FITRI UTAMINIGRUM

LOCAL IMAGE FEATURE

Pixel Neighborhood-based Feature

- The most important for texture analysis is to describe the spatial behavior of intensity values in any given neighborhood.
- Local binary pattern (LBP) is one of the most-widely used approach – mainly for face recognition.
- LBP is used for texture analysis too.

Local Binary Pattern (LBP)

For each PIXEL of an image, a BINARY CODE is produced
→ to make a new matrix with the new value (binary to decimal value).

$$LBP_{p,r}(N_c) = \sum_{p=0}^{P-1} g\left(N_p - N_c\right)2^p$$

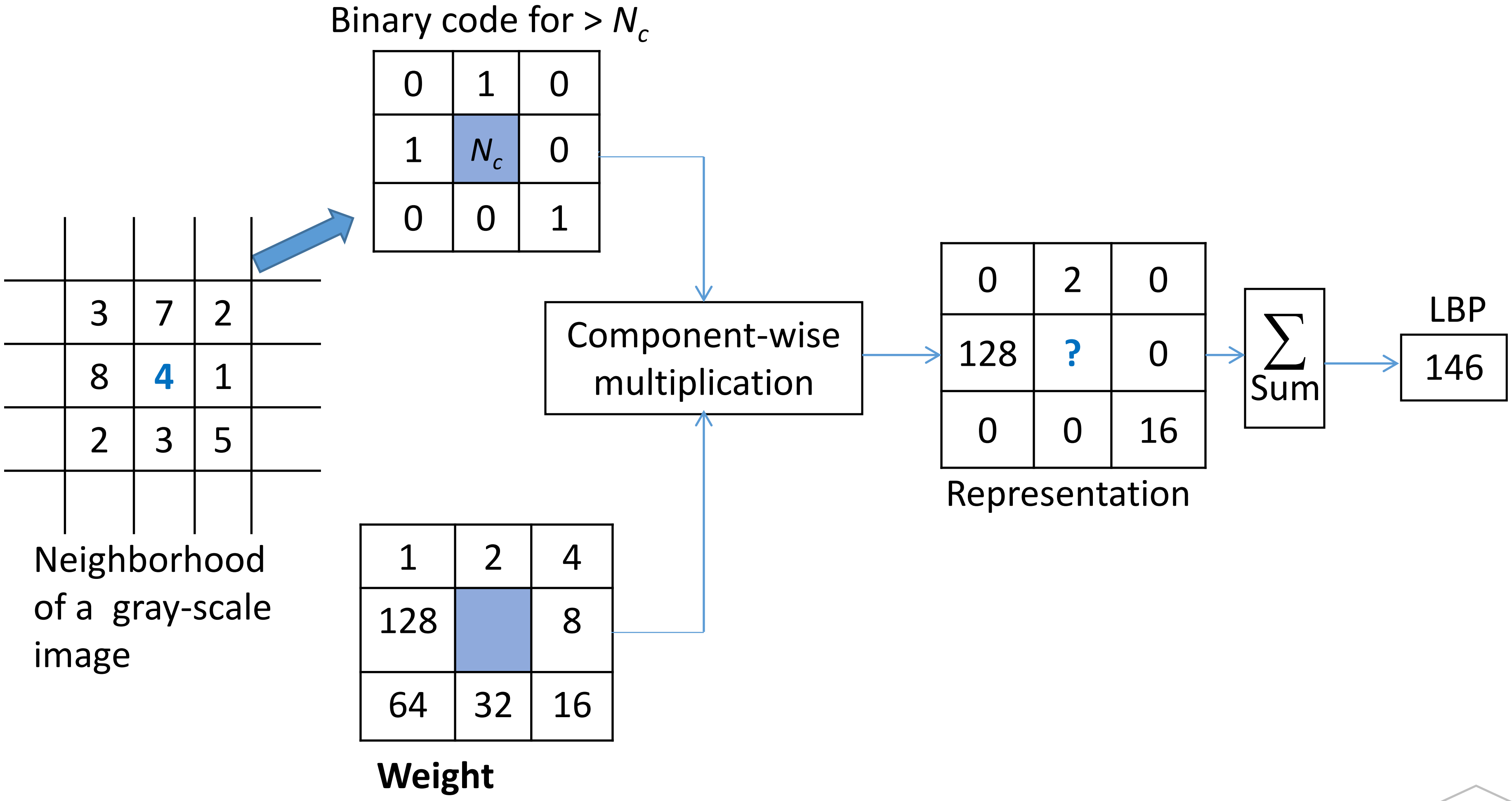
where,
neighborhood pixels (N_p) in each block →
is thresholded by its center pixel value (N_c)
p→ sampling points (e.g., $p = 0, 1, \ldots, 7$ for a 3x3 cell, where P = 8)
r → radius (for 3x3 cell, it is 1).

Binary threshold function $g(x)$ is,

$$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Slides from Dr. Shahera Hossain

Computation of Local Binary Pattern



Slides from Dr. Shahera Hossain

Computation of LBP

6	5	2
7	6	1
9	8	7

Example

(LSB) 1	0	0
1		0
1	1	1

Threshold

1	2	4
128		8
64	32	16

Weights

Binary Pattern:	1 (MSB)	1	1	1	0	0	0	1 (LSB)

Code/Weight (2^p)**:**	1 x 2^7	1 x 2^6	1 x 2^5	1 x 2^4	0 x 2^3	0x 2^2	0x 2^1	1 x 2^0
	= 128	= 64	= 32	= 16	= 0	= 0	= 0	= 1

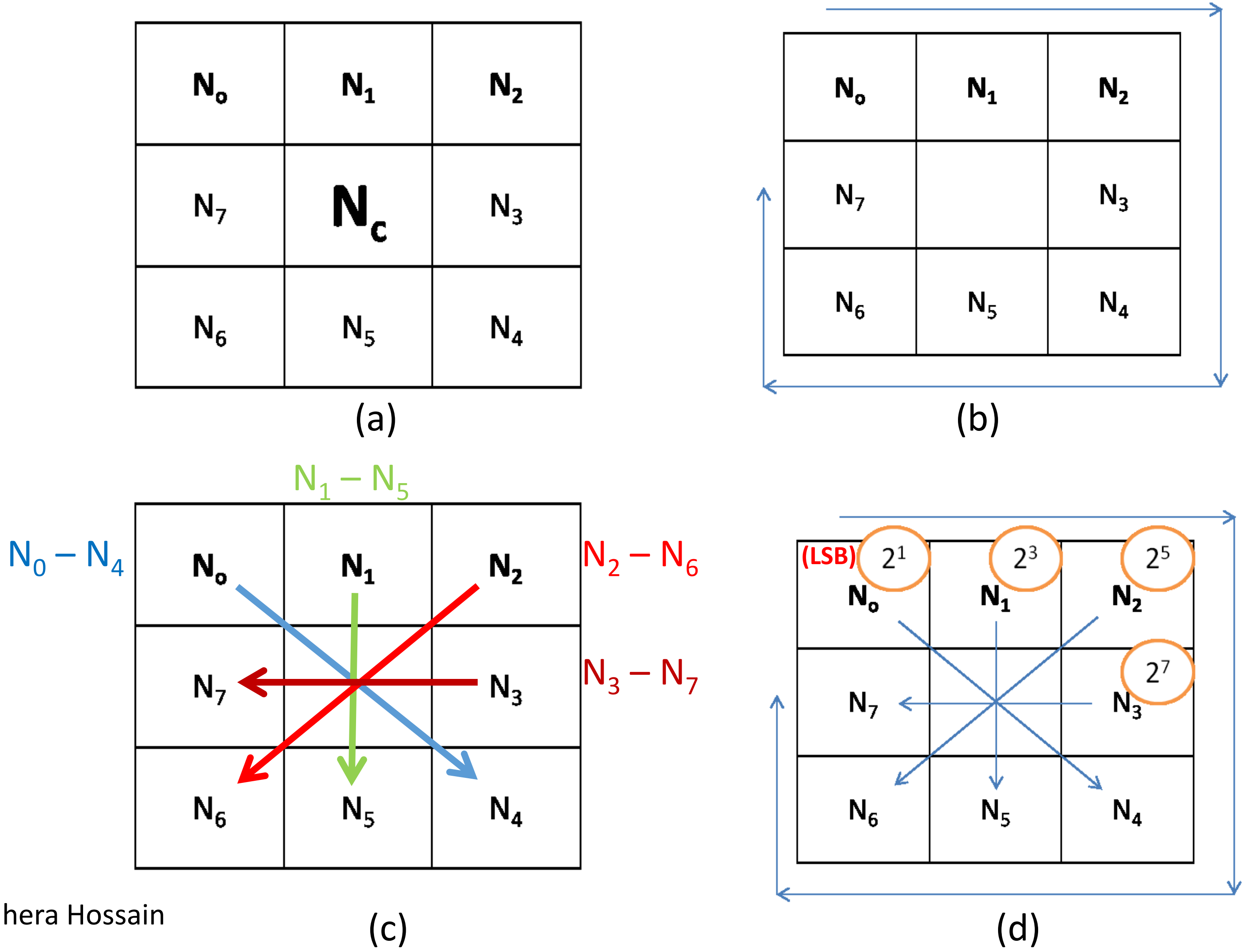
LBP:	**1 + 0 + 0 + 0 + 16 + 32 + 64 + 128 = 241**

Slides from Dr. Shahera Hossain

A New Method: DCLBP

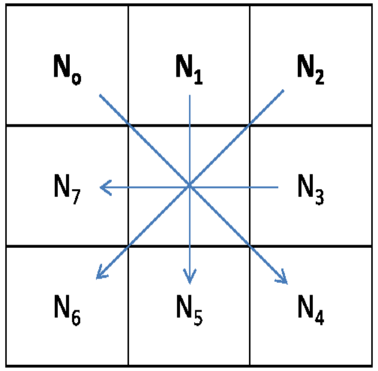
- A new method called *diagonal-crisscross local binary pattern* (DCLBP) for texture representation is proposed recently.
- Basic concept: An image feature should take *diagonal* pixel variations as well as *horizontal* and *vertical* (crisscross) pixel variations in the neighborhood

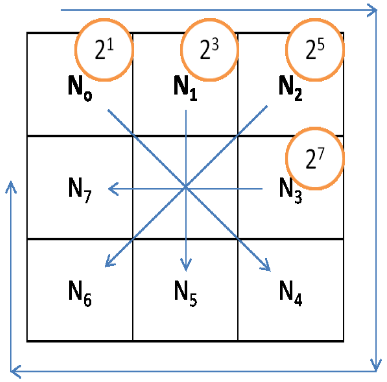
Diagonal-Crisscross Local Binary Pattern (DCLBP)



Slides from Dr. Shahera Hossain

Diagonal-Crisscross Local Binary Pattern (DCLBP)





a) 3x3 image patch

b) Start from N_0 pixel position

c) Get differences for **front-diagonal values** $(N_0 - N_4)$, in **vertical** direction $(N_1 - N_5)$, for **back-diagonal** direction $(N_2 - N_6)$, and in **horizontal** direction $(N_3 - N_7)$

d) Multiply each difference with 2^1, 2^3, 2^5 and 2^7 sequentially to *compute a new value*

e) Take the *mean* value of *newly-computed value* and the *central pixel value*

Slides from Dr. Shahera Hossain

Diagonal-Crisscross Local Binary Pattern (DCLBP)

$$DCLBP_{p,r}(N_c) = \frac{\left[\left(\sum_{k=0}^{|P|-1} \vartheta\left(\delta_{k,|P|+k}\right) \times 2^{p_k \in P}\right) + N_c\right]}{2}$$

$p_0 = 2^1$
$p_1 = 2^3$
$p_2 = 2^5$
$p_3 = 2^7$

where,

Sampling-point set is, $P = \{1, 3, 5, 7\}$

Cardinality of the set is, $|P| = 4$

Difference parameter δ is,

$$\delta_{k,|P|+k} = \left(N_k - N_{|P|+k}\right)$$

The binary threshold function $\vartheta(\delta)$ is,

$$\vartheta(\delta) = \begin{cases} 0, & \delta < 0 \\ 1, & \delta \geq 0 \end{cases}$$

Slides from Dr. Shahera Hossain

Diagonal-Crisscross Local Binary Pattern (DCLBP)

Here, the cardinality, $|P| = 4$ that demonstrates that it has 4 different-possible values as '2^1, 2^3, 2^5 and 2^7'.

when $k = 0,$ we get

$\delta_{0,4+0} \times 2^1 = \delta_{0,4} \times 2^1$

which, covers the front-*diagonal* difference of $(N_0 - N_4)$.

Similarly,

when $k = 1, \delta_{1,5} \times 2^3$

covers the *vertical* difference of $(N_1 - N_5)$;

when $k = 2, \delta_{2,6} \times 2^5$

covers the back-*diagonal* difference of $(N_2 - N_6)$;

when $k = 3, \delta_{3,7} \times 2^7$

covers the *horizontal* difference of $(N_3 - N_7)$.

Through this manner, we get the new central pixel value for each patch.

Diagonal-Crisscross Local Binary Pattern (DCLBP)

6	5	2
7	6	1
9	8	7

Example

-1	-3	-7
	6	-6

$$\delta_{k,|P|+k} = (N_k - N_{|P|+k})$$

0	0	0
	6	0

$$\vartheta(\delta) = \begin{cases} 0, & \delta < 0 \\ 1, & \delta \geq 0 \end{cases}$$

	3	

$$DCLBP_{p,r}(N_c) = \frac{\left[\left(\sum_{k=0}^{|P|-1} \vartheta\left(\delta_{k,|P|+k}\right) \times 2^{p_k \in P}\right) + N_c\right]}{2}$$

$$= \frac{0 \times 2 + 0 \times 8 + 0 \times 32 + 0 \times 128 + 6}{2}$$

$$= 3$$

Median-RILBP

$$\widetilde{LBP^{\propto}_{p,r}}(N_p) = median\{\widetilde{\rho(LBP_{p,r}(N_p), i)}\}|_{p=0,1,\ldots,P-1}$$

where,

- $\propto$ symbolizes ‘rotational-invariant’ nature;
- p is pattern (e.g., 8 for a 3x3 patch);
- r is radius (1 for 3x3 patch);
- Rho ($\rho(\quad)$) is the rotational function where it circularly does bitwise right-shift operation.
- i is 8 when if $P = 8$. so 8 times rotation
- The bit-shift is done 8 times if $P = 8$. The concept here is to rotate the Pneighbors.
- Finally, compute the $\widetilde{median}$ value that the neighbor chain may represent.

Median-Rotation Invariant LBP

3	7	2
8	**4**	1
2	3	5

0	1	0
1	N_c	0
0	0	1

10010010

- For example, if a usual computation provides a binary pattern 10010010, then by the bit-wise right-shift operation, we get 01001001, 10100100, 01010010, 00101001, 10010100, 01001010, 00100101.
- From these 8 binary patterns, we take the median of them.
- The concept here is to rotate the neighbors and compute the value that the neighbor chain may represent.

Mean-RILBP

$$\overline{LBP_{p,r}^{\propto}(N_p)} = \overline{mean\{\rho(LBP_{p,r}(N_p), i)\}|_{p=0,1,\ldots,P-1}}$$

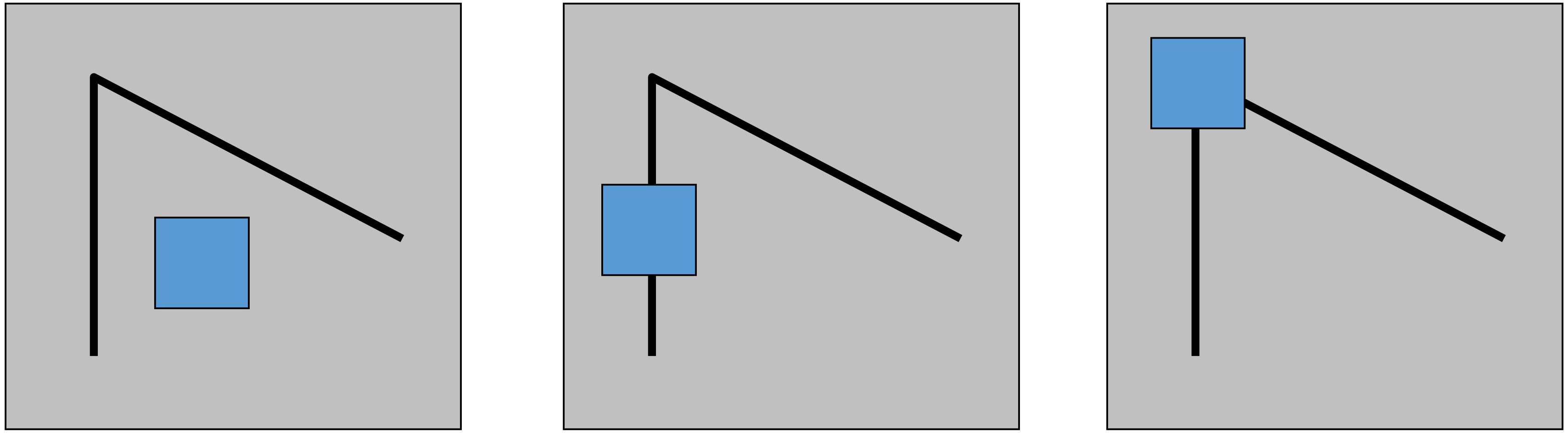
where,

- $\propto$ symbolizes 'rotational-invariant' nature;
- p is pattern (e.g., 8 for a 3x3 patch);
- r is radius (1 for 3x3 patch);
- Rho ($\rho(\quad)$) is the rotational function where it circularly does bitwise right-shift operation.
- i is 8 when if $P = 8$. so 8 times rotation
- The bit-shift is done 8 times if $P = 8$. The concept here is to rotate the P neighbors.
- Finally, compute the MEAN value.

Local measures of uniqueness

Suppose we only consider a small window of pixels

- What defines whether a feature is a good or bad candidate?

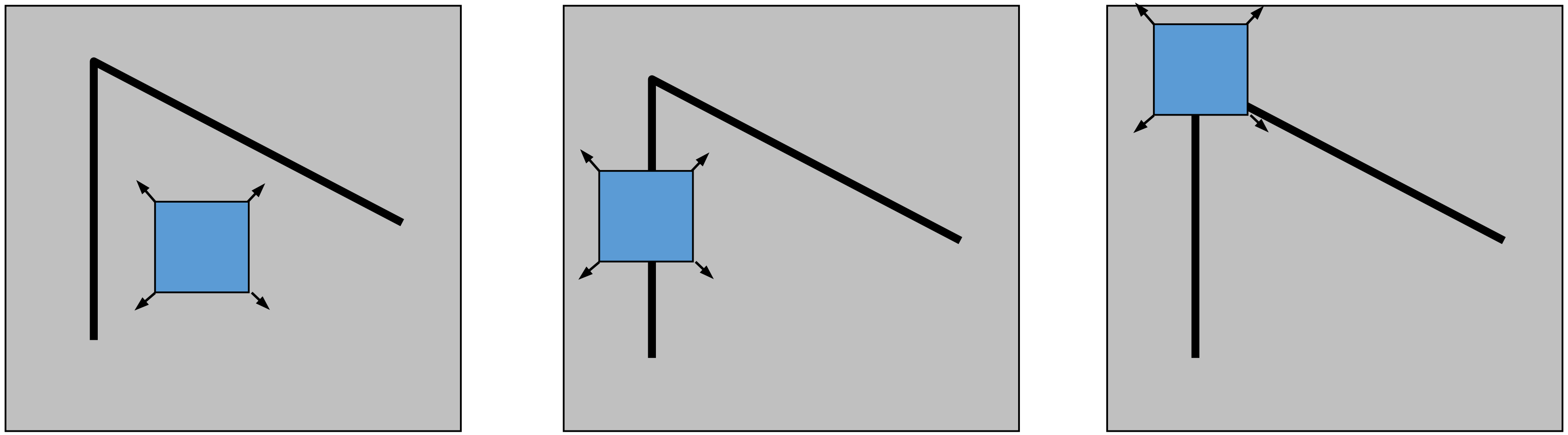


Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Feature detection

Local measure of feature uniqueness

- How does the window change when you shift it?
- Shifting the window in *any direction* causes a *big change*



"flat" region:
no change in all directions

"edge":
no change along the edge direction

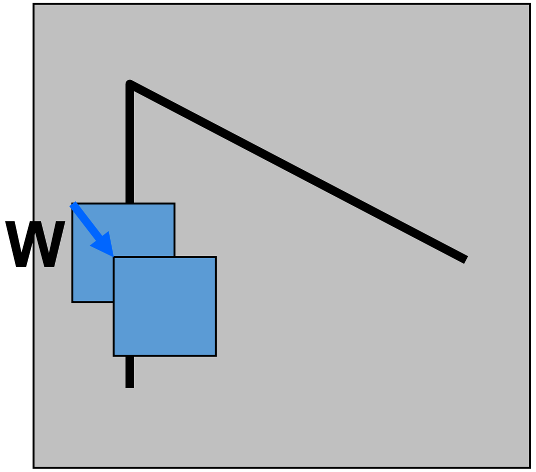
"corner":
significant change in all directions

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

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 by Summing up the Squared Differences (SSD)
- this defines an SSD "error" of *E(u,v)*:



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

Small motion assumption

Taylor Series expansion of I:

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

If the motion (u,v) is small, we can do a first-order Tylor.

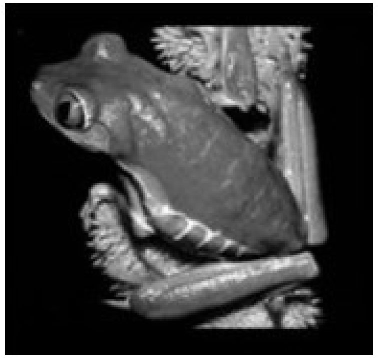
$$\begin{aligned} I(x+u, y+v) &\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x,y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

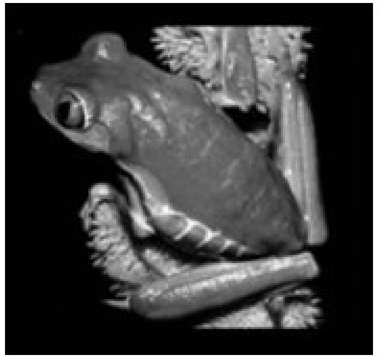
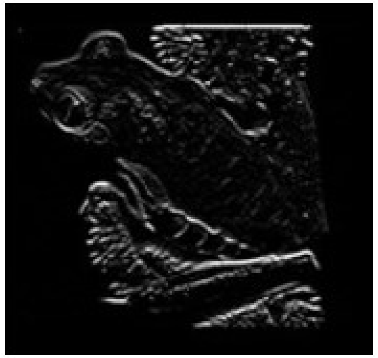
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Feature detection

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

$I_x =$  $* \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} =$ 

$I_y =$  $* \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} =$ 

Feature detection: the math

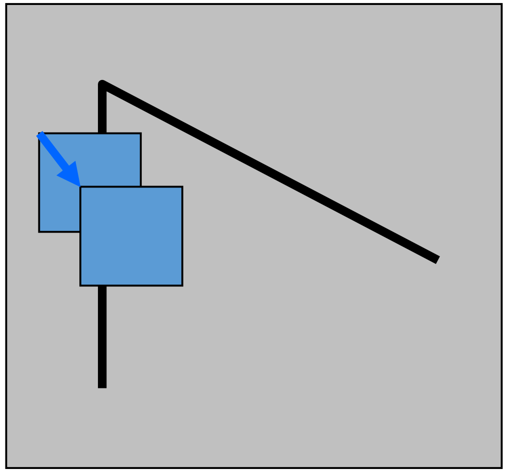
Consider shifting the window W by (u,v)

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

W

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

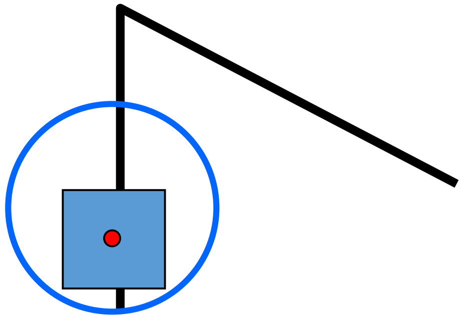
$$\approx \sum_{(x,y)\in W} \left[[I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix}\right]^2$$

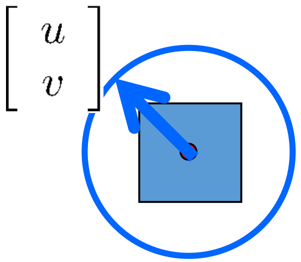


Feature detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$





For the example above

- You can move the center of the blue window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of ***H***

Quick eigenvalue/eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

- The eigenvalues are found by solving: $det(A - \lambda I) = 0$
- In our case, ***A*** = ***H*** is a 2x2 matrix, so we have $det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$
- The solution: $\lambda_\pm = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$

Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Feature detection

Example 1 The matrix A has two eigenvalues $\lambda = 1$ and $\lambda = 1/2$. Look at $\det(A - \lambda I)$:

$$A = \begin{bmatrix} .8 & .3 \\ .2 & .7 \end{bmatrix} \quad \det \begin{bmatrix} .8-\lambda & .3 \\ .2 & .7-\lambda \end{bmatrix} = \lambda^2 - \frac{3}{2}\lambda + \frac{1}{2} = (\lambda - 1)\left(\lambda - \frac{1}{2}\right).$$

I factored the quadratic into $\lambda - 1$ times $\lambda - \frac{1}{2}$, to see the two eigenvalues $\boldsymbol{\lambda} = \mathbf{1}$ and $\boldsymbol{\lambda} = \frac{1}{2}$. For those numbers, the matrix $A - \lambda I$ becomes *singular* (zero determinant). The eigenvectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are in the nullspaces of $A - I$ and $A - \frac{1}{2}I$.

$(A - I)\boldsymbol{x}_1 = 0$ is $A\boldsymbol{x}_1 = \boldsymbol{x}_1$ and the first eigenvector is $(.6, .4)$.

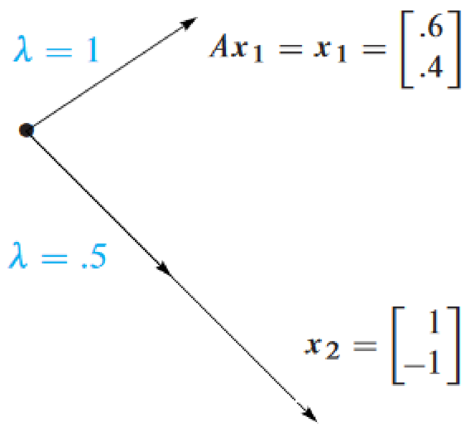
$(A - \frac{1}{2}I)\boldsymbol{x}_2 = 0$ is $A\boldsymbol{x}_2 = \frac{1}{2}\boldsymbol{x}_2$ and the second eigenvector is $(1, -1)$:

$$\boldsymbol{x}_1 = \begin{bmatrix} .6 \\ .4 \end{bmatrix} \qquad \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} .8 & .3 \\ .2 & .7 \end{bmatrix} \begin{bmatrix} x11 \\ x12 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

0.8 x11 + 0.3 x12 = 1

0.2 x11 + 0.7 x12 = 1



Feature detection

$$\begin{vmatrix} 0{,}8-\lambda & 0{,}3 \\ 0{,}2 & 0{,}7-\lambda \end{vmatrix} \Rightarrow (0{,}8-\lambda)(0{,}7-\lambda) - (0{,}2 \times 0{,}3)$$

$$\Rightarrow [0{,}56 - 0{,}8\lambda - 0{,}7\lambda + \lambda^2] - 0{,}06$$

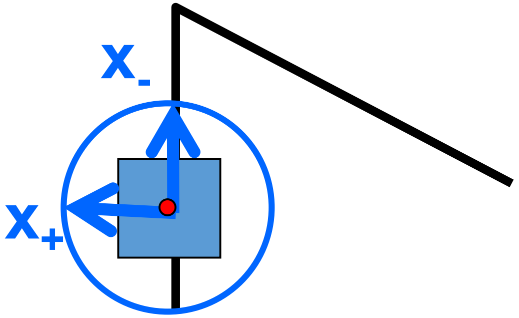
$$\Rightarrow \lambda^2 - 1{,}5\lambda + 0{,}5$$

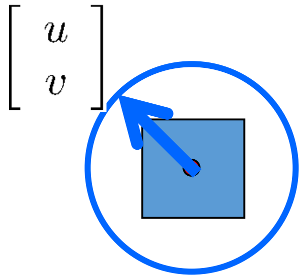
$$\Rightarrow \lambda^2 - \frac{3}{2}\lambda + \frac{1}{2}$$

Feature detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u \ v] \underbrace{\begin{bmatrix} I_x^2 & I_xI_y \\ I_yI_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$





Eigenvalues and eigenvectors of H

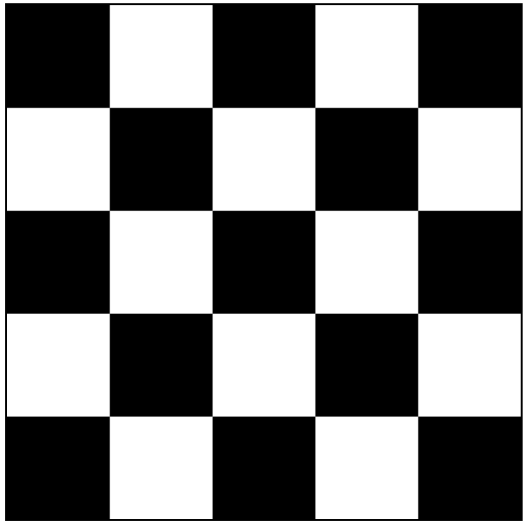
- Define shifts with the smallest and largest change (E value)
- x_+ = direction of **largest** increase in E.
- λ_+ = amount of increase in direction x_+
- x_- = direction of **smallest** increase in E.
- λ- = amount of increase in direction x_+

$$\begin{aligned} Hx_+ &= \lambda_+ x_+ \\ Hx_- &= \lambda_- x_- \end{aligned}$$

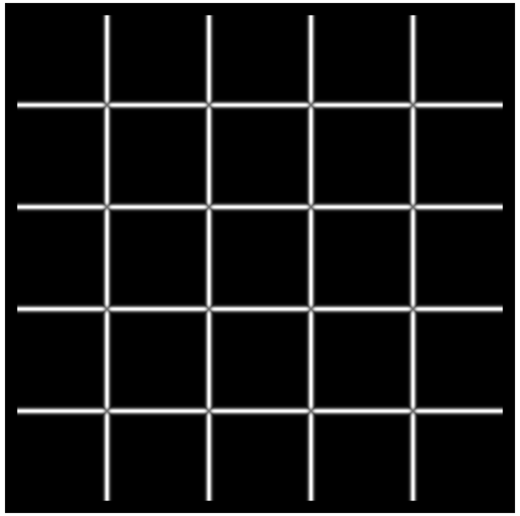
Feature detection: the math

Want *E(u,v)* to be ***large*** for small shifts in ***all*** directions

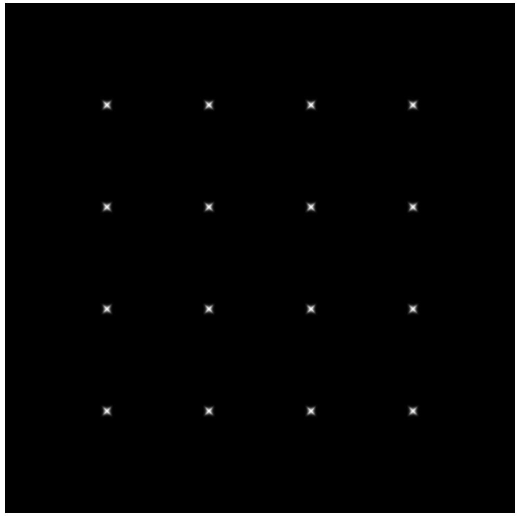
- the *minimum* of *E(u,v)* should be large, over all unit vectors [u v]
- this minimum is given by the smaller eigenvalue (λ_-) of ***H***



I



λ_+

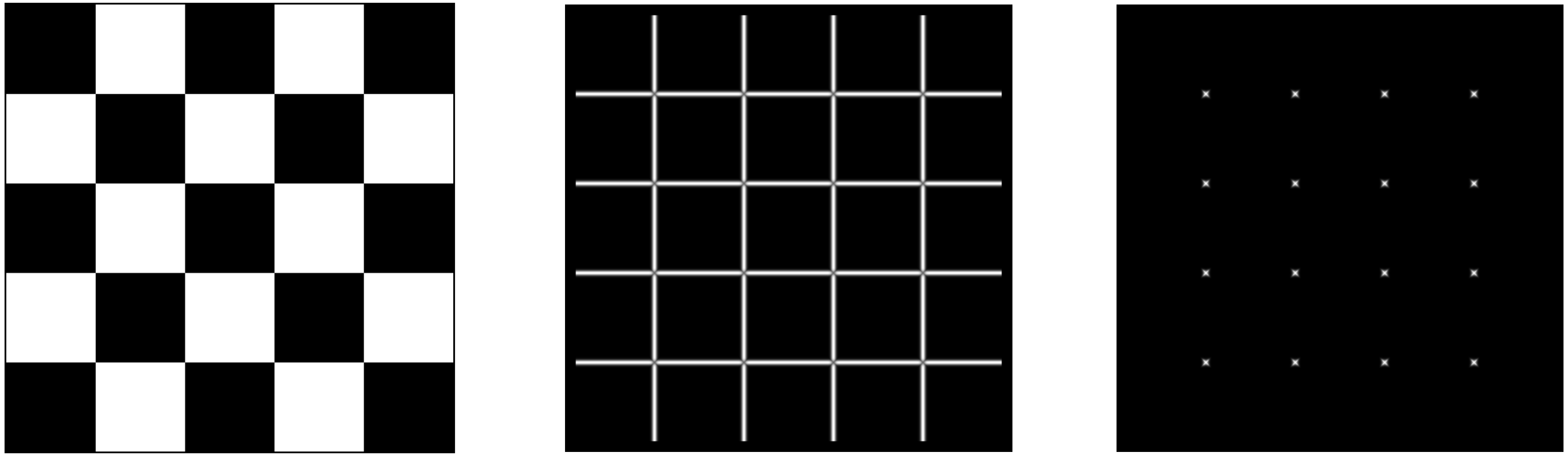


λ_-

Feature detection summary

Here's what you do

- Compute the gradient at each point in the image
- Create the ***H*** matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_- > threshold)
- Choose those points where λ_- is a local maximum as features



I $\qquad$ λ_+ $\qquad$ λ_-

Harris Detector: Mathematics

Window-averaged squared change of intensity induced by shifting the image data by $[u,v]$:

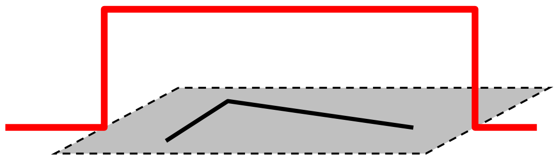
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

Shifted intensity

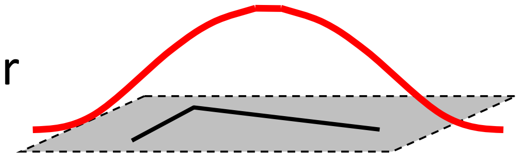
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Harris Detector: Mathematics

Expanding I(x,y) in a Taylor series expansion, we have, for small shifts [u,v], a quadratic approximation to the error surface between a patch and itself, shifted by [u,v]:

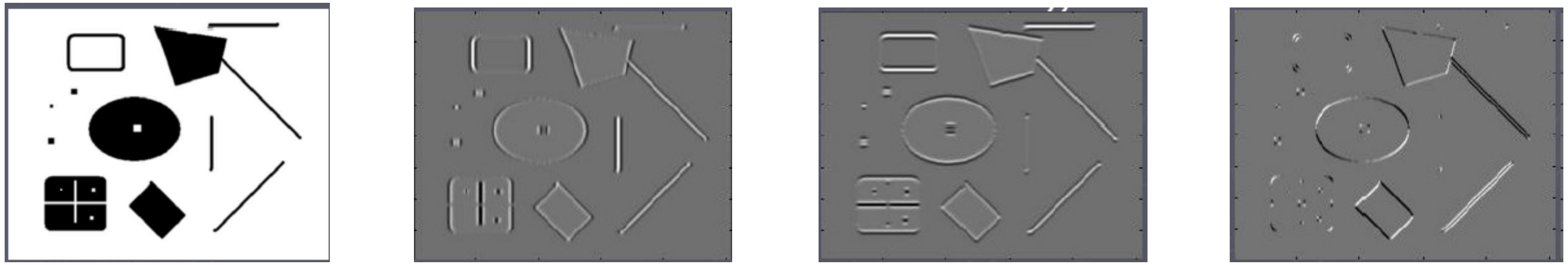
$$E(u,v) \cong [u,v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix computed from image derivatives:

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

Harris Detector: Mathematics

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



Notation: $I_x \Leftrightarrow \frac{\partial I}{\partial x}$ $I_y \Leftrightarrow \frac{\partial I}{\partial y}$ $I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$

K. Grauman

What does this matrix reveal?

First, consider an axis-aligned corner:

$$M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means dominant gradient directions align with x or y axis

Look for locations where **both** λ's are large.

If either λ is close to 0, then this is **not** corner-like.

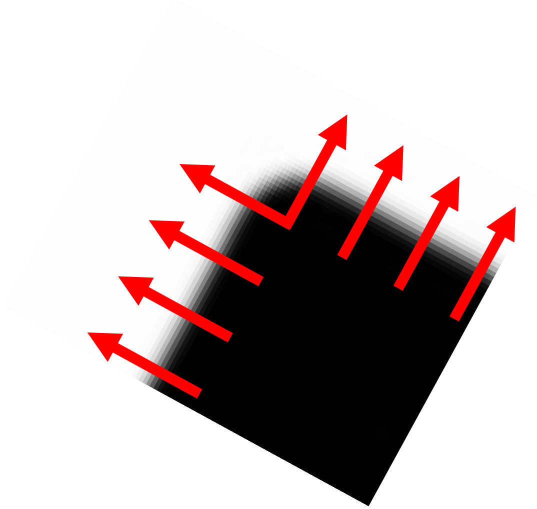
What if we have a corner that is not aligned with the image axes?

K. Grauman

What does this matrix reveal?

Since *M* is symmetric, we have

$$M = X \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} X^T$$

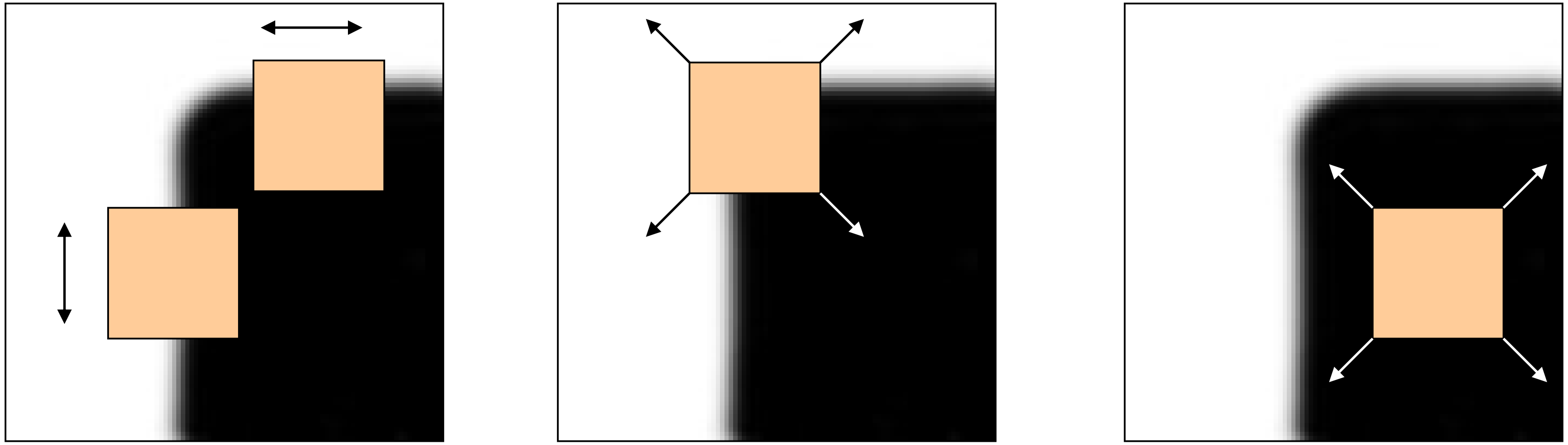


$$Mx_i = \lambda_i x_i$$

The *eigenvalues* of *M* reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.

K. Grauman

Corner response function



"edge":

$\lambda_1 >> \lambda_2$

$\lambda_2 >> \lambda_1$

"corner":

λ_1 and λ_2 are large,

$\lambda_1 \sim \lambda_2$

"flat" region:

λ_1 and λ_2 are small

Adapted from A. Efros, D. Frolova, D. Simakov, K. Grauman

Harris Detector: Mathematics

Measure of corner response:

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

$$\det M = \lambda_1 \lambda_2$$

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

(k – empirical constant, k = 0.04-0.06)

D. Frolova, D. Simakov

Harris Detector: Summary

- Compute image gradients I_x and I_y for all pixels
- For each pixel
  - Compute $M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$

    by looping over neighbors x, y
  - compute $R = \det M - k\left(\operatorname{trace} M\right)^2$

(k :empirical constant, k = 0.04-0.06)

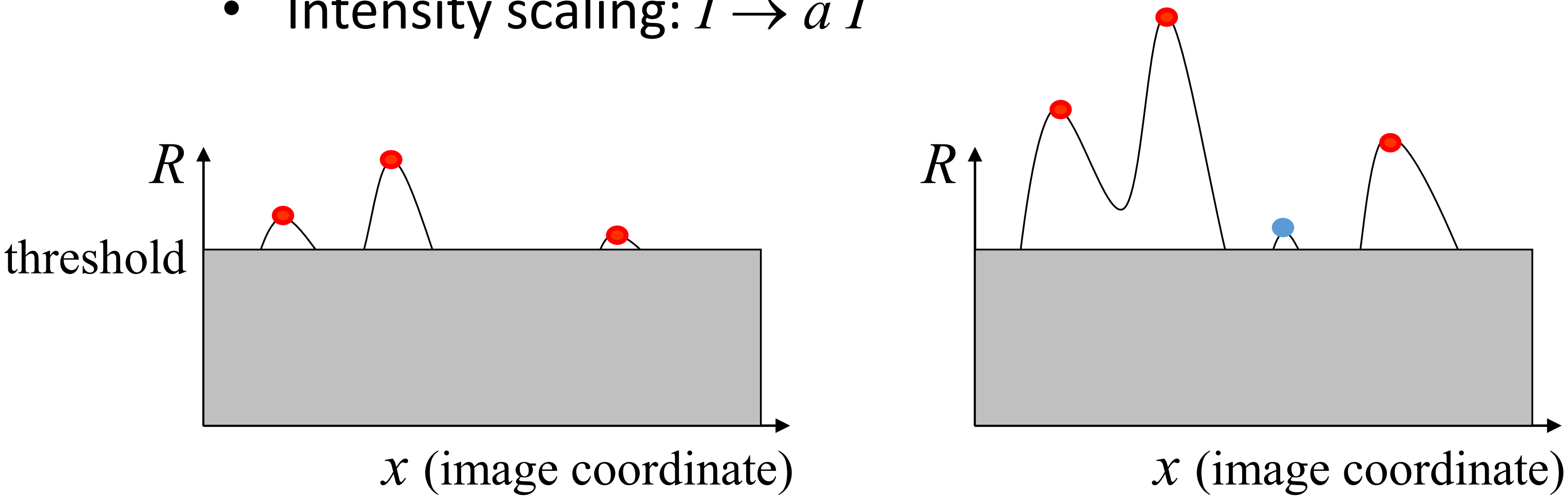
- Find points with large corner response function R (R > threshold)
- Take the points of locally maximum R as the detected feature points (i.e., pixels where R is bigger than for all the 4 or 8 neighbors).

D. Frolova, D. Simakov

Affine intensity change

$$I \rightarrow a\,I + b$$

- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
- Intensity scaling: $I \rightarrow a\,I$



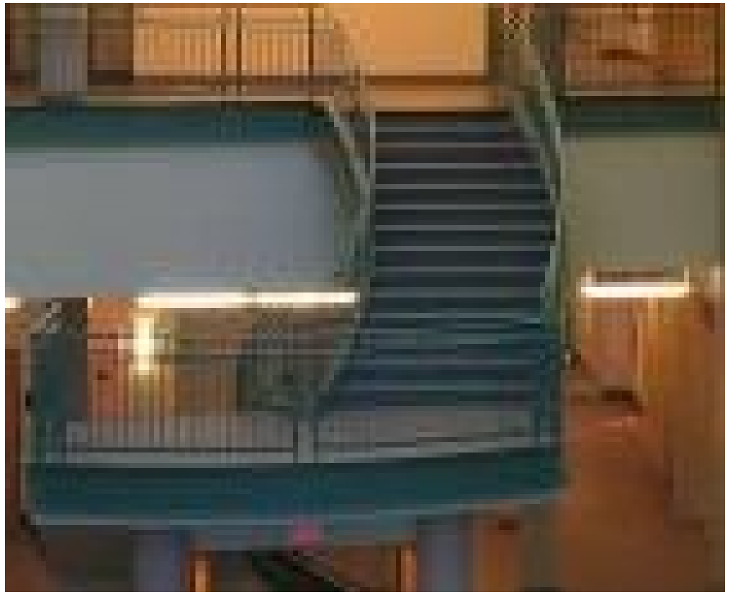
Partially invariant to affine intensity change

L. Lazebnik

Scaling

- Invariant to image scale?

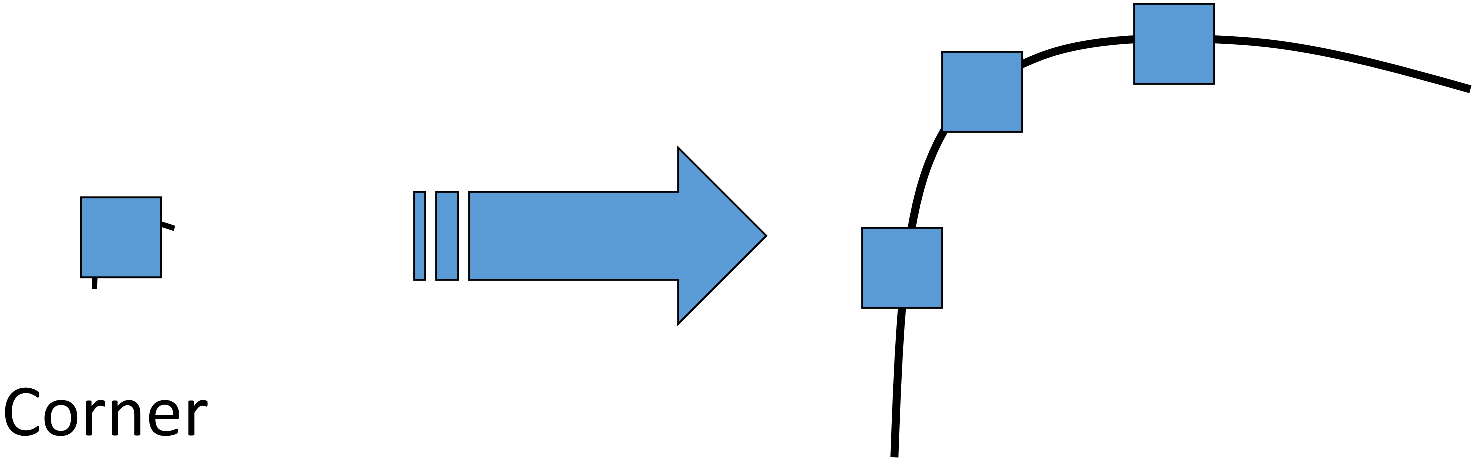
image



zoomed image

A. Torralba

Scaling



All points will be classified as edges

Corner location is not covariant to scaling!

L. Lazebnik

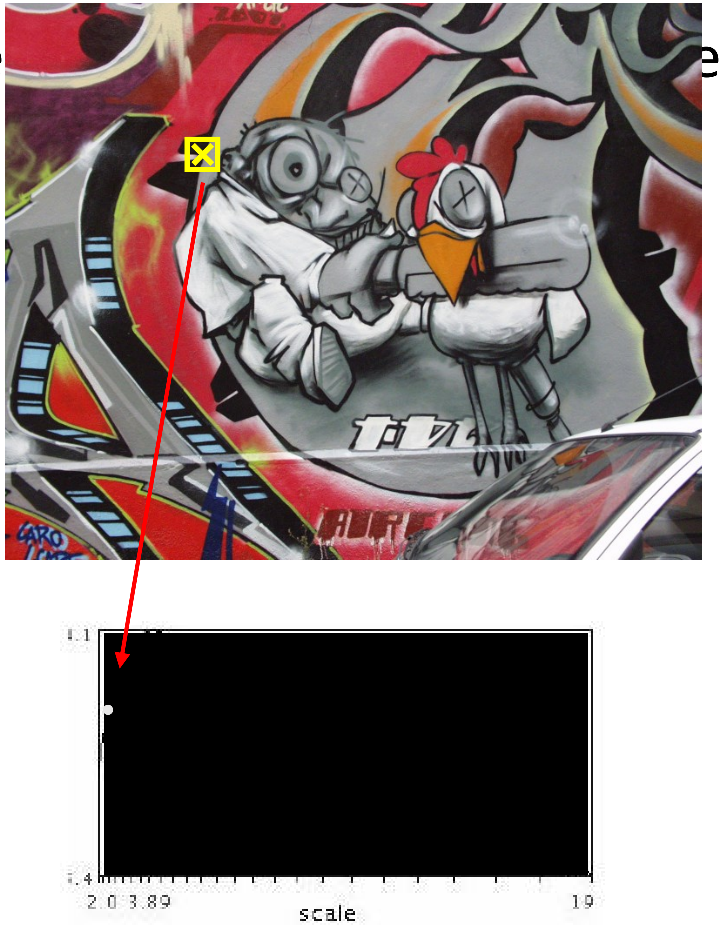
Automatic Scale Selection

$$f(I_{i_1 \ldots i_m}(x,\sigma)) = f(I_{i_1 \ldots i_m}(x',\sigma'))$$

How to find corresponding patch sizes?

Automatic Scale Selection

- Function re e (scale



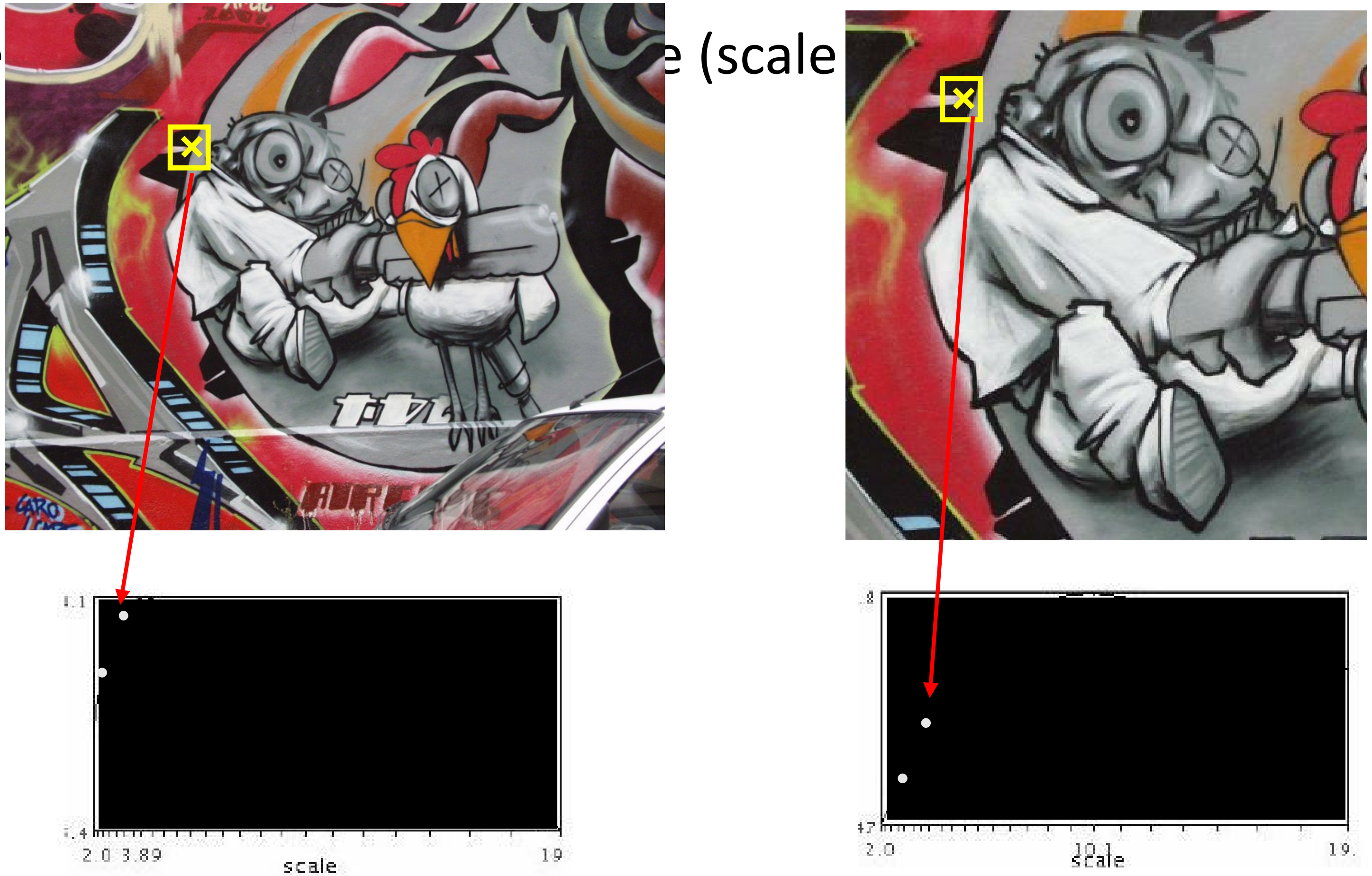
$f(I_{i_1 \ldots i_m}(x, \sigma))$

$f(I_{i_1 \ldots i_m}(x', \sigma))$

K. Grauman, B. Leibe

Automatic Scale Selection

- Function re e (scale



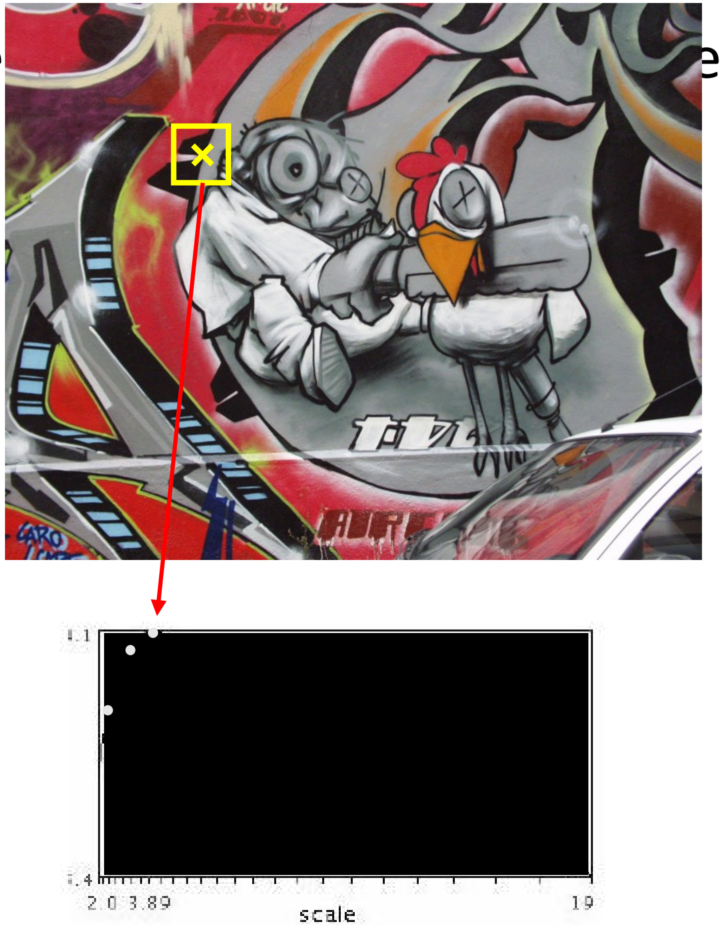
$f(I_{i_1 \ldots i_m}(x, \sigma))$

$f(I_{i_1 \ldots i_m}(x', \sigma))$

K. Grauman, B. Leibe

Automatic Scale Selection

- Function re e (scale



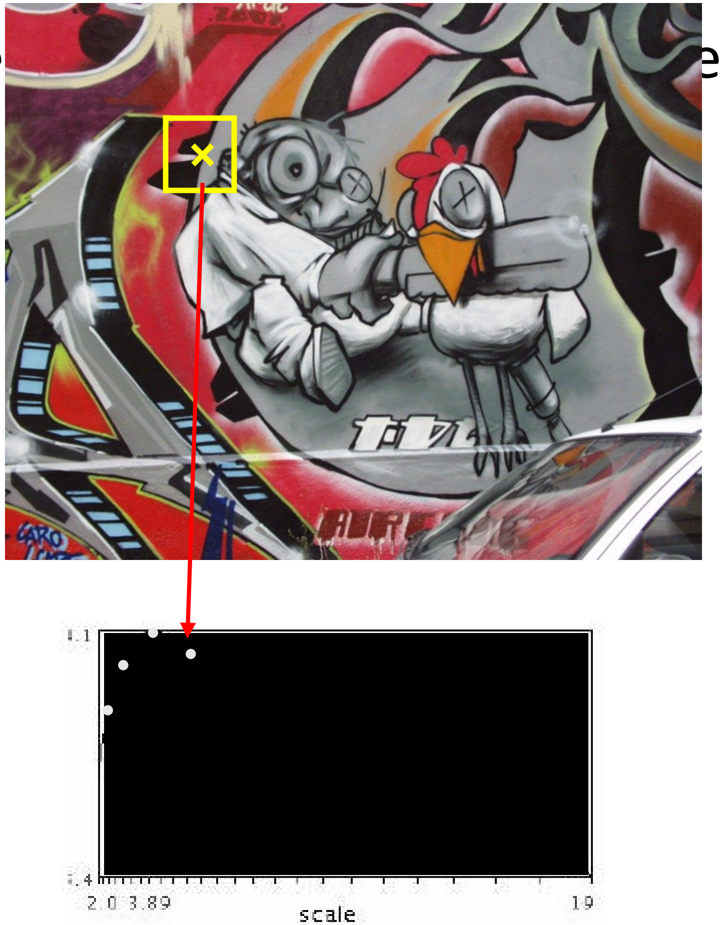
$$f(I_{i_1 \dots i_m}(x, \sigma))$$

$$f(I_{i_1 \dots i_m}(x', \sigma))$$

K. Grauman, B. Leibe

Automatic Scale Selection

- Function re e (scale



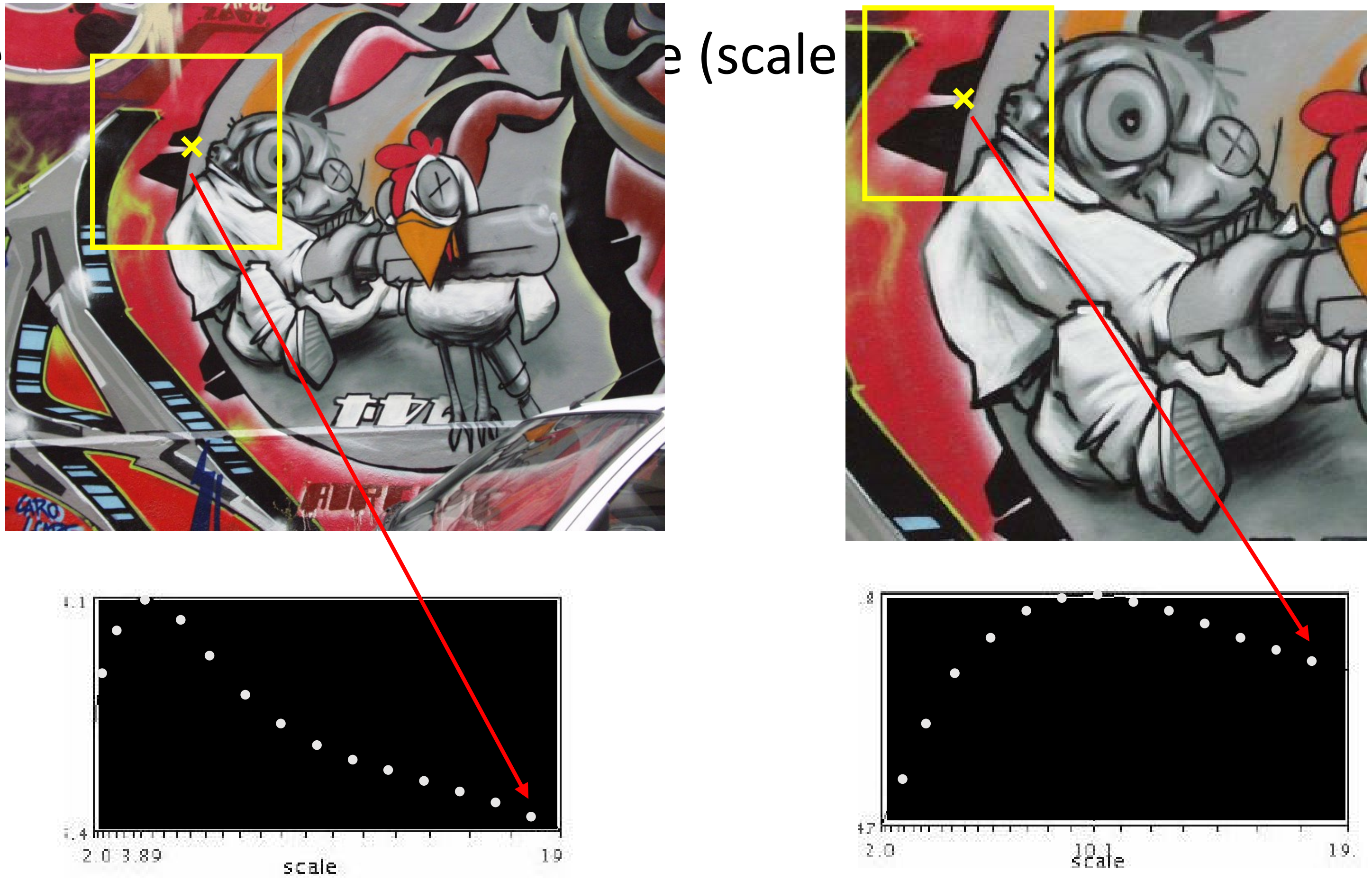
$f(I_{i_1\ldots i_m}(x,\sigma))$

$f(I_{i_1\ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

Automatic Scale Selection

- Function re… e (scale

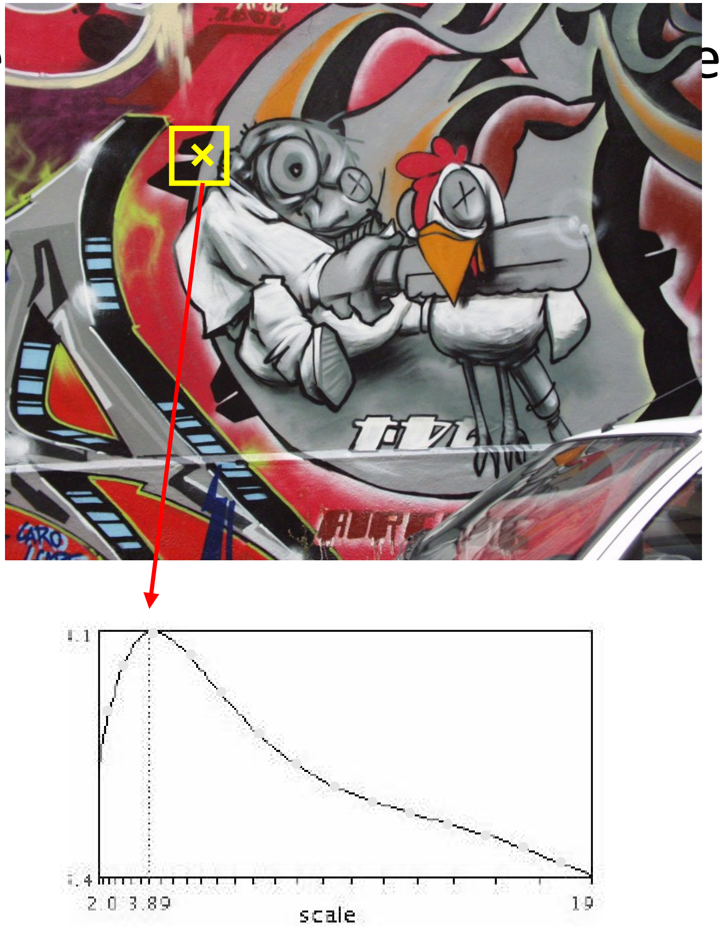


$f(I_{i_1 \dots i_m}(x, \sigma))$ $f(I_{i_1 \dots i_m}(x', \sigma))$

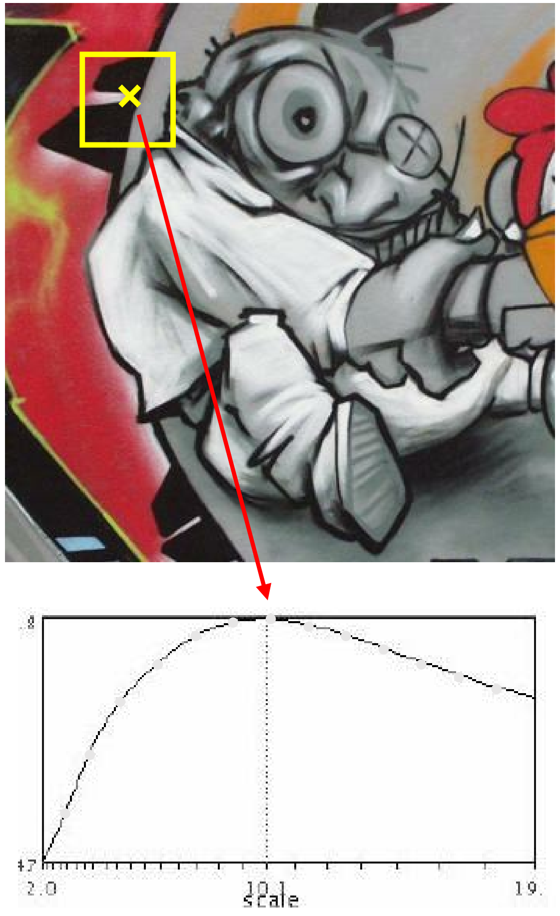
K. Grauman, B. Leibe

Automatic Scale Selection

- Function re e (scale



$f(I_{i_1 \dots i_m}(x, \sigma))$



$f(I_{i_1 \dots i_m}(x', \sigma'))$

K. Grauman, B. Leibe