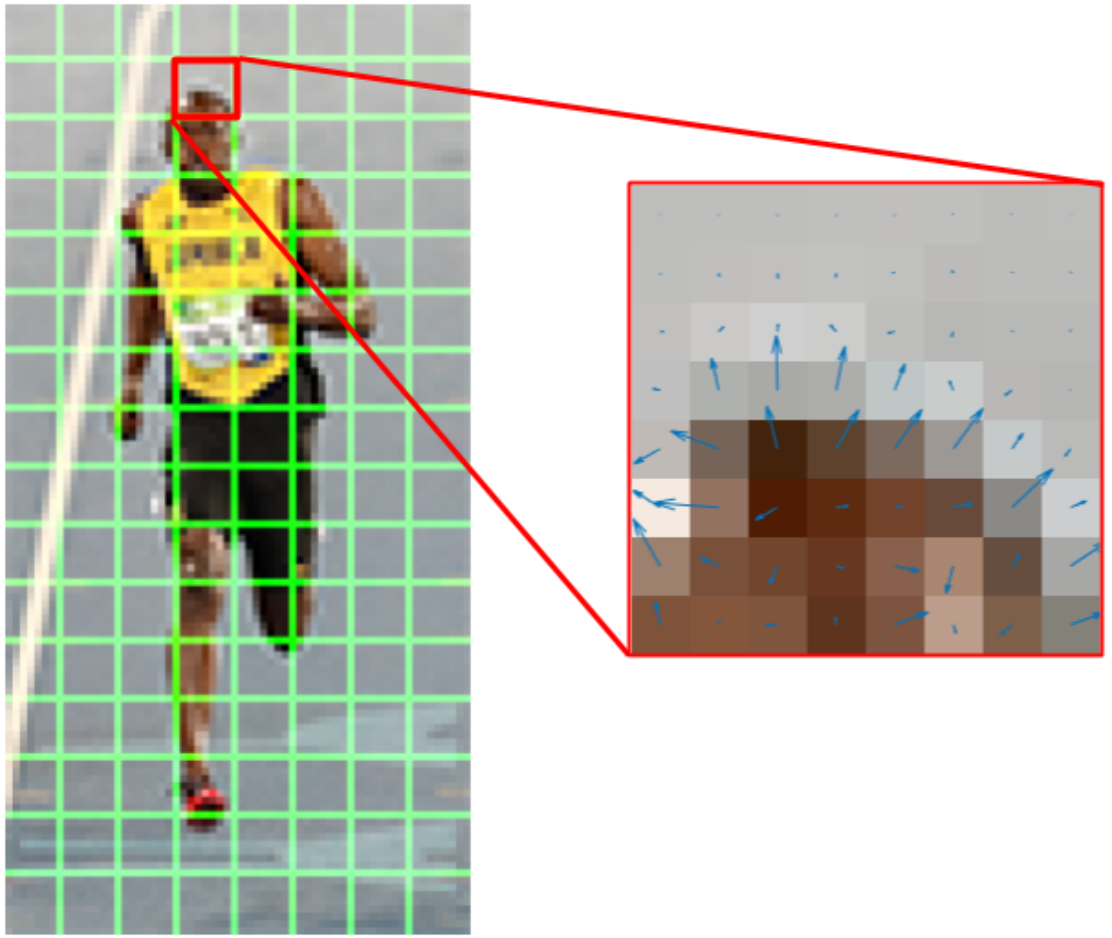


$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

$$\mathbf{\Theta} = \text{atan2}(\mathbf{G_y}, \mathbf{G_x})$$

Magnitude

Histogram of amplitude in cells



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

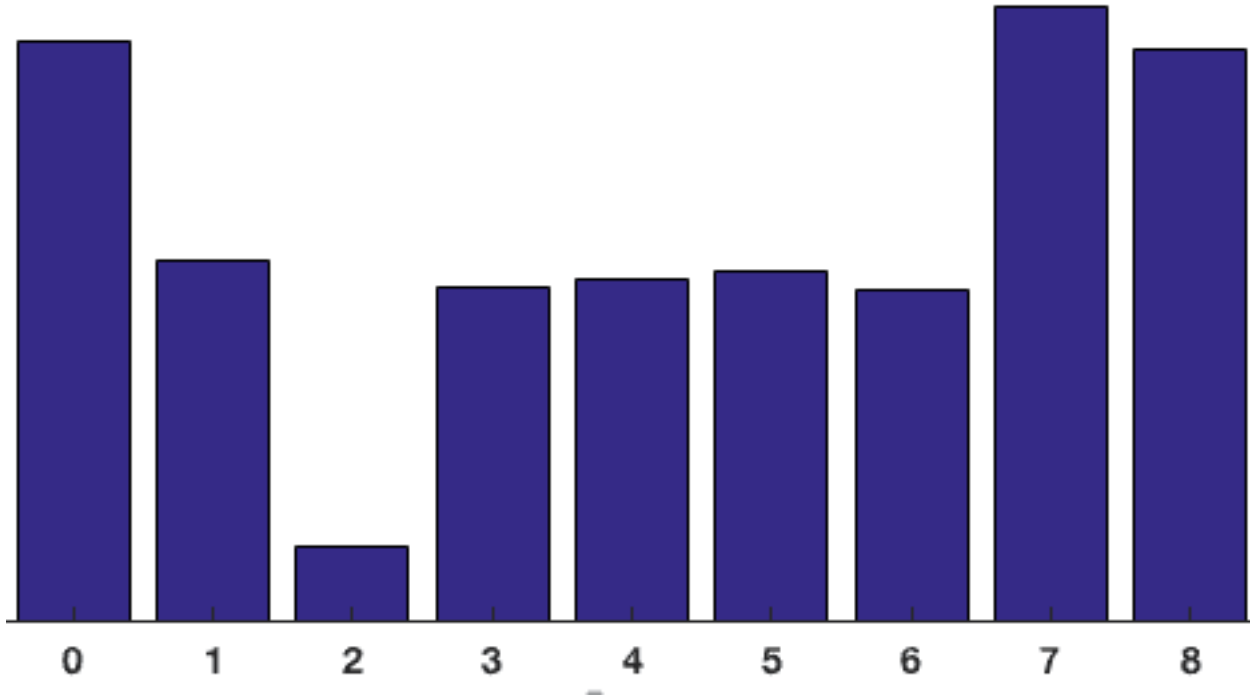
Gradient Direction

Histogram of amplitude in cells

θ

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction



$0 - 20^o$

$160^o - 180^o$

min_angle: idx = 8
max_angle: idx = 0
mod: mod = 5

bin_size = 9

Histogram of amplitude in cells

θ

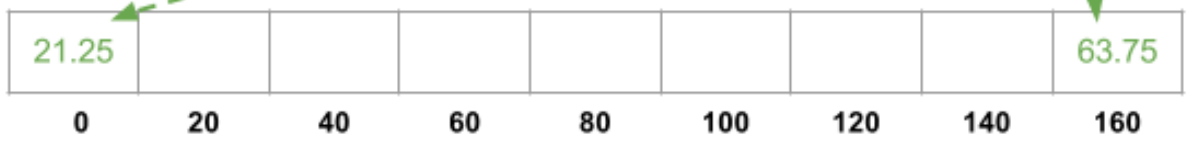
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

M

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude



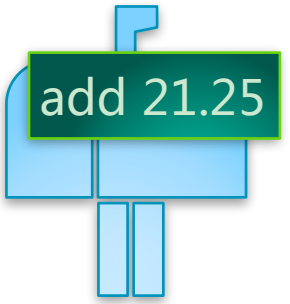
Histogram of Gradients

M = 85

165^o

160^o 180^o

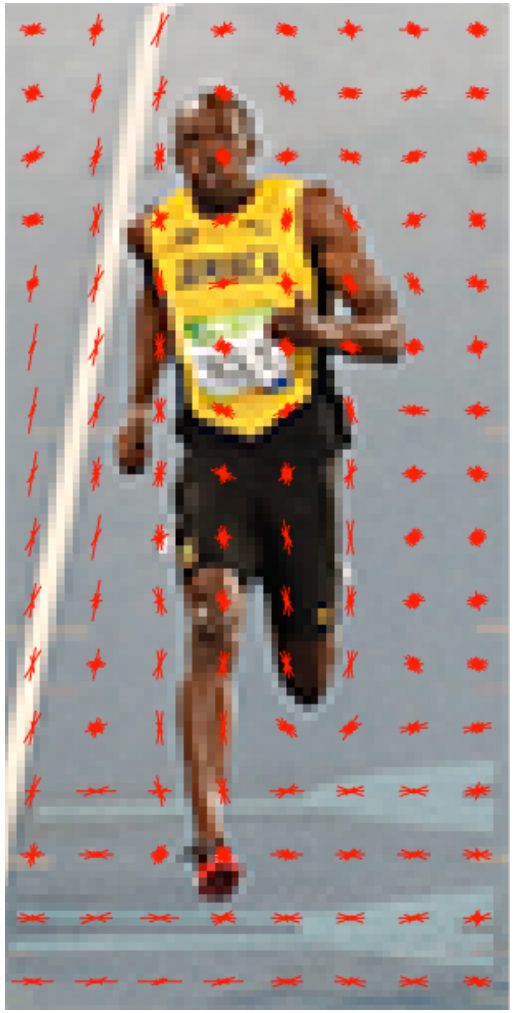
25% belongs to box 160
75% belongs to box 180

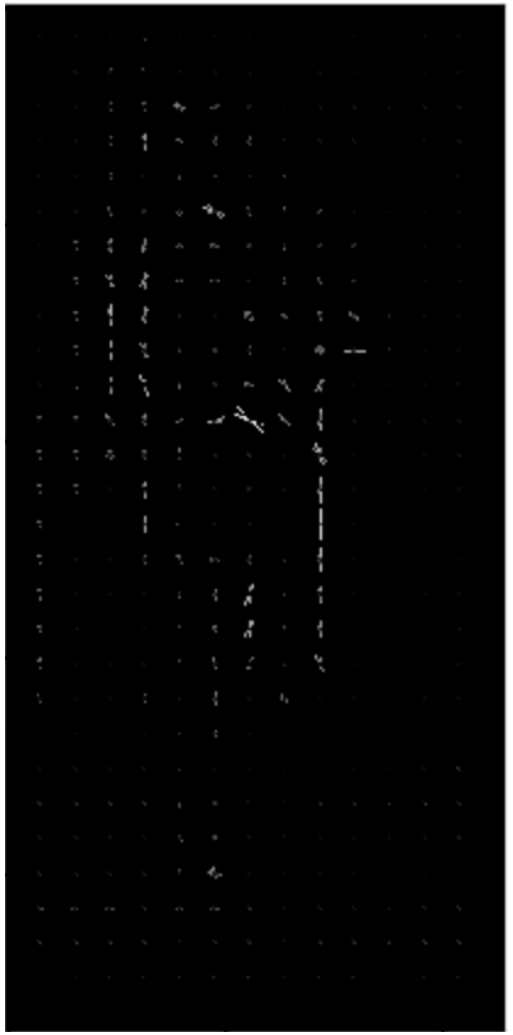


box 160 degree

box 180 degree

Histogram of amplitude in cells





Coding

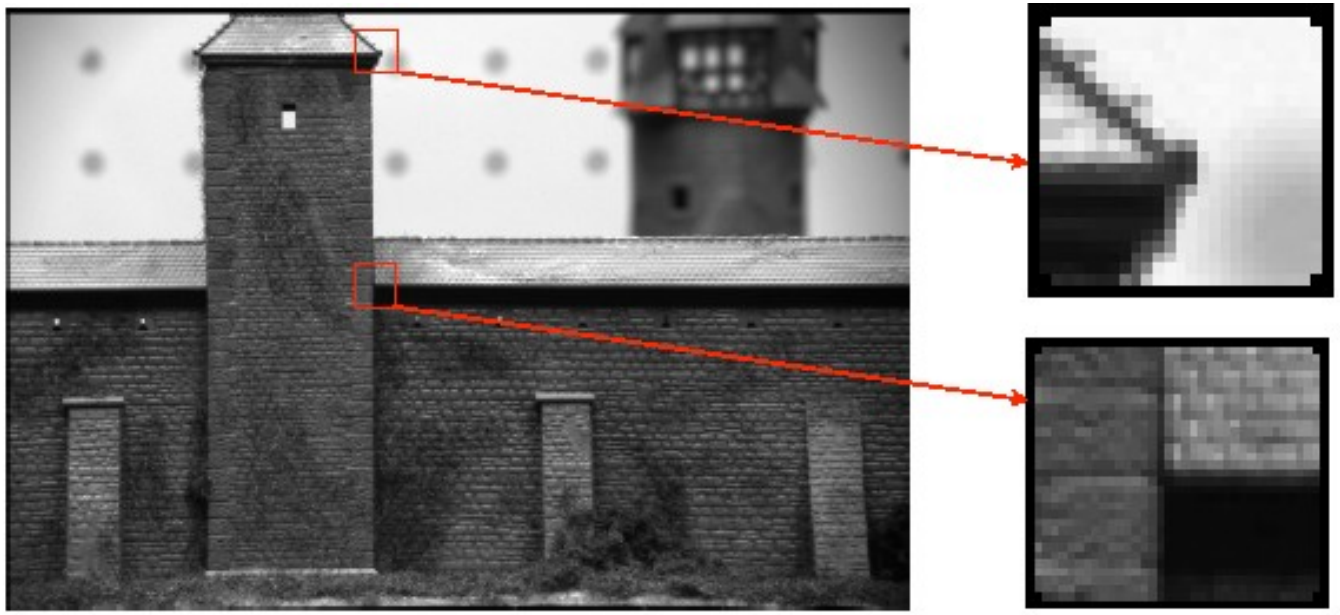
Fonction:

```
start = time.time()
        hog = cv2.HOGDescriptor()
        instances = []
        for img, label in images:
            # print img
            img = read_color_image(img)
            descriptor = hog.compute(img)
                descriptor = descriptor.ravel()
            pairing = Instance(descriptor, label)
            instances.append(pairing)
```

Wheel: https://github.com/danielshaving/CV_Features_HoG_Feature_Extraction

Table of contents

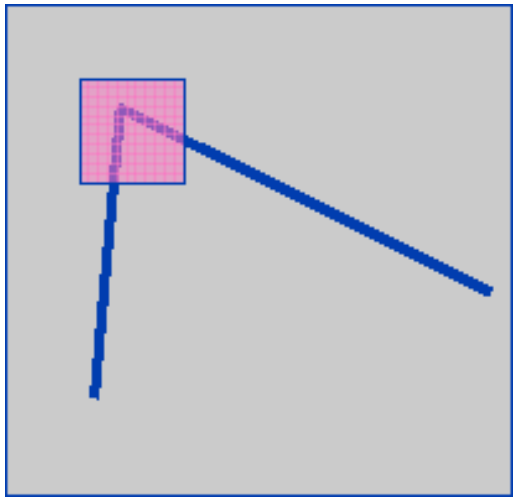
What are Corners?



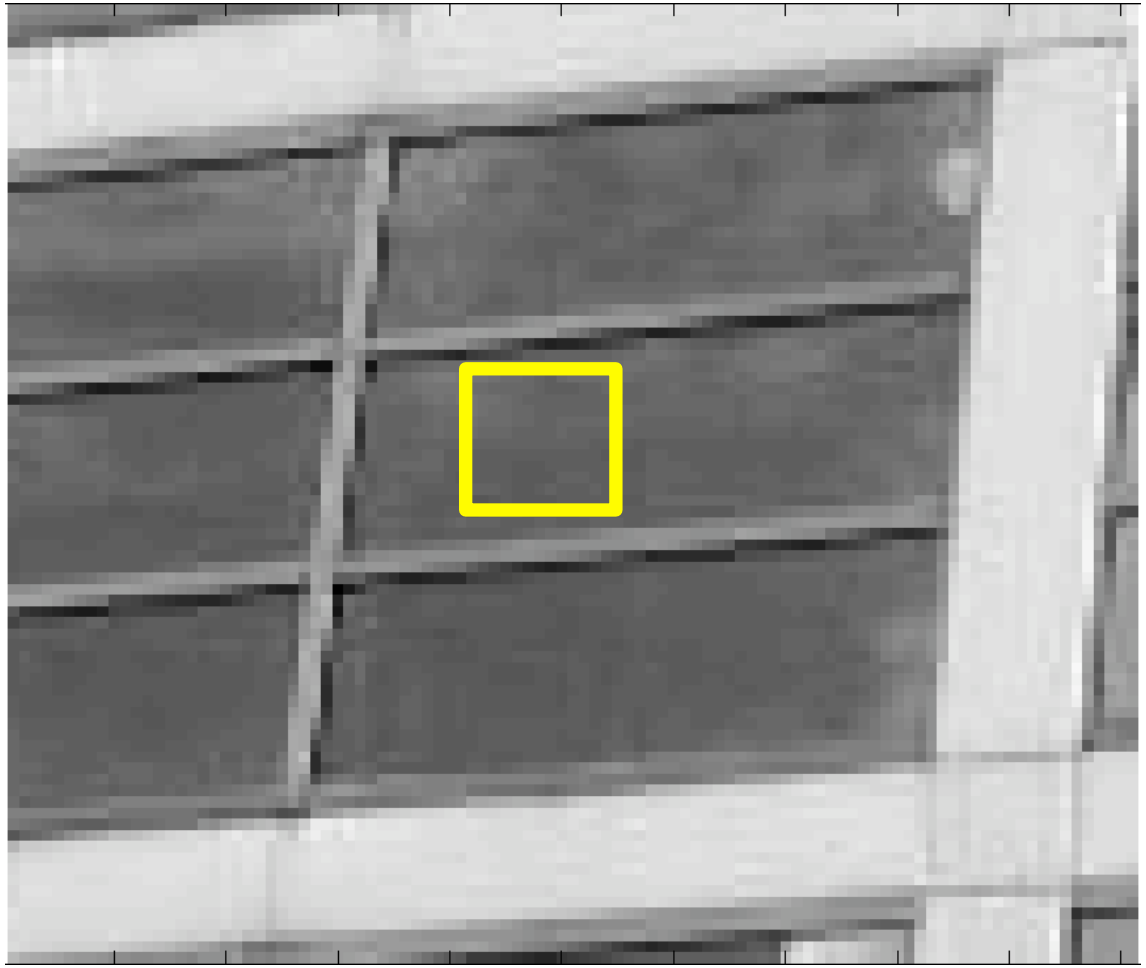
- Intuitively, junctions of contours.
- Generally more stable features over changes of viewpoint
- Intuitively, large variations in the neighborhood of the point in all directions
- They are good features to match!

Corner Points: Basic Idea

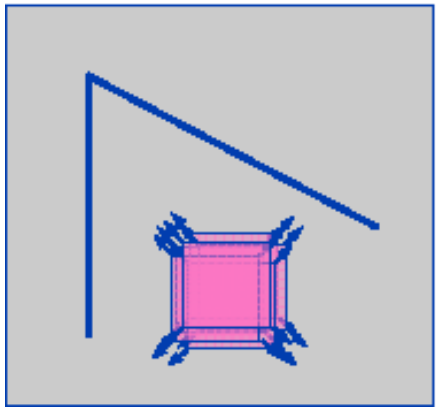
- We should easily recognize the point by looking at intensity values within a small window
- Shifting the window in *any direction* should yield a *large change* in appearance.



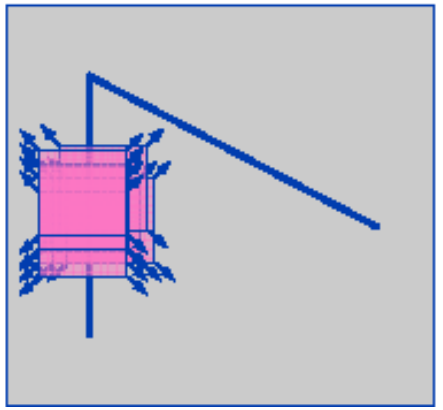
Appearance Change in Neighborhood of a Patch



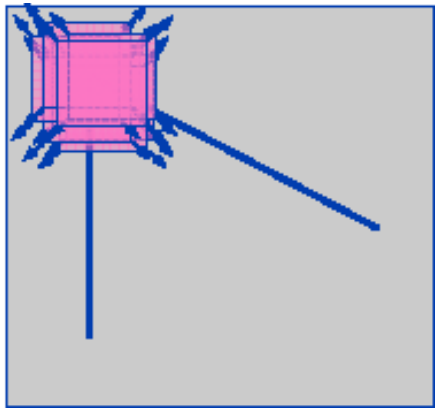
Harris Corner Detector: Basic Idea



"flat" region:
no change in
all directions



"edge":
no change along
the edge direction



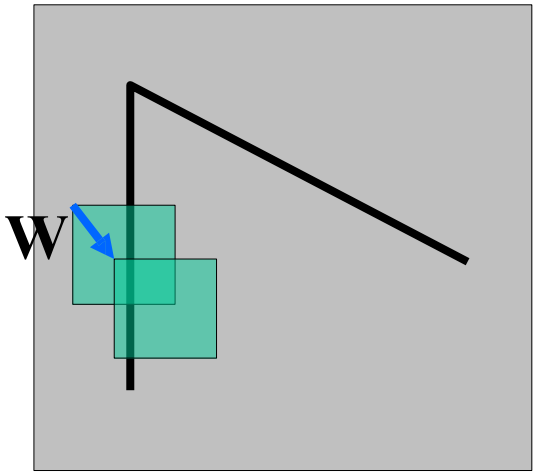
"corner":
significant change
in all directions

Harris corner detector gives a mathematical approach for determining which case holds.

Finding interest points (Feature Detection): the math

Consider shifting the window **W** by (u,v)

- how do the pixels in **W** change?
- compare each pixel before and after using the sum of squared differences (SSD)
- this defines an SSD "error" *E(u,v)*:



$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

Harris Detector: Mathematics

Change of intensity for the shift $[u,v]$:

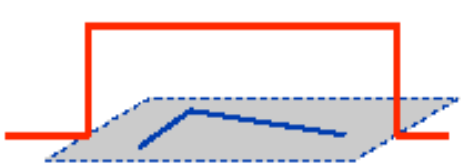
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function

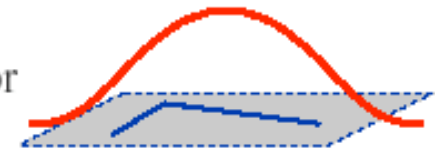
Shifted intensity

Intensity

Window function $w(x,y)$ =



or



1 in window, 0 outside

Gaussian

Harris Detector: Intuition

Change of intensity for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

Intensity

For nearly constant patches, this will be near 0. For very distinctive patches, this will be larger. Hence... we want patches where E(u,v) is LARGE.

Small motion assumption

Taylor Series expansion of I:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approx. is good

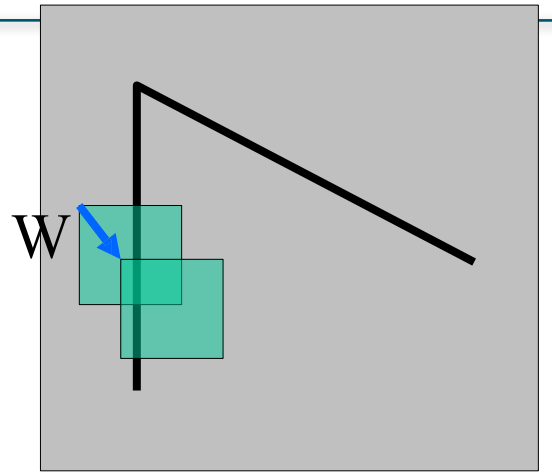
$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + [I_x \;\; I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

Plugging this into the formula on the previous slide…

Feature detection:the math



$$E(u,v) = \sum_{(x,y)\in W} [I(x+u,y+v) - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W} [I(x,y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W} \left[[I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

Harris Corner Derivation

$$\sum [I(x+u,y+v) - I(x,y)]^2$$

$$\approx \sum [I(x,y) + uI_x + vI_y - I(x,y)]^2 \qquad \text{First order approx}$$

$$= \sum u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2$$

$$= \sum \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \qquad \text{Rewrite as matrix equation}$$

$$= \begin{bmatrix} u & v \end{bmatrix} \left(\sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

Ellipse function and harris corner

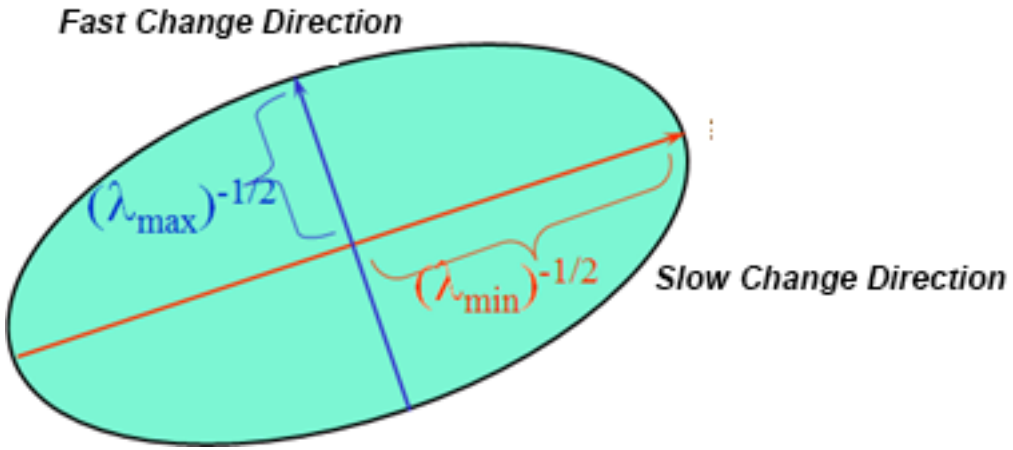
$$Au^2 + 2Buv + Cv^2 = 1$$

Ellipse

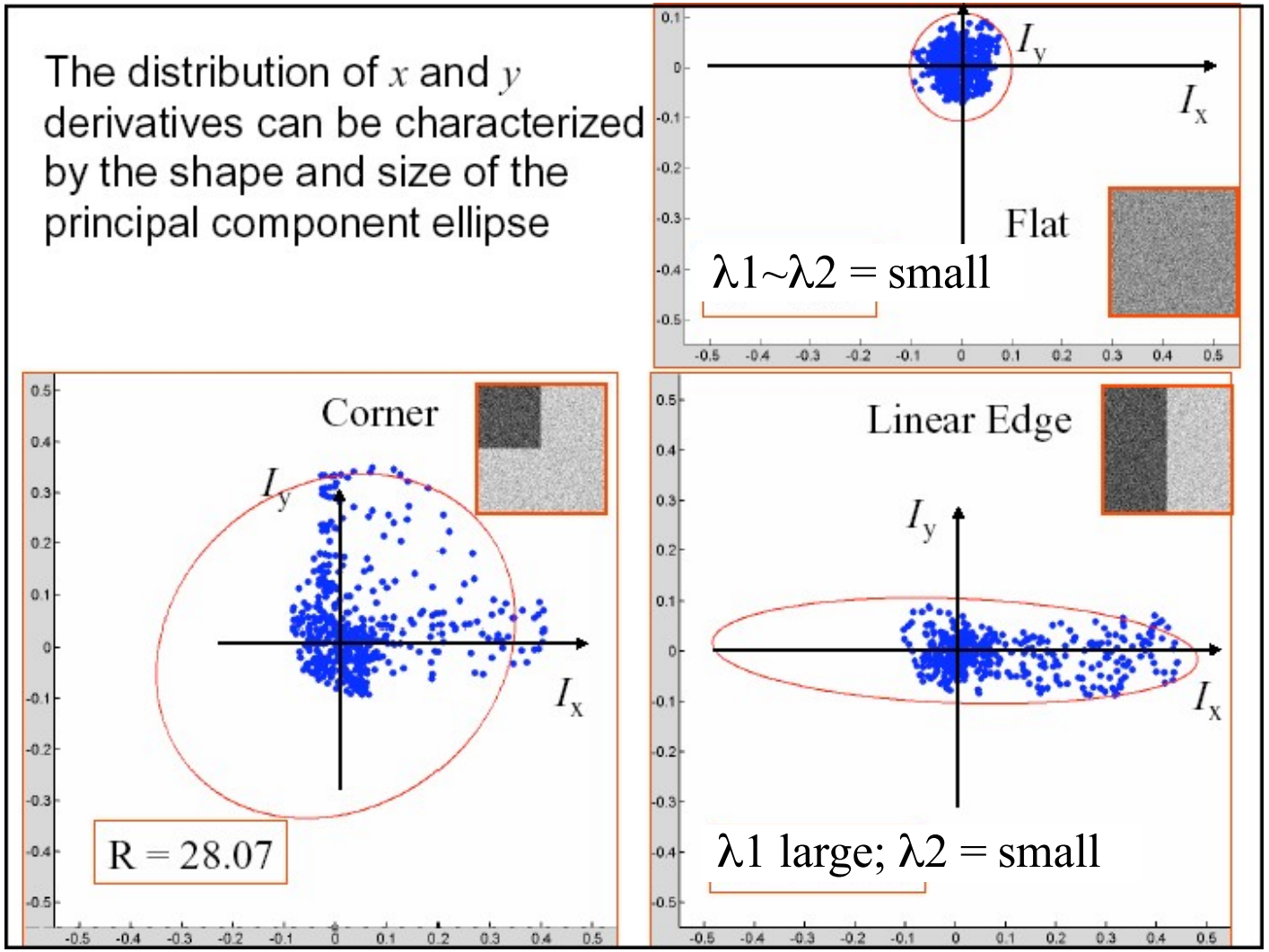
$$\begin{bmatrix} u \\ v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u & v \end{bmatrix} = 1$$

$$\begin{bmatrix} A = \sum_w I_x^2 & B = \sum_w I_{xy}^2 \\ B = \sum_w I_{xy}^2 & C = \sum_w I_y^2 \end{bmatrix}$$

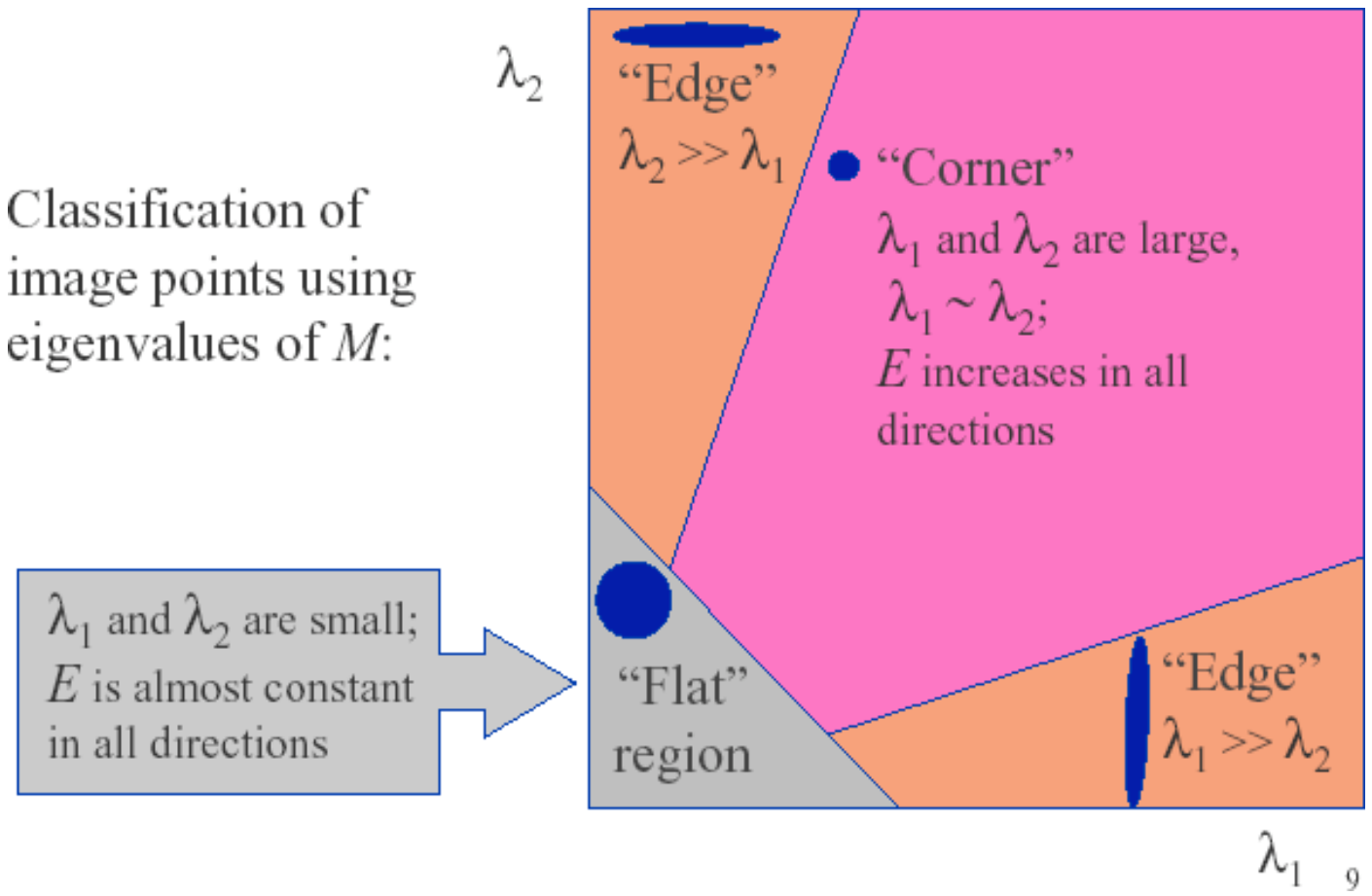
The short axis of ellipse is the fast change direction
The long axis of ellipse is the slow change direction



Fitting Ellipse to each Set of Points



Classification via Eigenvalues



Corner Response Measure

Measure of corner response:

$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

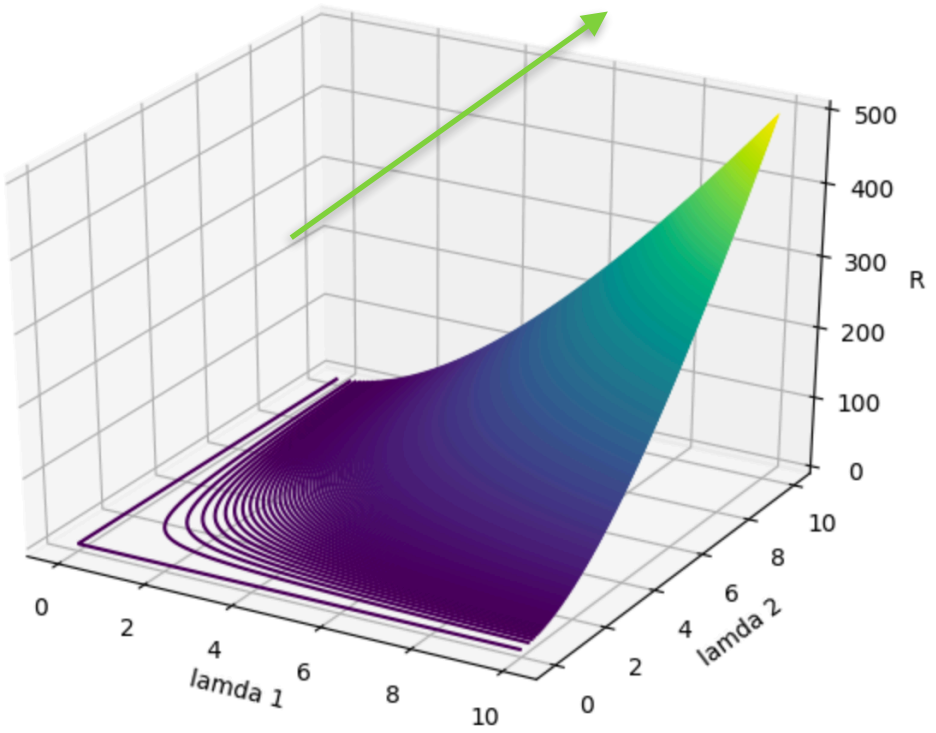
(k is an empirically determined constant; $k = 0.04$ - 0.06)

Corner Response Measure Simplification

To get rid of the weighting constant , it is often easier to use the quotient

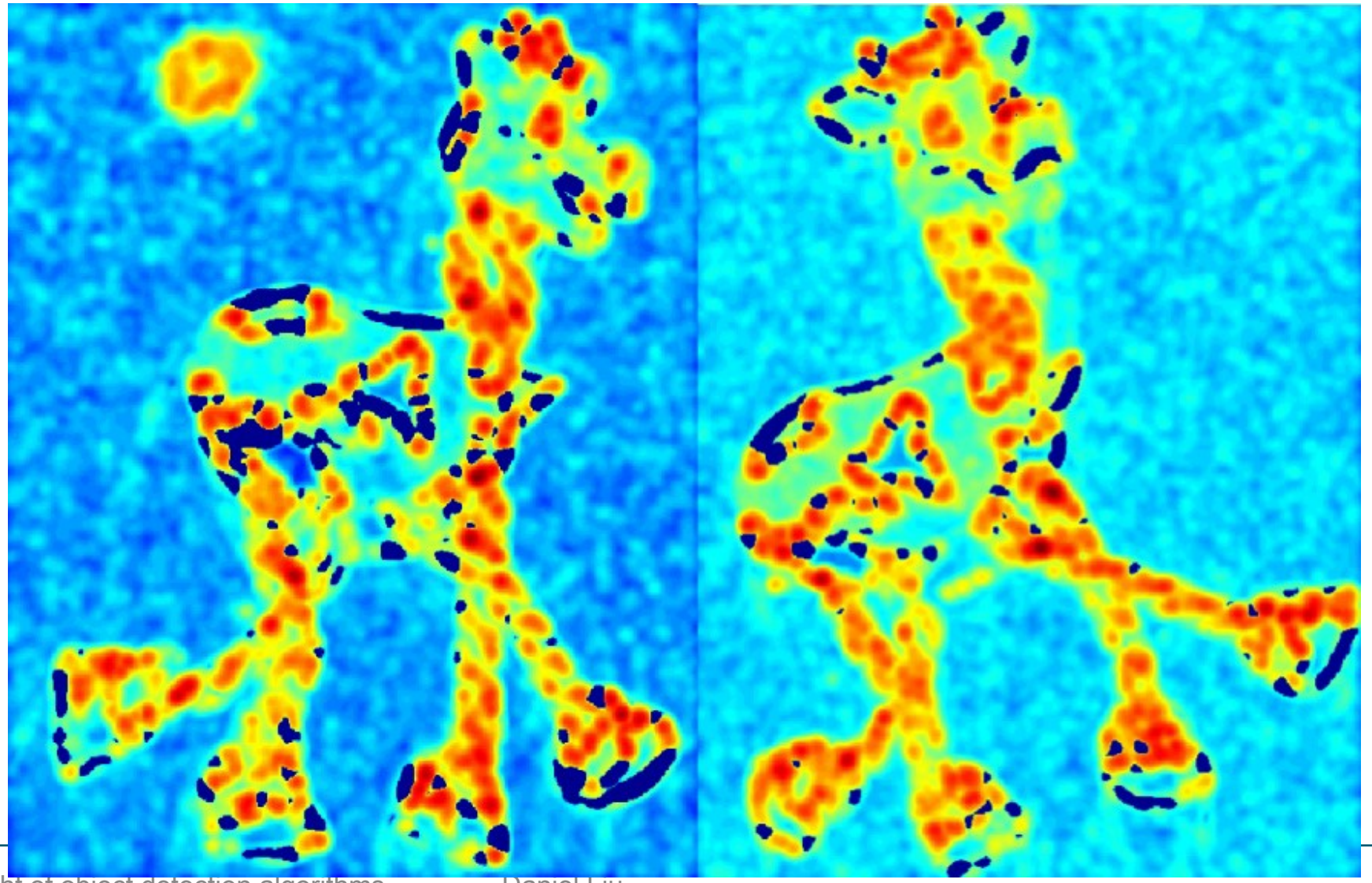
$$\rightarrow \frac{det\ (M)}{trace\ (M)} = \frac{I_x^2 I_y^2 - (I_x I_y)^2}{I_x^2 + I_y^2}$$

$$\frac{\lambda_1 \times \lambda_2}{\lambda_1 + \lambda_2}$$

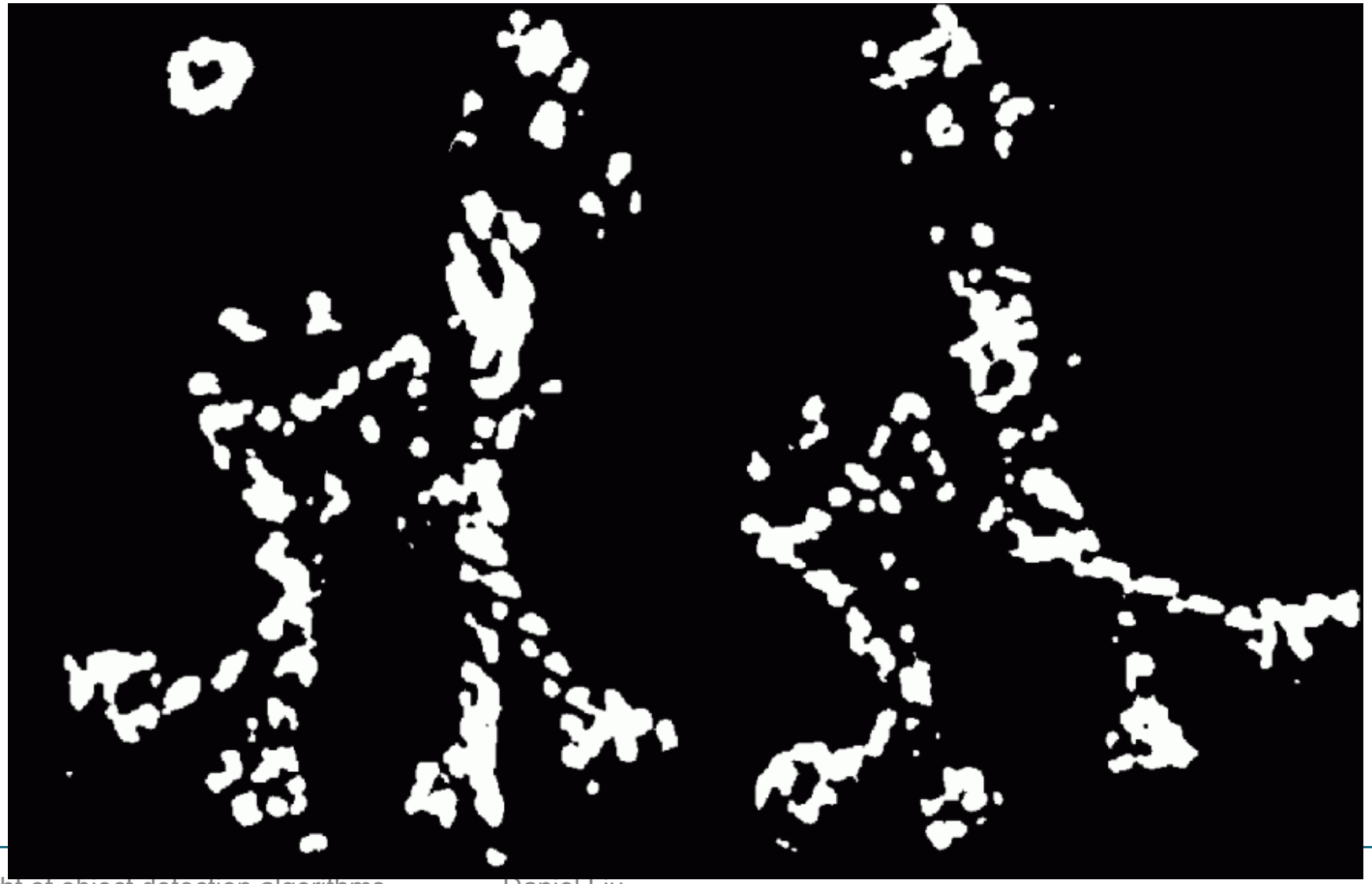


Harris detector example

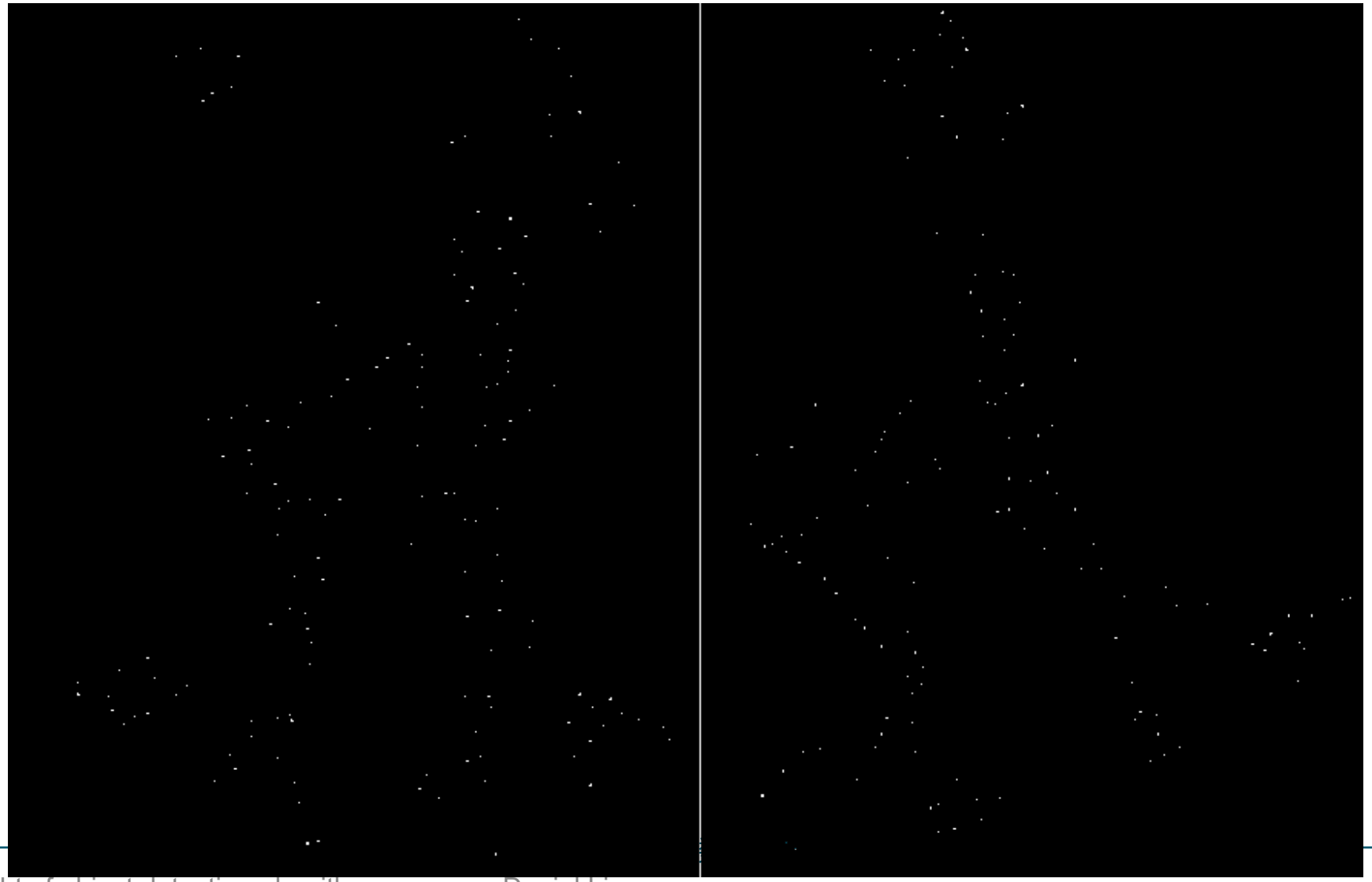
Corner Response (red high, blue low)



Threshold (Corner Response > threshold value)



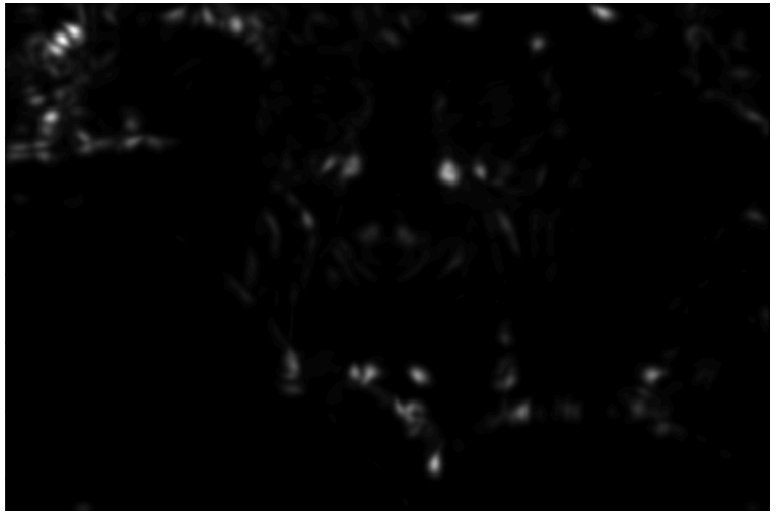
Threshold (Corner Response > threshold value)



Harris features (in red)

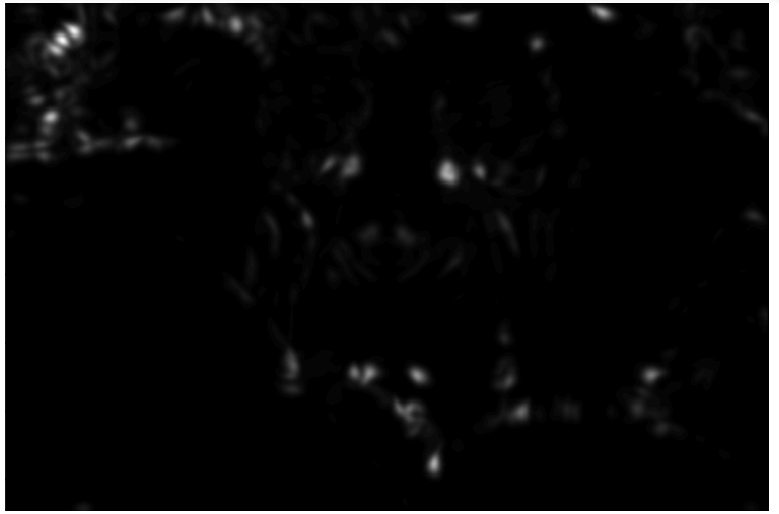
Harris Corner Points Codding

```
from scipy.ndimage import filters

def compute_harris_response(im,sigma=3):
  """ Compute the Harris corner detector response function
    for each pixel in a graylevel image. """

  # derivatives
  imx = zeros(im.shape)
  filters.gaussian_filter(im, (sigma,sigma), (0,1), imx)
  imy = zeros(im.shape)
  filters.gaussian_filter(im, (sigma,sigma), (1,0), imy)

  # compute components of the Harris matrix
  Wxx = filters.gaussian_filter(imx*imx,sigma)
  Wxy = filters.gaussian_filter(imx*imy,sigma)
  Wyy = filters.gaussian_filter(imy*imy,sigma)

  # determinant and trace
  Wdet = Wxx*Wyy - Wxy**2
  Wtr = Wxx + Wyy

  return Wdet / Wtr
```



Harris Corner Points Codding

```
def get_harris_points(harrisim,min_dist=10,threshold=0.1):
  """ Return corners from a Harris response image
    min_dist is the minimum number of pixels separating
    corners and image boundary. """

  # find top corner candidates above a threshold
  corner_threshold = harrisim.max() * threshold
  harrisim_t = (harrisim > corner_threshold) * 1

  # get coordinates of candidates
  coords = array(harrisim_t.nonzero()).T

  # ...and their values
  candidate_values = [harrisim[c[0],c[1]] for c in coords]

  # sort candidates
  index = argsort(candidate_values)

  # store allowed point locations in array
  allowed_locations = zeros(harrisim.shape)
  allowed_locations[min_dist:-min_dist,min_dist:-min_dist] = 1

  # select the best points taking min_distance into account
  filtered_coords = []
  for i in index:
    if allowed_locations[coords[i,0],coords[i,1]] == 1:
       filtered_coords.append(coords[i])
       allowed_locations[(coords[i,0]-min_dist):(coords[i,0]+min_dist),
             (coords[i,1]-min_dist):(coords[i,1]+min_dist)] = 0

   return filtered_coords
```



Characters of Harris Point

- Harris corner detection is not sensitive to changes in brightness and contrast.
- Harris corner detection has rotation invariance, but does not have scale invariance. As shown in the figure below, the corner points at small scales may be considered as image edges after being enlarged.

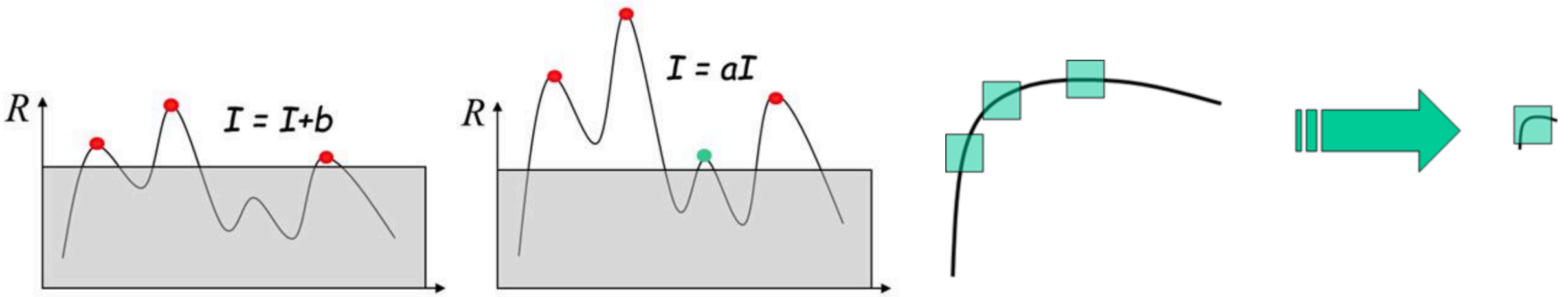


Table of contents

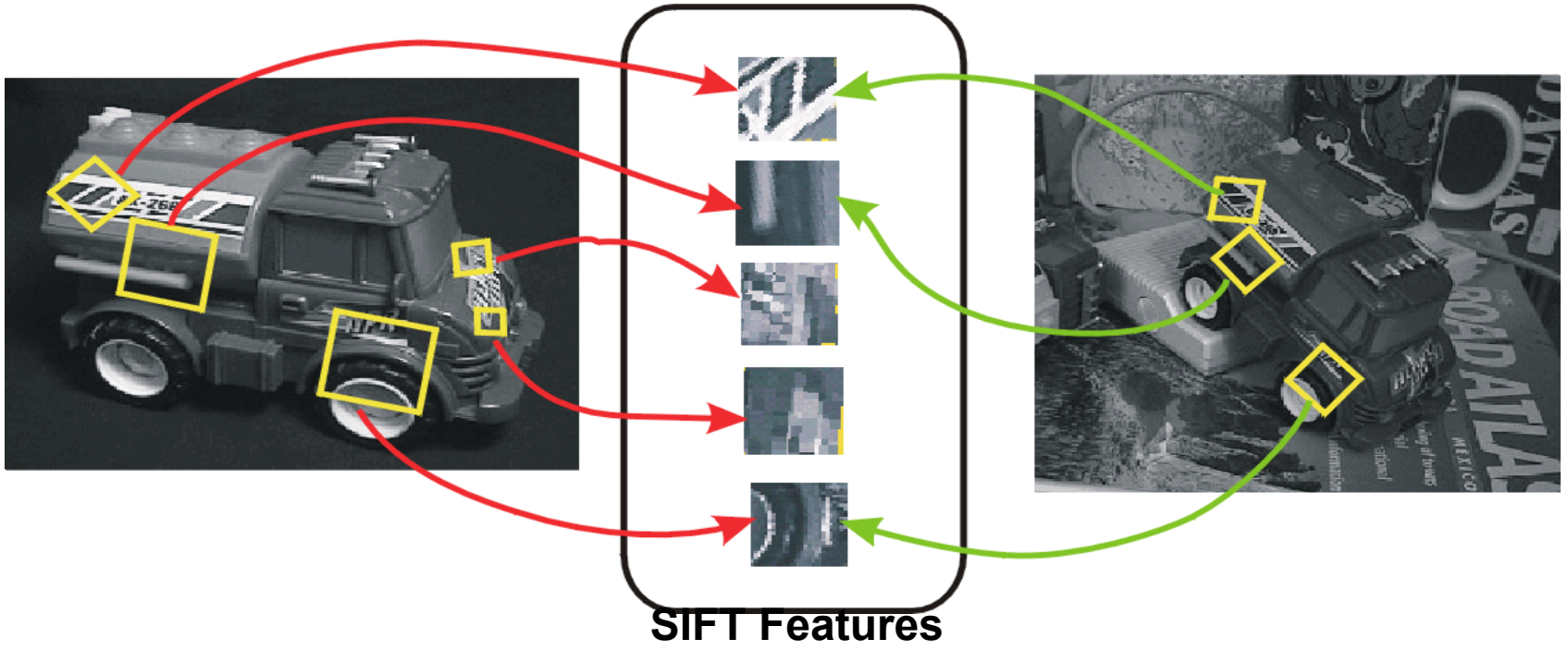
SIFT: Motivation

- The Harris operator is not invariant to scale and correlation is not invariant to rotation[1].
- For better image matching, Lowe's goal was to develop an interest operator that is invariant to scale and rotation.
- Also, Lowe aimed to create a descriptor that was robust to the variations corresponding to typical viewing conditions. The descriptor is the most-used part of SIFT.

[1]But Schmid and Mohr developed a rotation invariant descriptor for it in 1997.

Idea of SIFT (Scale Invariant Feature Transform)

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



Claimed Advantages of SIFT

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects
- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to wide range of differing feature types, with each adding robustness

Overall Procedure at a High Level

1. Scale-space extrema detection

   Search over multiple scales and image locations.

2. Keypoint localization

   Fit a model to detrmine location and scale.
   Select keypoints based on a measure of stability.

3. Orientation assignment

   Compute best orientation(s) for each keypoint region.

4. Keypoint description

   Use local image gradients at selected scale and rotation to describe each keypoint region.

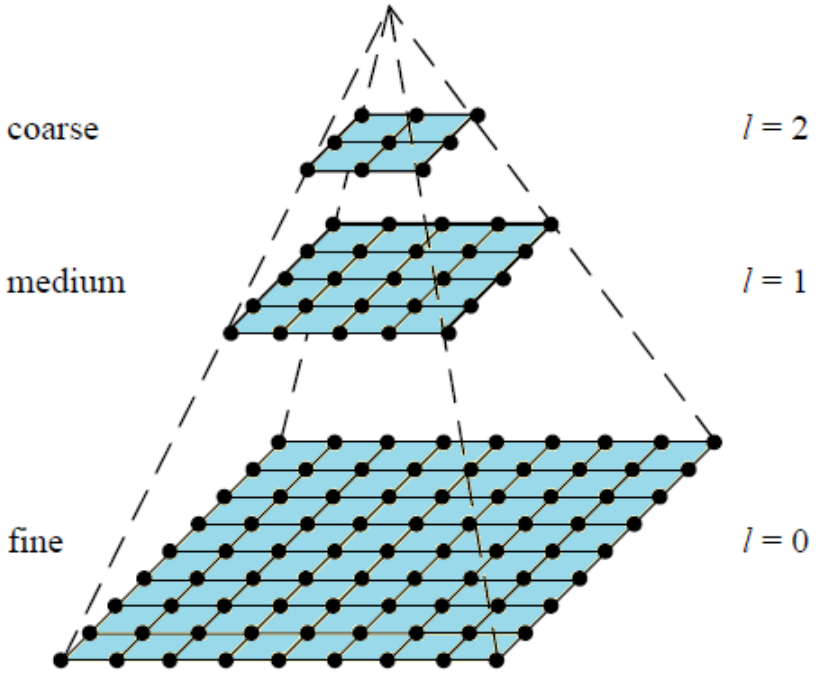
Scale-space extrema detection

- Goal: Identify locations and scales that can be repeatably assigned under different views of the same scene or object.
- Method: search for stable features across multiple scales using a continuous function of scale.
- Prior work has shown that under a variety of assumptions, the best function is a Gaussian function.
- The scale space of an image is a function $L(x,y,\sigma)$ that is produced from the convolution of a Gaussian kernel (at different scales) with the input image.

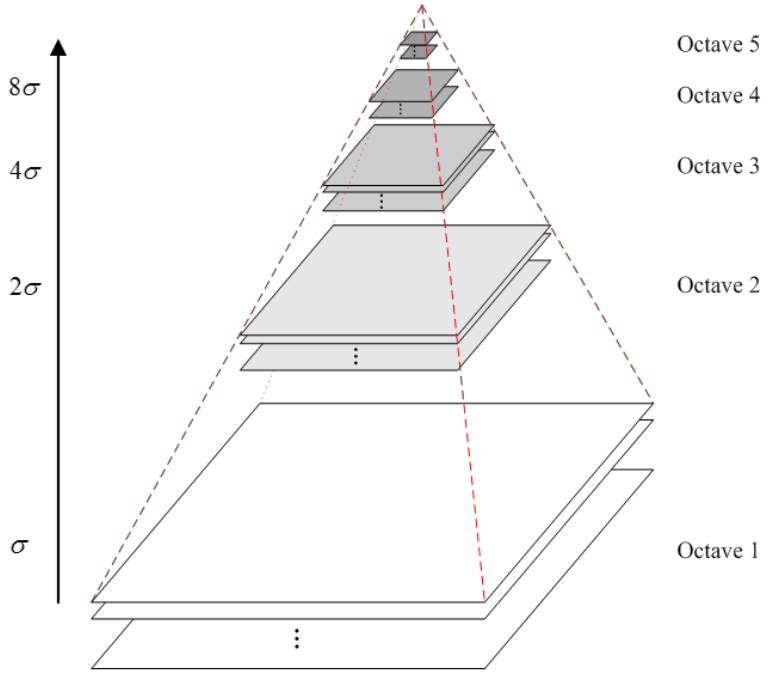
Claimed Advantages of SIFT

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects
- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to wide range of differing feature types, with each adding robustness

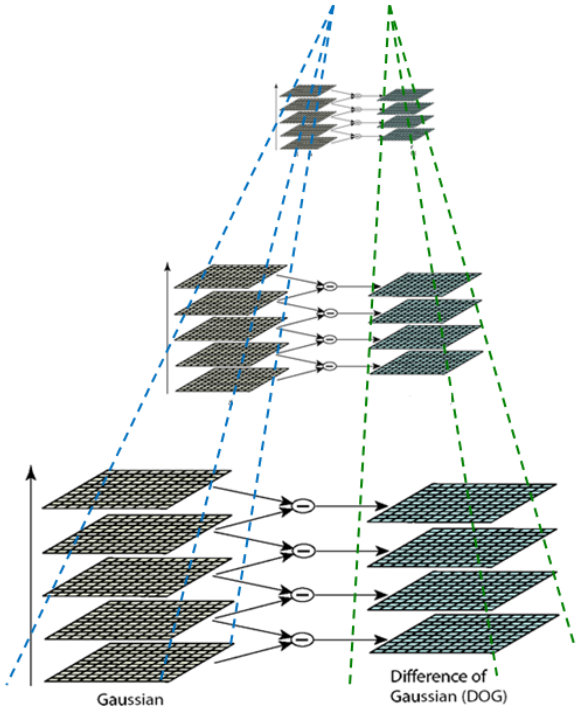
3 Pyramids: Subsampling, LoG, DoG



Subsampling

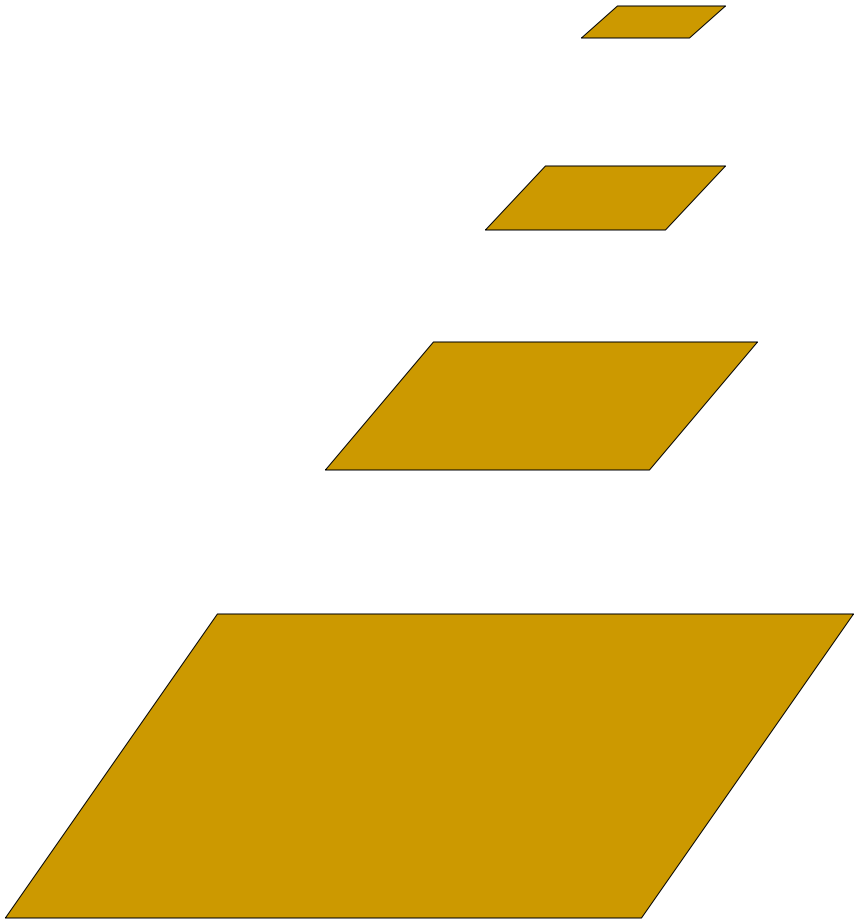


LoG



DoG

Step1 Subsampling: Image Pyramids



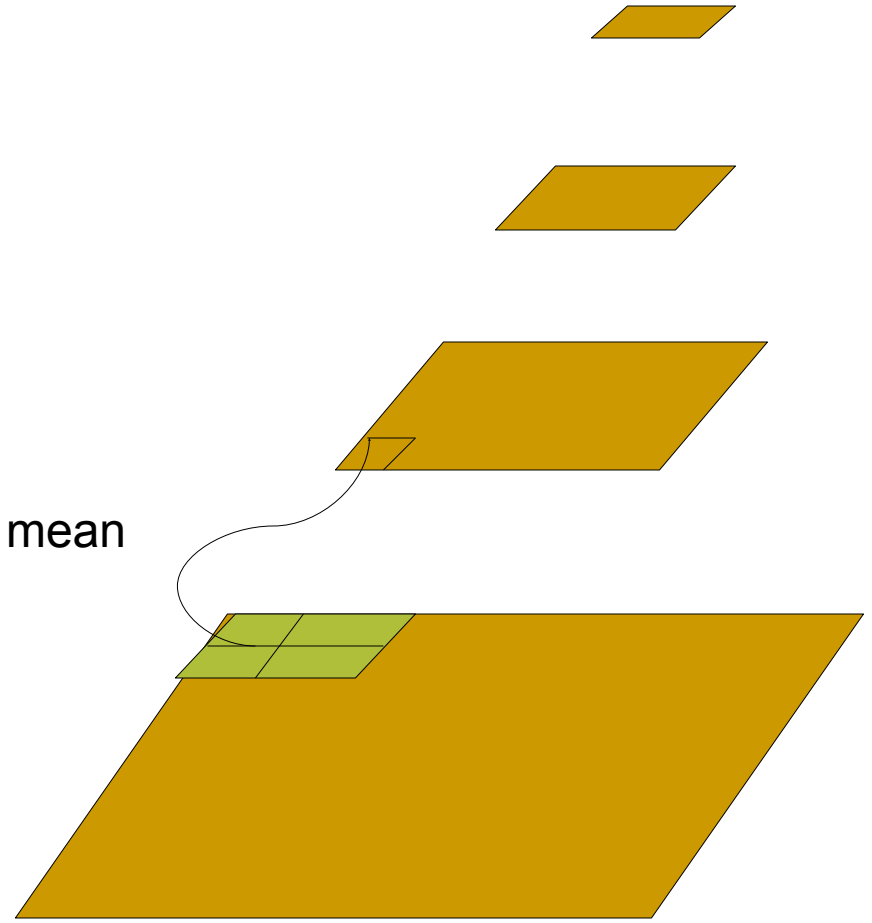
And so on.

3rd level is derived from the 2nd level according to the same funtion

2nd level is derived from the original image according to some function

Bottom level is the original image.

Step1 Subsampling: Mean Pyramid



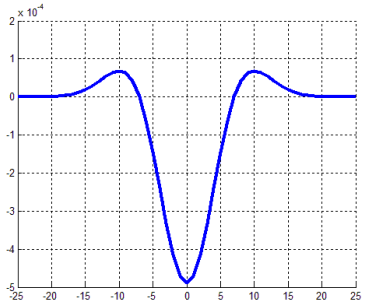
And so on.

At 3rd level, each pixel is the mean of 4 pixels in the 2nd level.

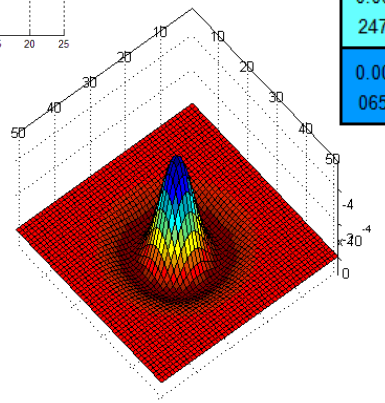
At 2nd level, each pixel is the mean of 4 pixels in the original image.

Bottom level is the original image.

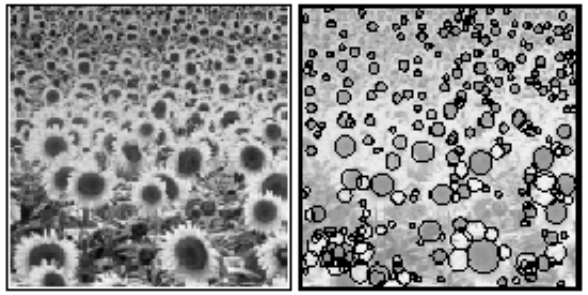
Step 2: LoG: Laplacian of Gaussian

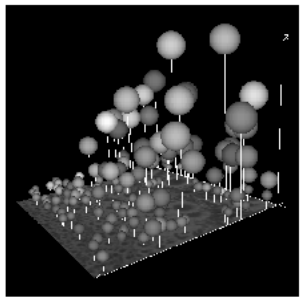


0.0000 06586	0.0004 24781	0.0133 0373	0.0004 24781	0.0000 06586
0.0004 24781	0.0273 984	0.1098 78	0.0273 984	0.0004 24781
0.0133 0373	0.1098 78	**0.4406 55**	0.1098 78	0.0133 0373
0.0004 24781	0.0273 984	0.1098 78	0.0273 984	0.0004 24781
0.0000 06586	0.0004 24781	0.0133 0373	0.0004 24781	0.0000 06586



- **Laplacian of Gaussian** kernel
  - Scale normalised (x by scale^2)
  - Proposed by Lindeberg
- Scale-space detection
  - Find local maxima across scale/space
  - A good “blob” detector





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

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

Step 2: Lowe's Scale-space Interest Points

- First smooth (Gaussian filter),
- Then, find zero-crossings (Laplacian filter):
  - $O(x,y) = \nabla^2(I(x,y) * G(x,y))$

Just another linear filter.

$$\nabla^2(f(x,y) \otimes G(x,y)) = \nabla^2 G(x,y) \otimes f(x,y)$$

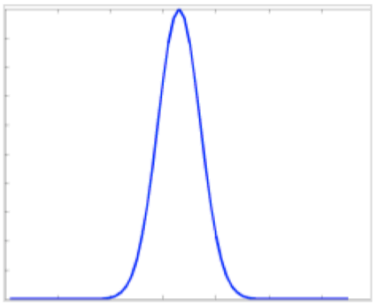
Laplacian of Gaussian-filtered image

Laplacian of Gaussian (LoG) -filtered image

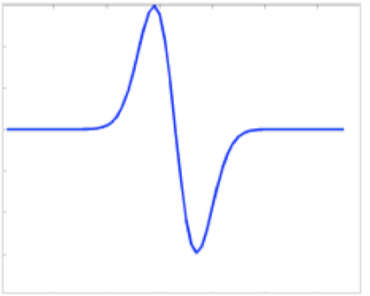
Do you see the distinction?

Derivate of Gaussian

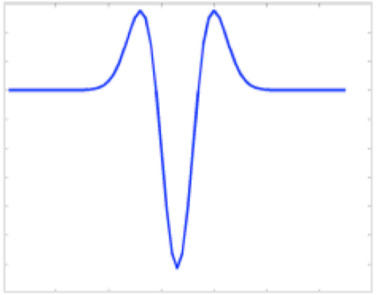
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

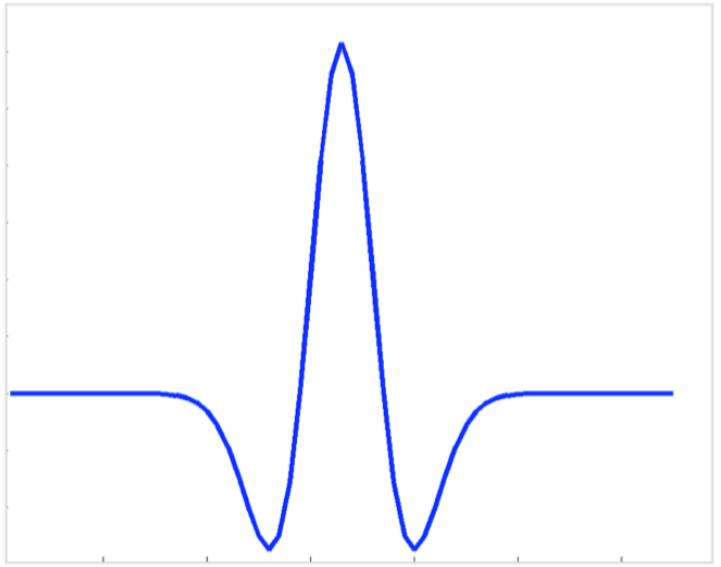


$$g''(x) = (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}) e^{-\frac{x^2}{2\sigma^2}}$$

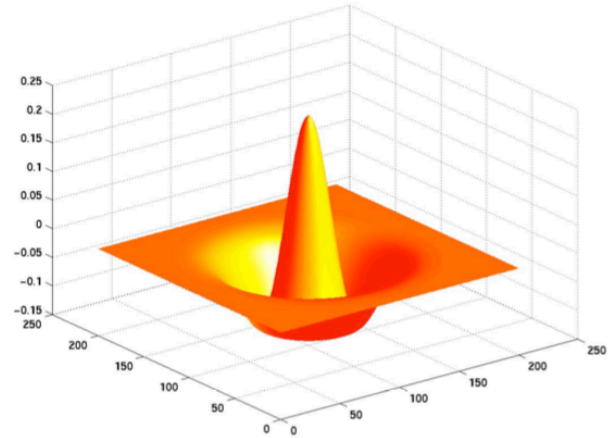


LoG: Maxico Hat

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

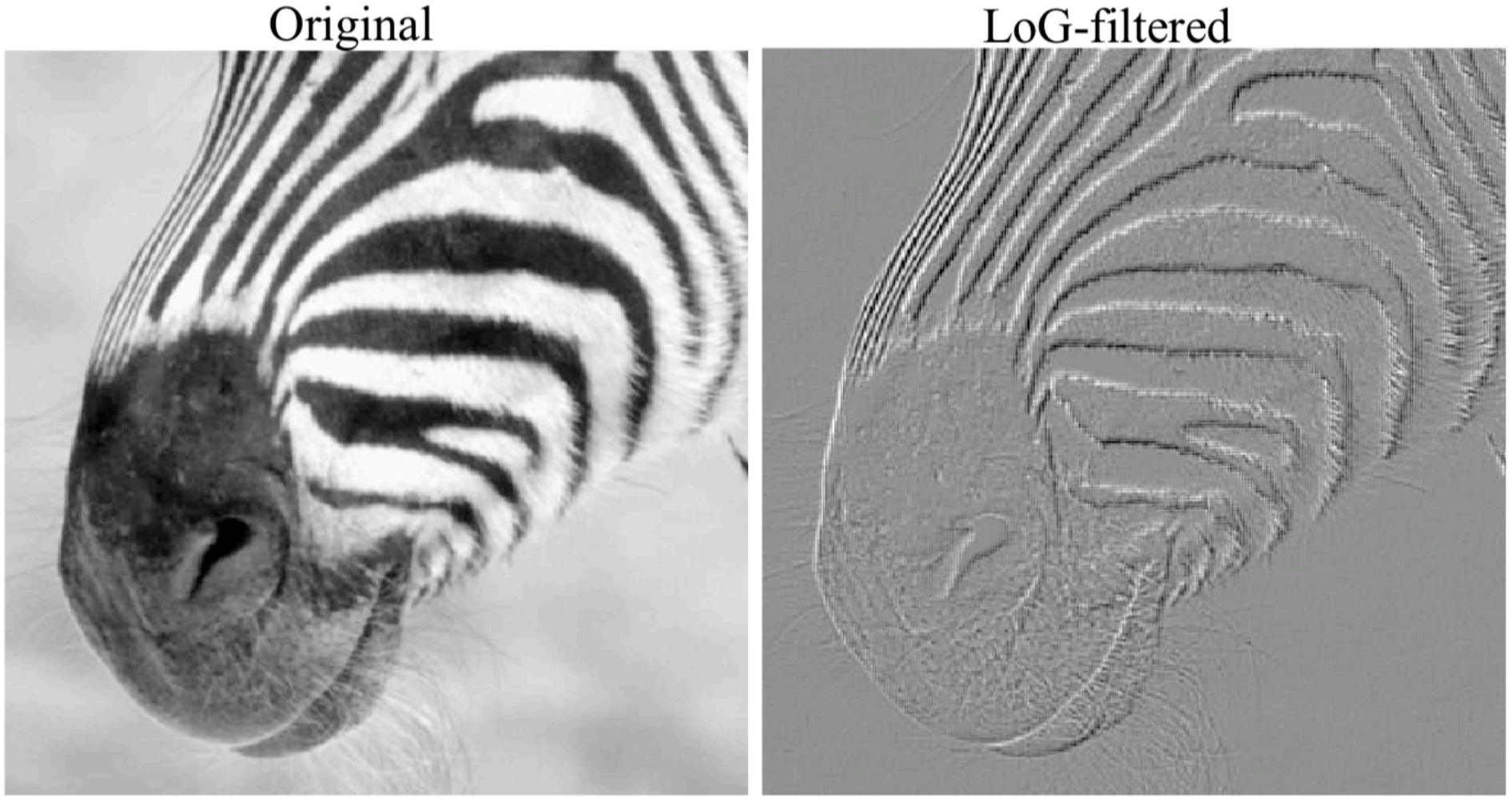


2D analog

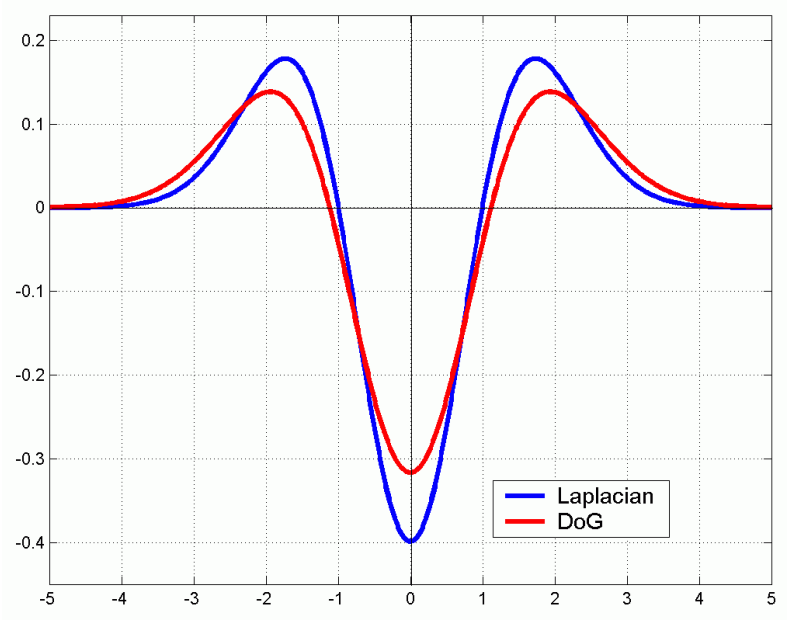


LoG "Mexican Hat"

LoG Extraction



LoG can be approximate by a Difference of two Gaussians (DoG) at different scales



- Gaussian is an ad hoc solution of heat diffusion equation

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G.$$

- Hence

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G.$$

- k is not necessarily very small in practice

DoG:

$$D(\mathbf{x}, \sigma) = [G_{k\sigma}(\mathbf{x}) - G_\sigma(\mathbf{x})] * I(\mathbf{x}) = [G_{k\sigma} - G_\sigma] * I = I_{k\sigma} - I_\sigma$$

Step 2: Lowe's Pyramid Scheme

- Scale space is separated into octaves:
  - Octave 1 uses scale σ
  - Octave 2 uses scale 2σ
  - etc.

- In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images.

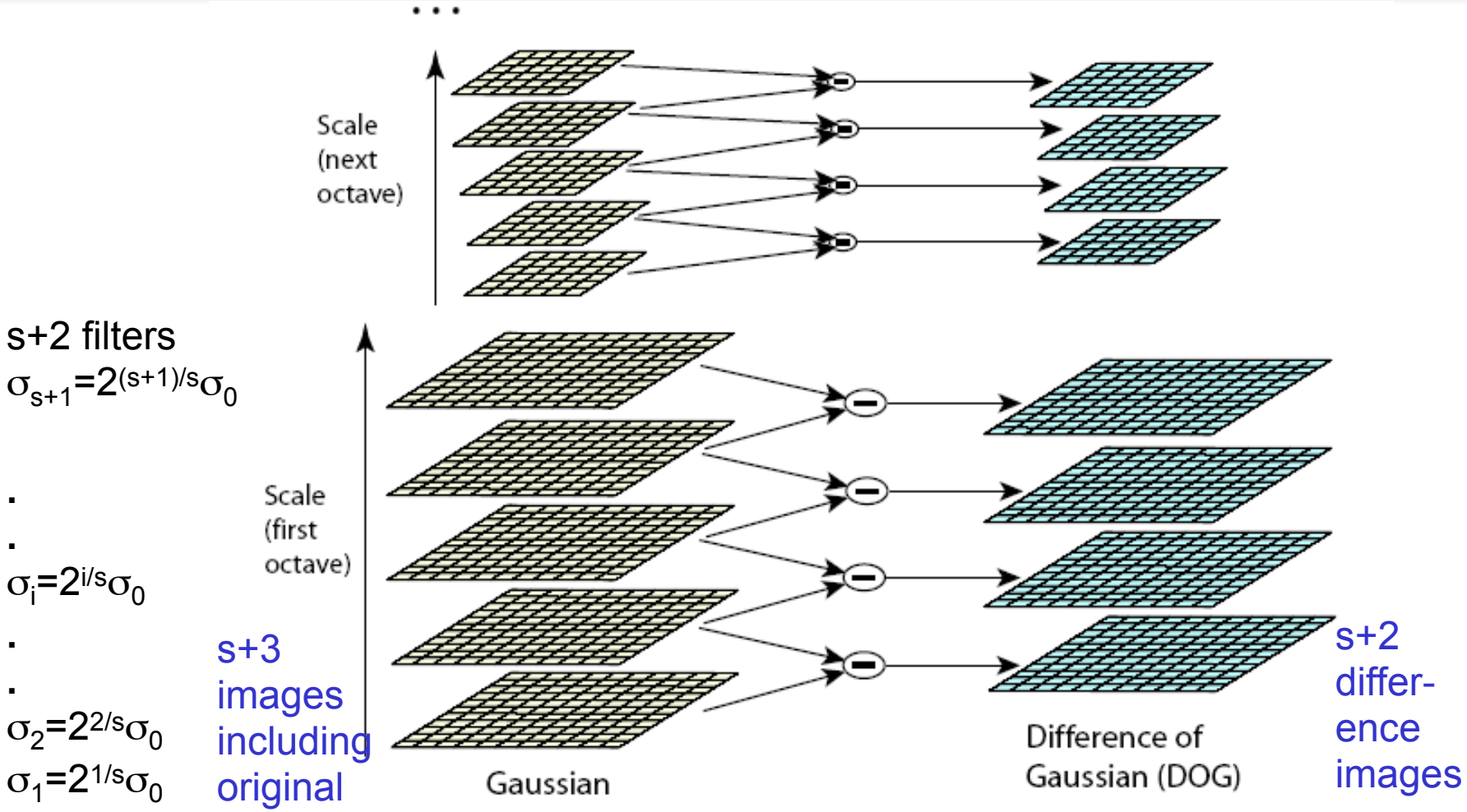
- Adjacent Gaussians are subtracted to produce the DOG

- After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image ¼ the size to start the next level.

Step 2: Lowe's Pyramid Scheme

$\sigma = 0$ $\sigma = 1$ $\sigma = 2^{\frac{1}{2}}$ $\sigma = 2$ $\sigma = 2^{\frac{3}{2}}$

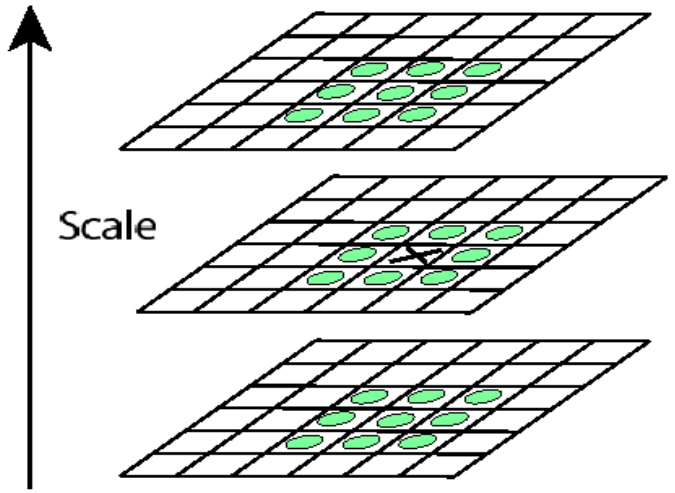
Step 3: Difference of Gaussian



The parameter **s** determines the number of images per octave.

Step4: Key point localization

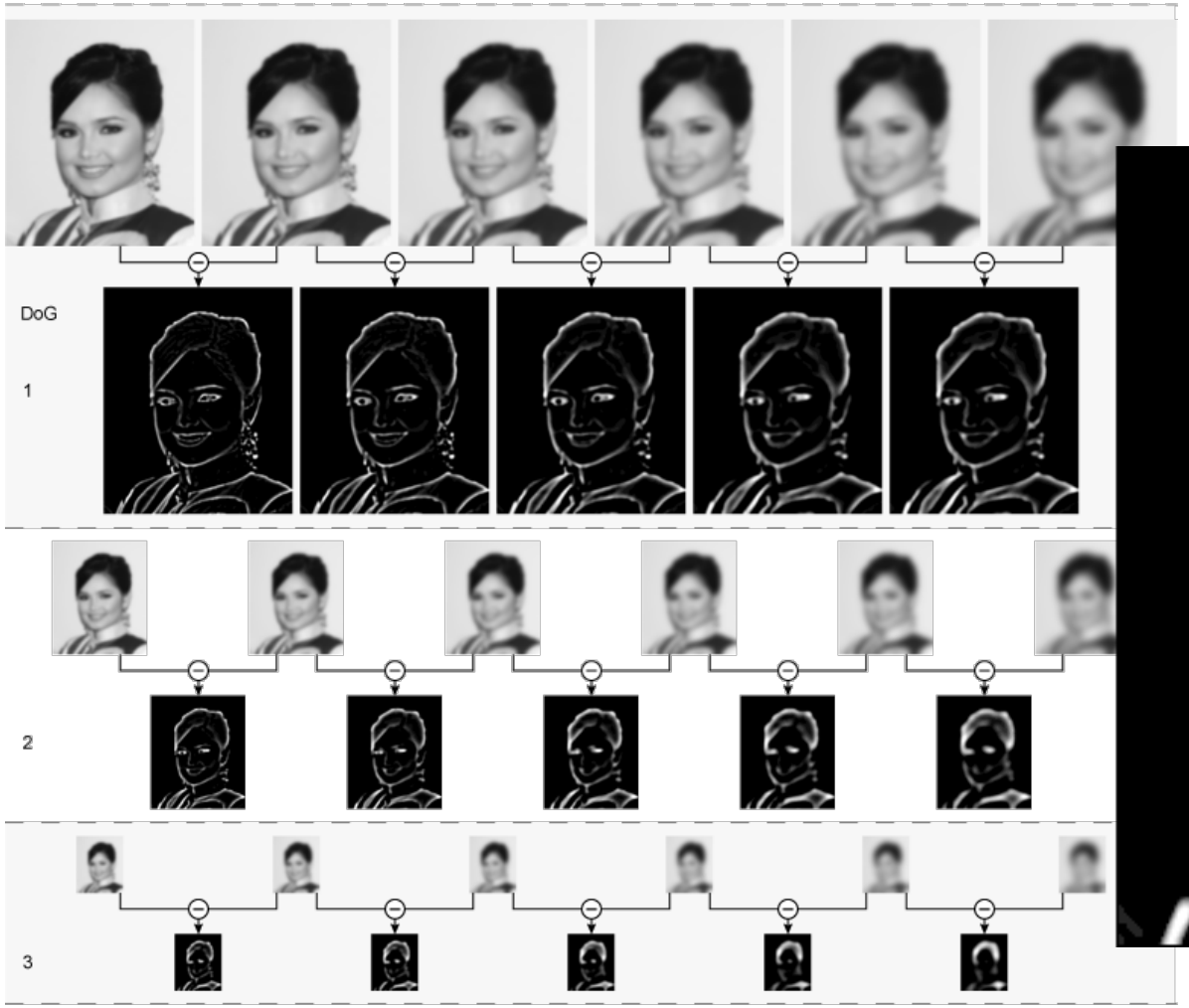
s+2 difference images.
top and bottom ignored.
s planes searched.

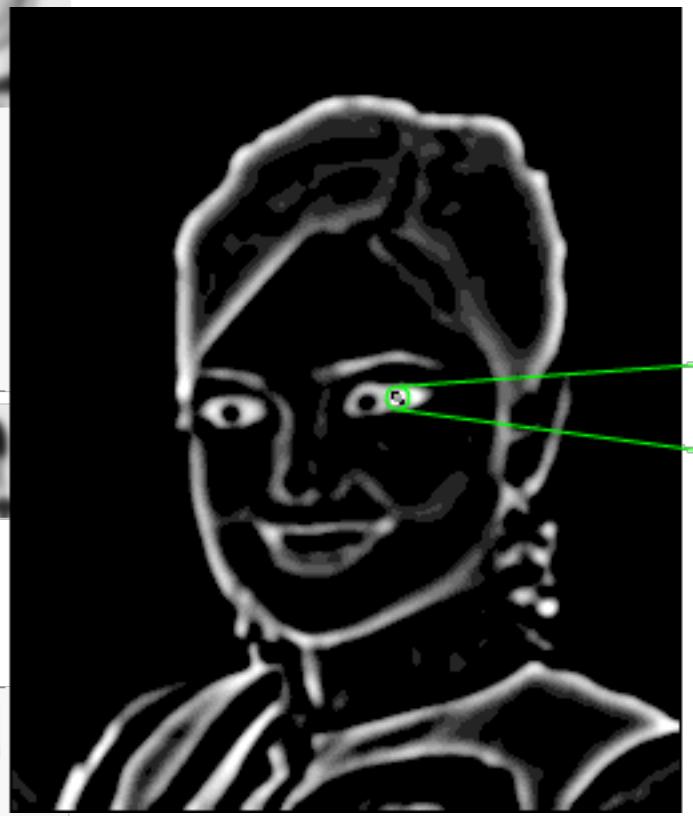


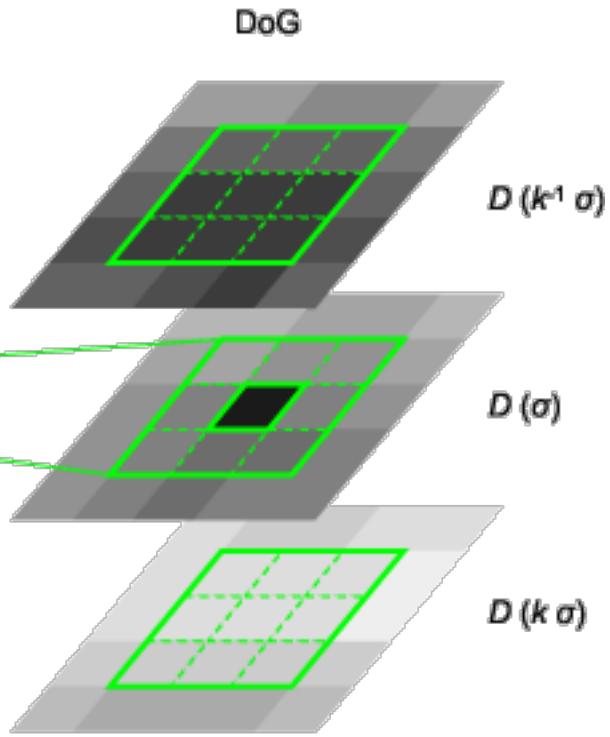
- Detect maxima and minima of difference-of-Gaussian in scale space
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below

For each max or min found, output is the **location** and the **scale**.

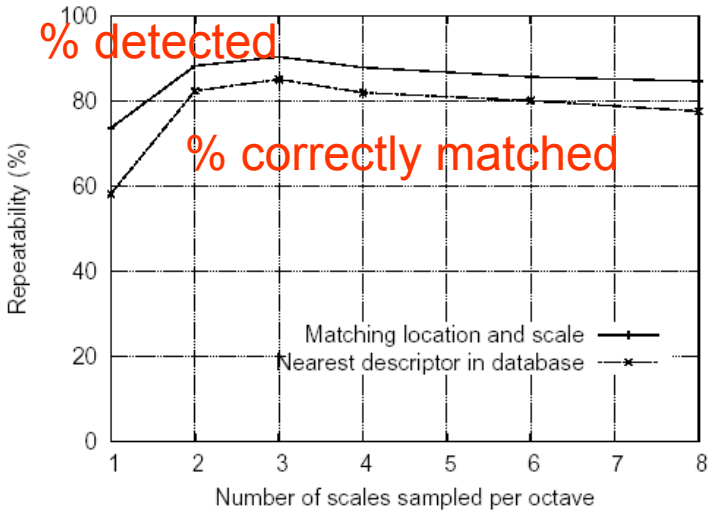
Keypoint localization



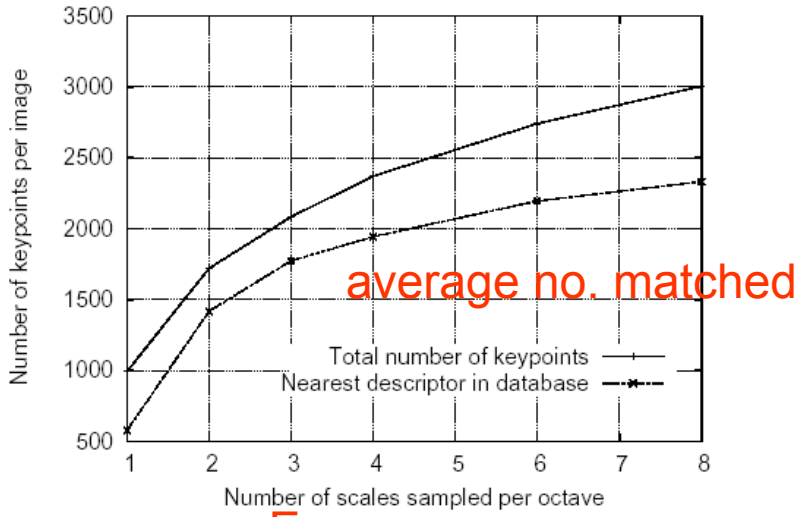




Scale-space extrema detection: experimental results over 32 images that were synthetically transformed and noise added.



Stability



Expense

- Sampling in scale for efficiency
  - How many scales should be used per octave? S=?
    - More scales evaluated, more keypoints found
    - S < 3, stable keypoints increased too
    - S > 3, stable keypoints decreased
    - S = 3, maximum stable keypoints found

Localizing SIFT Keypoints

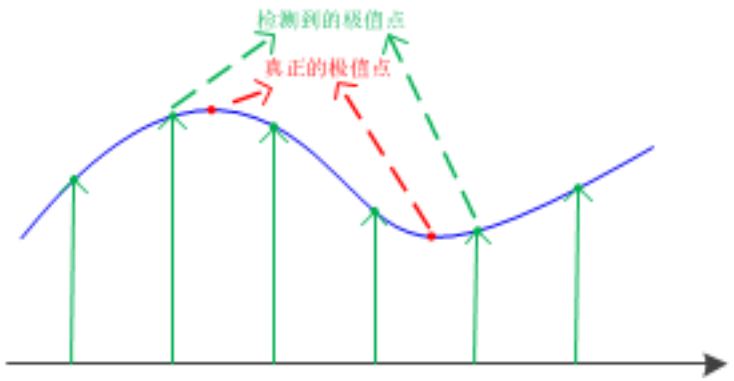
Better results by interpolating than by taking center of cell as location
– Fit quadratic to surrounding points

Taylor expansion around point
– Where D is difference of Gaussian

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Offset of extremum as location
– Use finite differences

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}$$



Step 5: Eliminating the Edge Response (Hessein Matrix)

- Reject flats:
  - $|D(\hat{\mathbf{x}})|$ < 0.03
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

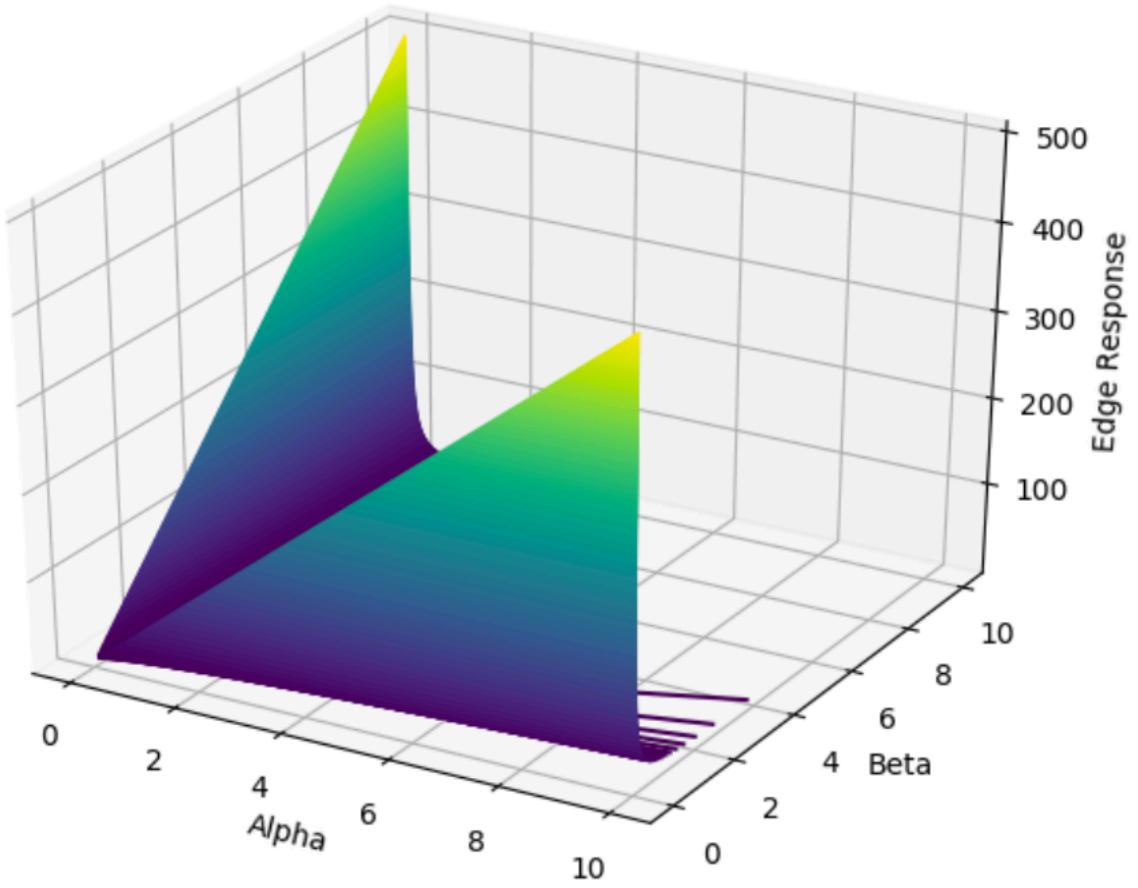
Let $r = \alpha/\beta$. So $\alpha = r\beta$

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

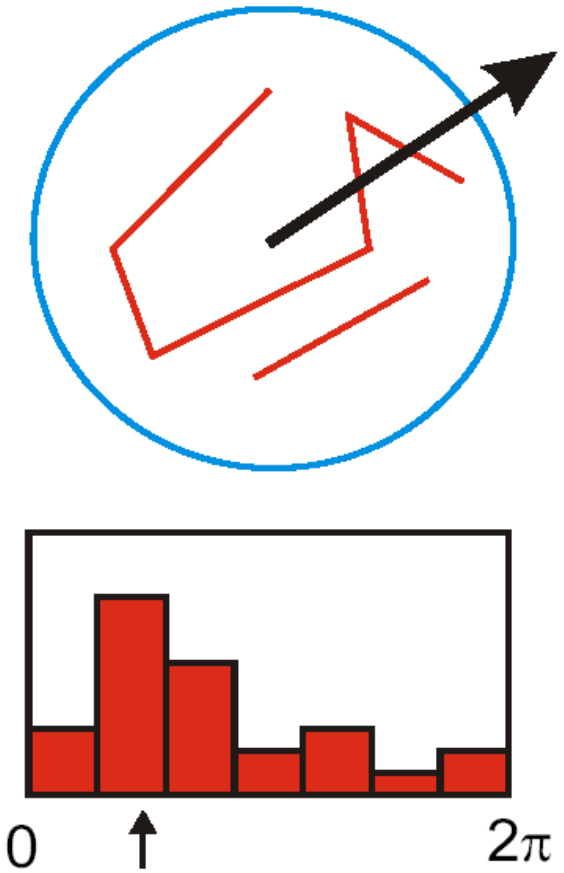
$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

  - r < 10
- What does this look like?

Edge Response (Hessein Matrix)



Step 6. Orientation assignment



- Create histogram of local gradient directions at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates (x, y, scale,orientation)

If 2 major orientations, use both.

Keypoint localization with orientation

233x189

832

initial keypoints

729

keypoints after
gradient threshold

536

keypoints after
ratio threshold

Step 7. Keypoint Descriptors

- At this point, each keypoint has
  - location
  - scale
  - orientation
- Next is to compute a descriptor for the local image region about each keypoint that is
  - highly distinctive
  - invariant as possible to variations such as changes in viewpoint and illumination

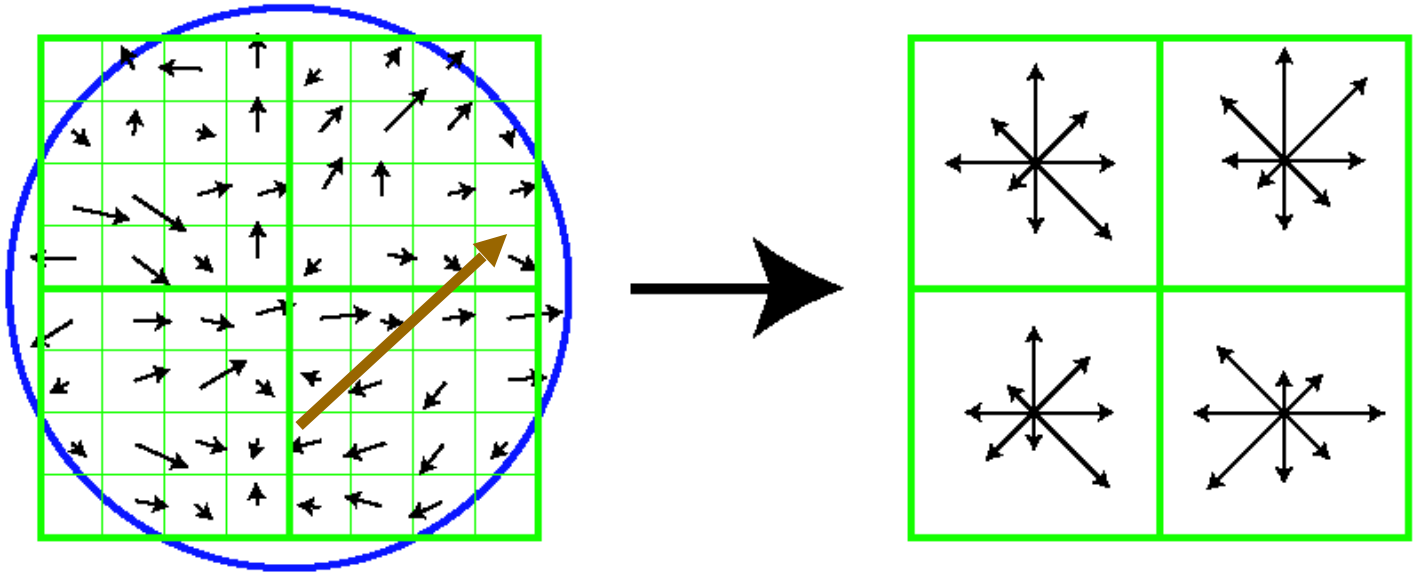
$$m(x,y) = \sqrt{\left(L(x+1,y) - L(x-1,y)\right)^2 + \left(L(x,y+1) - L(x,y-1)\right)^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

Lowe's Keypoint Descriptor

- use the normalized region about the keypoint
- compute gradient magnitude and orientation at each point in the region
- weight them by a Gaussian window overlaid on the circle
- create an orientation histogram over the 4 X 4 subregions of the window
- 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 times 8 directions gives a vector of 128 values.

Lowe’s Keypoint Descriptor (shown with 2 X 2 descriptors over 8 X 8)

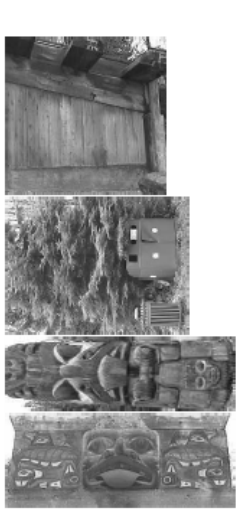


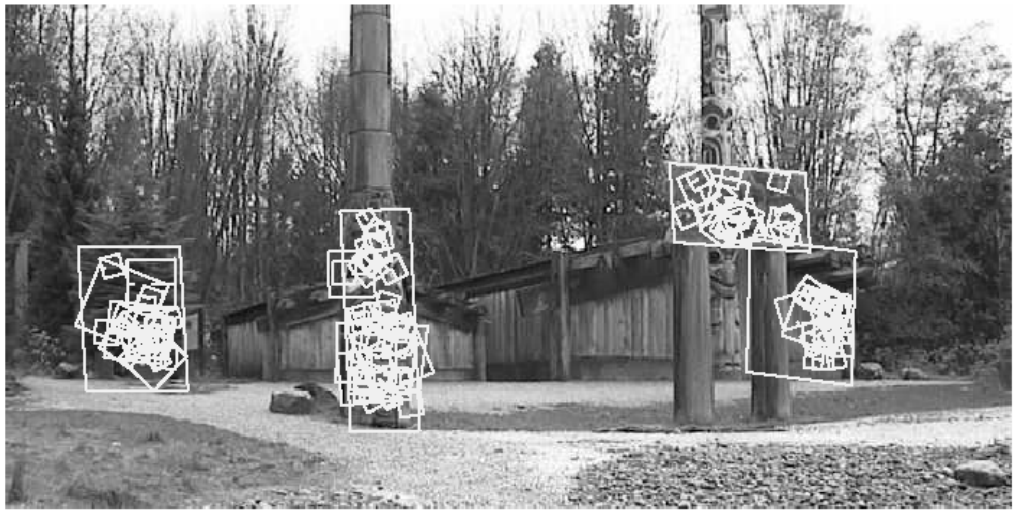
In experiments, 4x4 arrays of 8 bin histogram is used,
a total of 128 features for one keypoint

Uses for SIFT

- Feature points are used also for:
  - Image alignment (homography, fundamental matrix)
  - 3D reconstruction (e.g. Photo Tourism)
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … many others

Using SIFT for Matching “Objects”





Sift Match

Sony Aibo

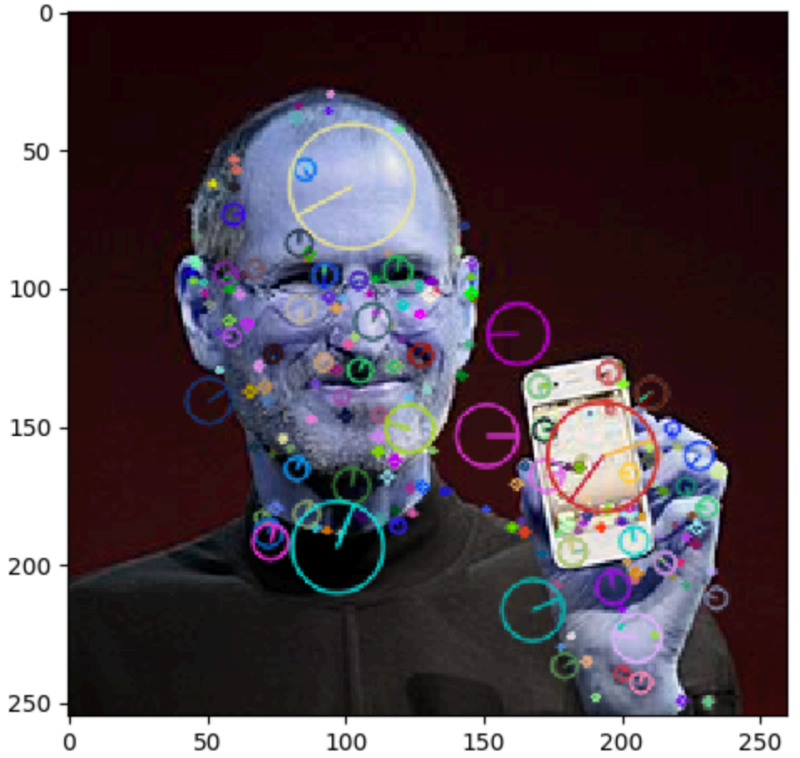
SIFT usage:

- Recognize charging station
- Communicate with visual cards
- Teach object recognition

Coding

Fonction:

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

im = cv2.imread('Images/jobs.jpg')
SIFT = cv2.xfeatures2d.SIFT_create()
SURF = cv2.xfeatures2d.SURF_create()
ORB = cv2.ORB_create()

kps_SIFT,descfs_SIFT = SIFT.detectAndCompute(im, None)
kps_SURF,descfs_SURF = SURF.detectAndCompute(im, None)
kps_ORB, descfs_ORB = ORB.detectAndCompute(im, None)


imgSIFT = cv2.drawKeypoints(im, kps_SIFT, None, color=(255,0,0))
imgSURF = cv2.drawKeypoints(imgSIFT, kps_SURF, None, color=(0,255,0))
imgORB = cv2.drawKeypoints(imgSURF, kps_ORB, None, color=(0,0,255))
```




```
# Now we drawn the gray image and overlay the Key Points (kp)
img = cv2.drawKeypoints(im, kps_SIFT, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Plot it to the screen, looks a little small
plt.imshow(imgSIFT)
plt.show()
```

Table of contents

Feature matching

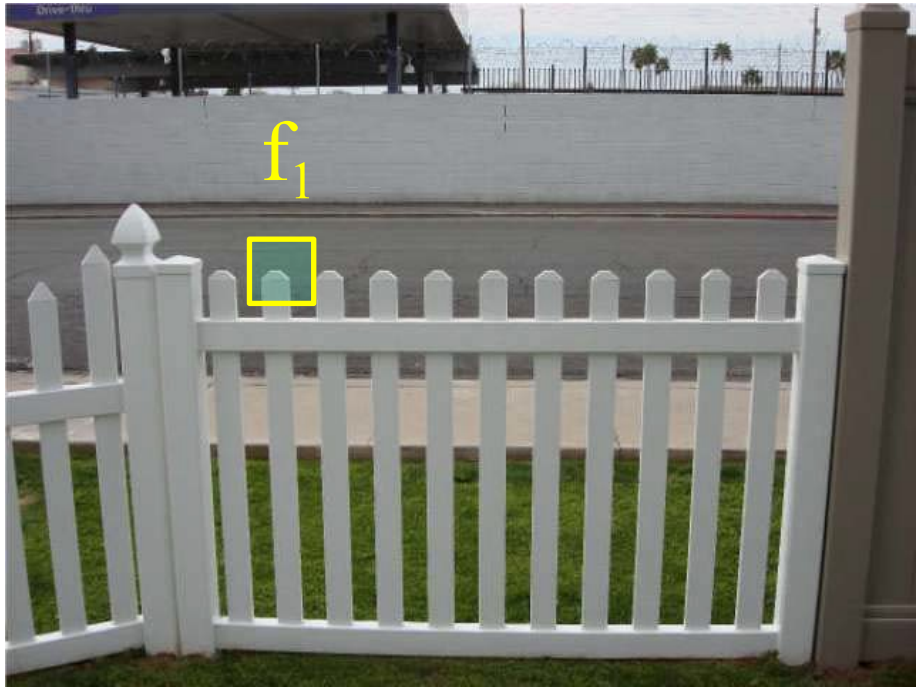
Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

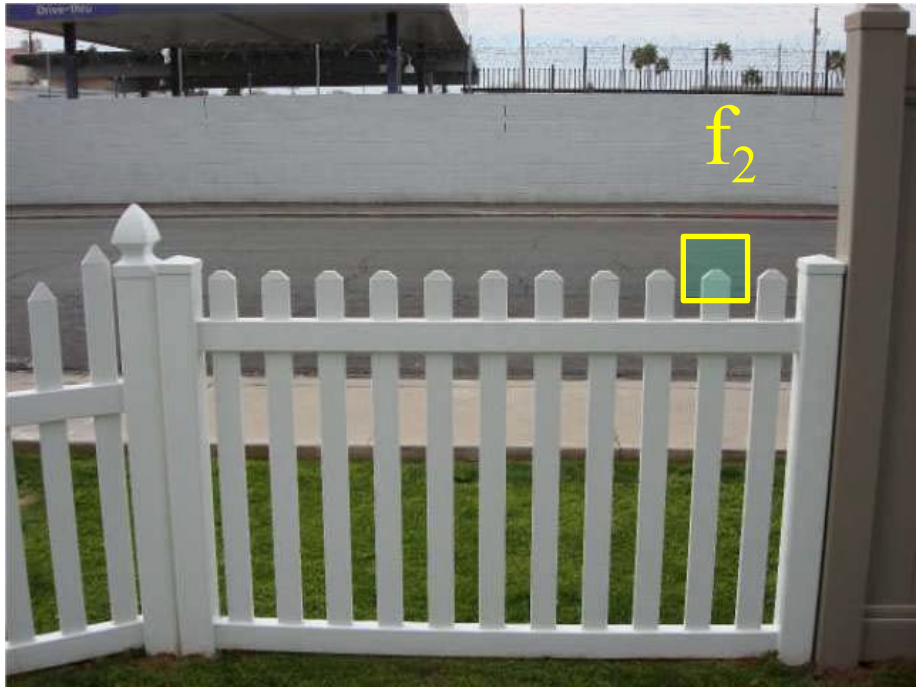
Feature distance: SSD

How to define the similarity between two features f_1, f_2?

- Simple approach is SSD(f_1, f_2)
  - sum of square differences between entries of the two descriptors
  - Doesn't provide a way to discard ambiguous (bad) matches



I_1

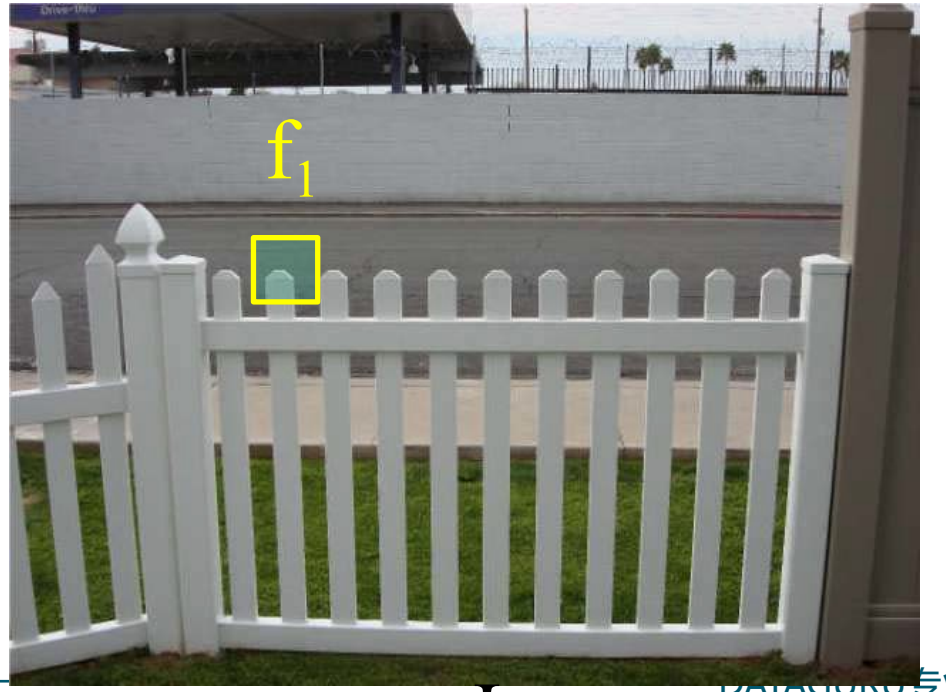


I_2

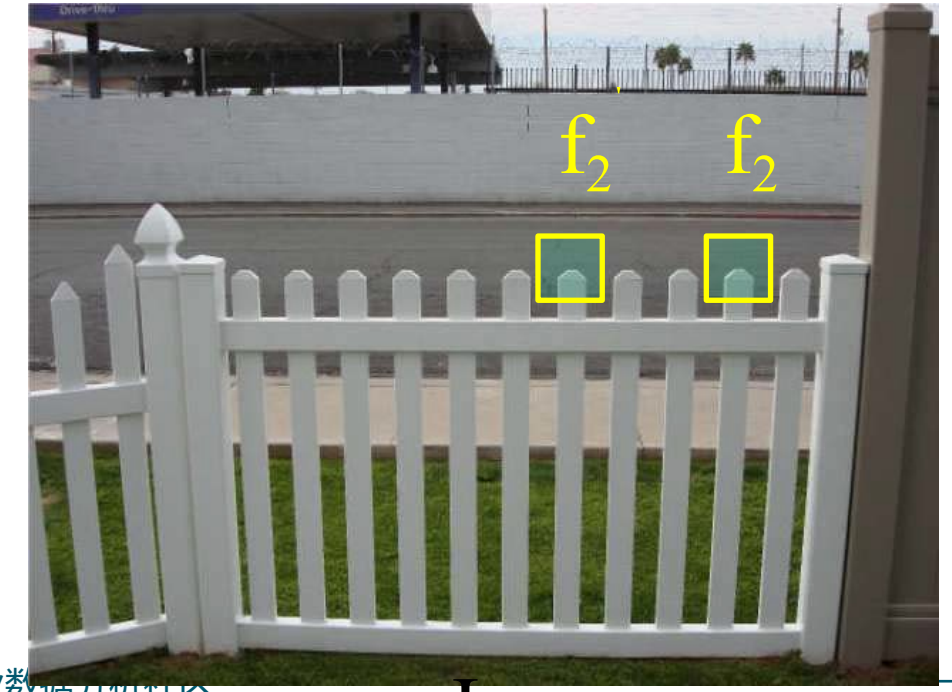
Feature distance: Ratio of SSDs

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = SSD(f_1, f_2) / SSD(f_1, f_2')
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - An ambiguous/bad match will have ratio close to 1
  - Look for unique matches which have low ratio

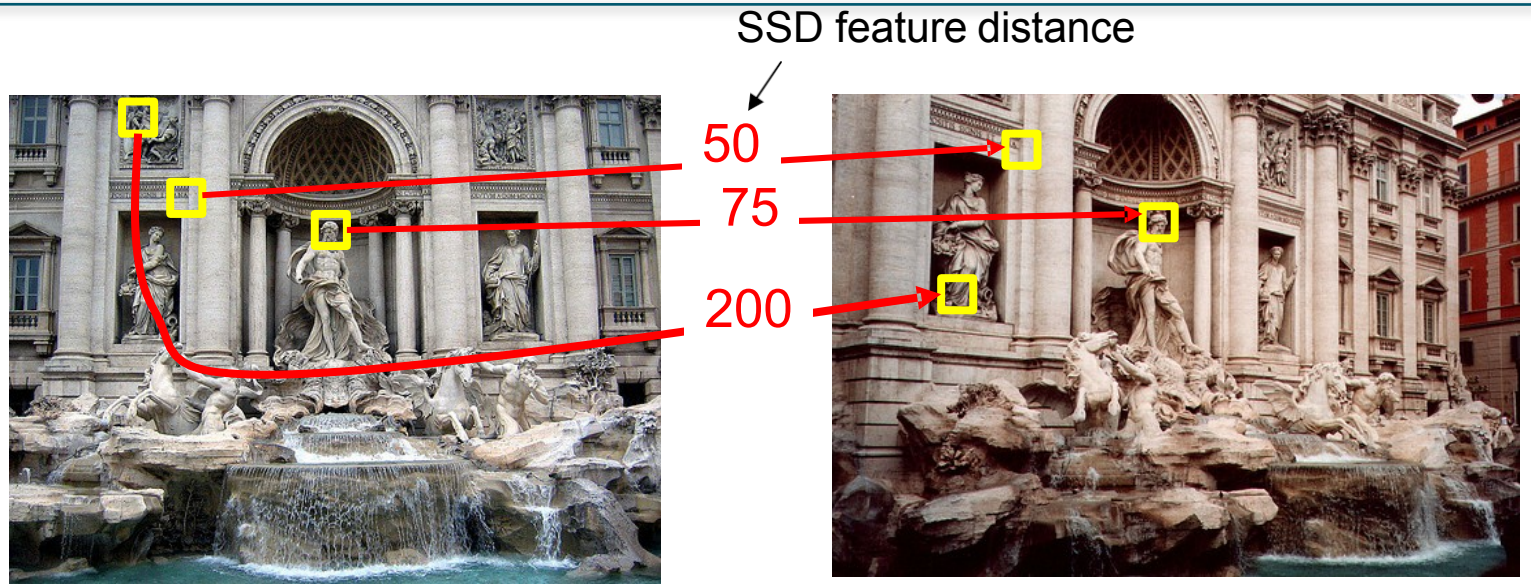


I_1



I_2

Image matching

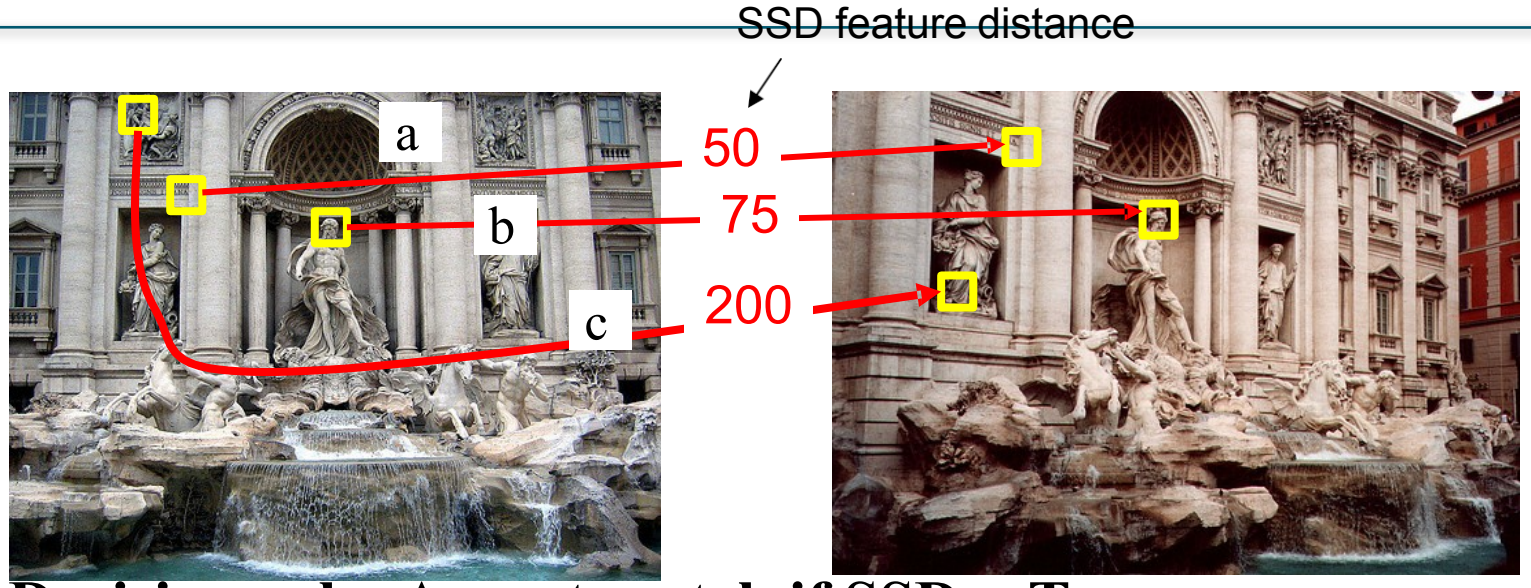


Suppose we use SSD
Small values are possible matches but how small?

Decision rule: Accept match if SSD < T
where T is a threshold

What is the effect of choosing a particular T?

Effect of threshold T



Decision rule: Accept match if SSD < T

Example: **Large T**

T = 250 ⇒ a, b, c are all accepted as matches

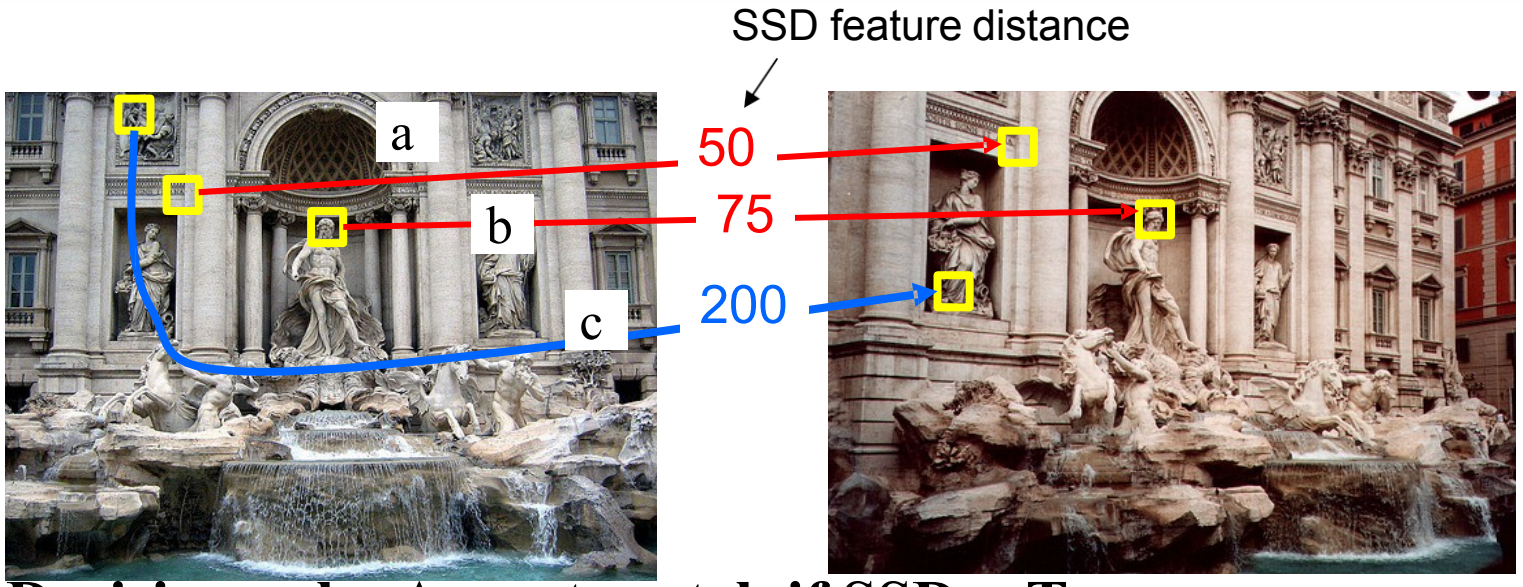
a and b are true matches (“**true positives**”)

– they are actually matches

c is a false match (“**false positive**”)

– actually not a match

Effect of threshold T



Decision rule: Accept match if SSD < T

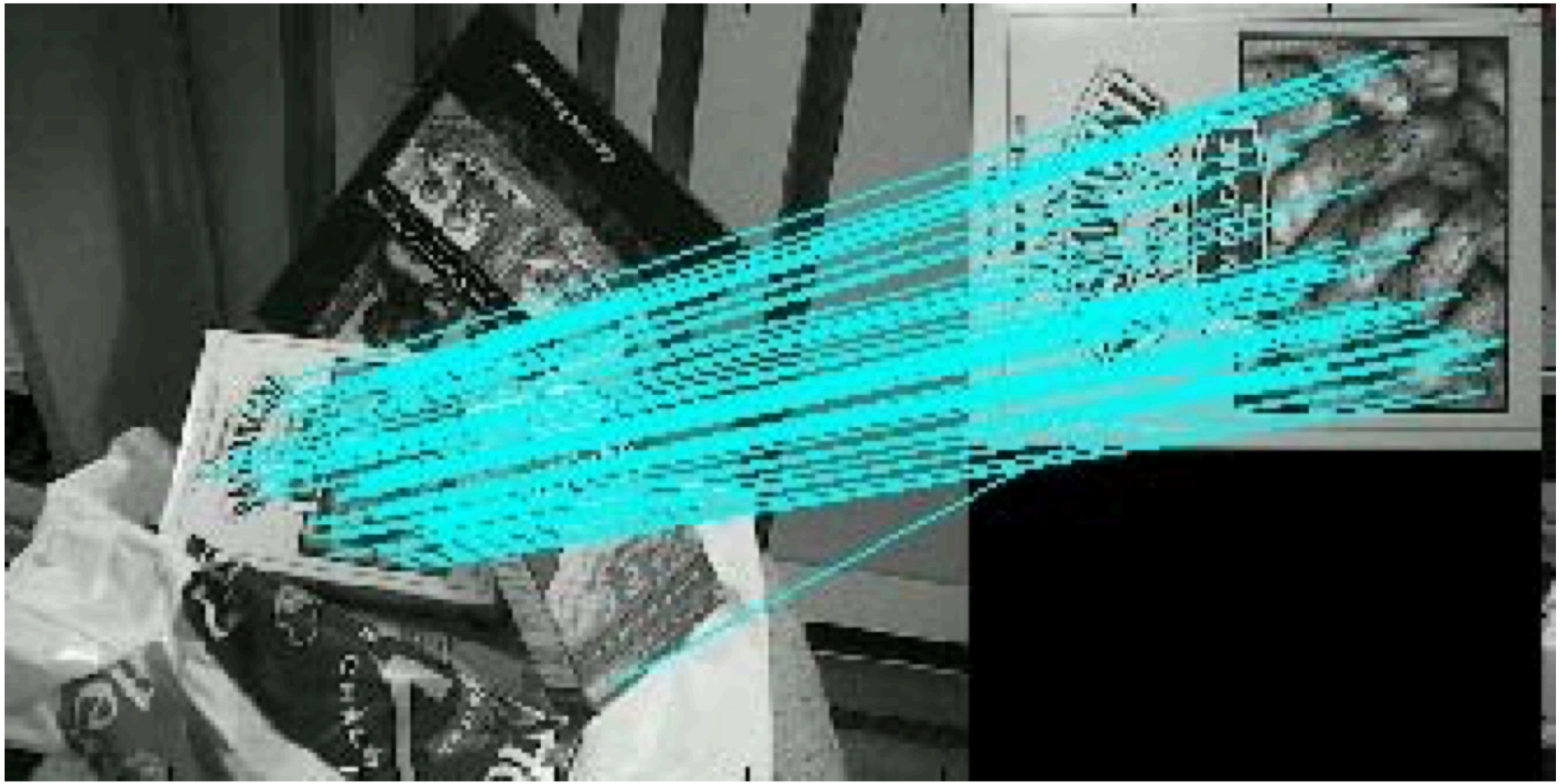
Example: **Smaller T**

T = 100 ⇒ only a and b are accepted as matches

a and b are true matches (“true positives”)

c is no longer a “false positive” (it is a “true negative”)

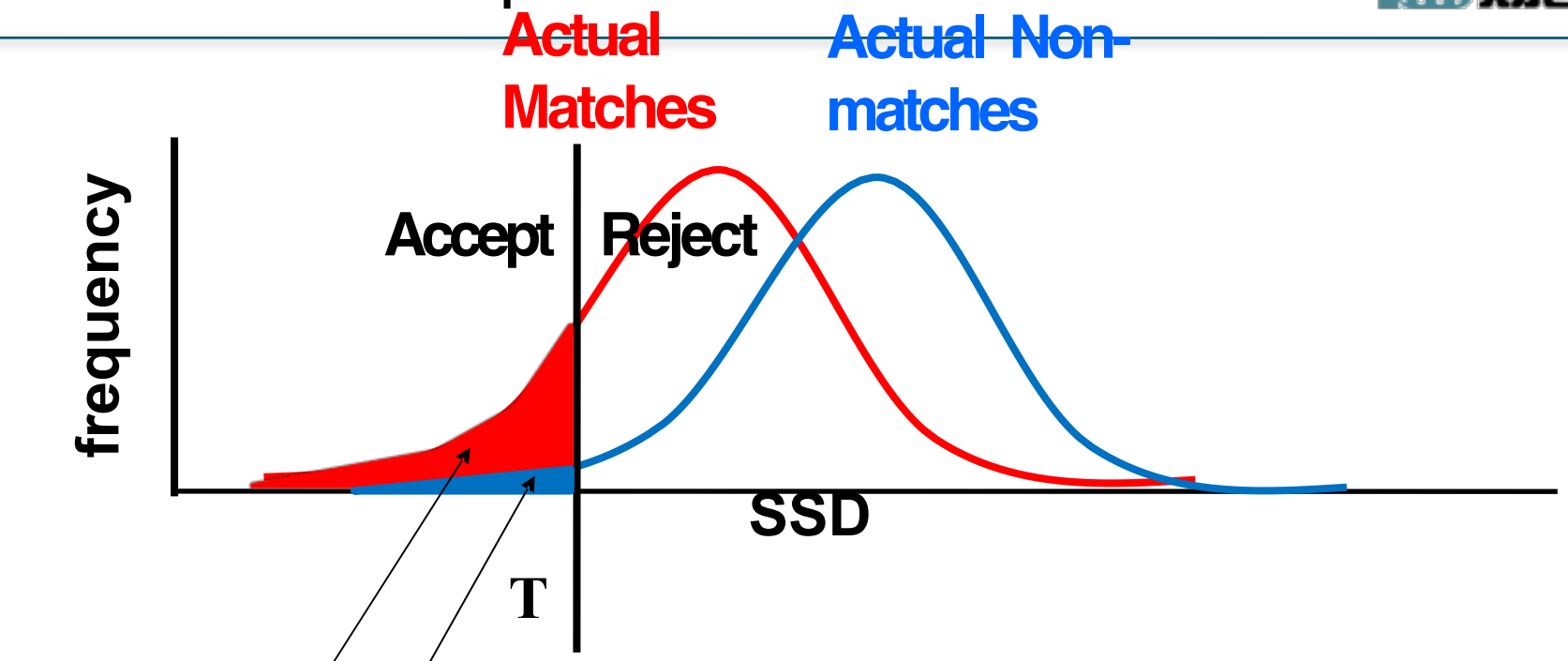
Image matching



Thanks

FAQ时间

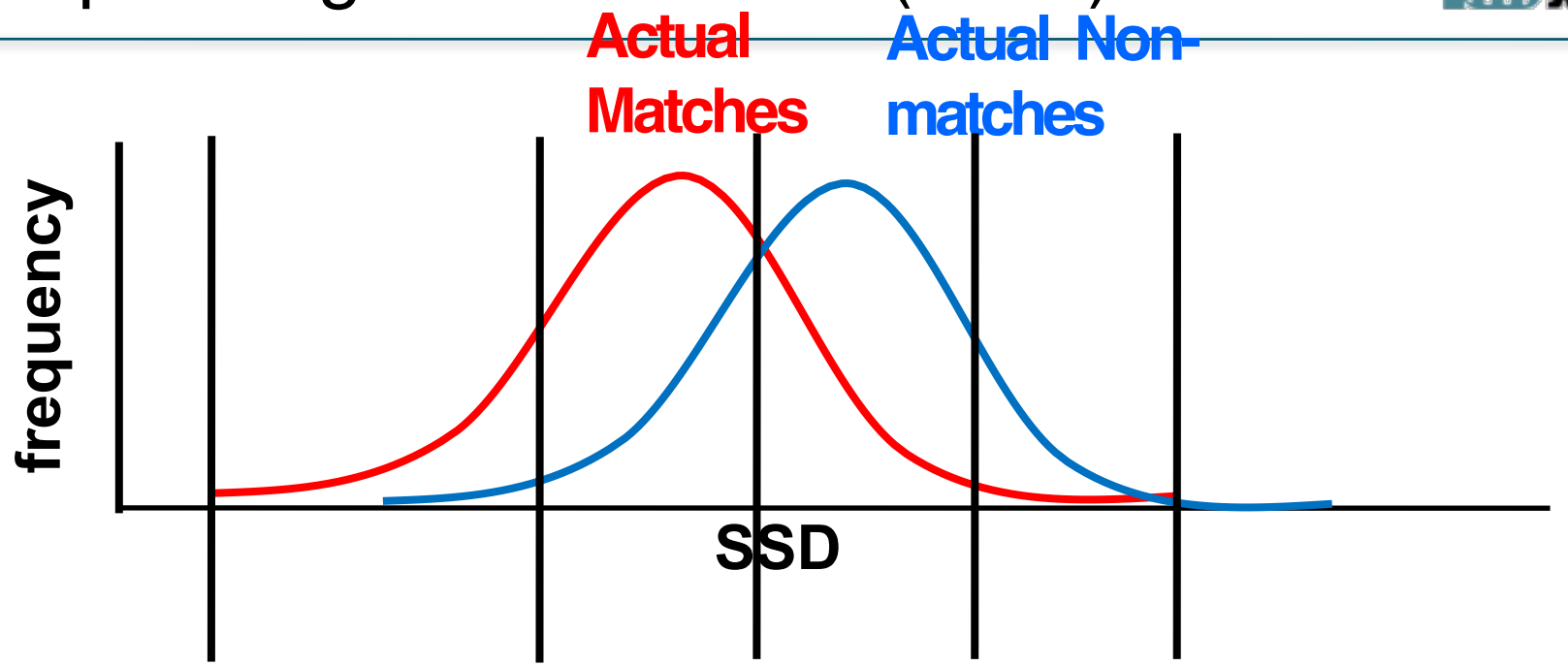
True positives and false positives

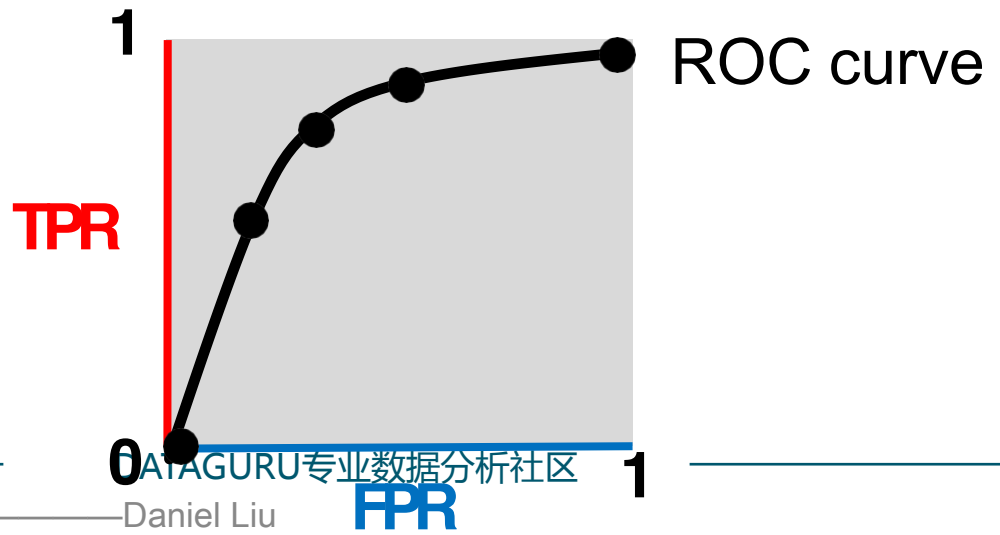


$$\text{True positive rate (TPR)} = \frac{\#\,\text{true positives}}{\#\,\text{actual matches}}$$

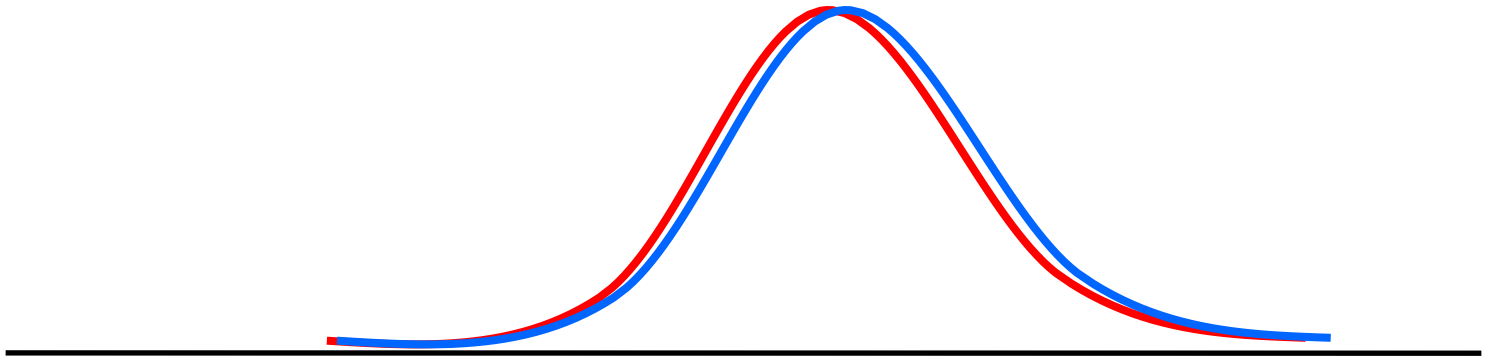
$$\text{False positive rate (FPR)} = \frac{\#\,\text{false positives}}{\#\,\text{actual nonmatches}}$$

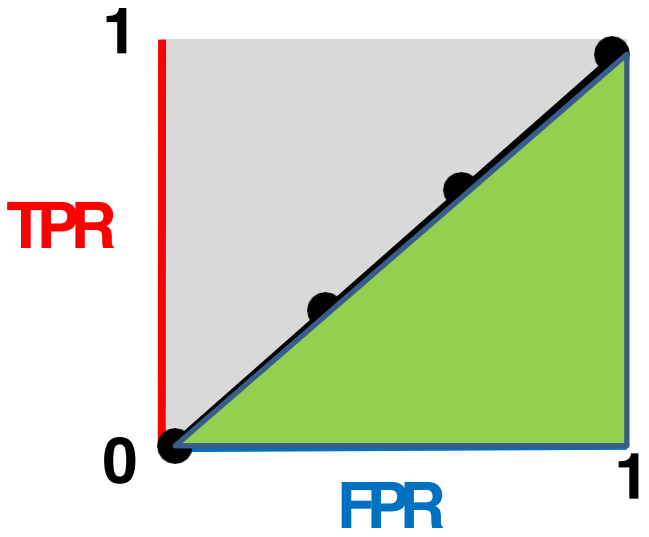
Receiver Operating Characteristic (ROC) curve





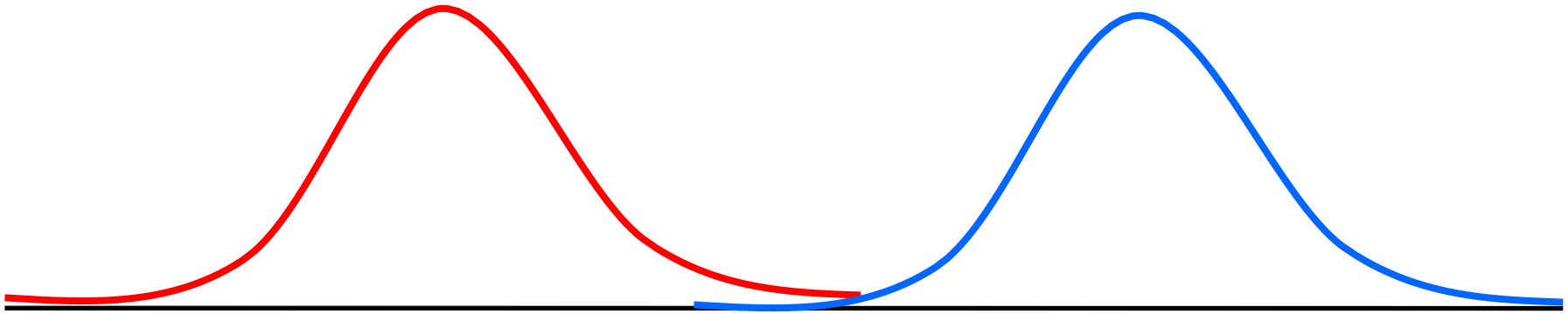
If the features selected were bad…

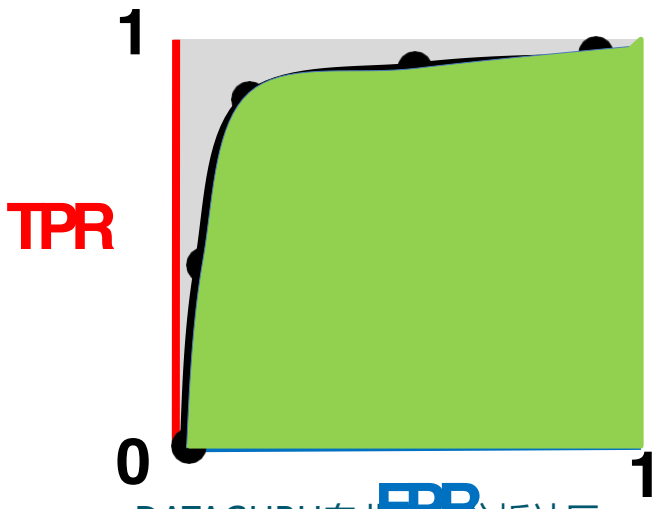




Area ≈ 0.5

If the features selected were good…



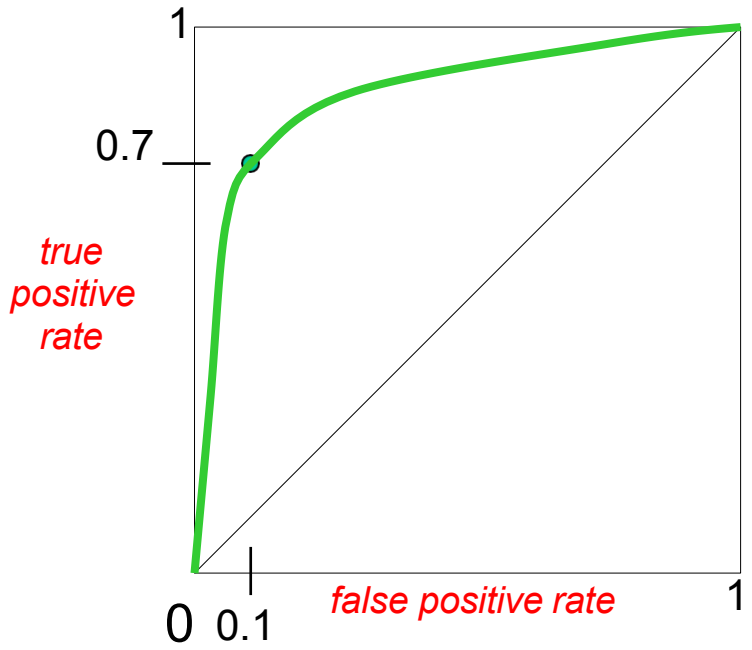


Area ≈ 1.0

Adapted from a slide by Shin Kira

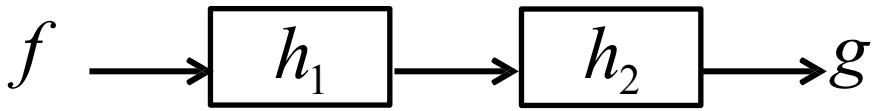
ROC Curve

- Useful for comparing different feature matching methods
- Pick method that maximizes area under the curve
- More info: http://en.wikipedia.org/wiki/Receiver_operating_characteristic



Properties of Convolution

- Commutative $a * b = b * a$
- Associative $(a * b) * c = a * (b * c)$
- Cascade system



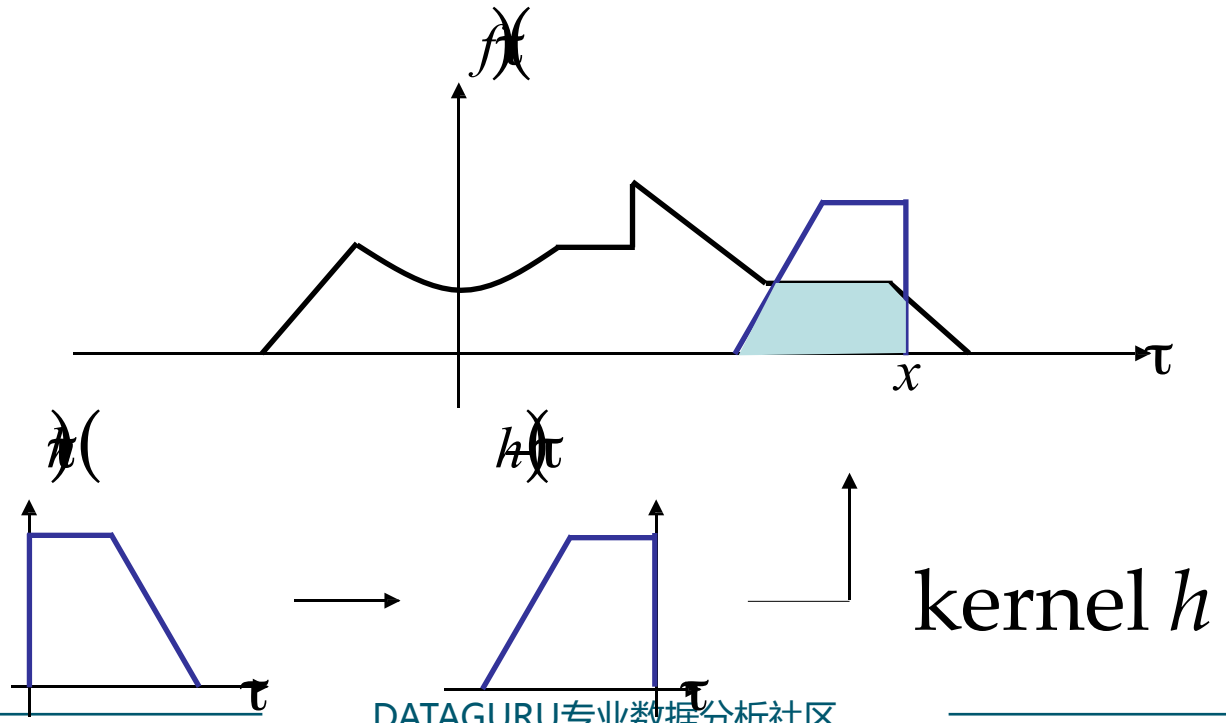
$= f \rightarrow h_1 * h_2 \rightarrow g$

$= f \rightarrow h_2 * h_1 \rightarrow g$

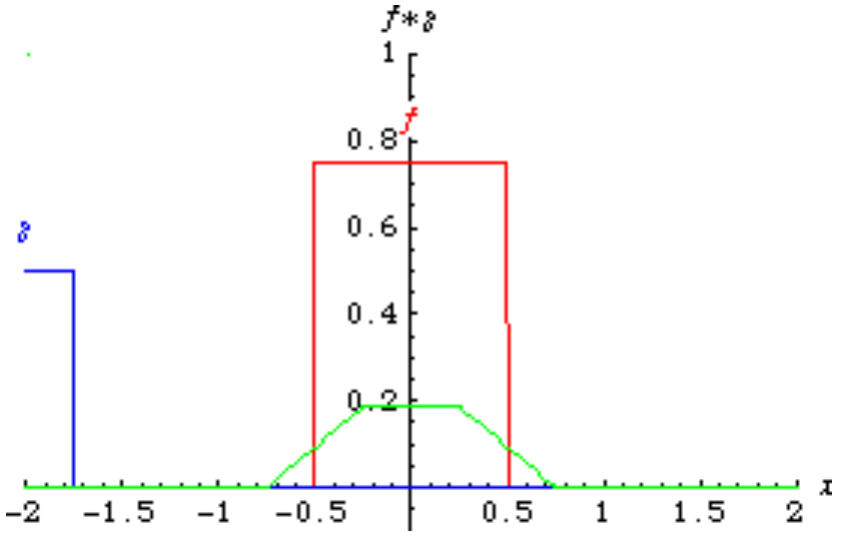
Convolution

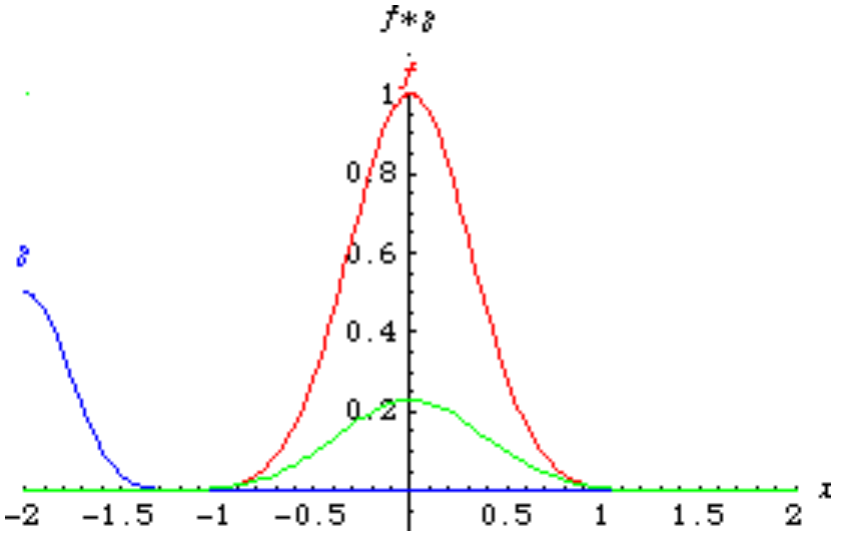
Convolution is linear and shift invariant

$$g = f * h$$



Convolution - Example





f

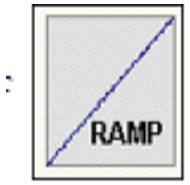
g

$f*g$

Gamma correction

Monitors have a intensity to voltage response curve which is roughly a 2.5 power function

Send v → actually display a pixel which has intensity equal to $v^{2.5}$



Graph of Input

Graph of Output L = V ^ 2.5

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

$\wp$ = 1.0; f(v) =v

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

$\wp$ = 2.5; f(v) = $v^{1/2.5} = v^{0.4}$

Blurred Image

Gaussian Smoothing

by Charles Allen Gillbert

by Harmon & Julesz

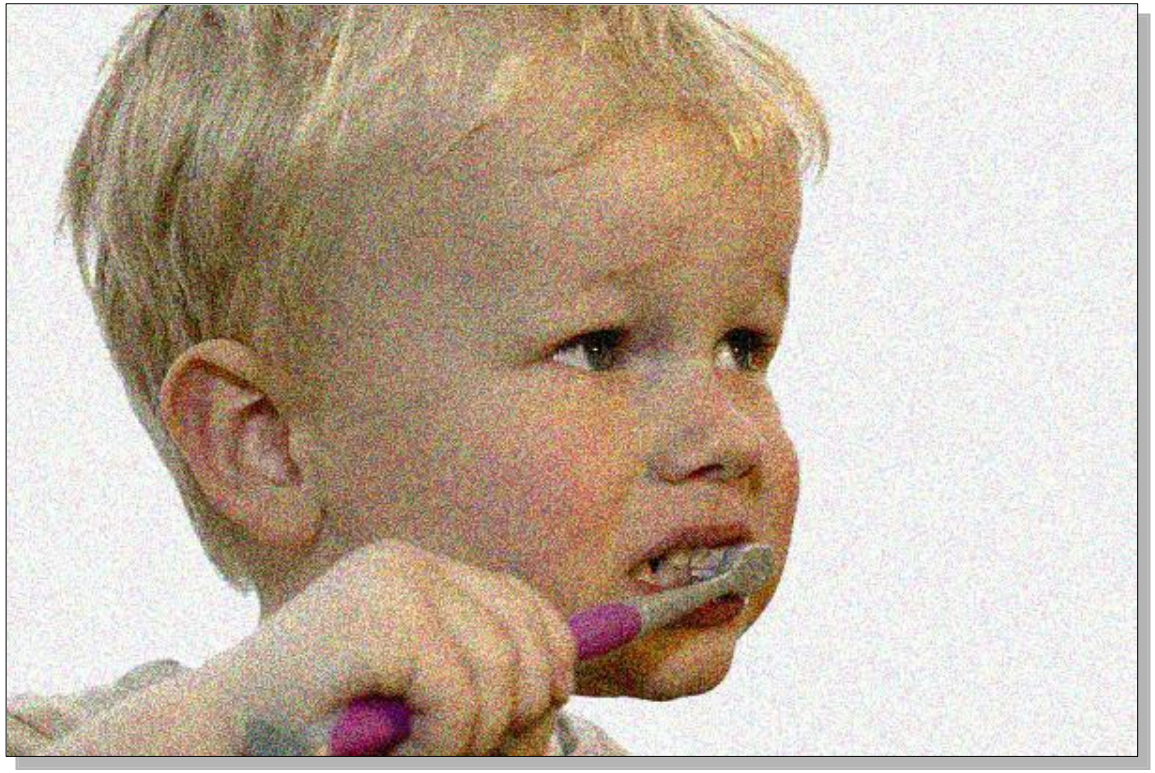
http://www.michaelbach.de/ot/cog_blureffects

Gaussian Smoothing

http://www.michaelbach.de/ot/cog_blureffects

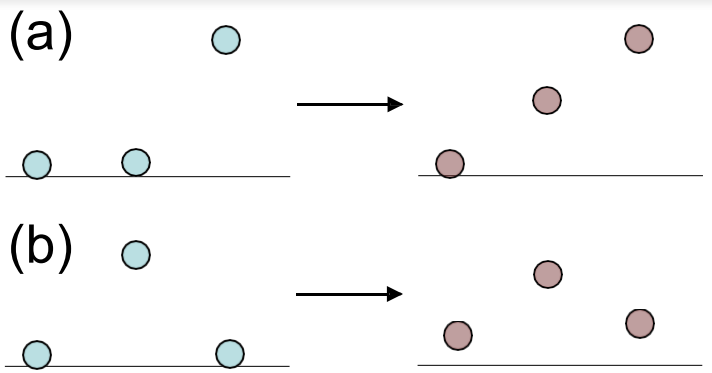
Sharpened Image

Noise

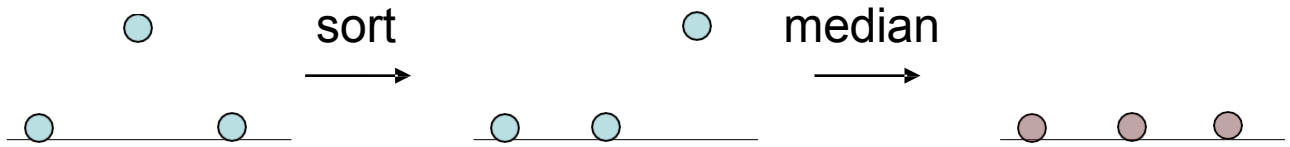


Median Filter

- Smoothing is averaging
  - (a) Blurs edges
  - (b) Sensitive to outliers



- Median filtering
  - Sort N^2-1 values around the pixel
  - Select middle value (median)



  - Non-linear (Cannot be implemented with convolution)

Median Filter

Can this be described as a convolution?

Original Image

Example: Noise Reduction

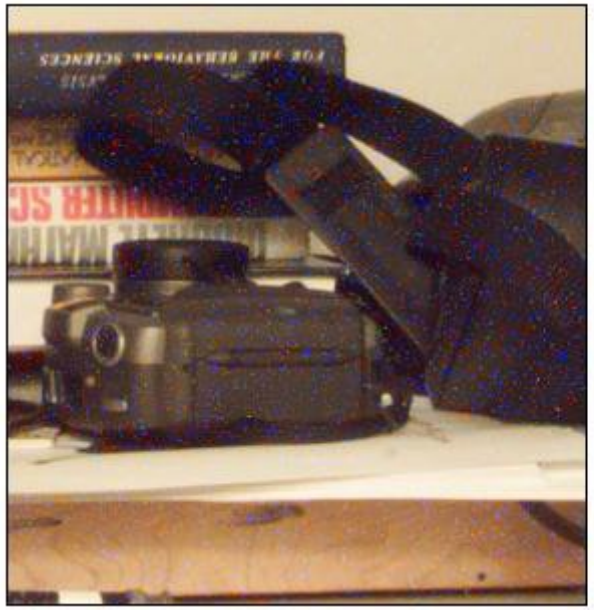


Image with noise

Median filter (5x5)

Example: Noise Reduction

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

Original image

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

Image with noise

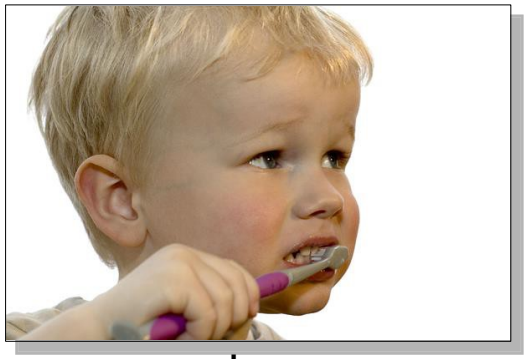
Median filter (5x5)

Original Image

Original Image

Warped Image

Warped Image



orig

+

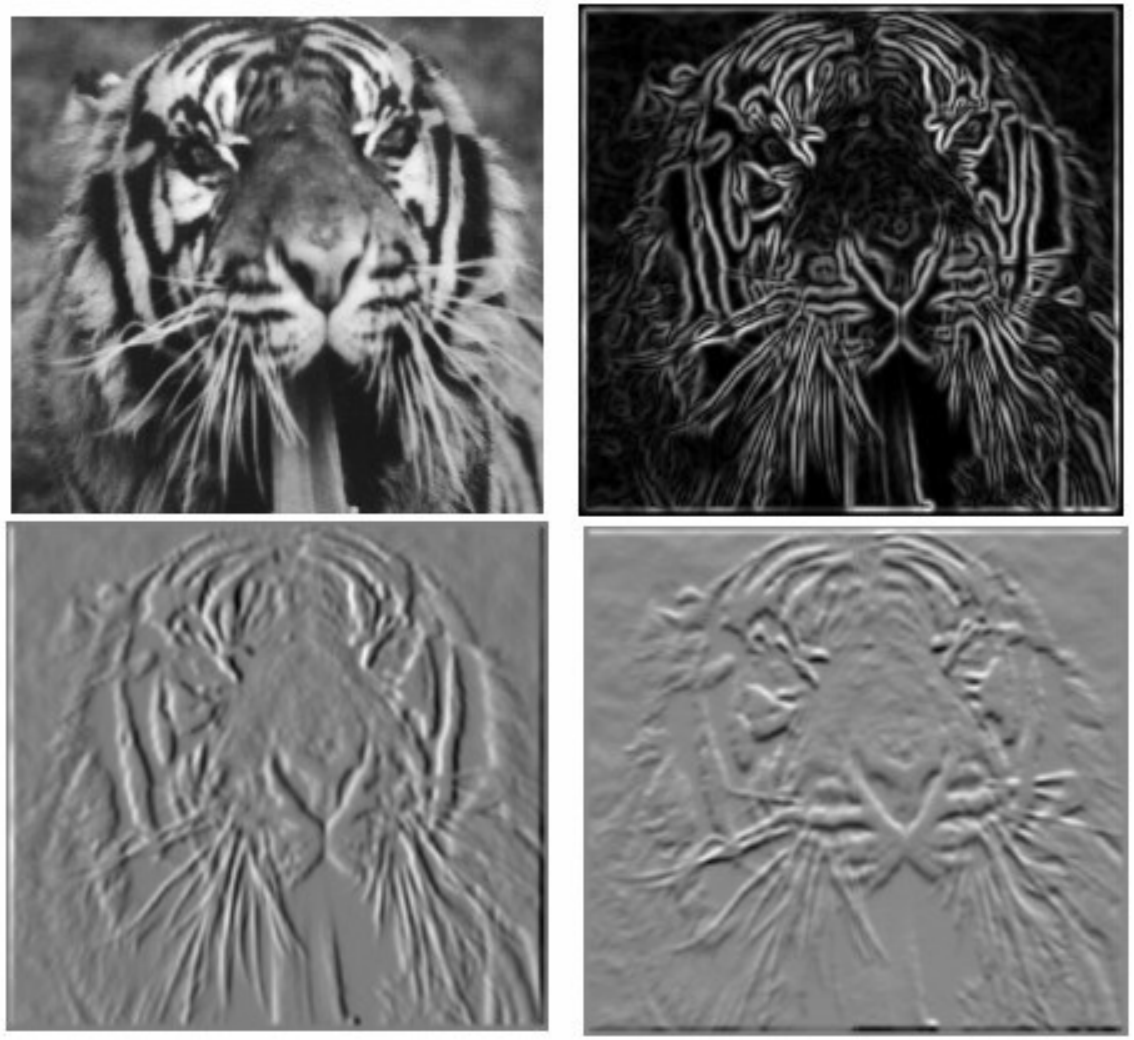
vector field

=

warped

how?

Finite differences: example



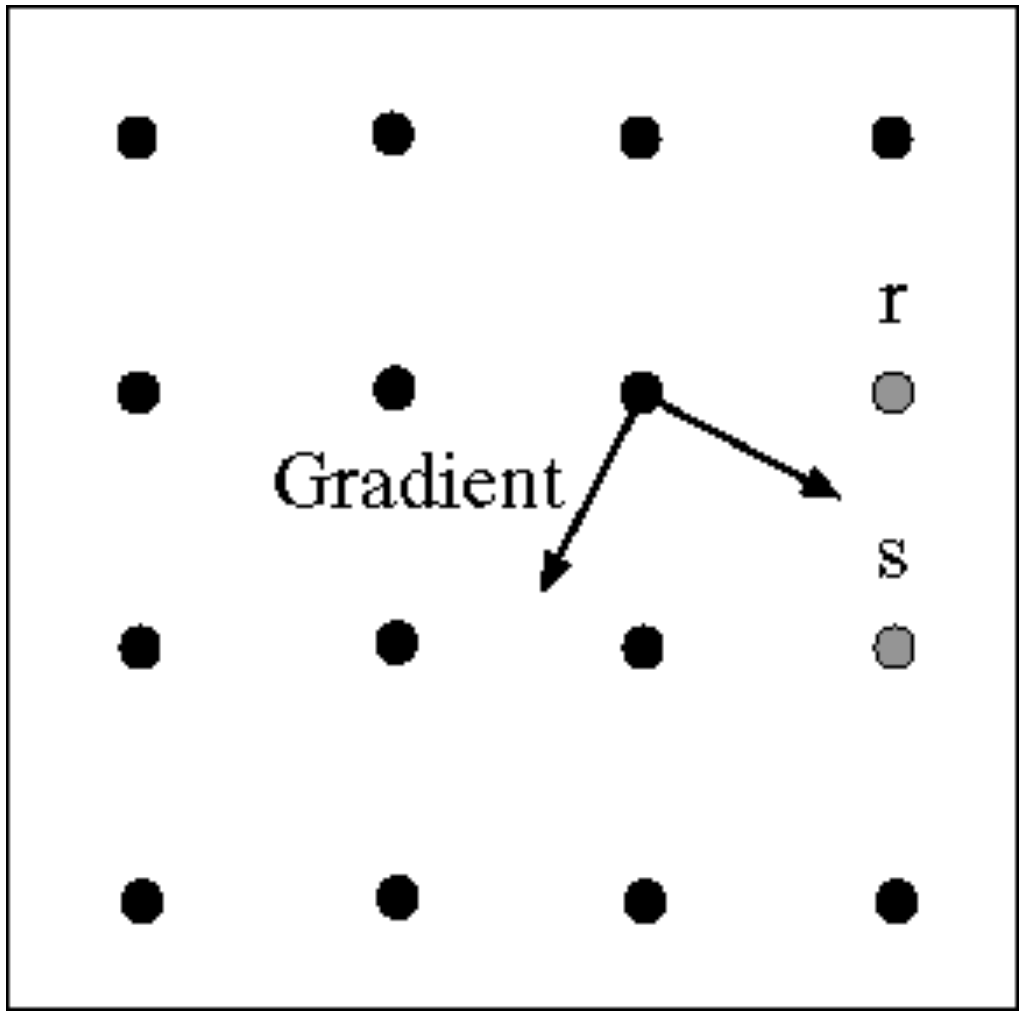
Which one is the gradient in the x-direction (resp. y-direction)?

Canny edge detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Linking of edge points

Source: D. Lowe, L. Fei-Fei

Edge linking



Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).

Source: D. Forsyth

X-Edge Detection

Y-Edge Detection