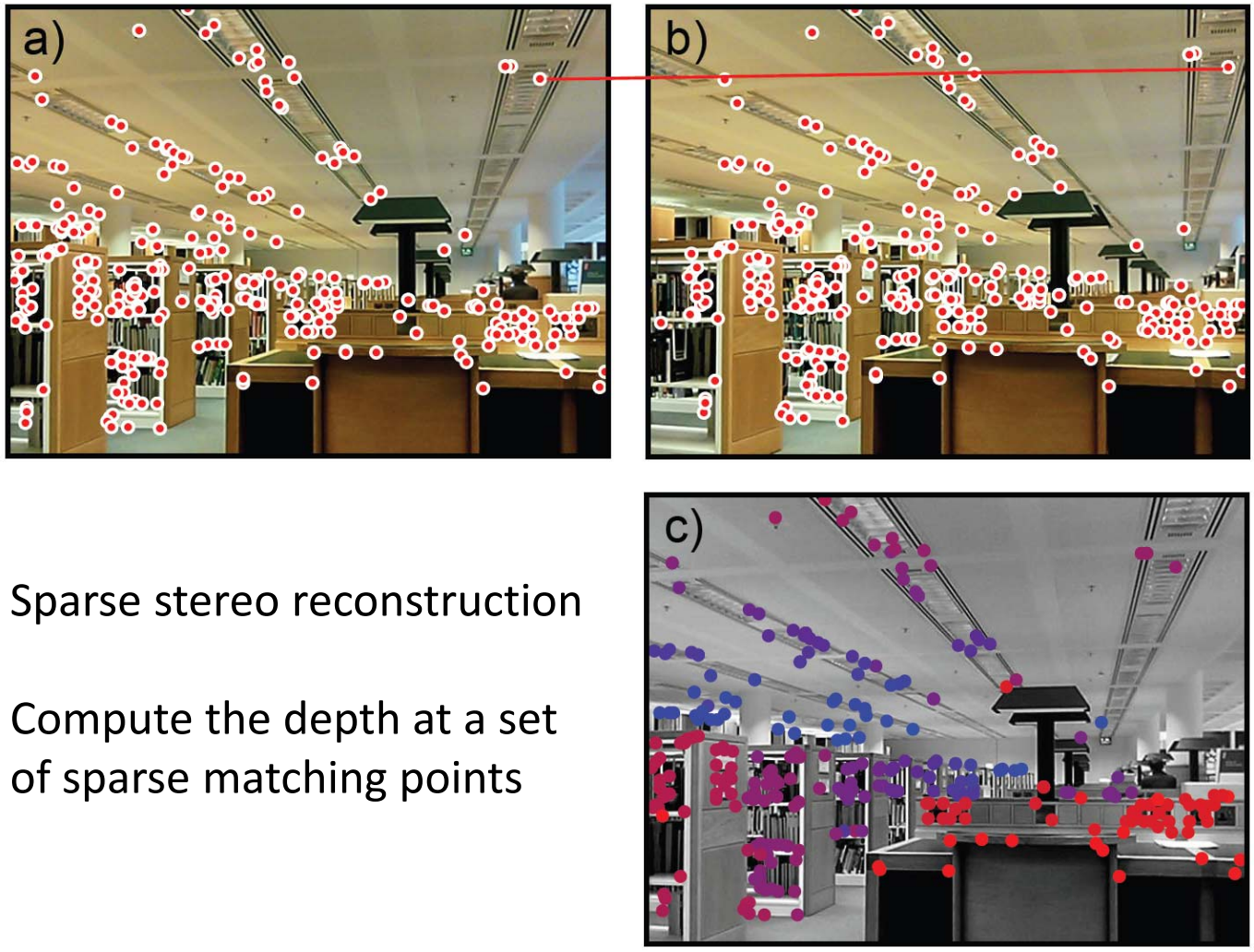
Computer vision: models, learning and inference

Chapter 13

Image preprocessing and feature extraction

Motivation (from Chapter 14)



Sparse stereo reconstruction

Compute the depth at a set of sparse matching points

Preprocessing

- The goal of pre-processing is
  - to try to reduce unwanted variation in image due to lighting, scale, deformation etc.
  - to reduce data to a manageable size
- Give the subsequent model a chance
- Preprocessing definition: deterministic transformation of pixels **p** to create data vector **x**
- Usually heuristics based on experience

Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

Normalization

- Fix first and second moments to standard values
- Remove contrast and constant additive luminance variations

Before

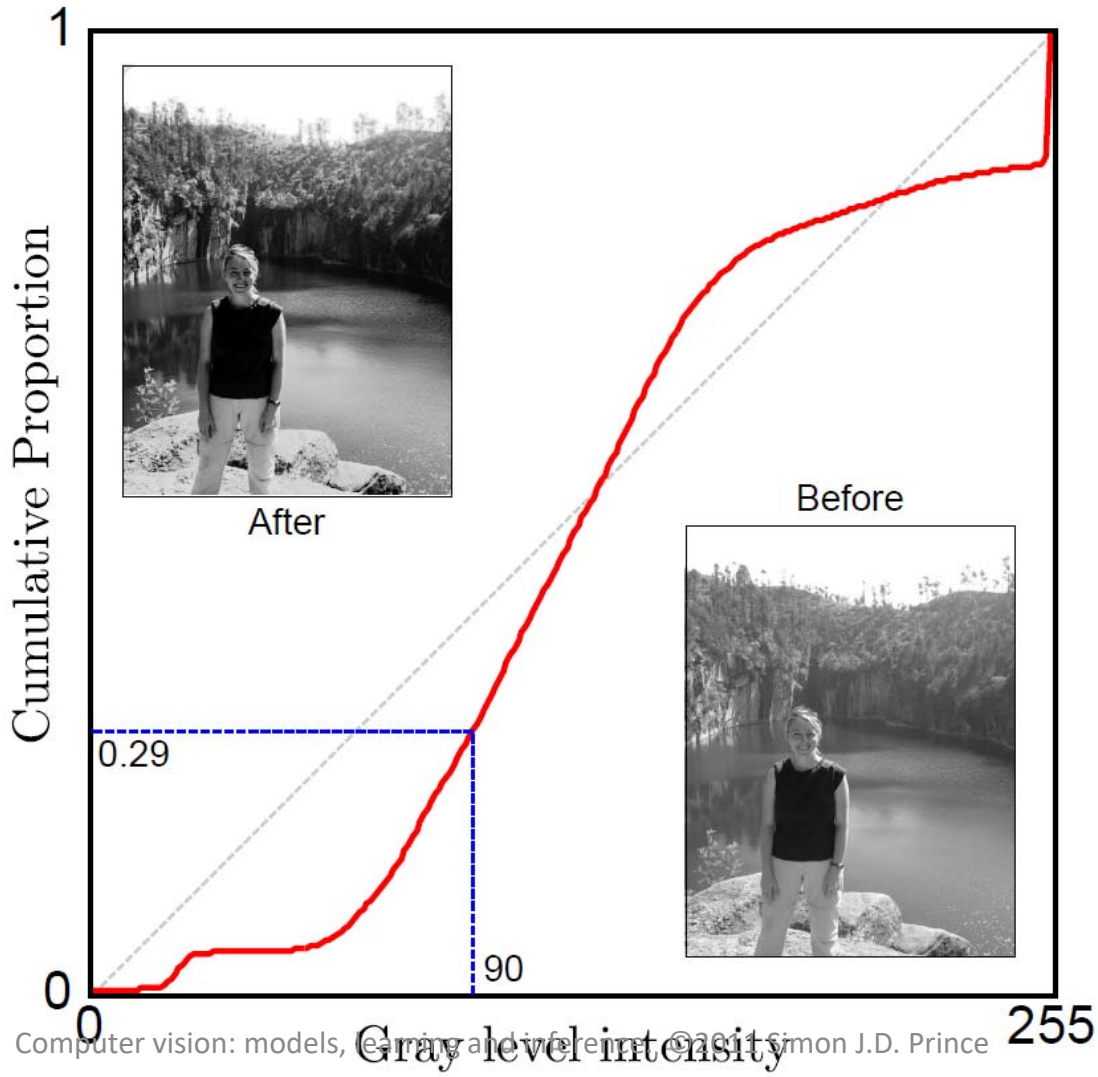
After

Histogram Equalization

Make all of the moments the same by forcing the histogram of intensities to be the same

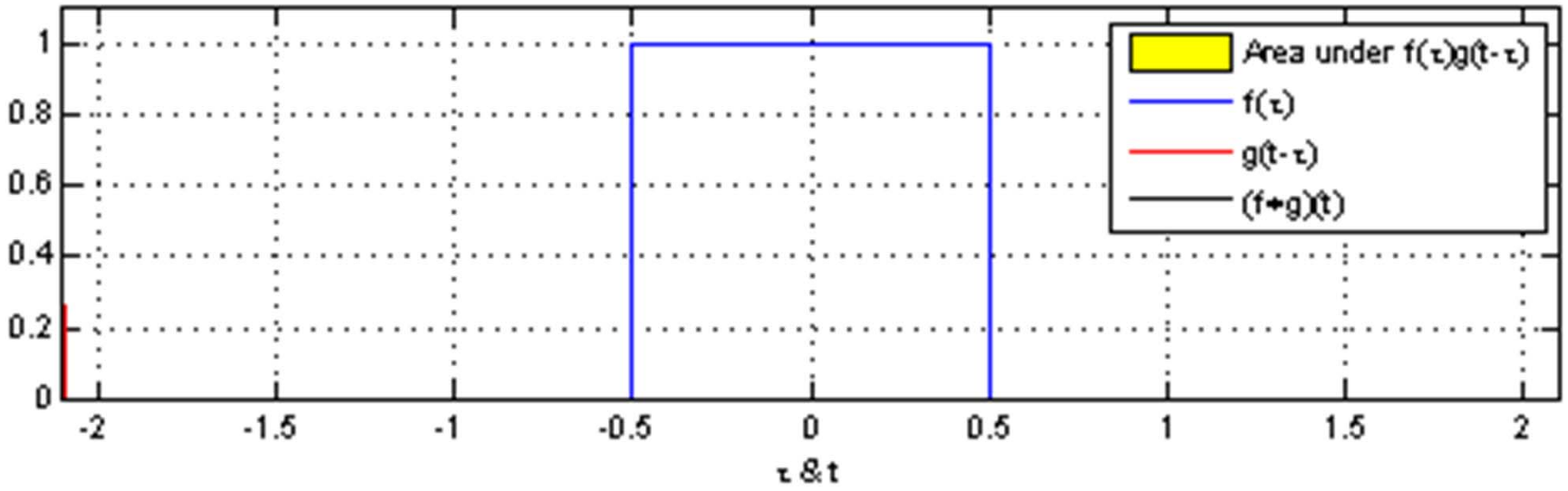
Before/ normalized/ Histogram Equalized

Histogram Equalization



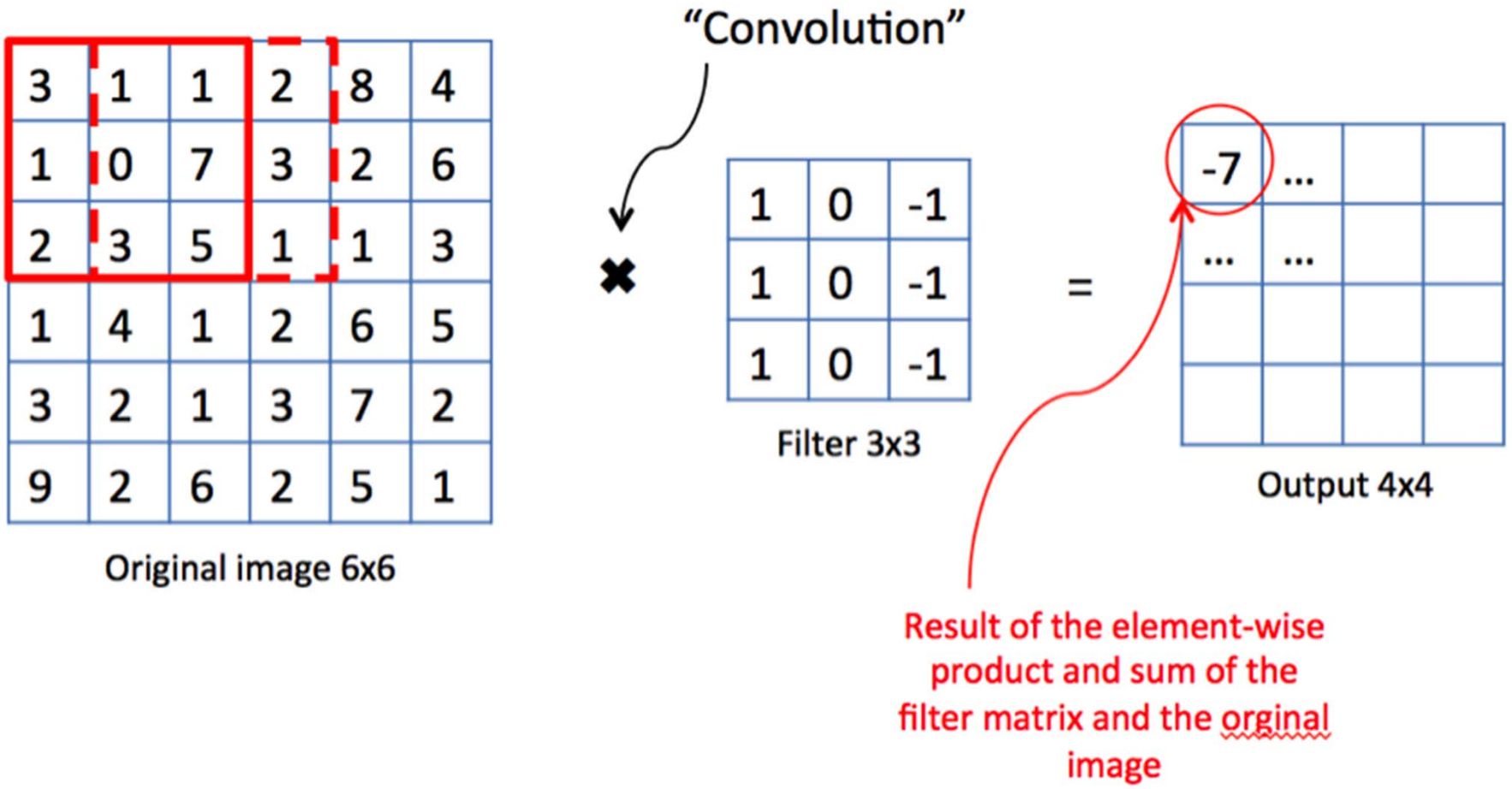
Convolution

$$x_{ij} = \sum_{m=-M}^{M} \sum_{n=-N}^{N} p_{i-m,j-n} f_{m,n}$$

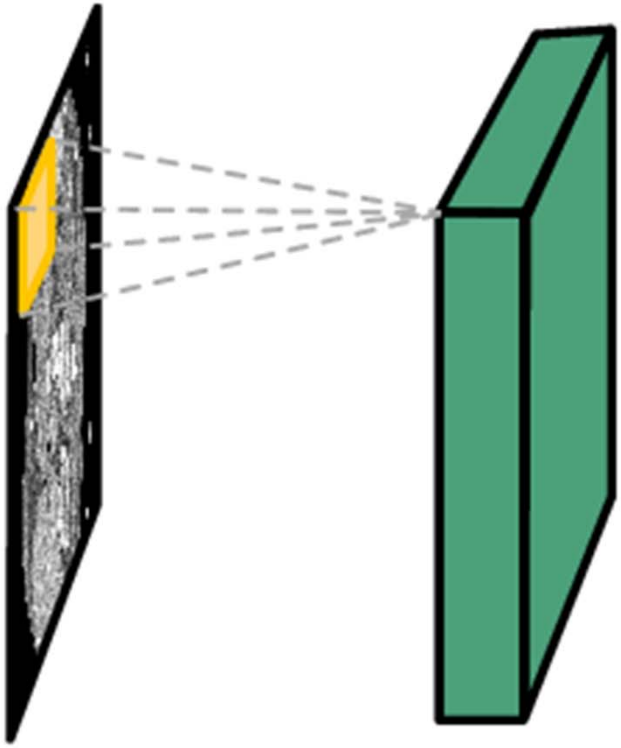


Wikipedia ~ convolution

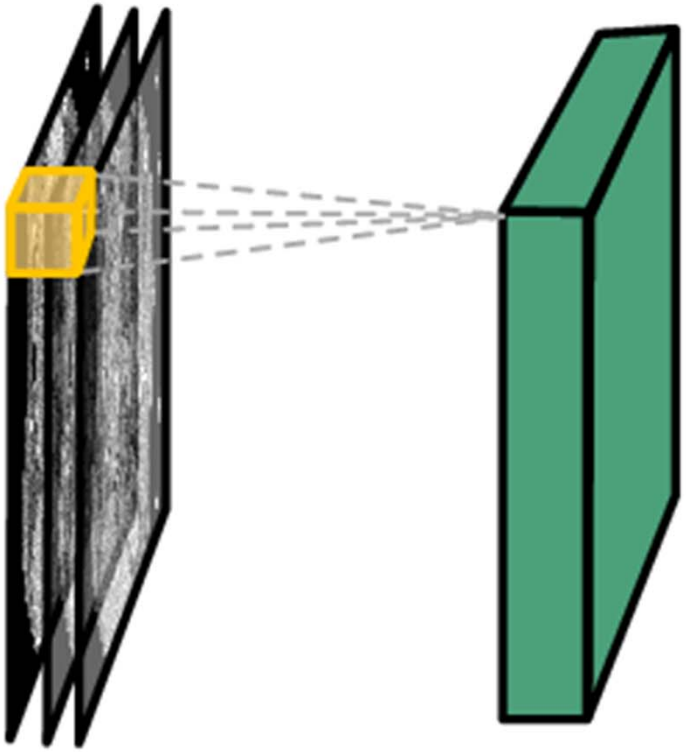
Convolution



Convolution



Conv2D



Conv3D

Convolution

Takes pixel image **P** and applies a filter **F**

$$x_{ij} = \sum_{m=-M}^{M} \sum_{n=-N}^{N} p_{i-m,j-n} f_{m,n}$$

Computes weighted sum of pixel values, where weights given by filter.

Easiest to see with a concrete example

Blurring (convolve with Gaussian)

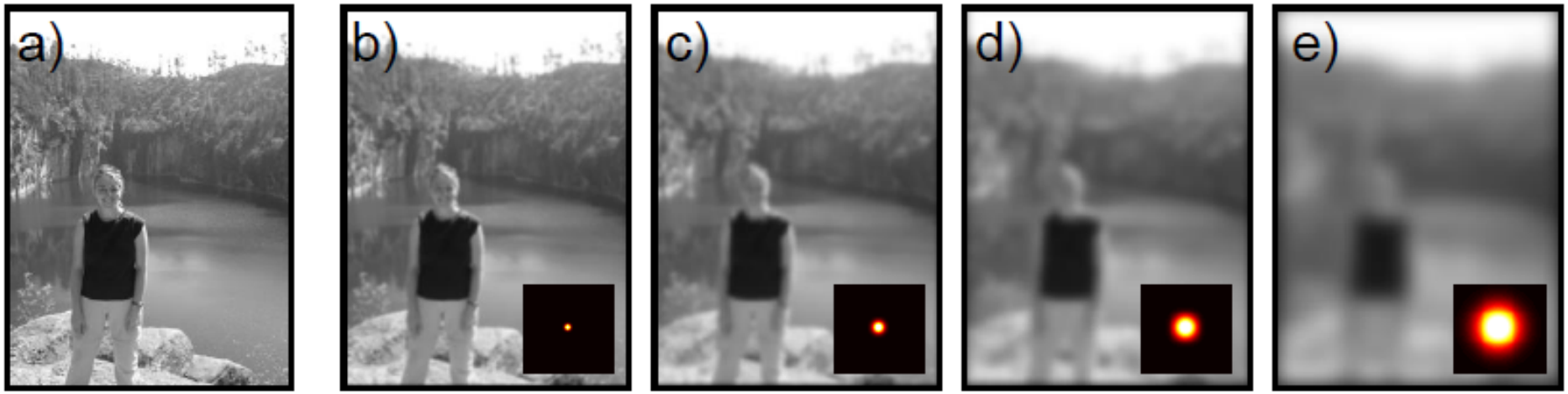
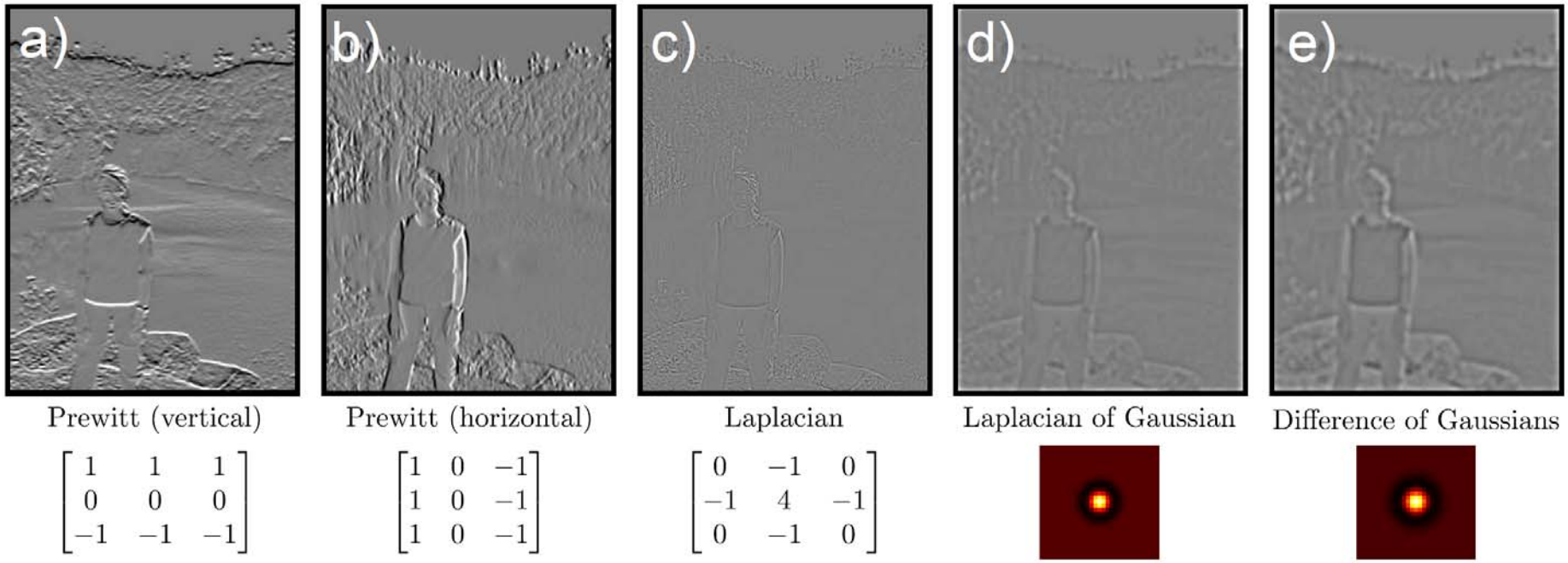


Figure B.3 Image blurring. a) Original image. b) Result of convolving with a Gaussian filter (filter shown in bottom right of image). The image is slightly blurred. c-e) Convolving with a filter of increasing standard deviation causes the resulting image to be increasingly blurred.

Gradient Filters



- Rule of thumb: big response when image matches filter

[OpenCV 3.x page 39] Blurring

```
import cv2
import numpy as np

img = cv2.imread('images/input.jpg')
rows, cols = img.shape[:2]
kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_3x3 = np.ones((3,3), np.float32) / 9.0 # Divide by 9 to
normalize
the kernel
kernel_5x5 = np.ones((5,5), np.float32) / 25.0 # Divide by 25 to
normalize
the kernel
cv2.imshow('Original', img)
# value -1 is to maintain source image depth
output = cv2.filter2D(img, -1, kernel_identity)
cv2.imshow('Identity filter', output)
output = cv2.filter2D(img, -1, kernel_3x3)
cv2.imshow('3x3 filter', output)
output = cv2.filter2D(img, -1, kernel_5x5)
cv2.imshow('5x5 filter', output)
cv2.waitKey(0)
```

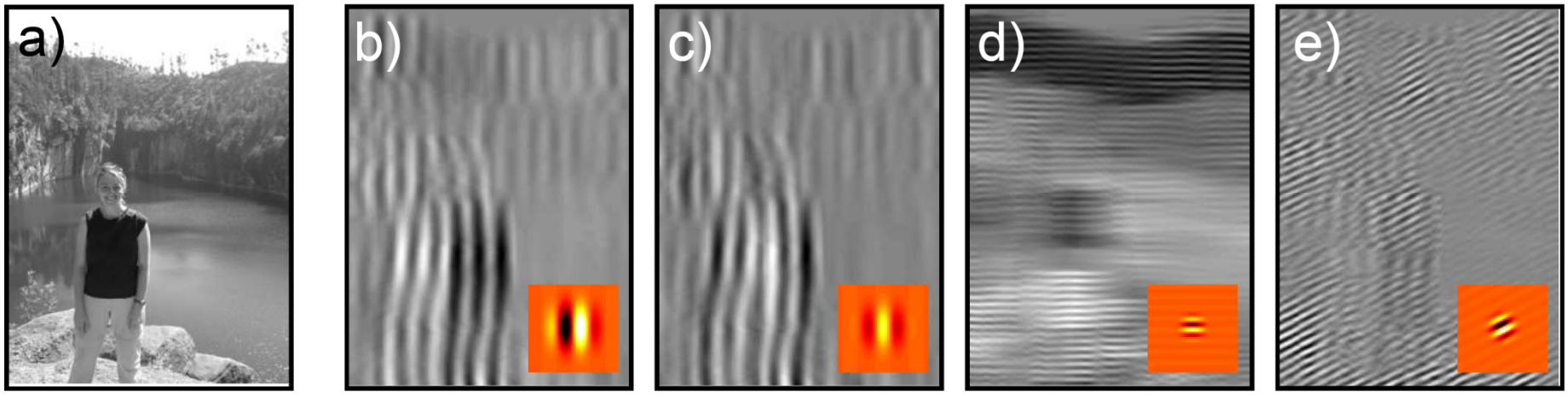
[OpenCV 3.x page 41] Motion blur

```
import cv2
import numpy as np
img = cv2.imread('images/input.jpg')
cv2.imshow('Original', img)
size = 15
# generating the kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur / size
# applying the kernel to the input image
output = cv2.filter2D(img, -1, kernel_motion_blur)
cv2.imshow('Motion Blur', output)
cv2.waitKey(0)
```

[OpenCV 3.x page 43] Sharpening

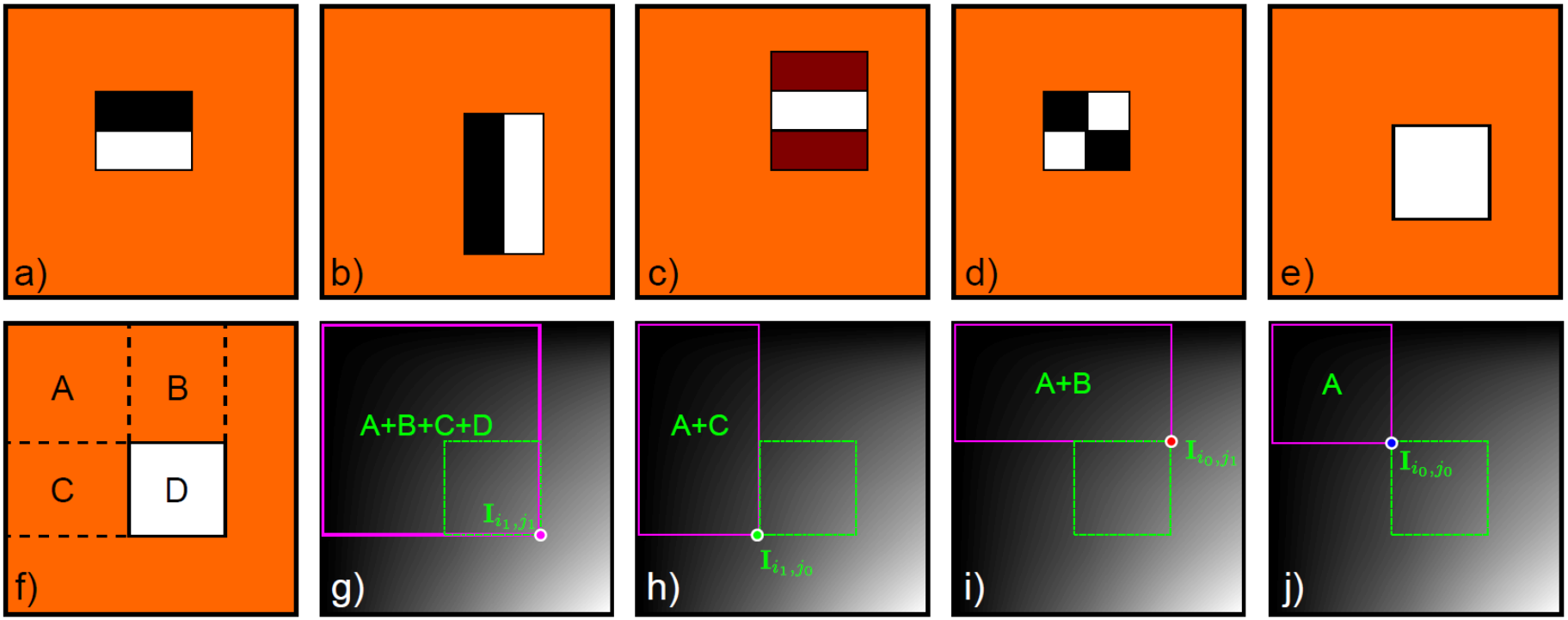
```
import cv2
import numpy as np
img = cv2.imread('images/input.jpg')
cv2.imshow('Original', img)
# generating the kernels
kernel_sharpen_1 = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
kernel_sharpen_2 = np.array([[1,1,1], [1,-7,1], [1,1,1]])
kernel_sharpen_3 = np.array([[-1,-1,-1,-1,-1],
[-1,2,2,2,-1],
[-1,2,8,2,-1],
[-1,2,2,2,-1],
[-1,-1,-1,-1,-1]]) / 8.0
# applying different kernels to the input image
output_1 = cv2.filter2D(img, -1, kernel_sharpen_1)
output_2 = cv2.filter2D(img, -1, kernel_sharpen_2)
output_3 = cv2.filter2D(img, -1, kernel_sharpen_3)
cv2.imshow('Sharpening', output_1)
cv2.imshow('Excessive Sharpening', output_2)
cv2.imshow('Edge Enhancement', output_3)
cv2.waitKey(0)
```

Gabor Filters



$$f_{mn} = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{m^2+n^2}{2\sigma^2}\right] \sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right]$$

Haar Filters

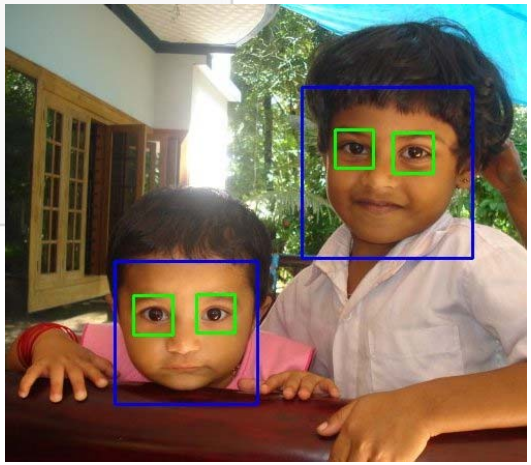


https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection

Face detection using Haar Cascades

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

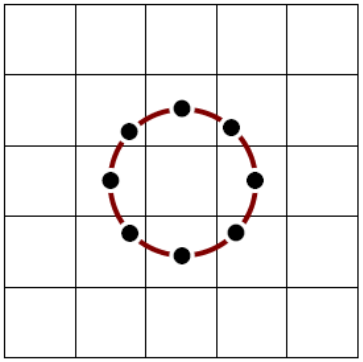


Local binary patterns

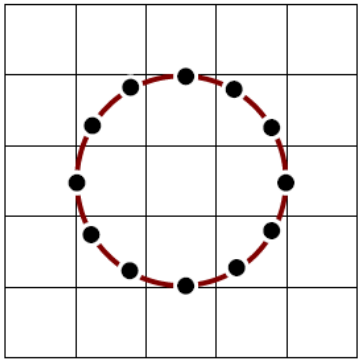
5	4	3
4	3	1
2	0	3

0: 1	1: 1	2: 1
7: 1		3: 0
6: 0	5: 0	4: 1

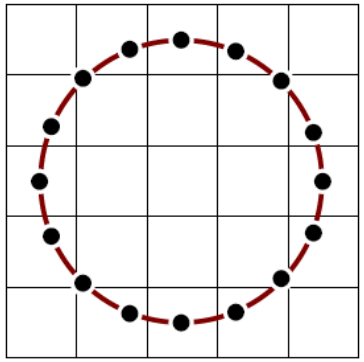
$$\text{LBP} = 10010111 = 151$$



$(P = 8, R = 1.0)$



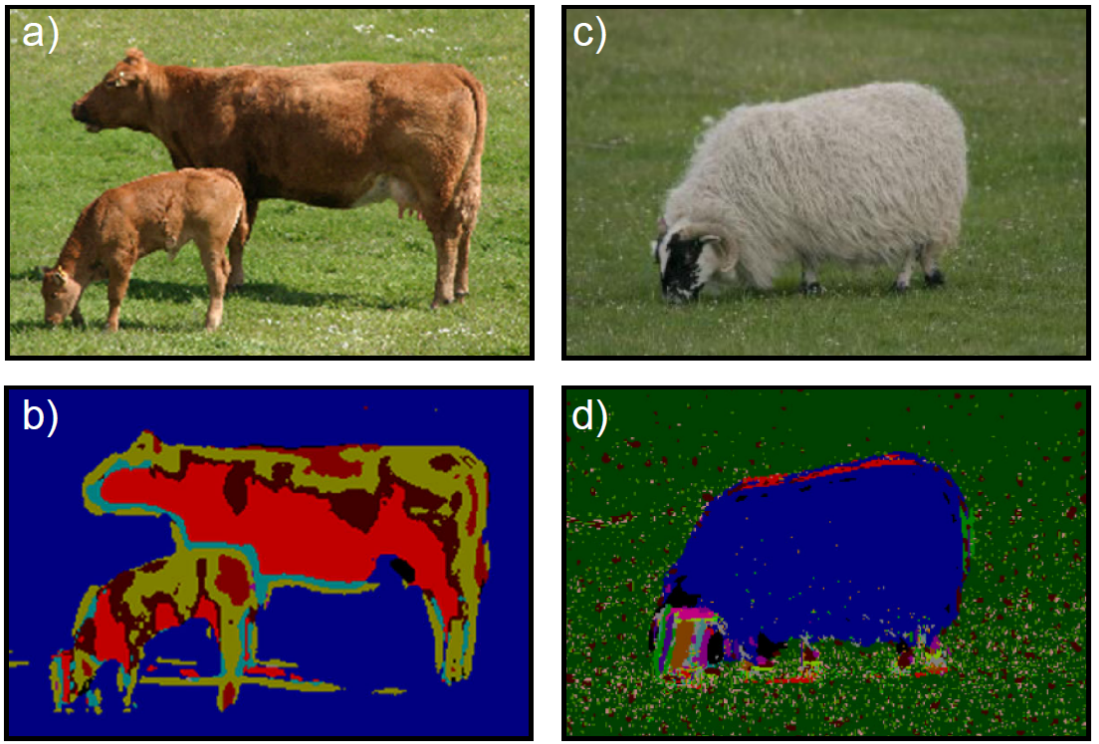
$(P = 12, R = 1.5)$



$(P = 16, R = 2.0)$

Textons

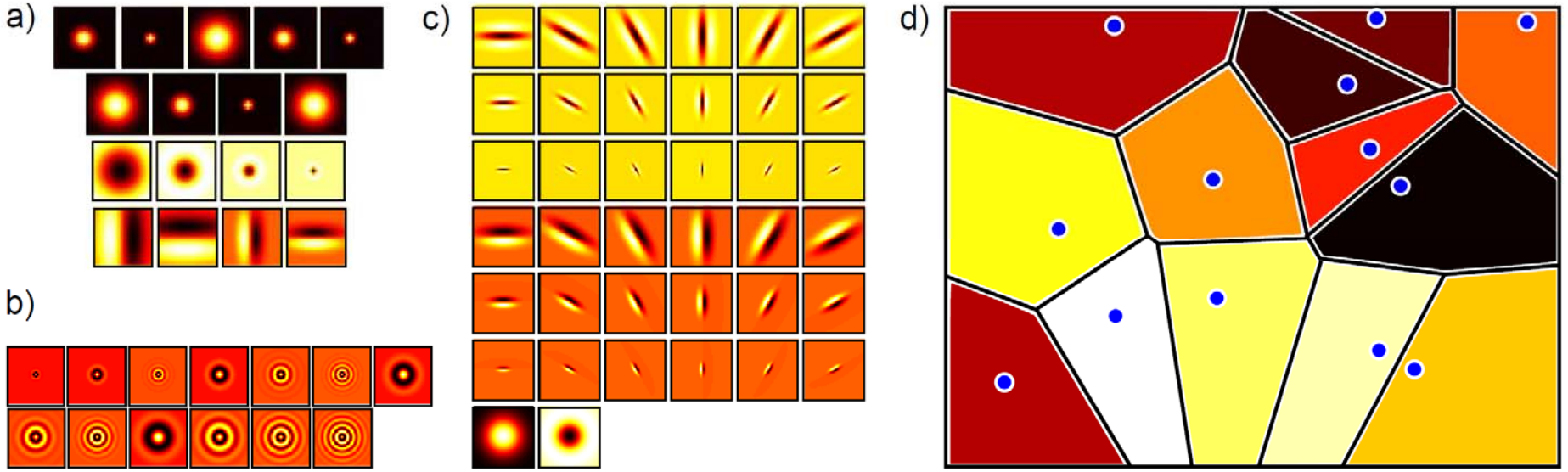
- An attempt to characterize texture
- Replace each pixel with integer representing the texture 'type'



Computing Textons

Take a bank of filters and apply to lots of images

Cluster in filter space



For new pixel, filter surrounding region with same bank, and assign to nearest cluster

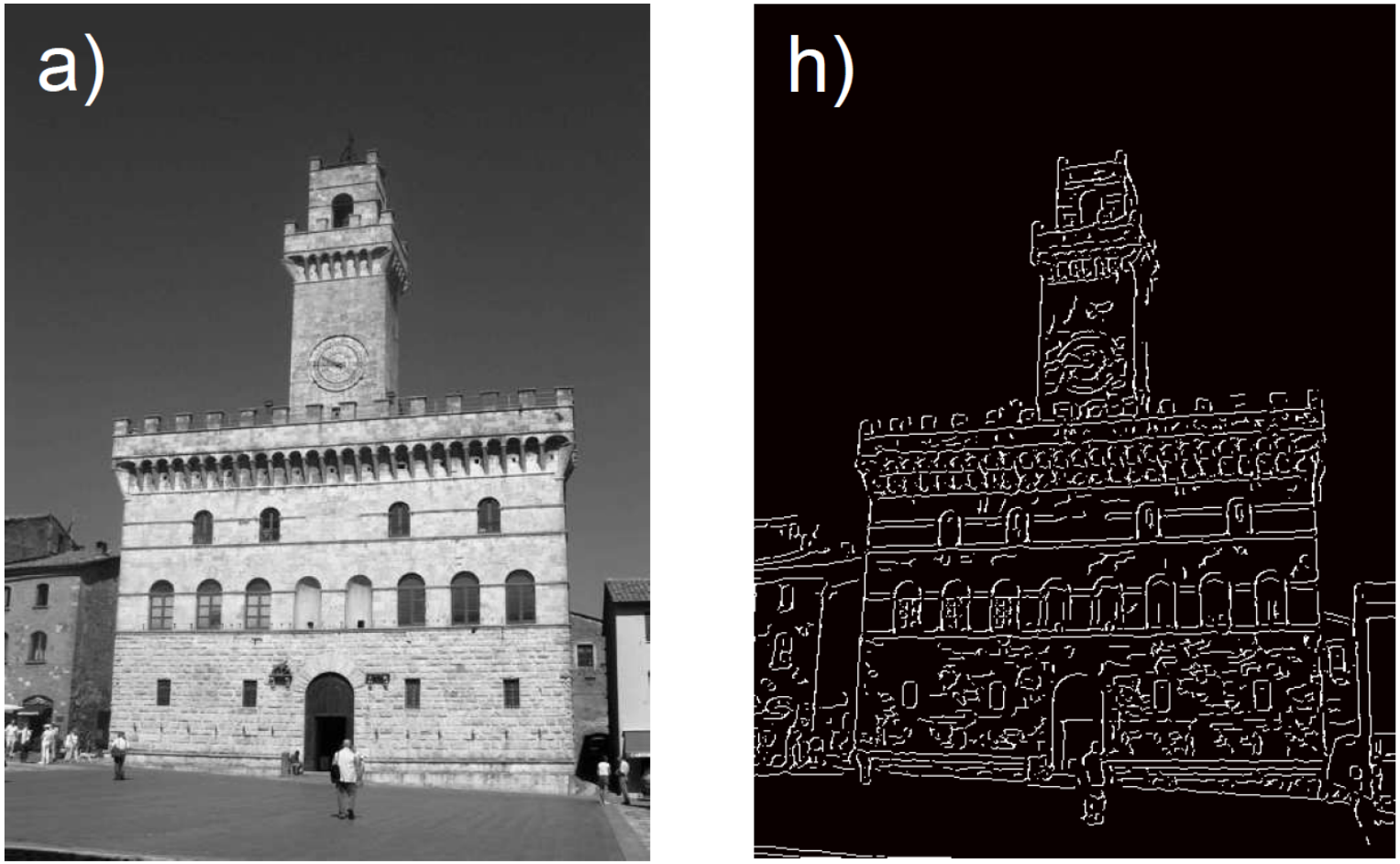
Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

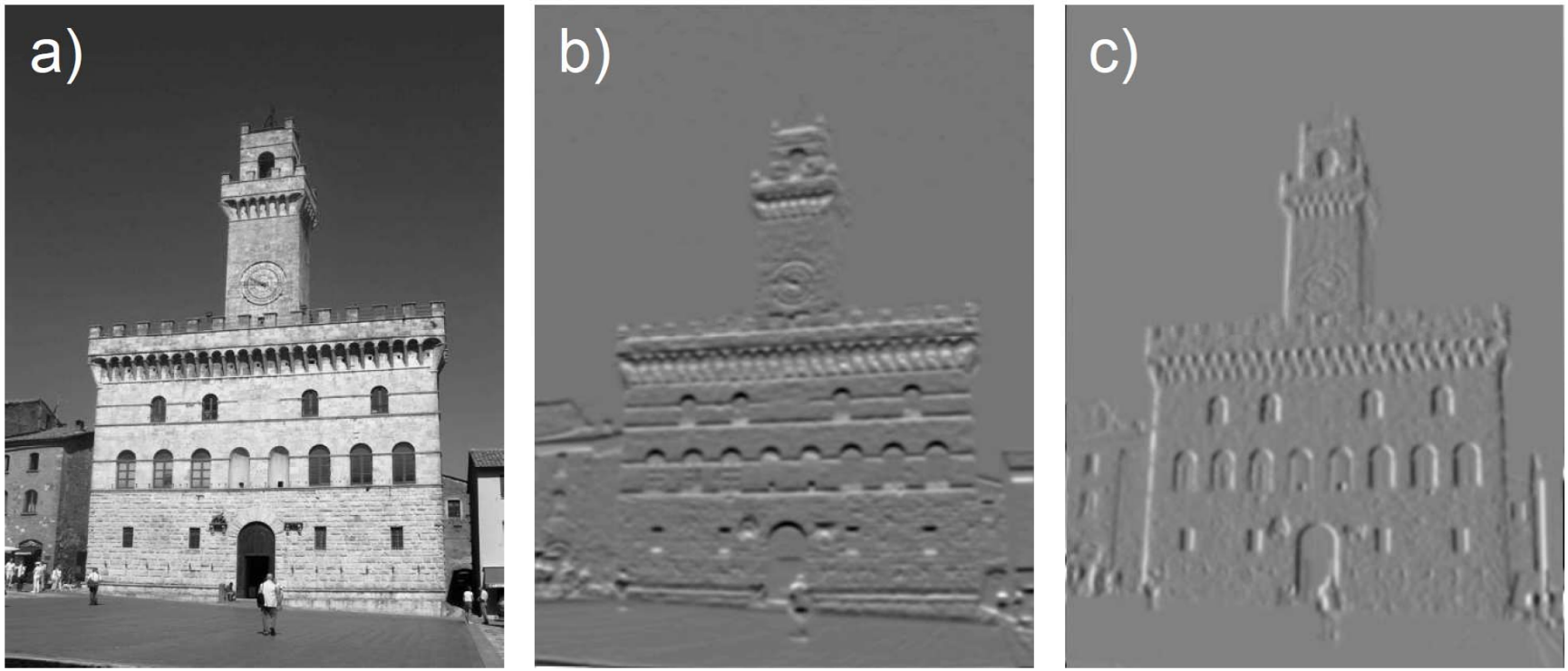
Edges

(from Elder and Goldberg 2000)

Canny Edge Detector

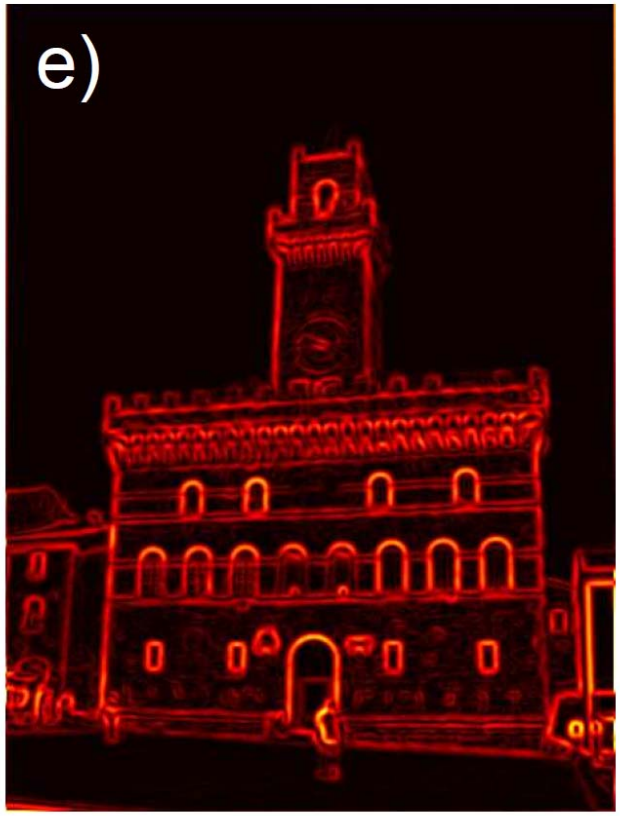


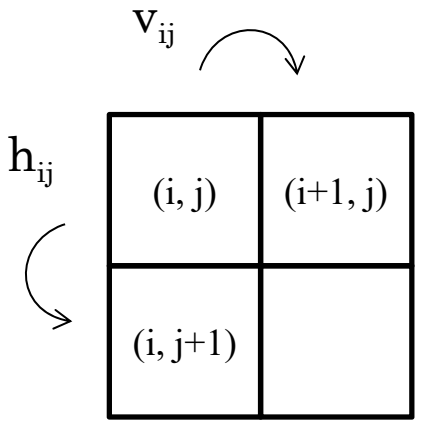
Canny Edge Detector



Compute horizontal and vertical gradient images **h** and **v**

Canny Edge Detector



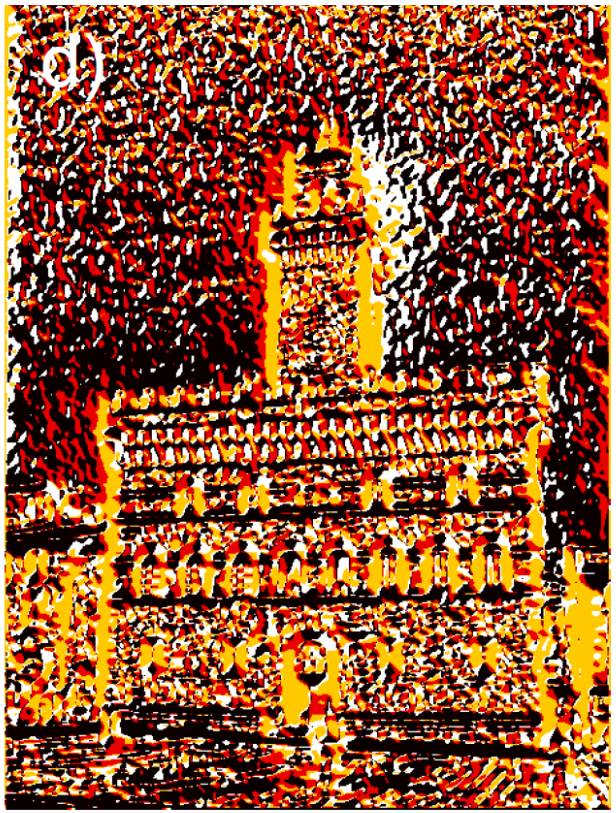


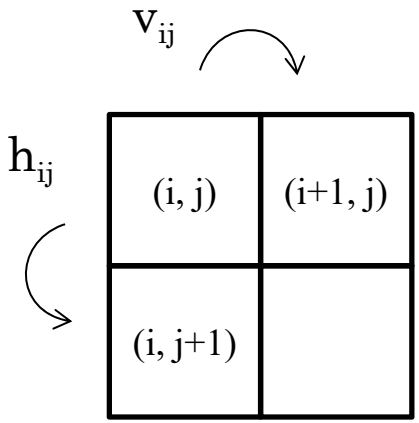
$$a_{ij} = \sqrt{h_{ij}^2 + v_{ij}^2}$$

$$\boldsymbol{\theta}_{ij} = \arctan[v_{ij}/h_{ij}]$$

Quantize to 4 directions

Canny Edge Detector



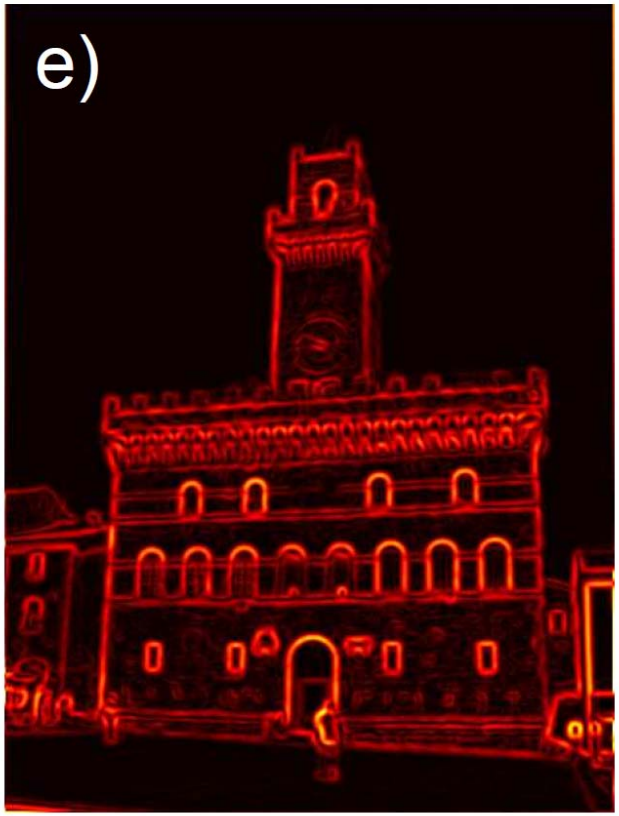


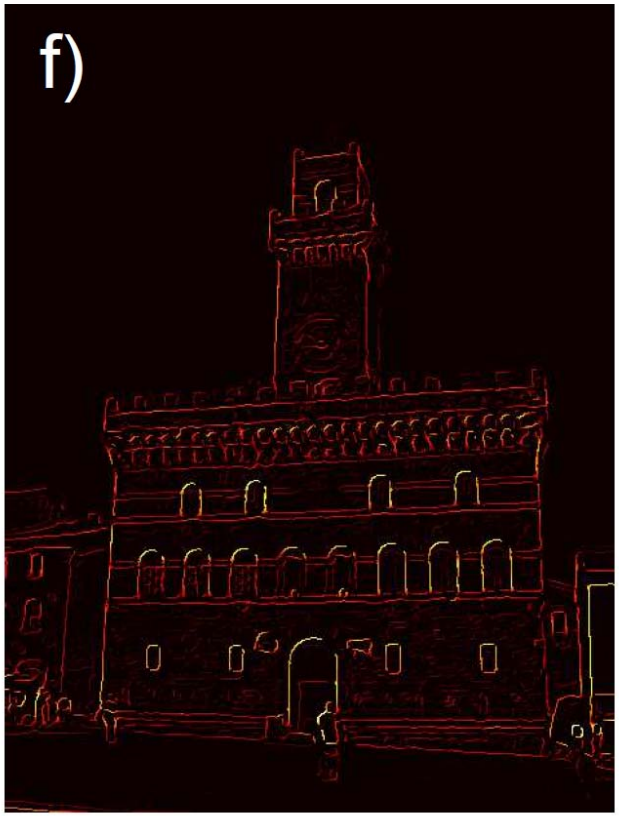
$$a_{ij} = \sqrt{h_{ij}^2 + v_{ij}^2}$$

$$\theta_{ij} = \arctan[v_{ij}/h_{ij}]$$

Quantize to 4 directions

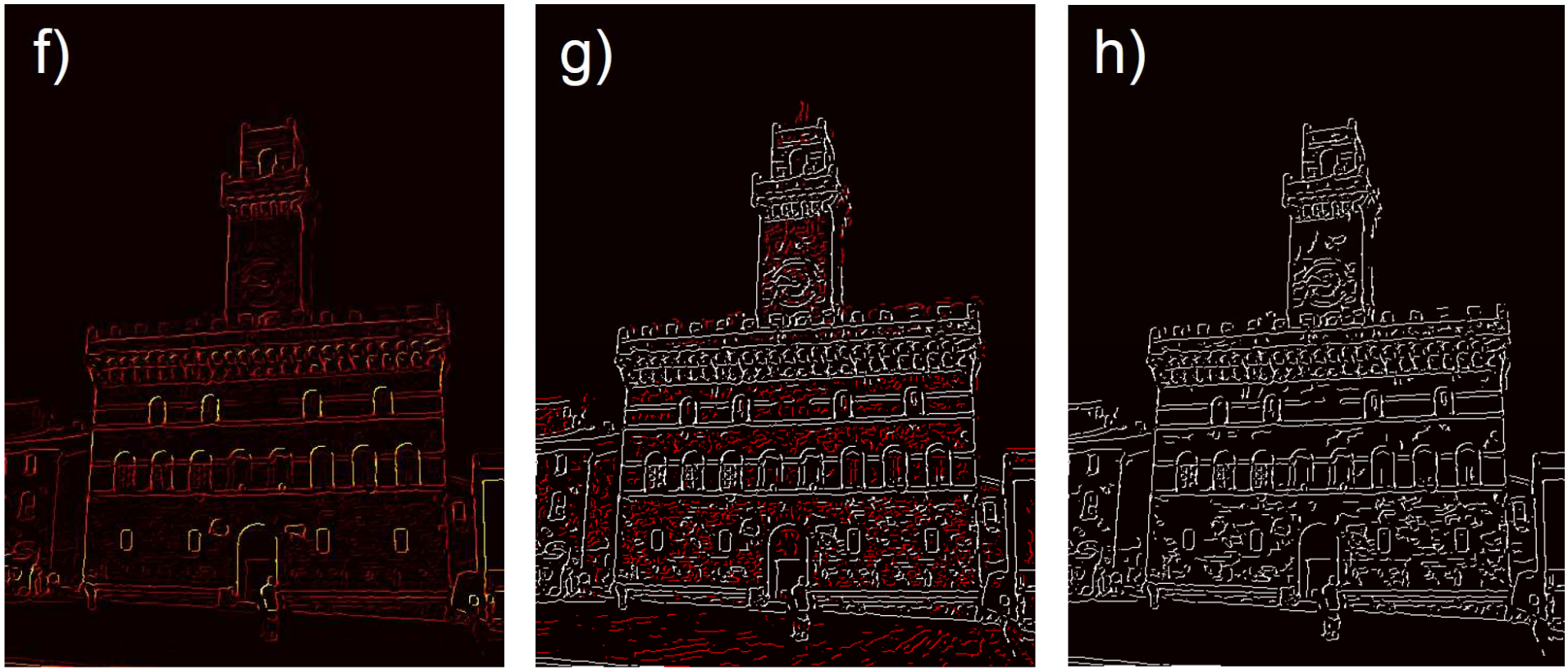
Canny Edge Detector





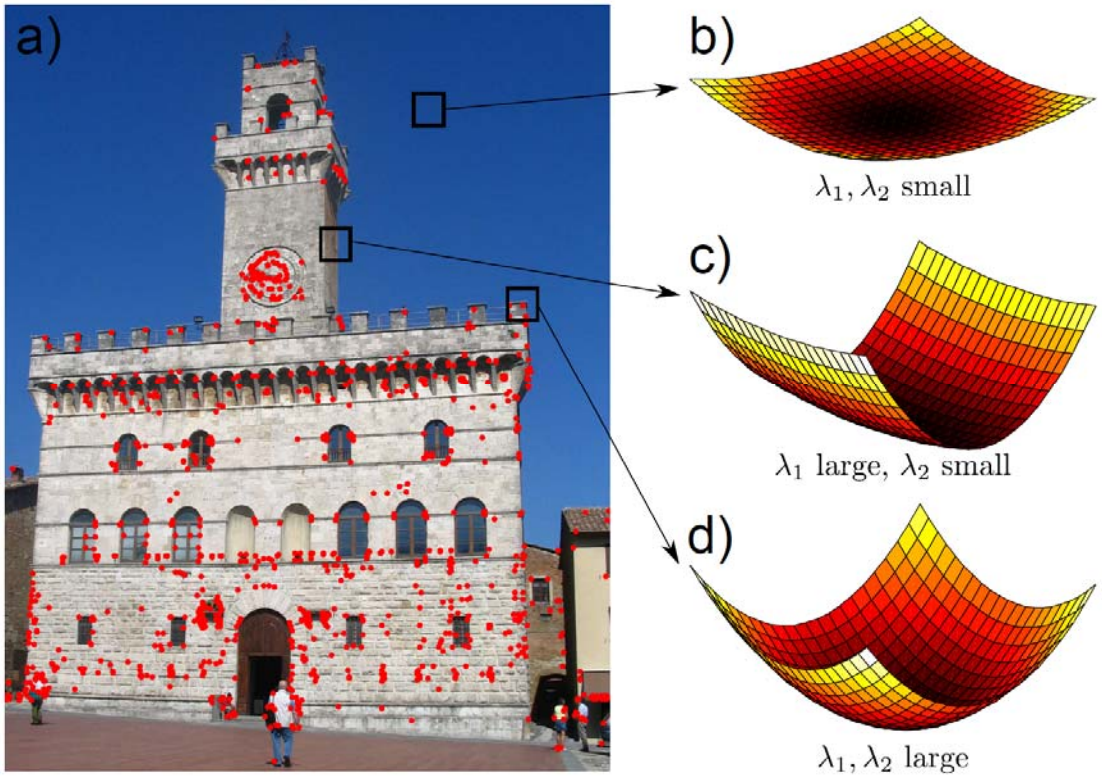
Non-maximal suppression

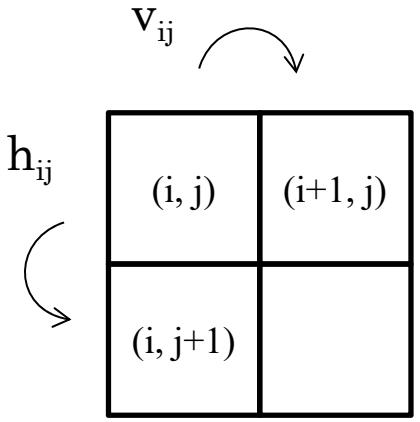
Canny Edge Detector



Hysteresis Thresholding

Harris Corner Detector





Make decision based on image structure tensor

$$\mathbf{S}_{ij} = \sum_{m=i-D}^{i+D} \sum_{n=j-D}^{i+D} w_{mn} \begin{bmatrix} h_{ij}^2 & h_{ij}v_{ij} \\ h_{ij}v_{ij} & v_{ij}^2 \end{bmatrix}$$

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html

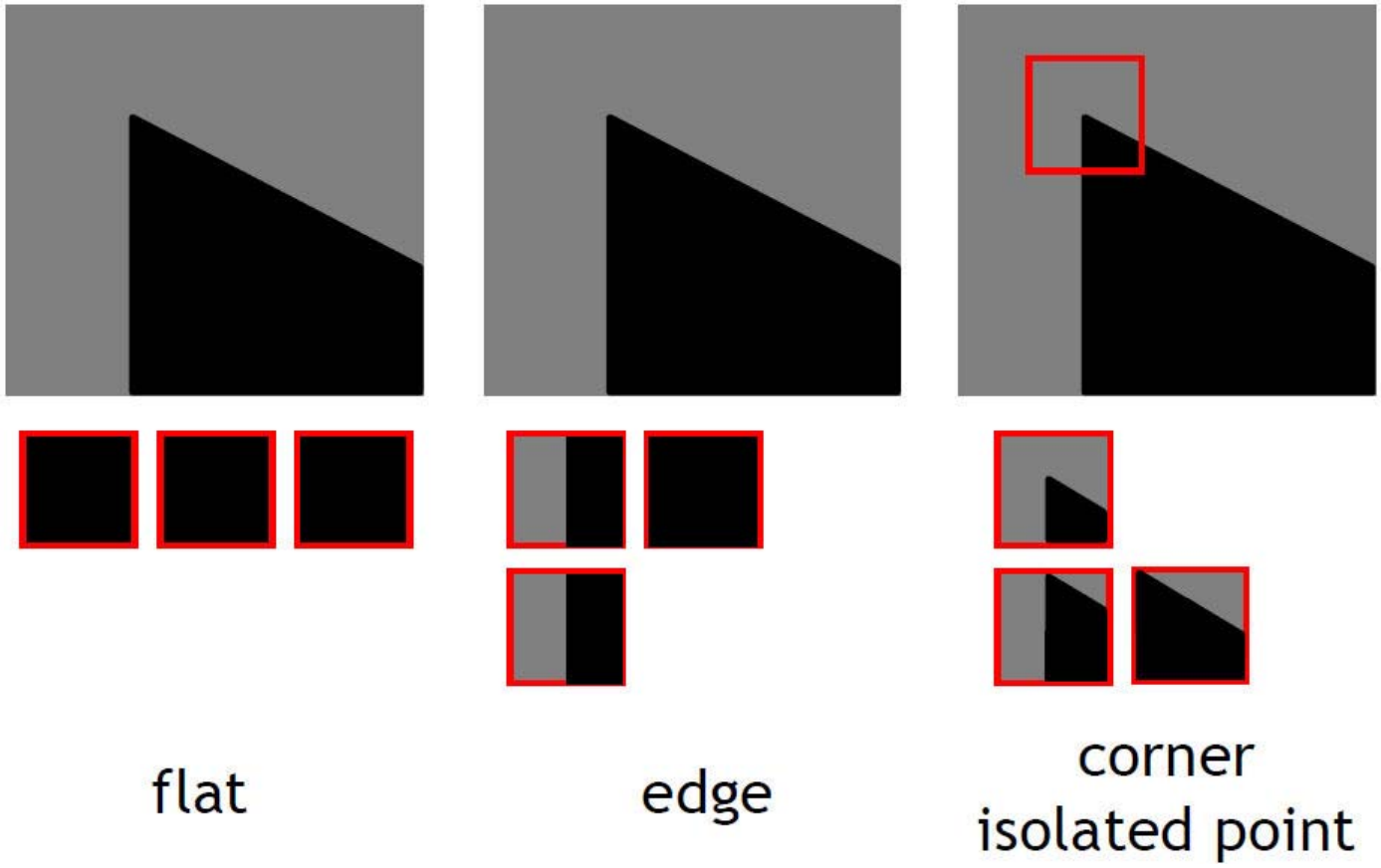
What are features?

Jigsaw puzzle games - how you do it?

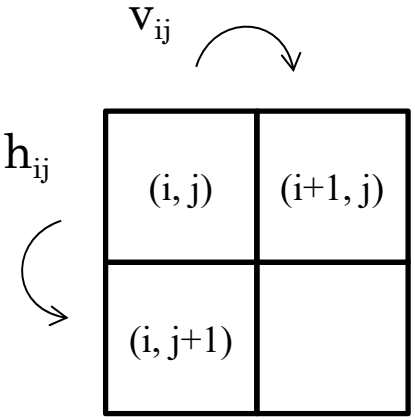
A,B: Flat, difficult
C,D: Edge, still difficult
E,F: Corner, good feature

- **Feature Detection**
  Looking for the regions in images which have maximum variation when moved (by a small amount) in all regions around it
- **Feature Description**
  Describing the region around the feature so that it can find it in other images

Moravec Corner detection (1980)



Moravec Corner detection (1980)

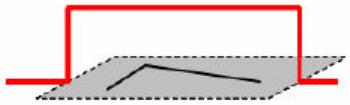


Change of intensity for the shift [u,v]:

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

Window function — Shifted intensity — Intensity

Window function $w(x,y)$ =

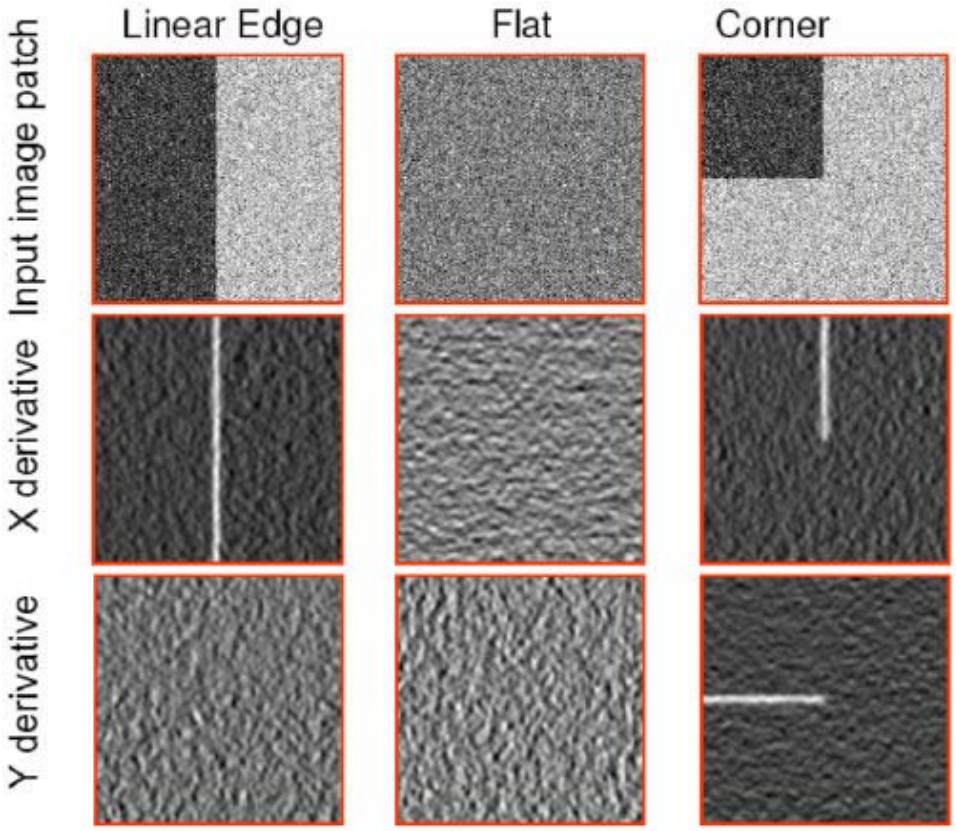


1 in window, 0 outside

Four shift: $(u,v) = \{(1,0),(1,1),(0,1),(-1,1)\}$

- ✓ The similarity is measured as a sum of absolute differences
- ✓ The corners are the pixels with a low similarity with its neighborhood

Different regions and their derivatives



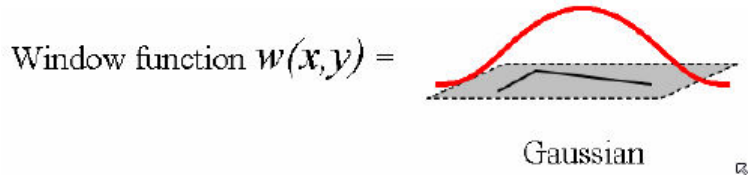
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

Harris Corner Detector

- Moravec Corner detection:
  - Noisy response due to a binary window function
  - → Use a Gaussian function
  - Only a set of shifts at every 45 degree is considered
  - → Consider all small shifts by Taylor's expansion
  - Only minimum of E is taken into account
  - → New corner measurement

Use a Gaussian function

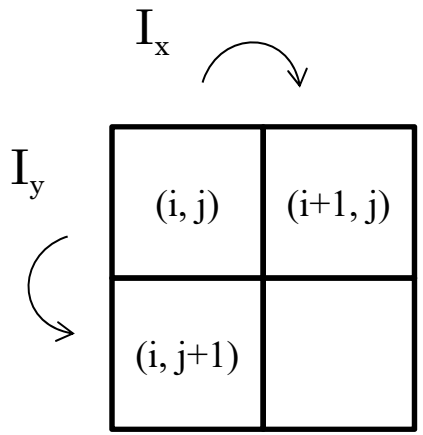
$$w(x,y) = exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$



Consider all small shifts by Taylor's expansion

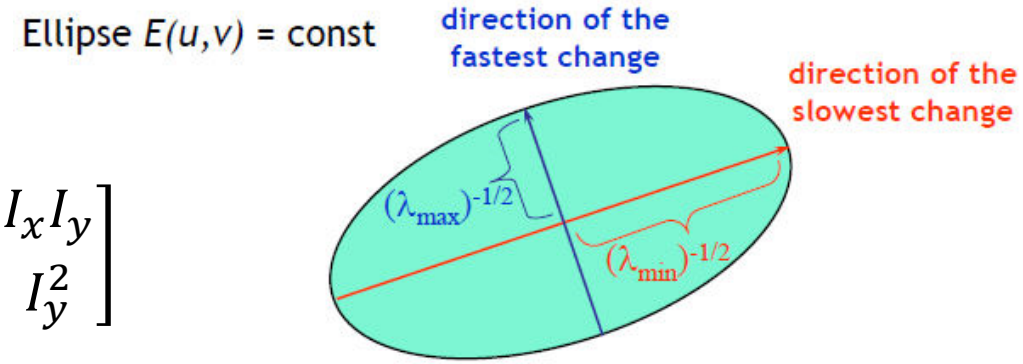
$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+u) - I(x,y)]^2$$

$$= \sum_{x,y} w(x,y)\left[I_x u + I_y v + O(u^2, v^2)\right]^2$$

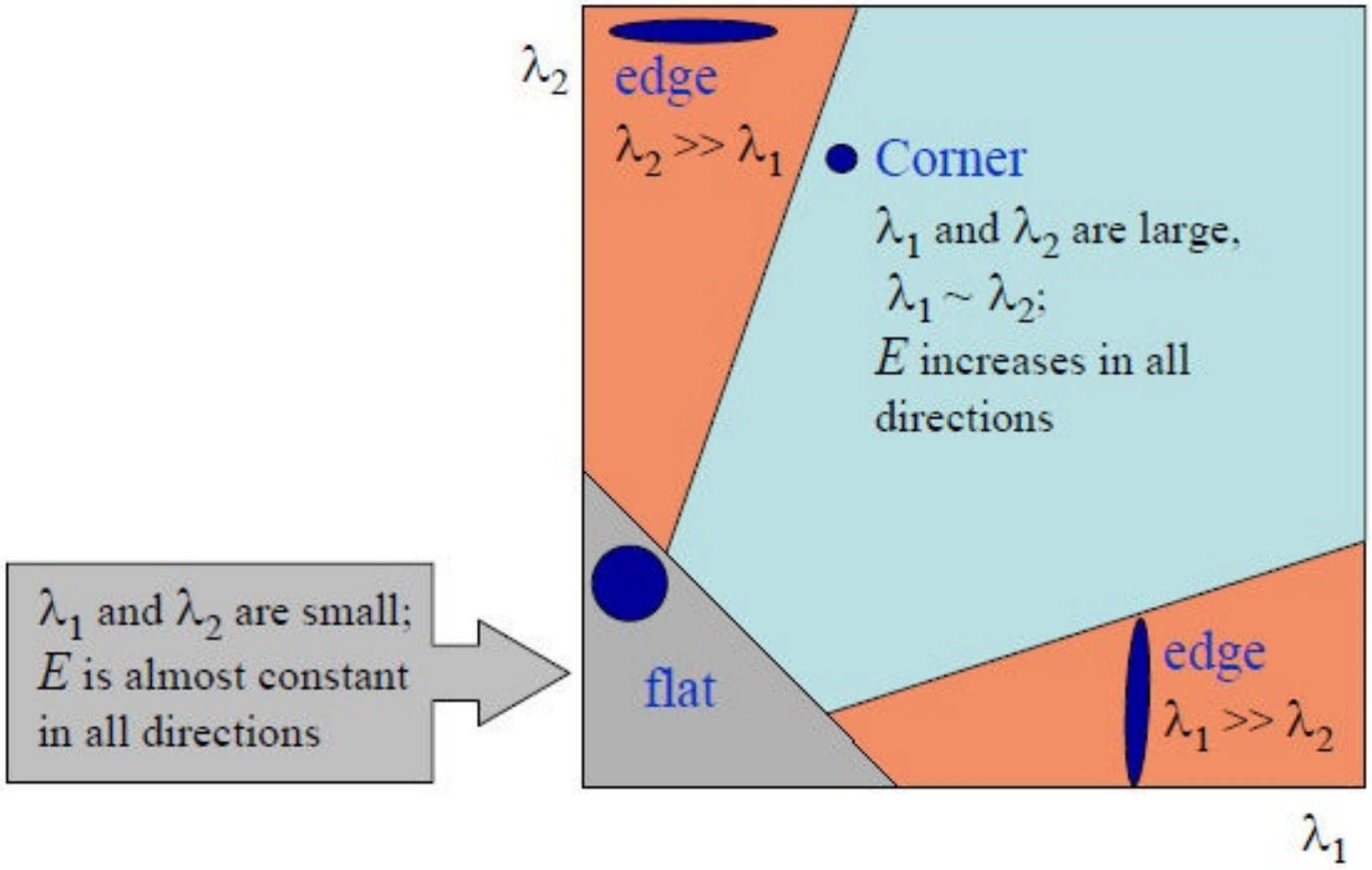


$$E(u,v) \approx [u\ v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



New corner measurement by λ_1 and λ_2



$$R = det(M) - k(trace(M))^2$$

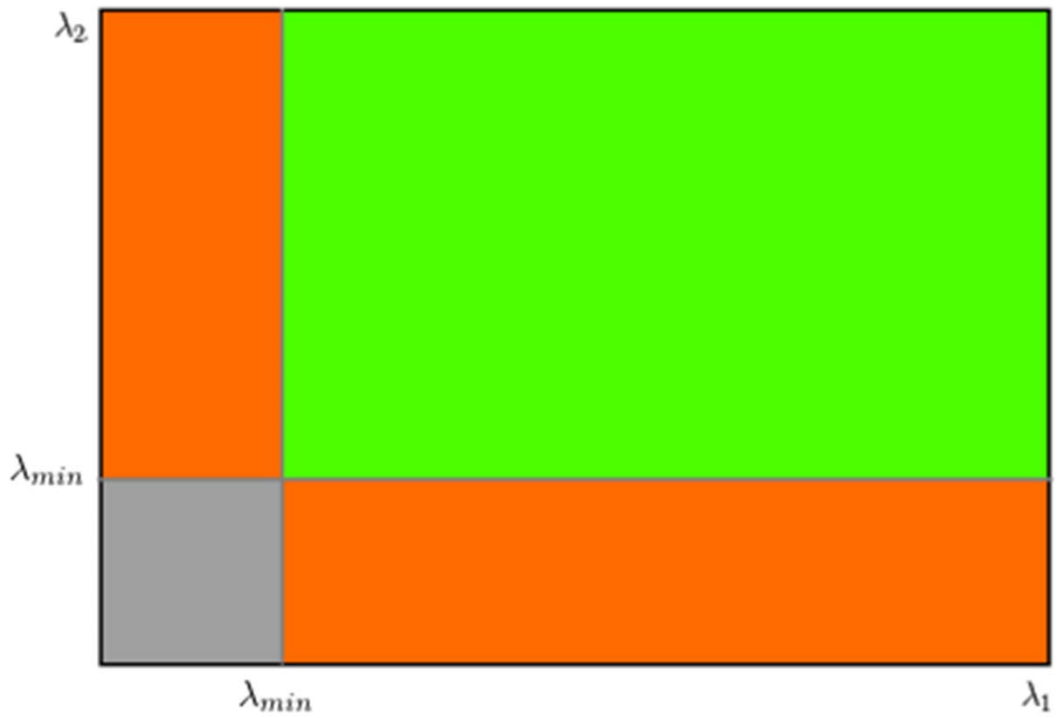
where

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigen values of M

So the values of these eigen values decide whether a region is corner, edge or flat.

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

Good Feature to Track

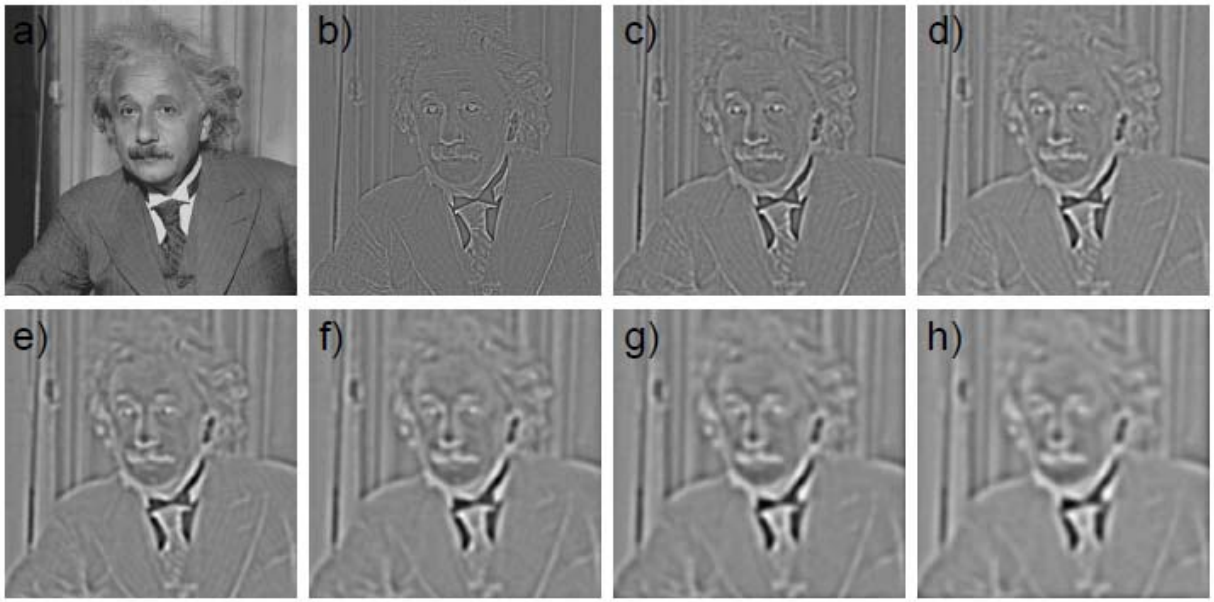


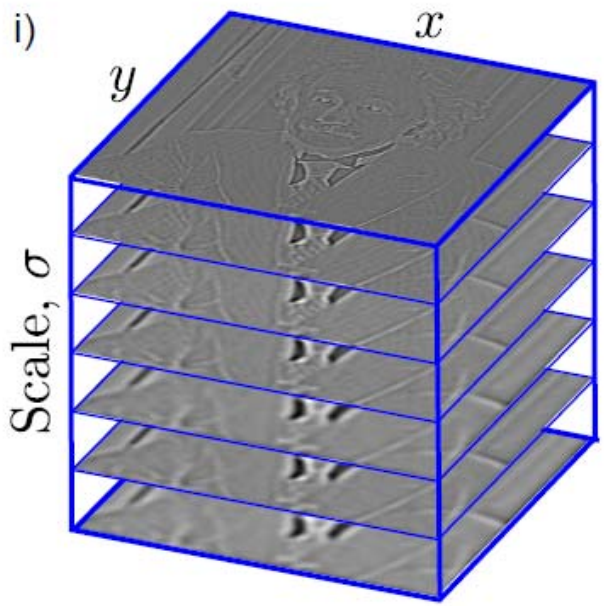
Harris Corner Detector in OpenCV

- OpenCV has the function **cv2.cornerHarris()** for this purpose.

```
import cv2
import numpy as np

filename = 'chessboard.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]

cv2.imshow('dst',img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

SIFT Detector



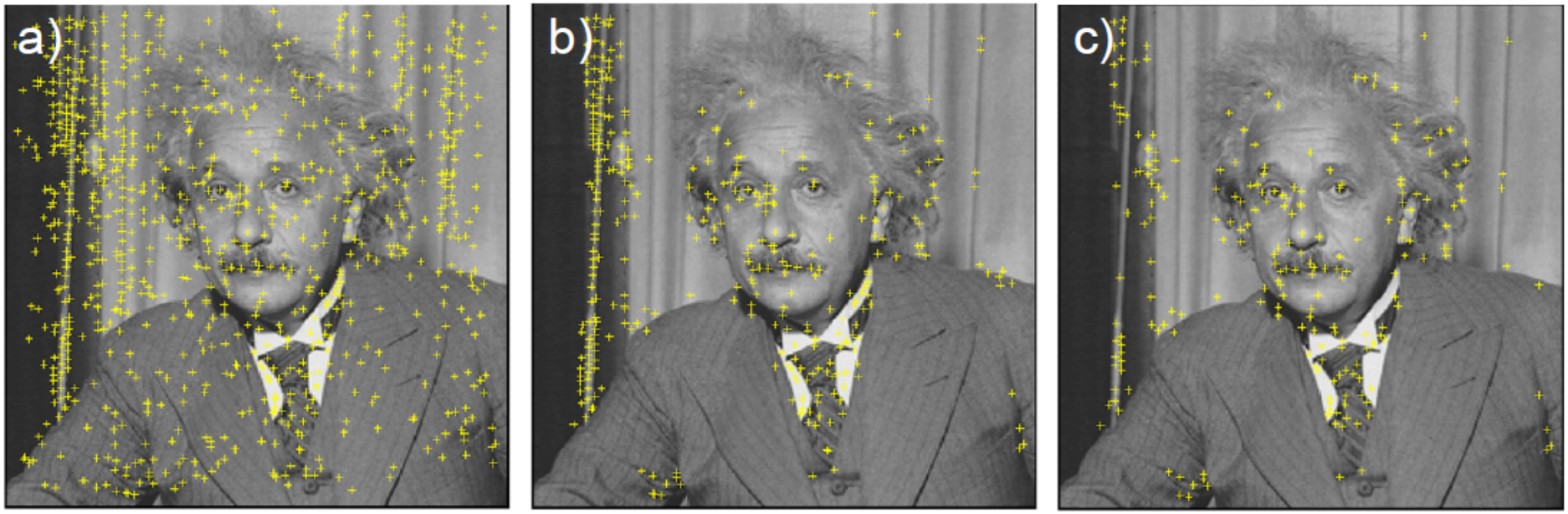


Filter with difference of Gaussian filters at increasing scales

Build image stack (scale space)

Find extrema in this 3D volume

SIFT Detector



Identified Corners

Remove those on edges

Remove those where contrast is low

Assign Orientation

Orientation assigned by looking at intensity gradients in region around point

Form a histogram of these gradients by binning.

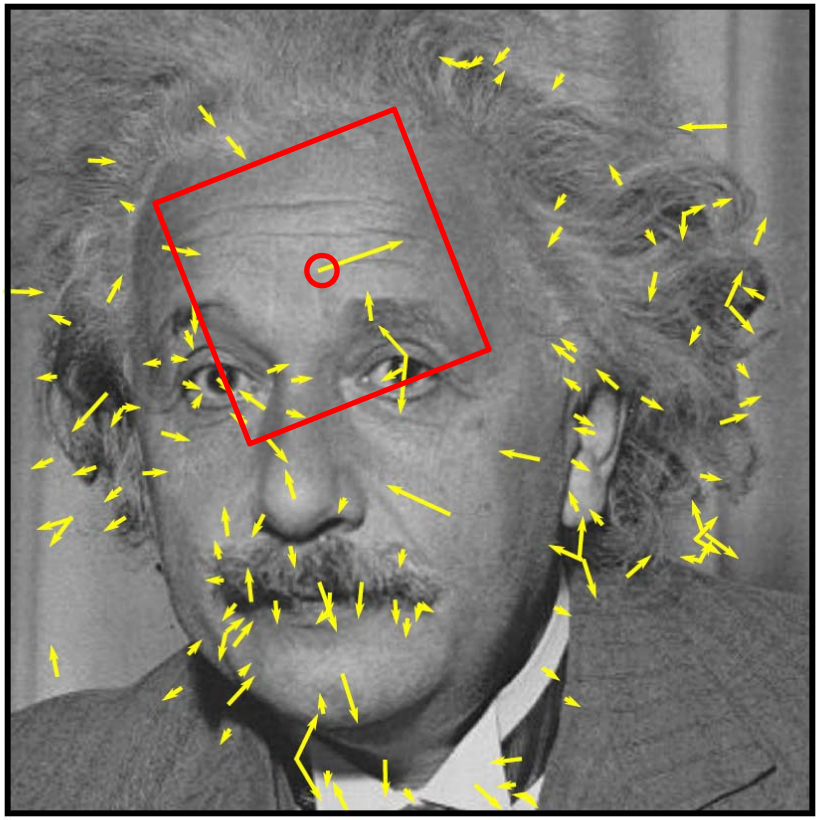
Set orientation to peak of histogram.

Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

Sift Descriptor

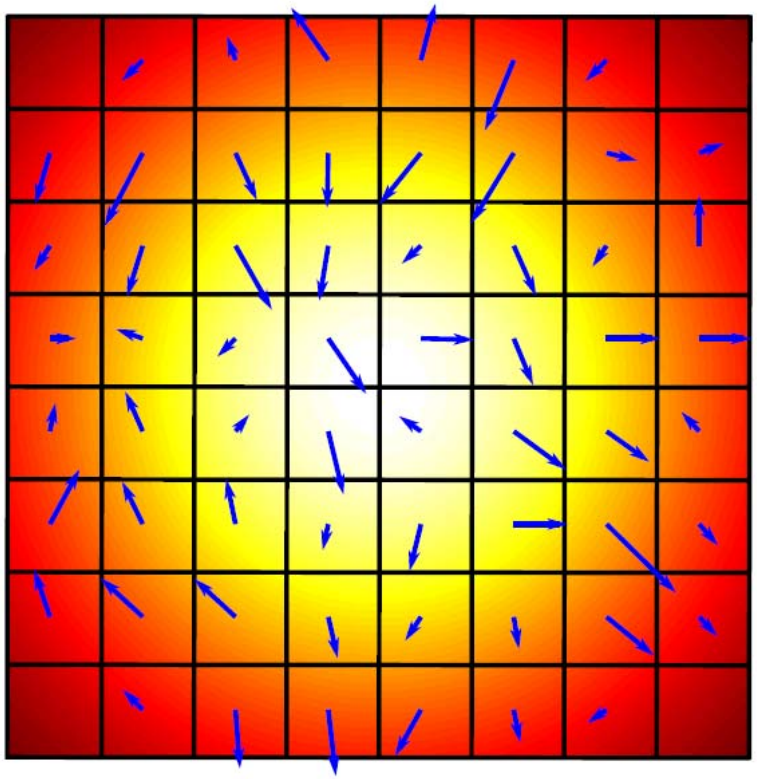
Goal: produce a vector that describes the region around the interest point.



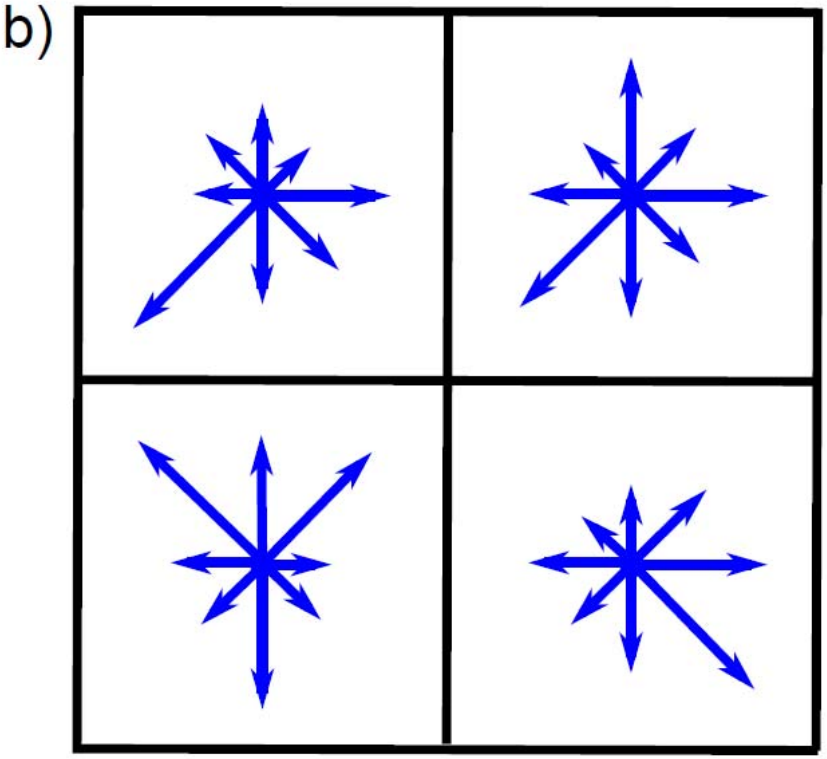
All calculations are relative to the orientation and scale of the keypoint

Makes descriptor invariant to rotation and scale

Sift Descriptor



1. Compute image gradients



2. Pool into local histograms
3. Concatenate histograms
4. Normalize histograms

HoG Descriptor

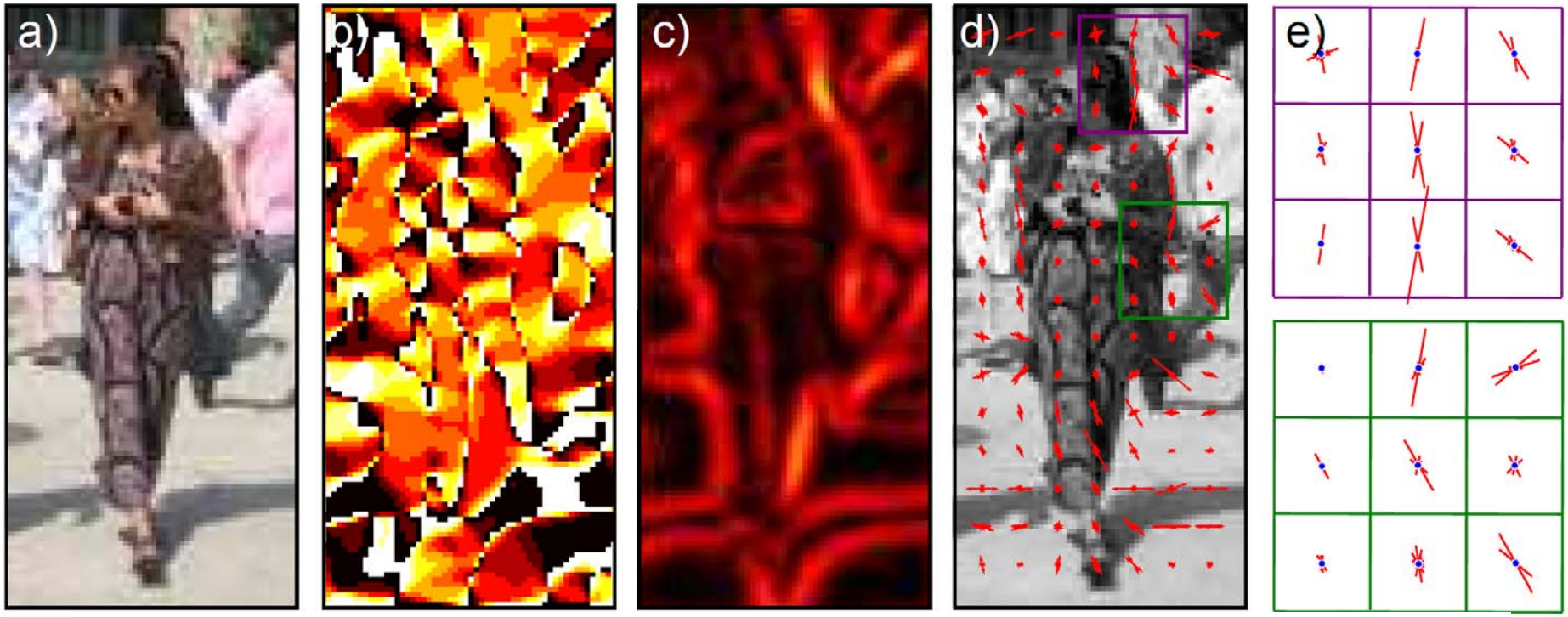
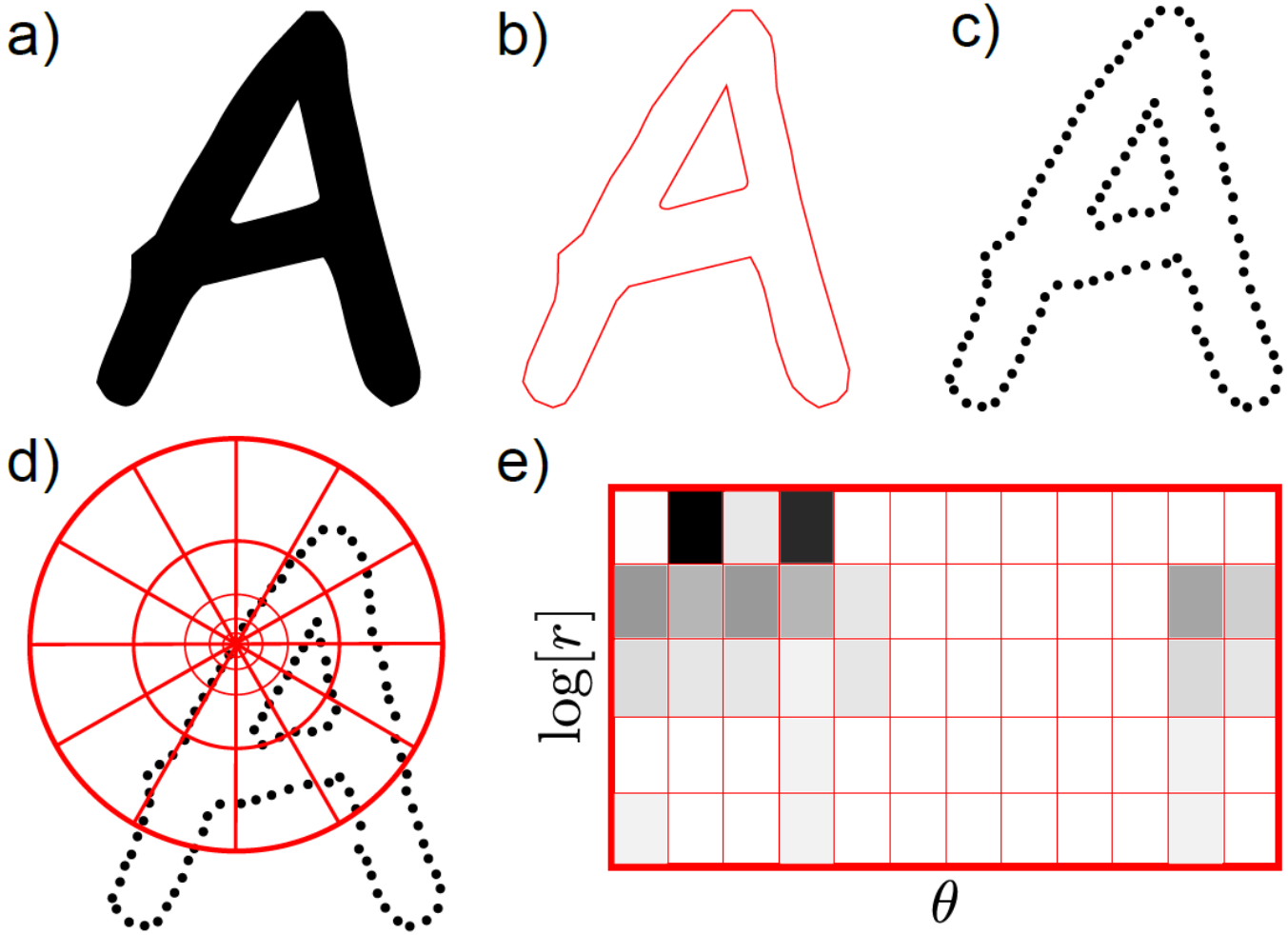


Figure 13.17 HOG descriptor. a) Original image. b) Gradient orientation, quantized into 9 bins from $0 - 180^o$. c) Gradient magnitude. d) Cell descriptors are 9D orientation histograms that are computed within 6×6 pixel regions. e) Block descriptors are computed by concatenating 3×3 blocks of cell descriptors. The block descriptors are normalized. The final HOG descriptor consists of the concatenated block descriptors.

Bag of words descriptor

- Compute visual features in image
- Compute descriptor around each
- Find closest match in library and assign index
- Compute histogram of these indices over the region
- Dictionary computed using K-means

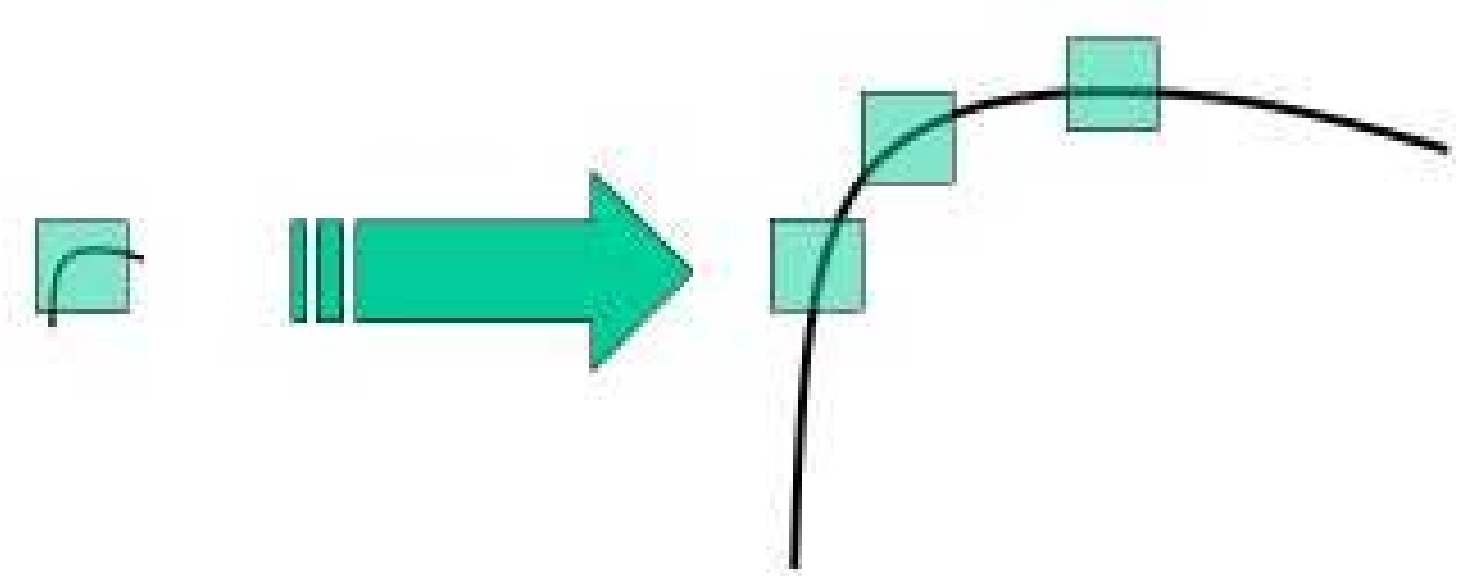
Shape context descriptor



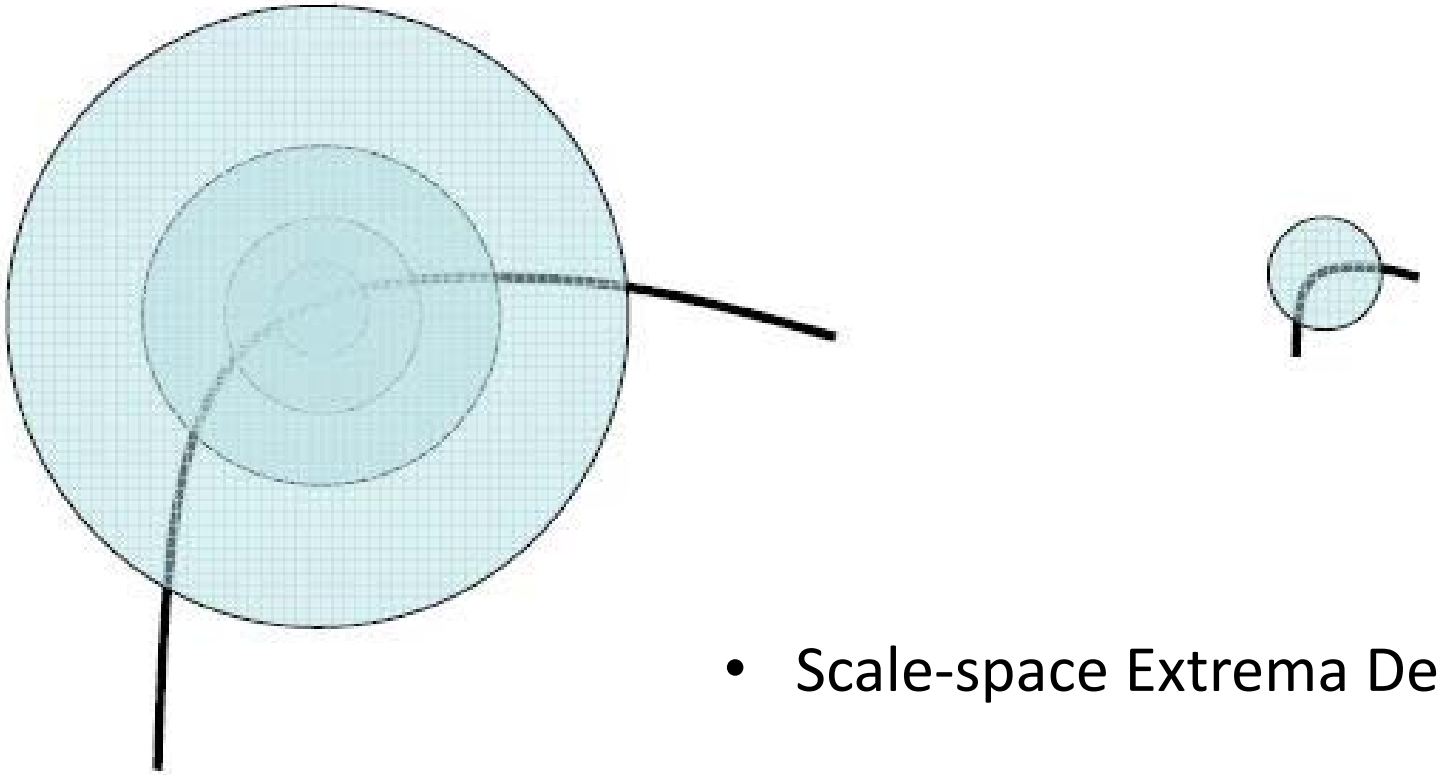
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

SIFT (Scale-Invariant Feature Transform)

Corner? Edge? – Scale-variant

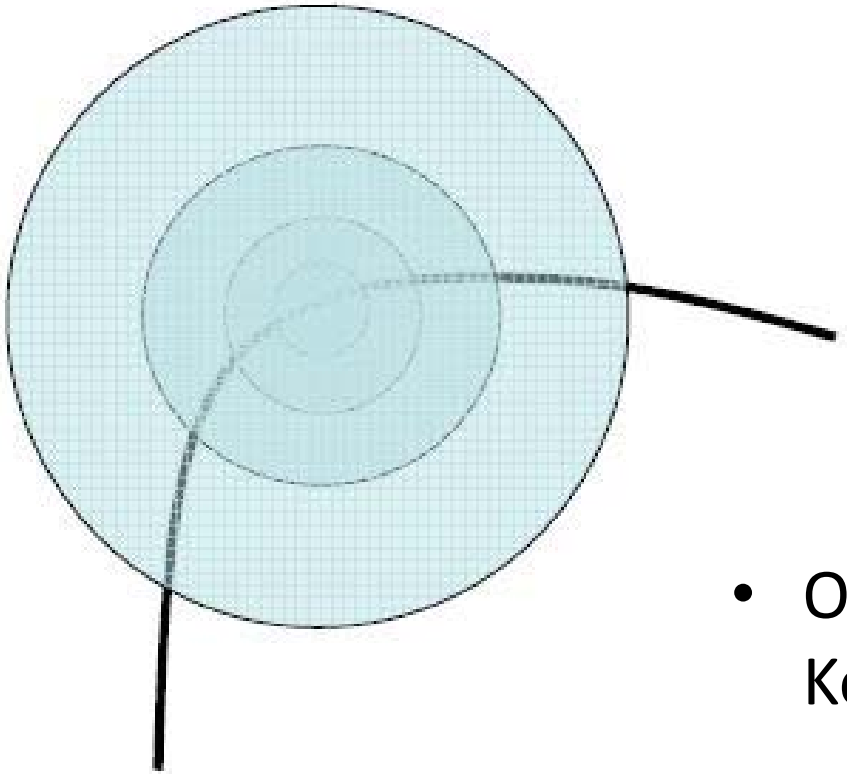


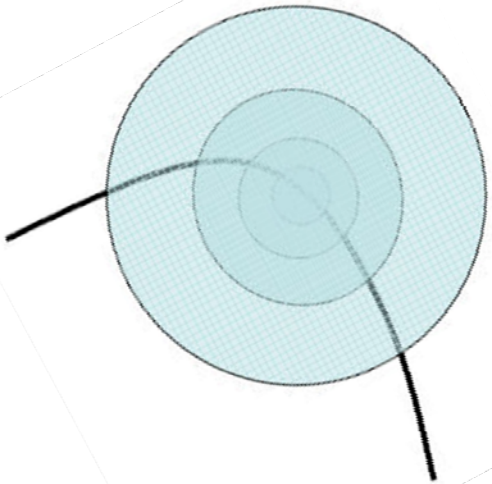
SIFT (Scale-Invariant Feature Transform)



- Scale-space Extrema Detection

SIFT (Scale-Invariant Feature Transform)





Same?

- Orientation Assignment
  Keypoint Description

SIFT (Scale-Invariant Feature Transform)

http://www.vision.cs.chubu.ac.jp/cvtutorial/PDF/02SIFTandMore.pdf

D. Lowe (ICCV99, IJCV04)

1. Keypoint Detection
   A. Scale-space keypoint detection
   B. Keypoint localization
2. Feature Description
   A. Orientation assignment
   B. Keypoint description

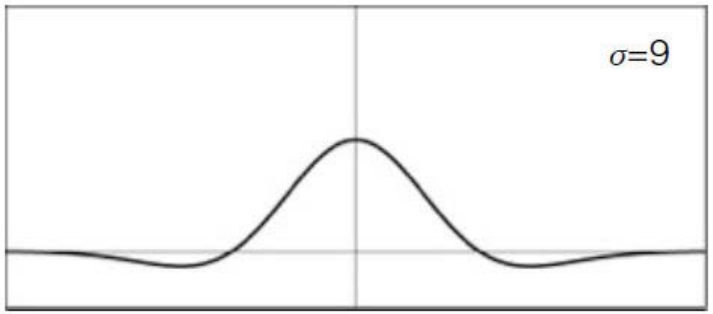
Scale detection - Laplacian-of-Gaussian

[Lindeberg 98]

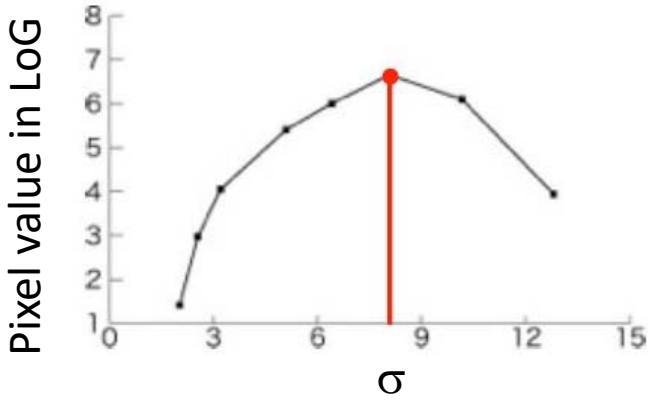
Change σ of LoG Operator

$$LoG = f(\sigma) = -\frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^6}\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

σ: scale, x,y: distance from target pixel



LoG operator



Difference-of-Gaussian (DoG)

~ Approximation of LoG

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

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

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

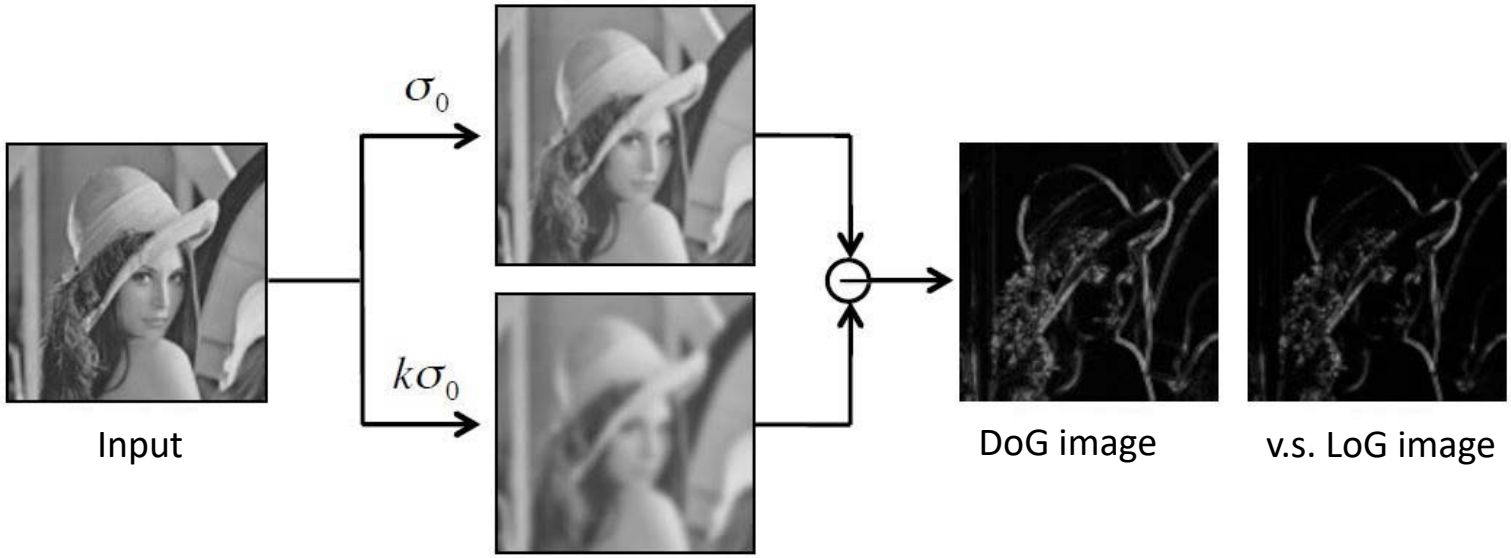
$L(x,y,\sigma)$: Smoothed image

$I(x,y)$: Input image

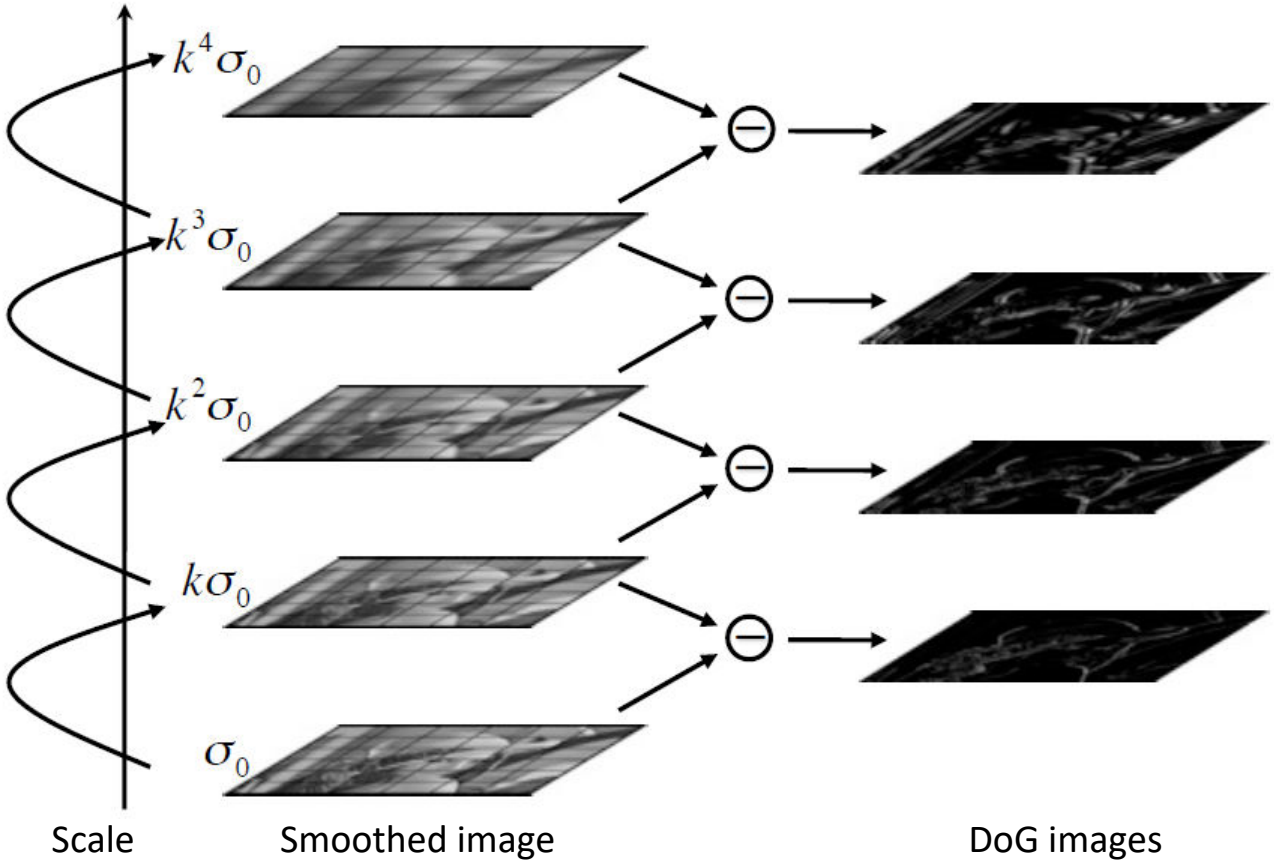
$G(x,y,\sigma)$: Gaussian function

$D(x,y,\sigma)$: DoG image

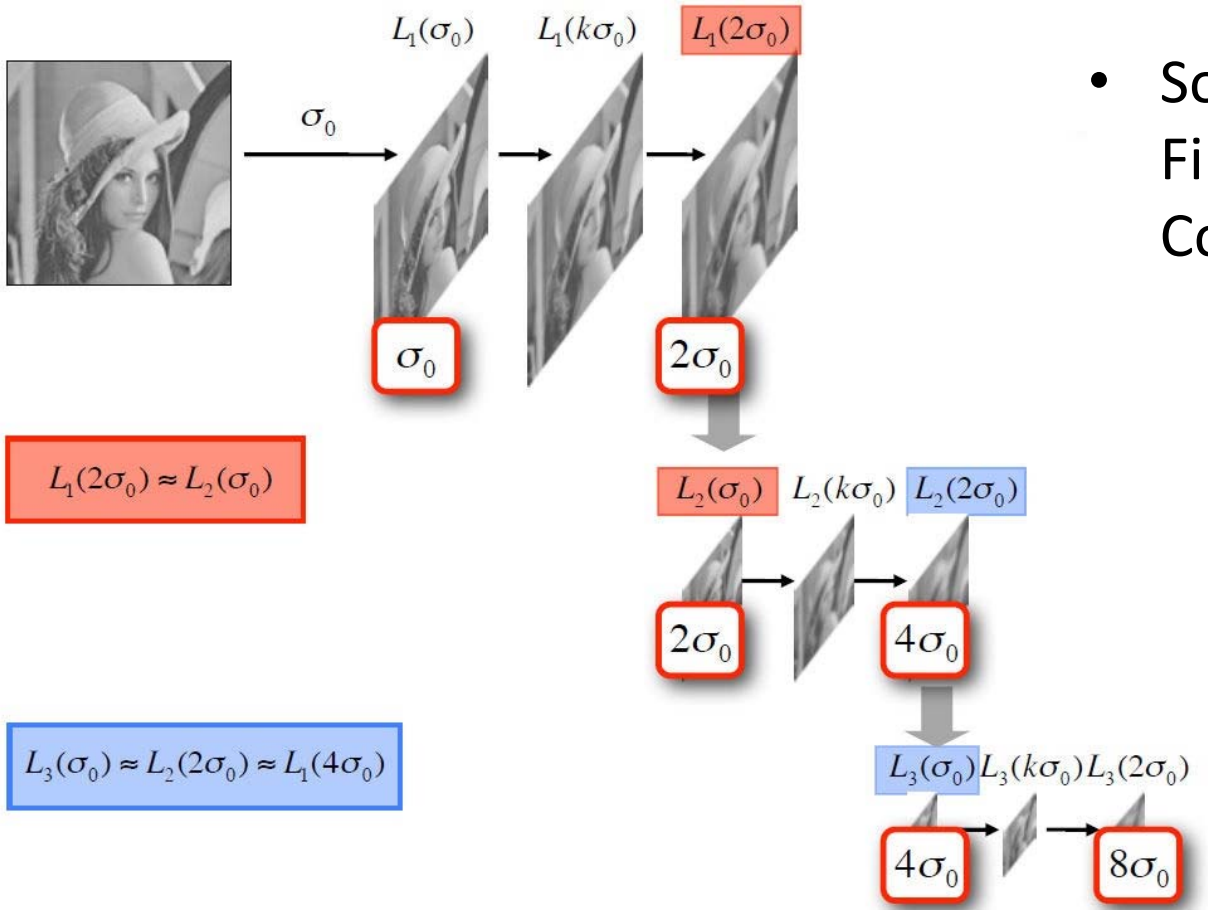
k : Increased ratio



Scale-space by DoG

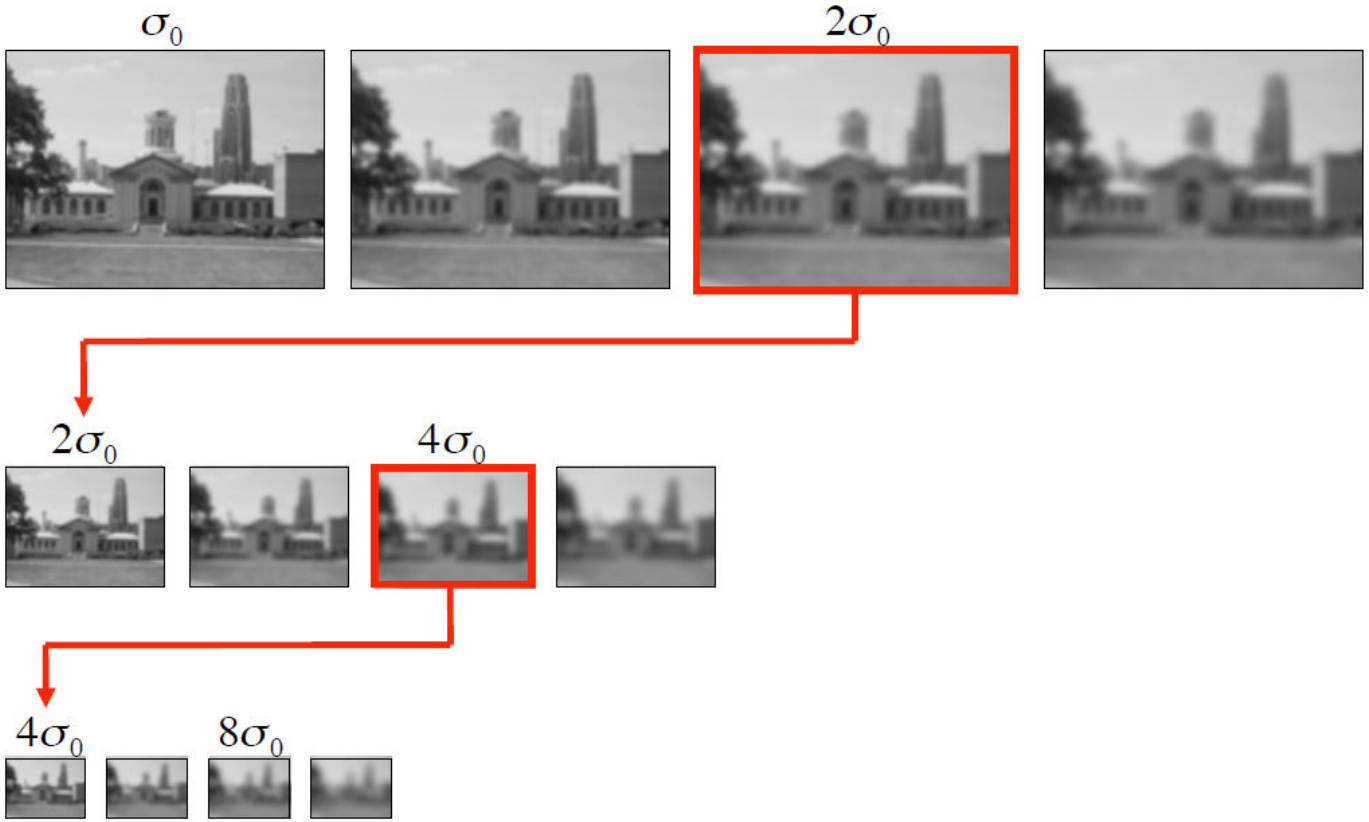


DoG with Down sampling

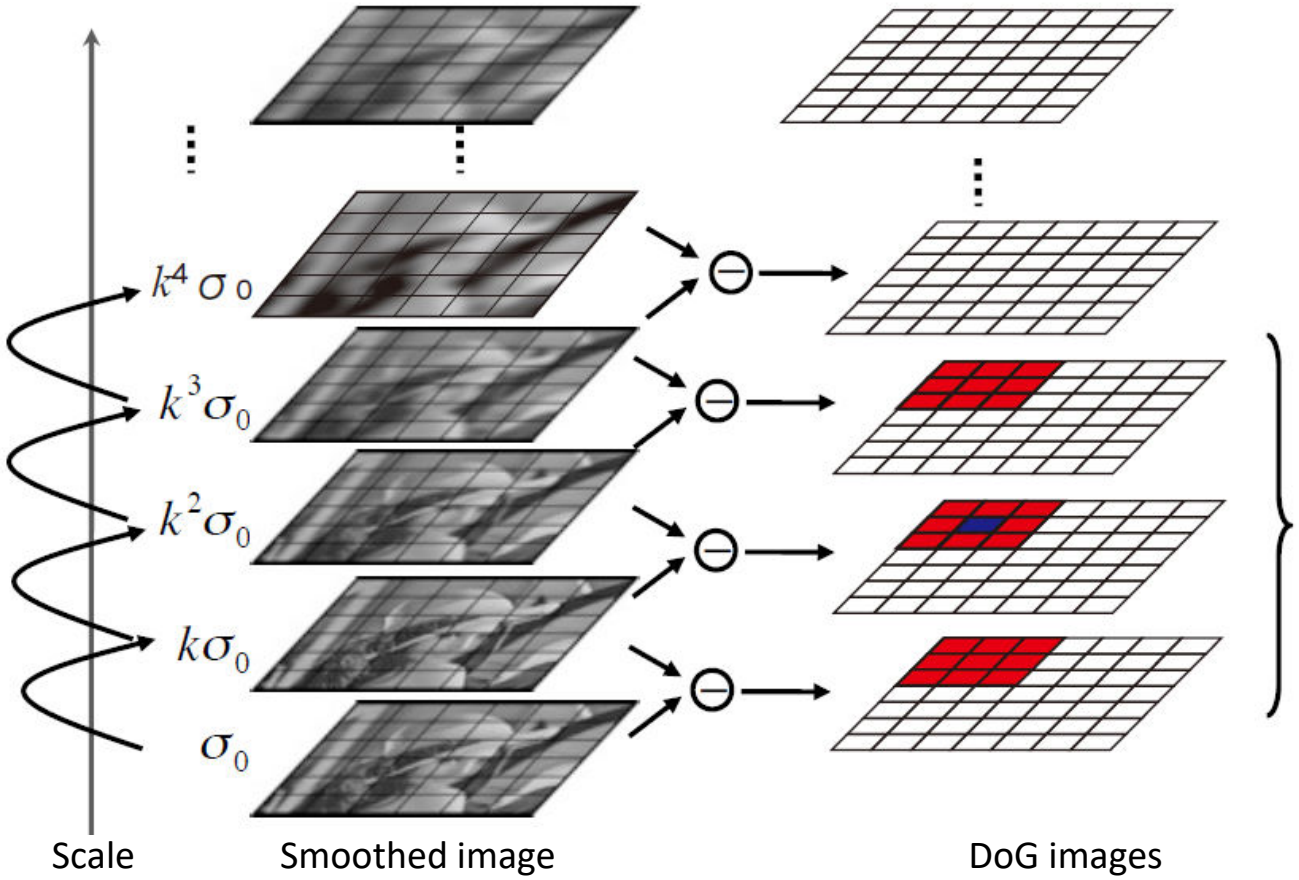


- Solve the problems:
  Filter edge
  Computational cost

DoG with Down sampling

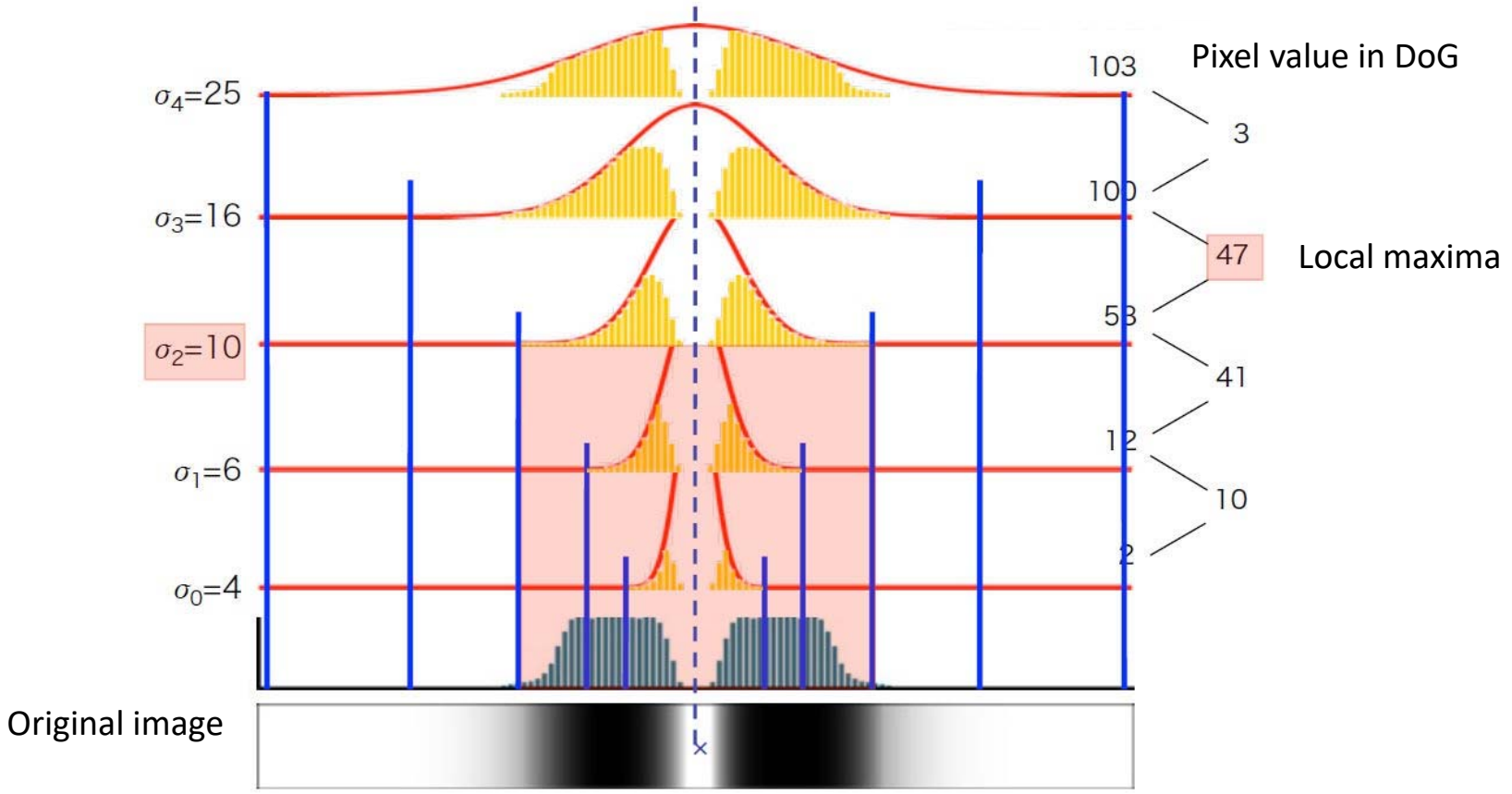


Scale-space by DoG

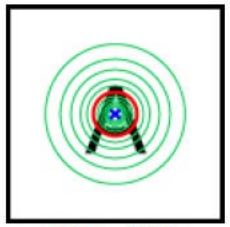


- Find peak from 3 images

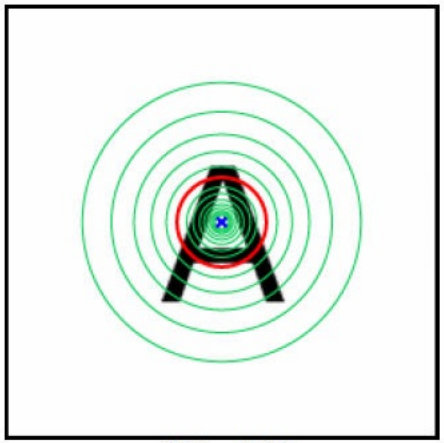
Scale-space by DoG



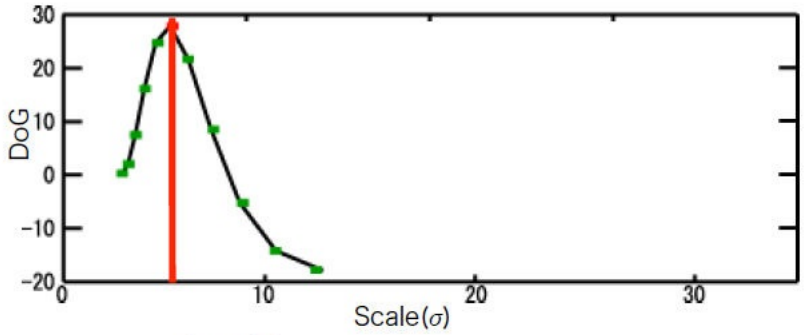
Local maxima in scale space



200×200

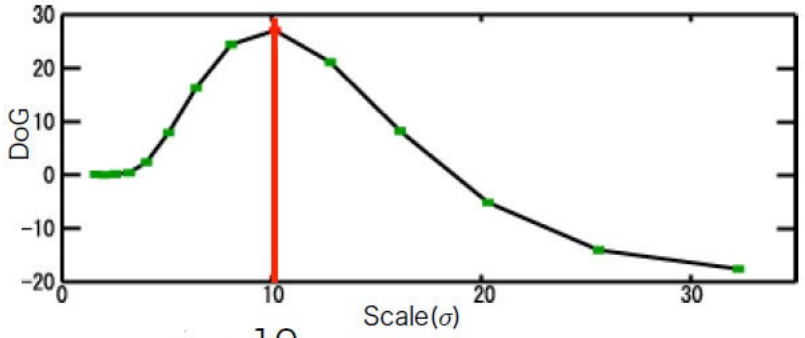


400×400



σ_1=5

σ_2=2σ_1



σ_2=10

SIFT (Scale-Invariant Feature Transform)

http://www.vision.cs.chubu.ac.jp/cvtutorial/PDF/02SIFTandMore.pdf

D. Lowe (ICCV99, IJCV04)

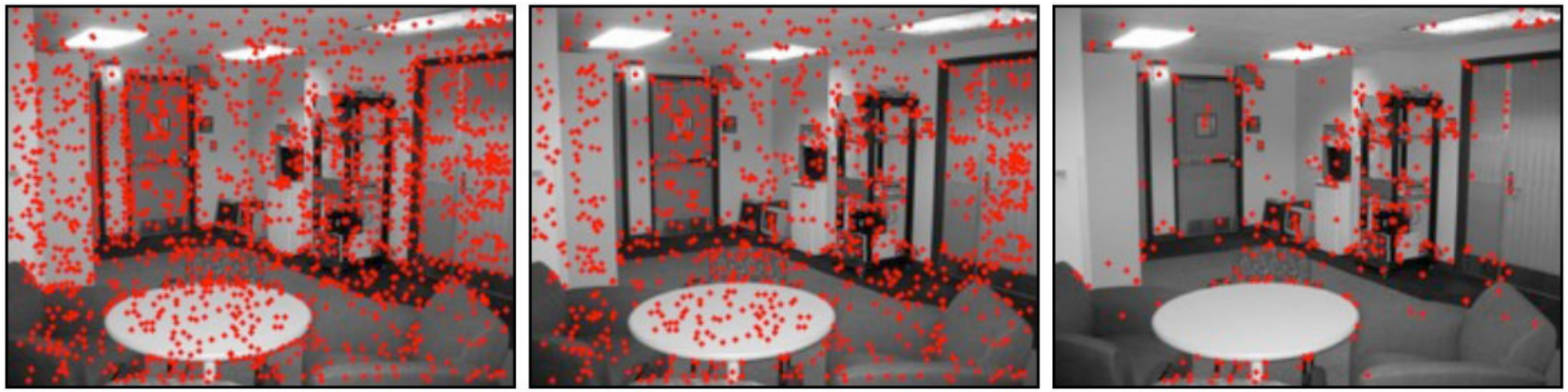
1. Keypoint Detection
   A. Scale-space keypoint detection
   B. Keypoint localization
2. Feature Description
   A. Orientation assignment
   B. Keypoint description

Keypoint localization

Not appropriate for keypoint:

Points on the edge – Similar appearance

Small pixel values in DoG image – Easily affected by noise

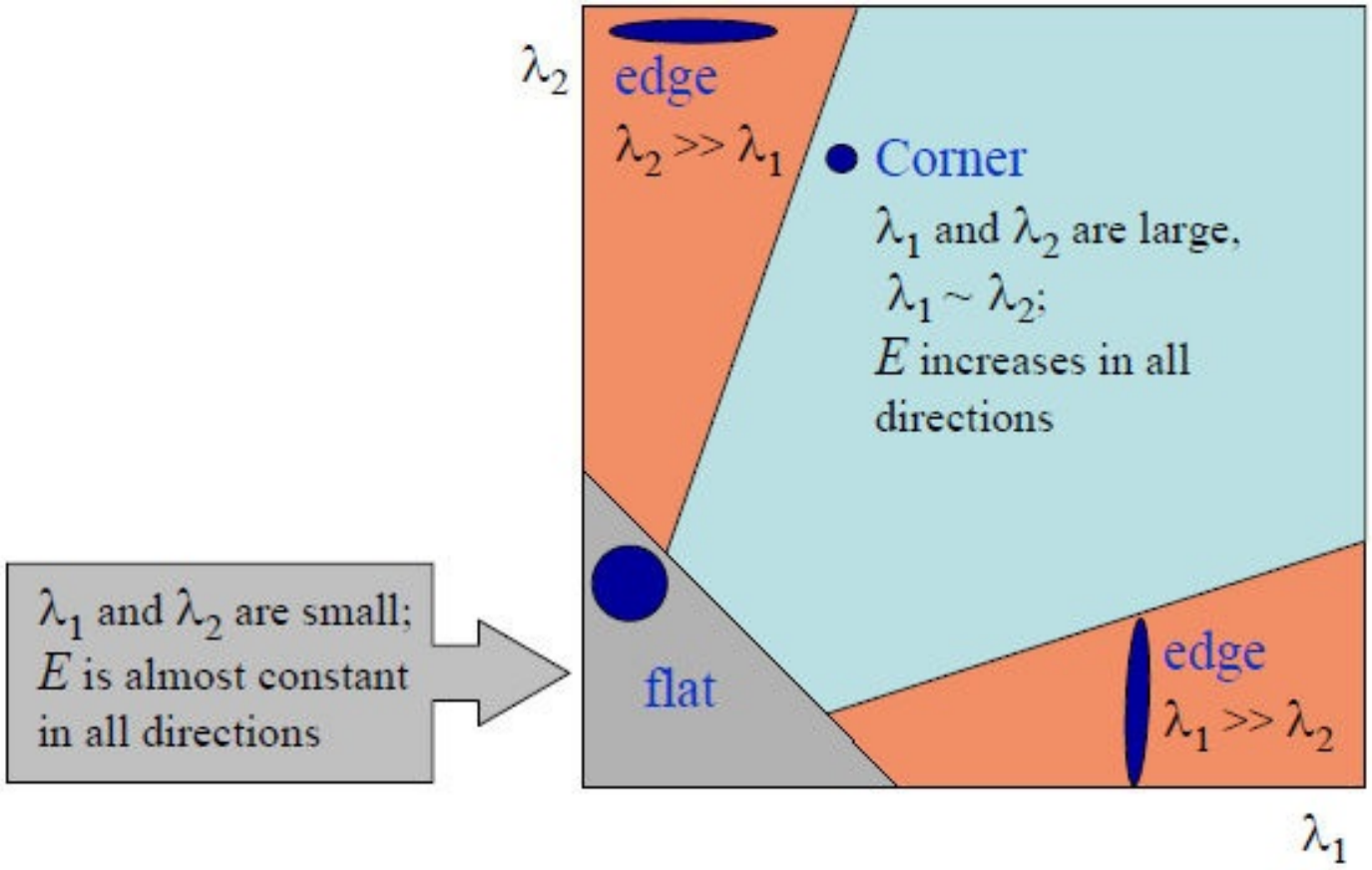


Keypoint candidates (1895)

Decreased by principal curve (1197)

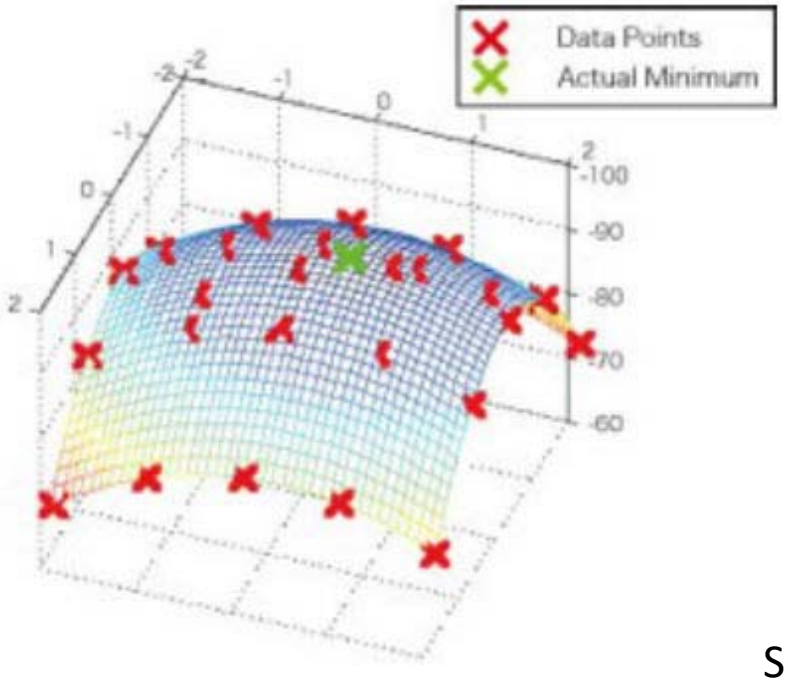
Decreased by contrast (421)

Principle curve by λ_1 and λ_2



Sub-pixel localization of keypoints

- Parabola fitting in 3D space



Taylor expansion of $\mathbf{x} = (x, y, \sigma)^T$

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

$$\frac{\partial D}{\partial \mathbf{x}} + \frac{\partial^2 D}{\partial \mathbf{x}^2}\hat{\mathbf{x}} = 0 \quad \Rightarrow \quad \frac{\partial^2 D}{\partial \mathbf{x}^2}\hat{\mathbf{x}} = -\frac{\partial D}{\partial \mathbf{x}}$$

$$\hat{\mathbf{x}} = \begin{bmatrix} x \\ y \\ \sigma \end{bmatrix} = -\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial xy} & \frac{\partial^2 D}{\partial x\sigma} \\ \frac{\partial^2 D}{\partial xy} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y\sigma} \\ \frac{\partial^2 D}{\partial x\sigma} & \frac{\partial^2 D}{\partial y\sigma} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix}$$

Sub-pixel minima

Corner with SubPixel Accuracy

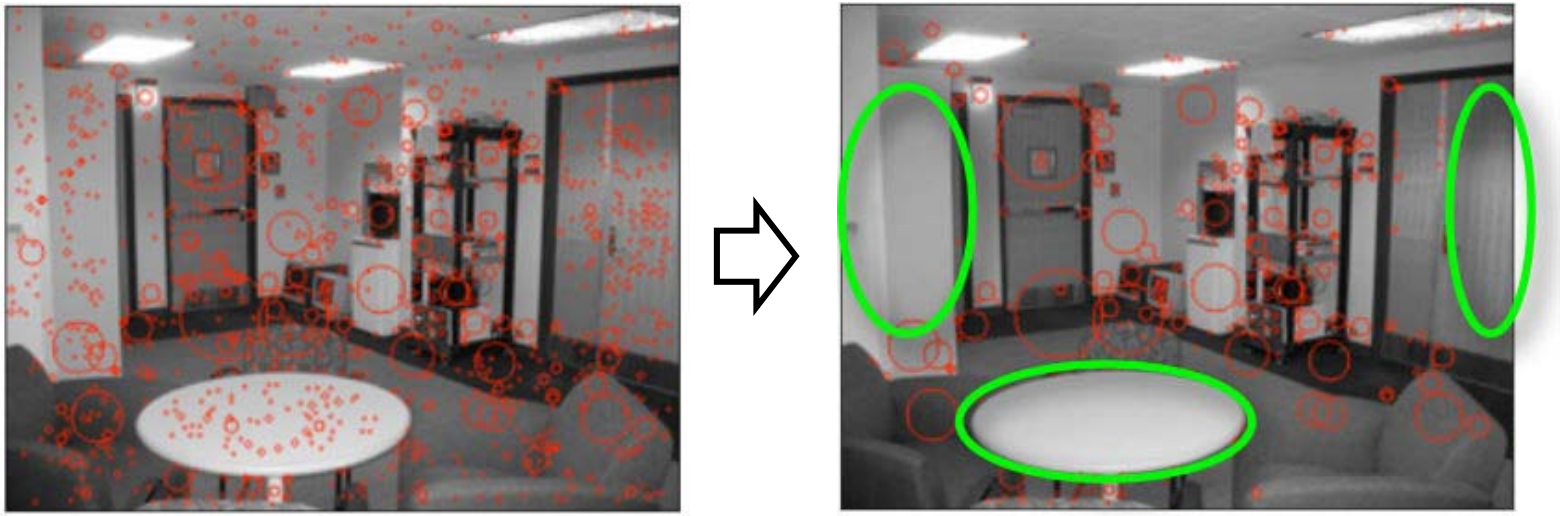
cv2.cornerSubPix() which further refines the corners detected with sub-pixel accuracy.

```python
import cv2
import numpy as np

filename = 'chessboard2.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# find Harris corners
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
dst = cv2.dilate(dst,None)
ret, dst = cv2.threshold(dst,0.01*dst.max(),255,0)
dst = np.uint8(dst)

# find centroids
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

# define the criteria to stop and refine the corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

# Now draw them
res = np.hstack((centroids,corners))
res = np.int0(res)
img[res[:,1],res[:,0]]=[0,0,255]
img[res[:,3],res[:,2]] = [0,255,0]

cv2.imwrite('subpixel5.png',img)
```

Relationship with λ_1, λ_2?

Thresholding of low-contrast points

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}} \quad < \text{ th ~ 0.03}$$



SIFT (Scale-Invariant Feature Transform)

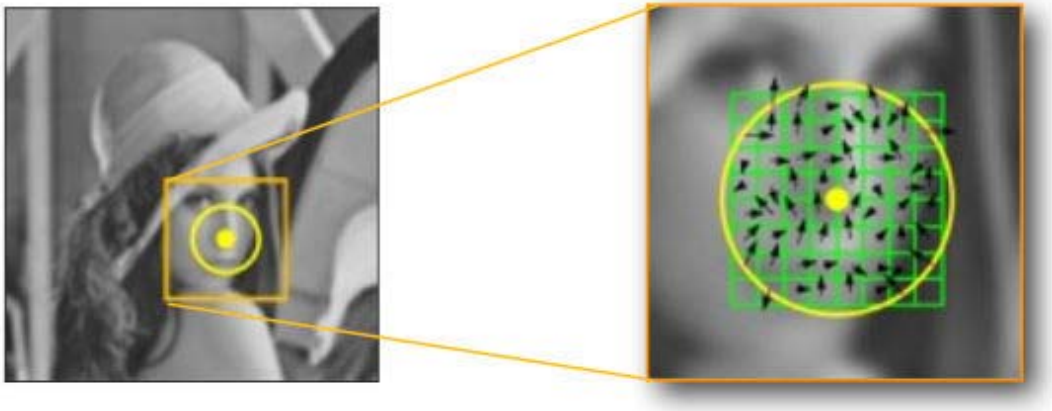
http://www.vision.cs.chubu.ac.jp/cvtutorial/PDF/02SIFTandMore.pdf

D. Lowe (ICCV99, IJCV04)

1. Keypoint Detection
   A. Scale-space keypoint detection
   B. Keypoint localization
2. Feature Description
   A. Orientation assignment
   B. Keypoint description

Orientation Assignment

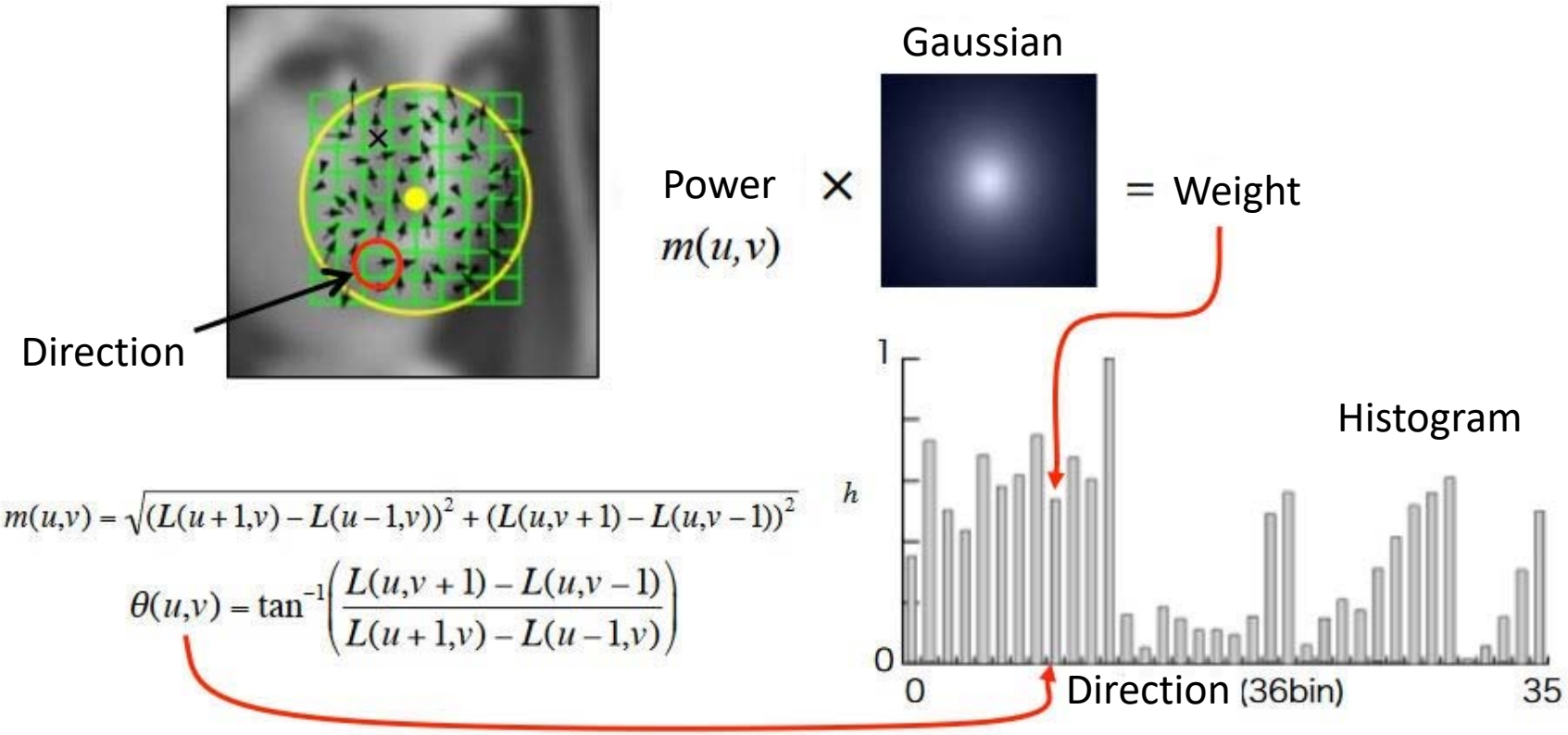
Gradient directions at the pixels



$$m(u,v) = \sqrt{(L(u+1,v) - L(u-1,v))^2 + (L(u,v+1) - L(u,v-1))^2}$$

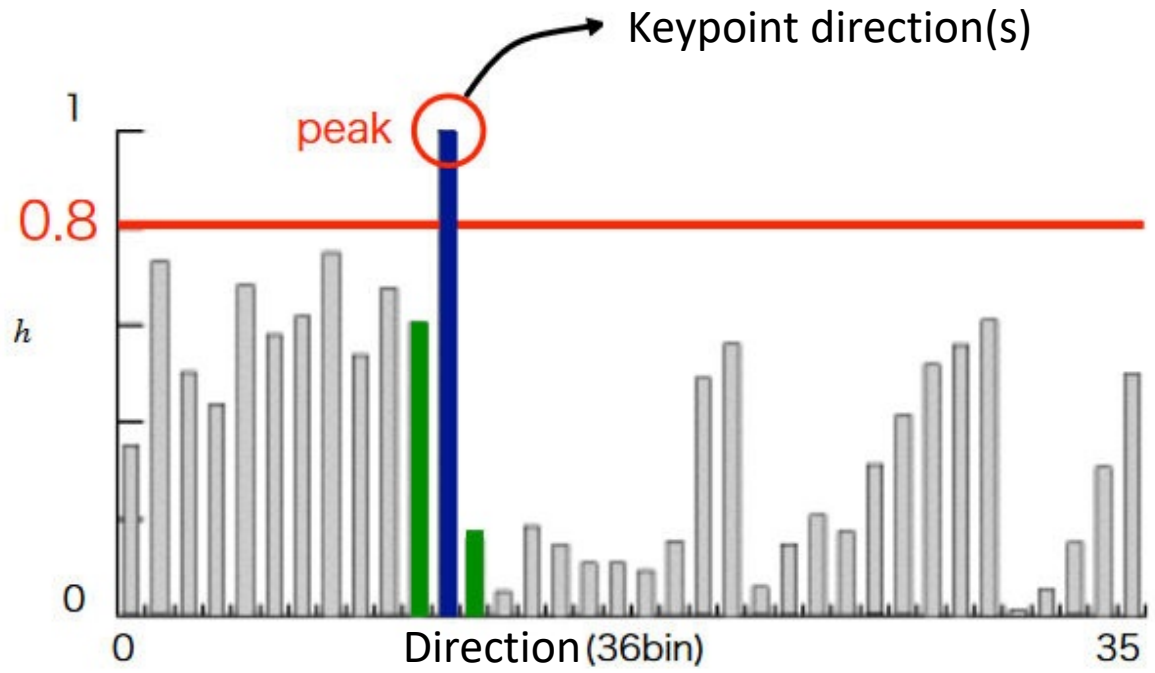
$$\theta(u,v) = \tan^{-1}\left(\frac{L(u,v+1) - L(u,v-1)}{L(u+1,v) - L(u-1,v)}\right)$$

Histogram of gradient direction



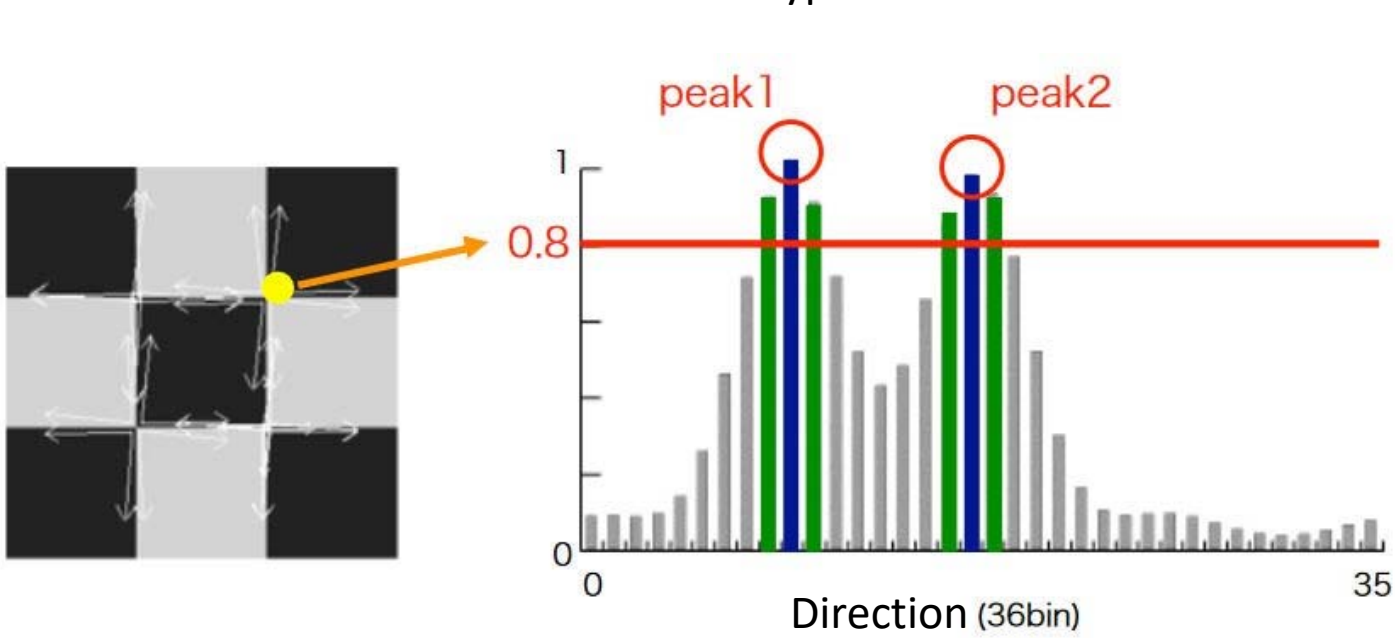
Orientation Assignment

- Over 80% of the maximum



Orientation Assignment

- Over 80% of the maximum



SIFT (Scale-Invariant Feature Transform)

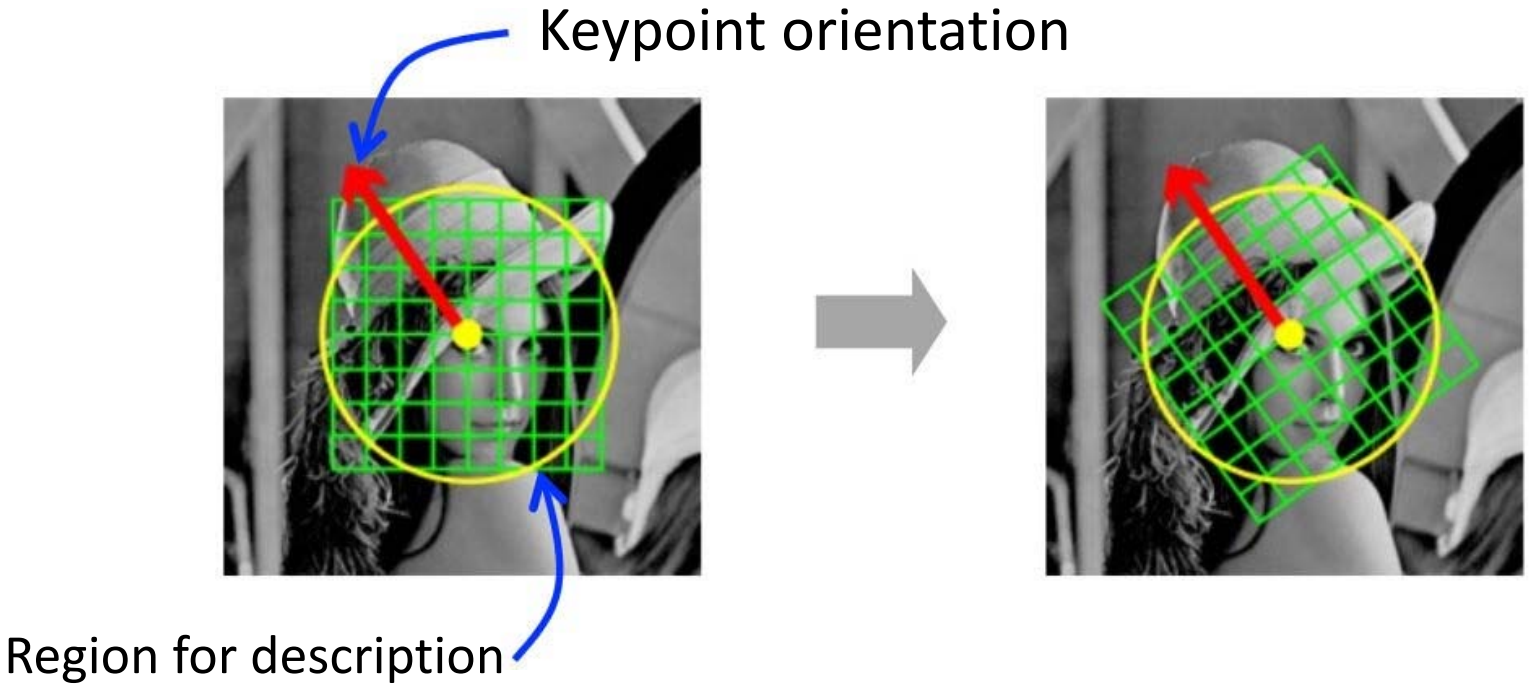
http://www.vision.cs.chubu.ac.jp/cvtutorial/PDF/02SIFTandMore.pdf

D. Lowe (ICCV99, IJCV04)

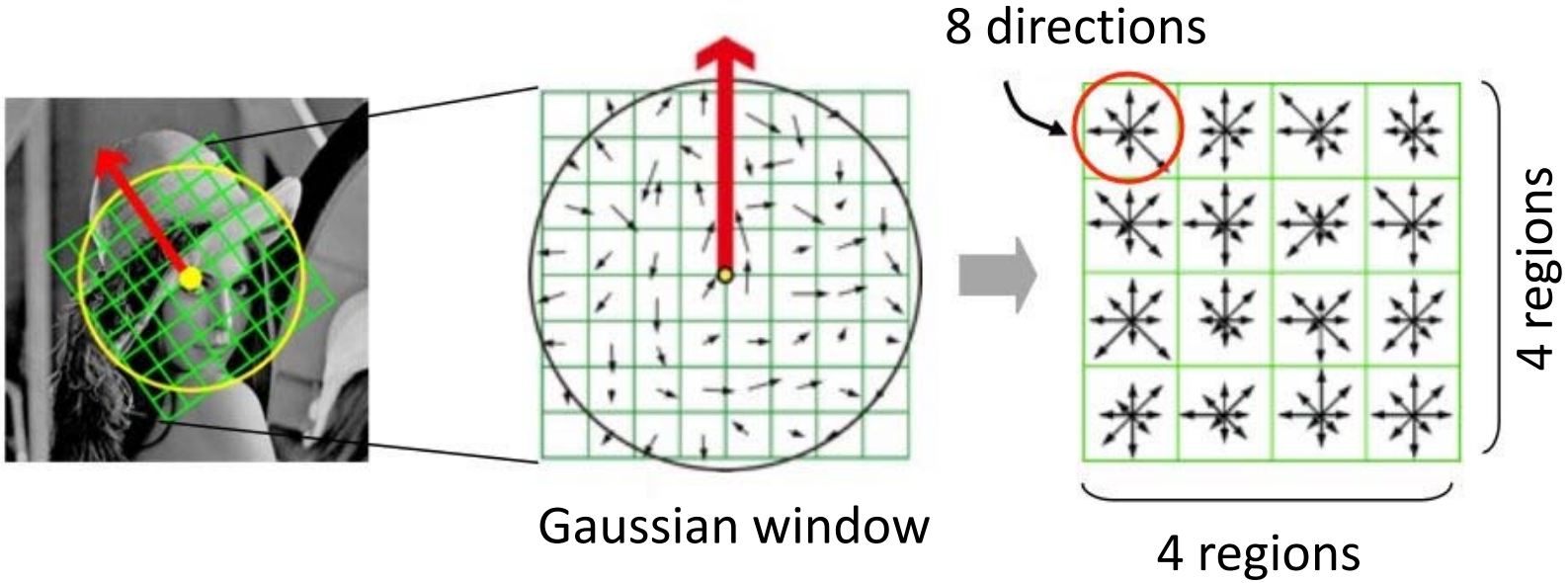
1. Keypoint Detection
   A. Scale-space keypoint detection
   B. Keypoint localization
2. Feature Description
   A. Orientation assignment
   B. Keypoint description

Keypoint description

Rotate by orientation



Keypoint description



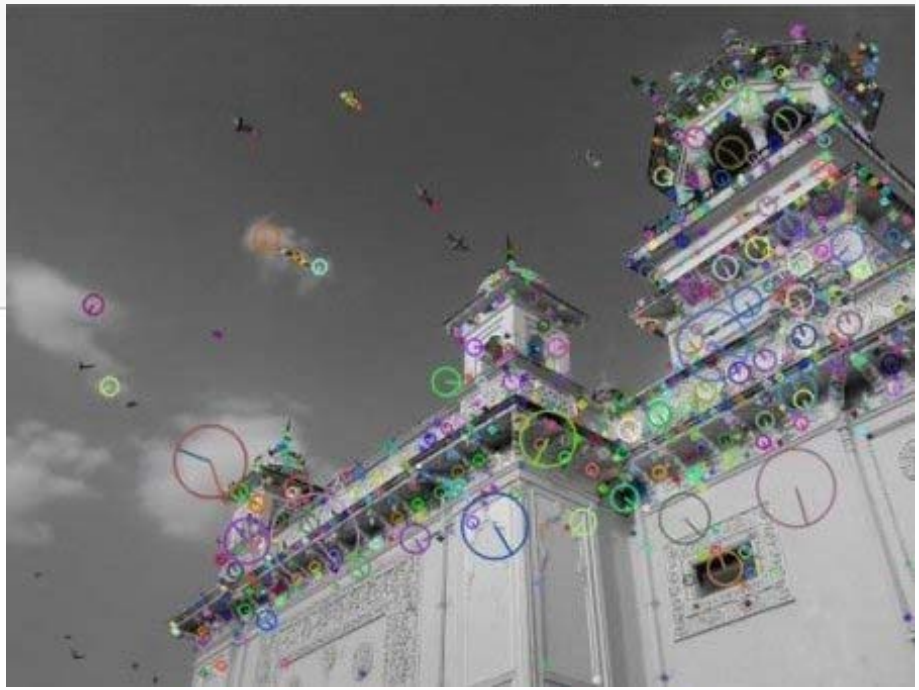
16 blocks around the keypoint → Histogram of 8 directions (45 degs)

→ 16 blocks x 8 directions = 128 dim descriptor

(+ Normalization for illumination robustness)

SIFT in OpenCV

```
import cv2
import numpy as np

img = cv2.imread('home.jpg')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT()
kp = sift.detect(gray,None)

img=cv2.drawKeypoints(gray,kp)

cv2.imwrite('sift_keypoints.jpg',img)
```



Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

Dimensionality Reduction

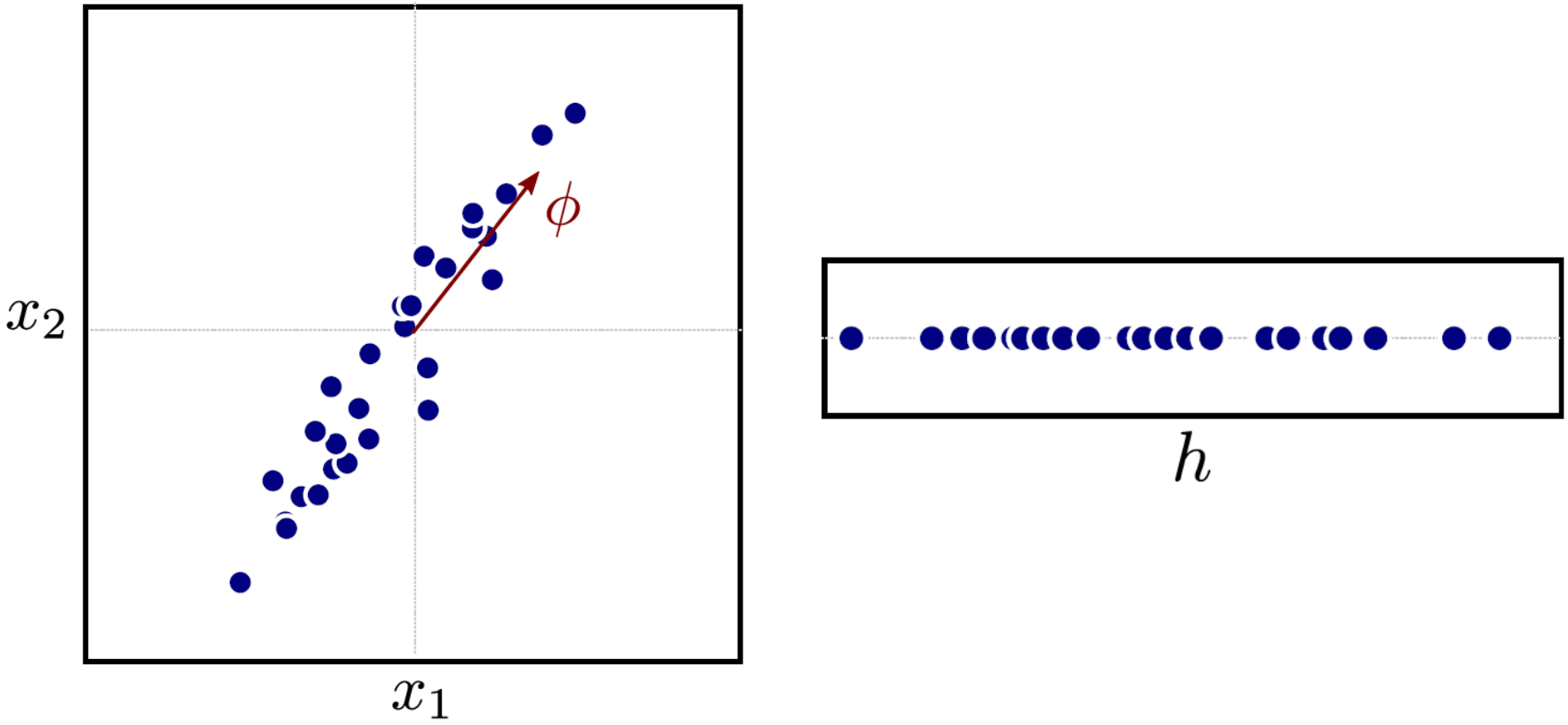
Dimensionality reduction attempt to find a low dimensional (or hidden) representation $\mathbf{h}$ which can approximately explain the data $\mathbf{x}$ so that

$$\mathbf{x} \approx f(\mathbf{h}, \boldsymbol{\theta})$$

where $f[\bullet, \bullet]$ is a function that takes the hidden variable and a set of parameters θ.

Typically, we choose the function family $f[\bullet, \bullet]$ and then learn $\mathbf{h}$ and θ from training data

Computing h



To compute the hidden value, take dot product with the vector ϕ

Least Squares Criterion

$$\hat{\boldsymbol{\theta}}, \hat{\mathbf{h}}_{1\ldots I} = \underset{\boldsymbol{\theta}, \mathbf{h}_{1\ldots I}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \left(\mathbf{x}_i - f\left[\mathbf{h}_i, \boldsymbol{\theta}\right]\right)^T \left(\mathbf{x}_i - f\left[\mathbf{h}_i, \boldsymbol{\theta}\right]\right) \right]$$

Choose the parameters $\boldsymbol{\theta}$ and the hidden variables $\mathbf{h}$ so that they minimize the least squares approximation error (a measure of how well they can reconstruct the data $\mathbf{x}$).

Simple Example

$$\mathbf{x}_i \approx \boldsymbol{\phi} h_i + \boldsymbol{\mu}$$

Approximate each data example $\mathbf{x}$ with a scalar value h.

Data is reconstructed by multiplying h by a parameter ϕ and adding the mean vector μ.

... or even better, lets subtract μ from each data example to get mean-zero data

Simple Example

$$\mathbf{x}_i \approx \boldsymbol{\phi} h_i$$

Approximate each data example $\mathbf{x}$ with a scalar value h.

Data is reconstructed by multiplying h by a factor $\boldsymbol{\phi}$.

Criterion:

$$\hat{\boldsymbol{\phi}}, \hat{h}_{1\ldots I} = \underset{\boldsymbol{\phi}, h_{1\ldots I}}{\operatorname{argmin}} \left[E\right] = \underset{\boldsymbol{\phi}, h_{1\ldots I}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \left(\mathbf{x}_i - \boldsymbol{\phi} h_i\right)^T \left(\mathbf{x}_i - \boldsymbol{\phi} h_i\right)\right]$$

Criterion

$$\hat{\boldsymbol{\phi}}, \hat{h}_{1\ldots I} = \underset{\boldsymbol{\phi}, h_{1\ldots I}}{\operatorname{argmin}} \left[E\right] = \underset{\boldsymbol{\phi}, h_{1\ldots I}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \left(\mathbf{x}_i - \boldsymbol{\phi} h_i\right)^T \left(\mathbf{x}_i - \boldsymbol{\phi} h_i\right)\right]$$

Problem: the problem is non-unique. If we multiply $\mathbf{f}$ by any constant α and divide each of the hidden variables $h_{1\ldots I}$ by the same constant we get the same cost. (i.e. $(\mathbf{f}\alpha)\,(\mathbf{h}_i/\alpha) = \mathbf{f}\mathbf{h}_i$)

Solution: We make the solution unique by constraining the length of $\mathbf{f}$ to be 1 using a Lagrange multiplier.

Criterion

Now we have the new cost function:

$$\begin{aligned} E &= \sum_{i=1}^{I} (\mathbf{x}_i - \boldsymbol{\phi} h_i)^T (\mathbf{x}_i - \boldsymbol{\phi} h_i) + \lambda(\boldsymbol{\phi}^T \boldsymbol{\phi} - 1) \\ &= \sum_{i=1}^{I} \mathbf{x}_i^T \mathbf{x}_i - 2 h_i \boldsymbol{\phi}^T \mathbf{x}_i + h_i^2 + \lambda(\boldsymbol{\phi}^T \boldsymbol{\phi} - 1). \end{aligned}$$

To optimize this we take derivatives with respect to $\boldsymbol{\phi}$ and h_i, equate the resulting expressions to zero and re-arrange.

Solution

$$\hat{h}_i = \hat{\boldsymbol{\phi}}^T \mathbf{x}_i$$

To compute the hidden value, take dot product with the vector $\boldsymbol{\phi}$

Solution

$$\hat{h}_i = \hat{\boldsymbol{\phi}}^T \mathbf{x}_i$$

To compute the hidden value, take dot product with the vector $\boldsymbol{\phi}$

$$\sum_{i=1}^{I} \mathbf{x}_i \mathbf{x}_i^T \hat{\boldsymbol{\phi}} = \lambda \hat{\boldsymbol{\phi}}$$

or

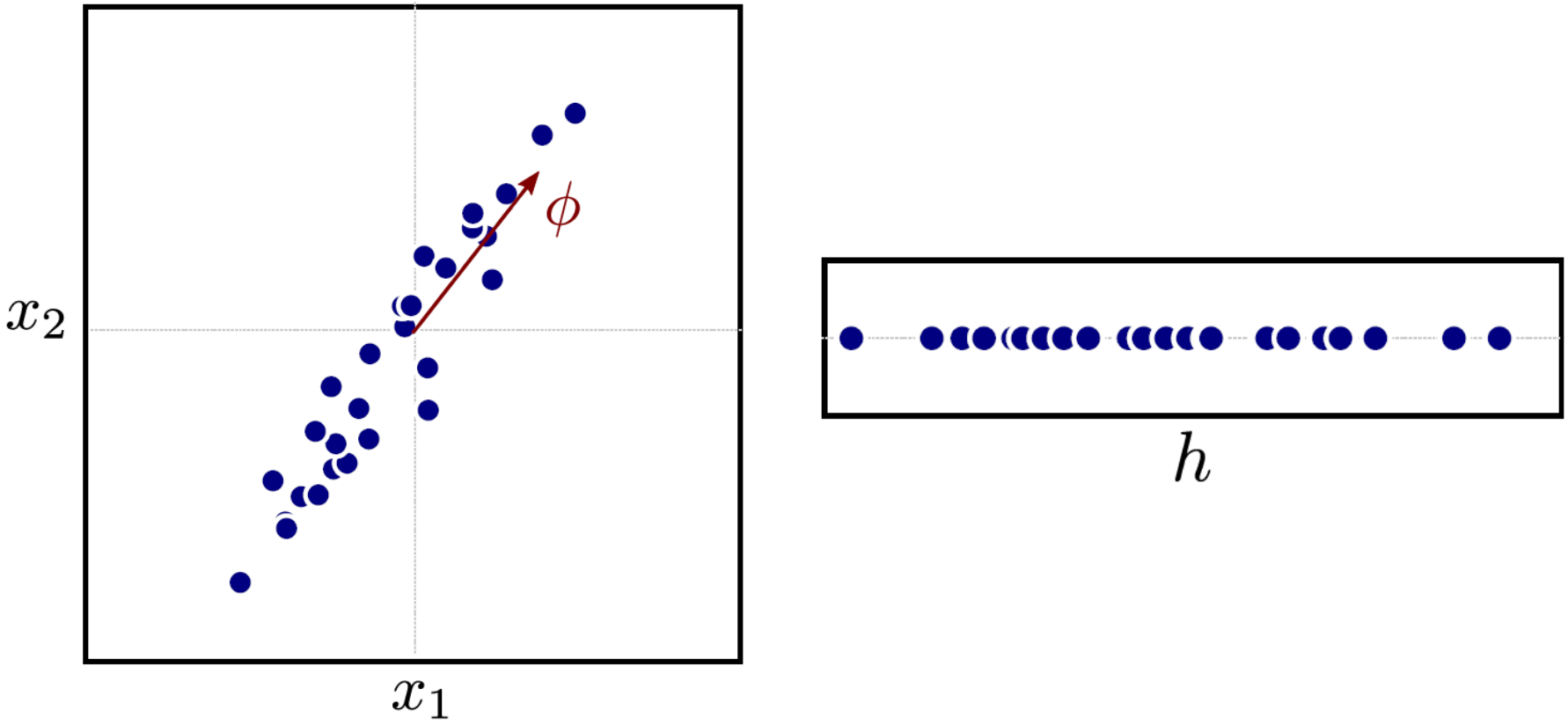
$$\mathbf{X}\mathbf{X}^T \hat{\boldsymbol{\phi}} = \lambda \hat{\boldsymbol{\phi}}$$

where

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_I]$$

To compute the vector $\boldsymbol{\phi}$, compute the first eigenvector of the scatter matrix $\mathbf{X}\mathbf{X}^T$.

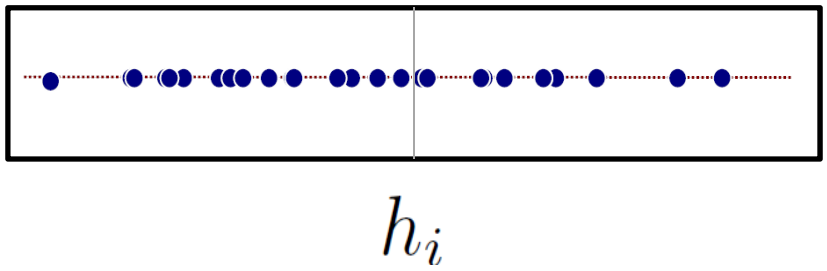
Computing h

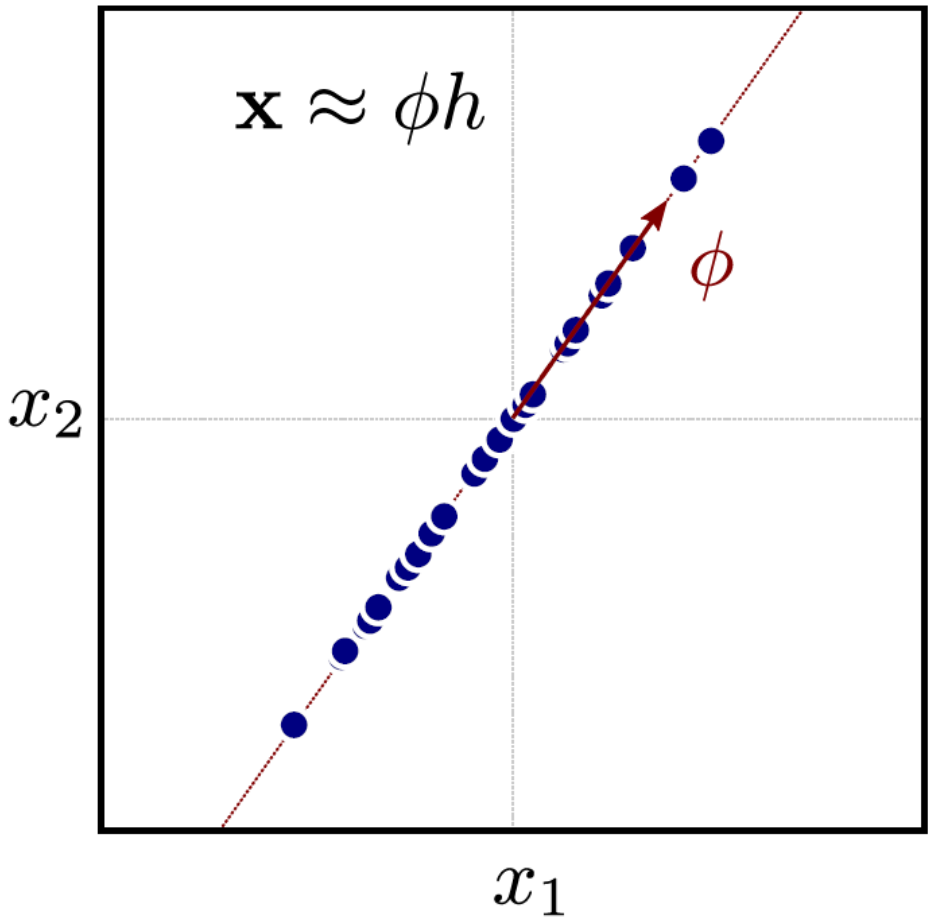


To compute the hidden value, take dot product with the vector ϕ

Reconstruction

$$\mathbf{x}_i \approx \boldsymbol{\phi} h_i$$





To reconstruct, multiply the hidden variable h by vector $\boldsymbol{\phi}$.

Principal Components Analysis

Same idea, but not the hidden variable $\mathbf{h}$ is multi-dimensional. Each components weights one column of matrix $\mathbf{F}$ so that data is approximated as

$$\mathbf{x}_i \approx \boldsymbol{\Phi}\mathbf{h}_i$$

This leads to cost function:

$$\boldsymbol{\Phi}, \hat{\mathbf{h}}_{1\ldots I} = \underset{\boldsymbol{\Phi},\mathbf{h}_{1\ldots I}}{\operatorname{argmin}}\left[E\right] = \underset{\boldsymbol{\Phi},\mathbf{h}_{1\ldots I}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}\left(\mathbf{x}_i - \boldsymbol{\Phi}\mathbf{h}_i\right)^T\left(\mathbf{x}_i - \boldsymbol{\Phi}\mathbf{h}_i\right)\right]$$

This has a non-unique optimum so we enforce the constraint that $\mathbf{F}$ should be a (truncated) rotation matrix and $\mathbf{F}^T\mathbf{F}=\mathbf{I}$

PCA Solution

$$\mathbf{h}_i = \mathbf{\Phi}^T \mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of $\mathbf{\Phi}$.

To compute the matrix $\mathbf{\Phi}$, compute the first D_h eigenvectors of the scatter matrix $\mathbf{X}\mathbf{X}^T$.

The basis functions in the columns of $\mathbf{\Phi}$ are called principal components and the entries of $\mathbf{h}$ are called loadings

Dual PCA

Problem: PCA as described has a major drawback. We need to compute the eigenvectors of the scatter matrix

$$\mathbf{X}\mathbf{X}^T$$

But this has size D_x x D_x. Visual data tends to be very high dimensional, so this may be extremely large.

Solution: Reparameterize the principal components as weighted sums of the data

$$\mathbf{\Phi} = \mathbf{X}\mathbf{\Psi}$$

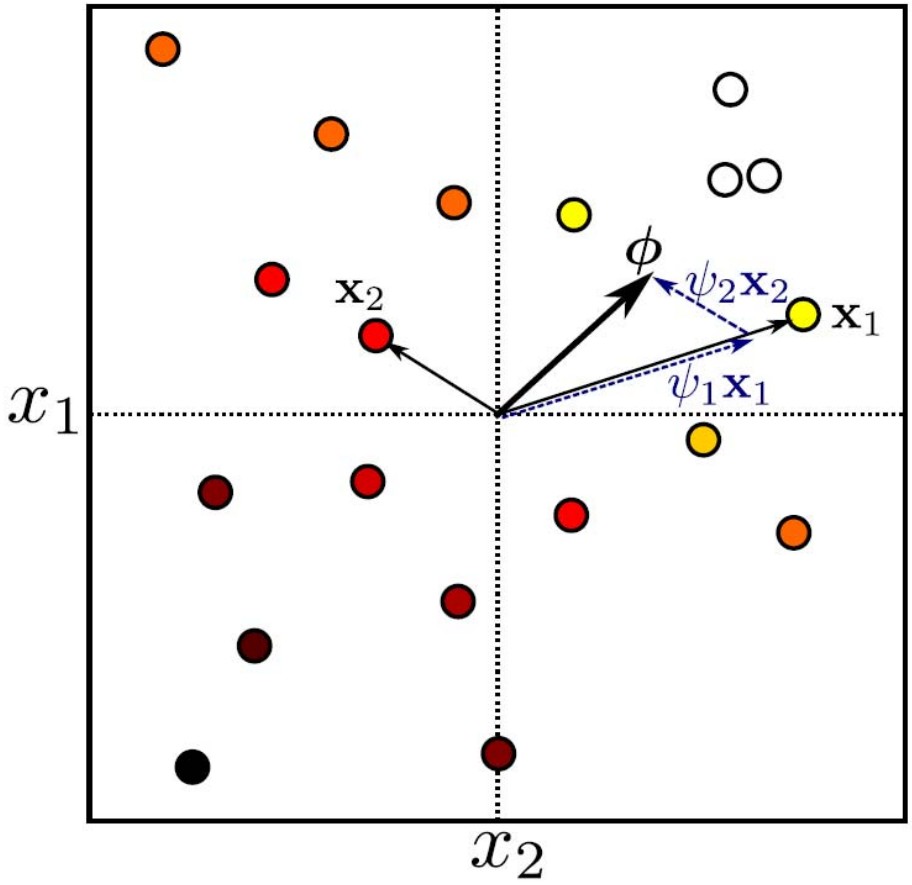
...and solve for the new variables Ψ.

Geometric Interpretation

$$\Phi = \mathbf{X}\Psi$$

Each column of Φ can be described as a weighted sum of the original datapoints.

Weights given in the corresponding columns of the new variable Ψ.



Motivation

Solution: Reparameterize the principal components as weighted sums of the data

$$\mathbf{\Phi} = \mathbf{X}\mathbf{\Psi}$$

...and solve for the new variables $\mathbf{\Psi}$.

Why? If the number of datapoints I is less than the number of observed dimension D_x then the $\mathbf{\Psi}$ will be smaller than $\mathbf{\Phi}$ and the resulting optimization becomes easier.

Intuition: we are not interested in principal components that are not in the subspace spanned by the data anyway.

Cost functions

Principal components analysis

$$\boldsymbol{\Phi}, \hat{\mathbf{h}}_{1\ldots I} = \underset{\boldsymbol{\Phi}, \mathbf{h}_{1\ldots I}}{\operatorname{argmin}} [E] = \underset{\boldsymbol{\Phi}, \mathbf{h}_{1\ldots I}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} (\mathbf{x}_i - \boldsymbol{\Phi}\mathbf{h}_i)^T (\mathbf{x}_i - \boldsymbol{\Phi}\mathbf{h}_i) \right]$$

...subject to $\boldsymbol{\Phi}^T\boldsymbol{\Phi}=\mathbf{I}$.

Dual principal components analysis

$$E = \sum_{i=1}^{I} (\mathbf{x}_i - \mathbf{X}\boldsymbol{\Psi}\mathbf{h}_i)^T (\mathbf{x}_i - \mathbf{X}\boldsymbol{\Psi}\mathbf{h}_i)$$

...subject to $\boldsymbol{\Phi}^T\boldsymbol{\Phi}=\mathbf{I}$ or $\boldsymbol{\Psi}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\Psi}=\mathbf{I}$.

Solution

$$\mathbf{h}_i = \boldsymbol{\Psi}^T\mathbf{X}^T\mathbf{x}_i = \boldsymbol{\Phi}^T\mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of $\boldsymbol{\Phi}=\boldsymbol{\Psi}\mathbf{X}$.

Solution

$$\mathbf{h}_i = \boldsymbol{\Psi}^T \mathbf{X}^T \mathbf{x}_i = \boldsymbol{\Phi}^T \mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of $\boldsymbol{\Phi}=\boldsymbol{\Psi}\mathbf{X}$.

To compute the matrix $\boldsymbol{\Psi}$, compute the first D_h eigenvectors of the inner product matrix $\mathbf{X}^T\mathbf{X}$.

The inner product matrix has size I x I.

If the number of examples I is less than the dimensionality of the data D_x then this is a smaller eigenproblem.

K-Means algorithm

Approximate data with a set of means

$$\mathbf{x}_i \approx \boldsymbol{\mu}_{h_i}$$

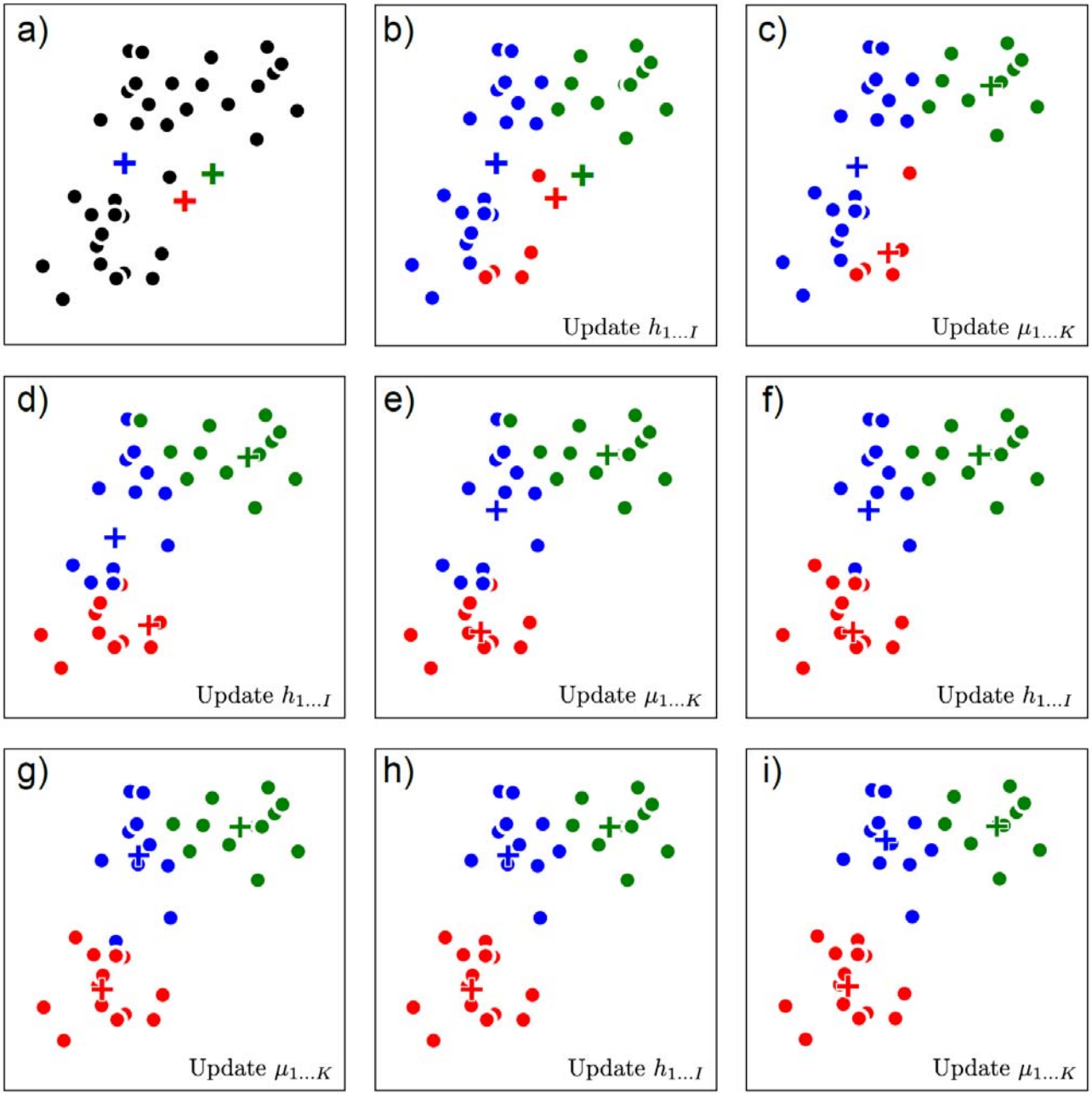
Least squares criterion

$$\hat{\boldsymbol{\mu}}_{1\ldots K}, \hat{h}_{1\ldots I} = \underset{\boldsymbol{\mu},h}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right)^T \left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right) \right]$$

Alternate minimization

$$\hat{h}_i = \underset{h_i}{\operatorname{argmin}} \left[\left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right)^T \left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right) \right]$$

$$\begin{aligned} \hat{\boldsymbol{\mu}}_k &= \underset{\boldsymbol{\mu}_k}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \left[\left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right)^T \left(\mathbf{x}_i - \boldsymbol{\mu}_{h_i}\right) \right] \right] \\ &= \frac{\sum_{i=1}^{I} \mathbf{x}_i \delta[h_i - k]}{\sum_{i=1}^{I} \delta[h_i - k]}, \end{aligned}$$


a)
b)
Update $h_{1...I}$
c)
Update $\mu_{1...K}$
d)
Update $h_{1...I}$
e)
Update $\mu_{1...K}$
f)
Update $h_{1...I}$
g)
Update $\mu_{1...K}$
h)
Update $h_{1...I}$
i)
Update $\mu_{1...K}$