Invariance in Feature Detection

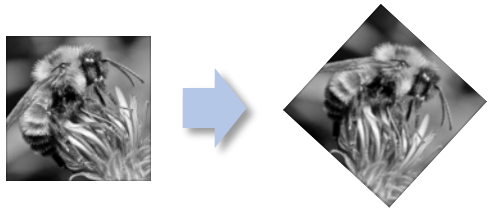
Invariance and equivariance

- Second criterion of corner detection: repeatability
- Suppose Image is transformed in some way
  - Image translation
  - Image rotation
  - Scaling of intensity
- How *should* the corners change?
  - Location?
  - Whether a corner is detected or not?
- How *does* the output of Harris corner detector change?

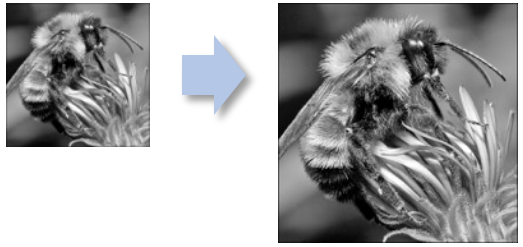
Image transformations

- Geometric

Rotation

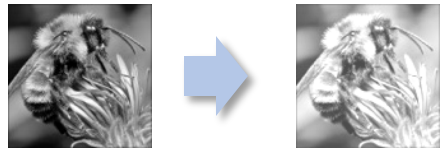


Scale



- Photometric

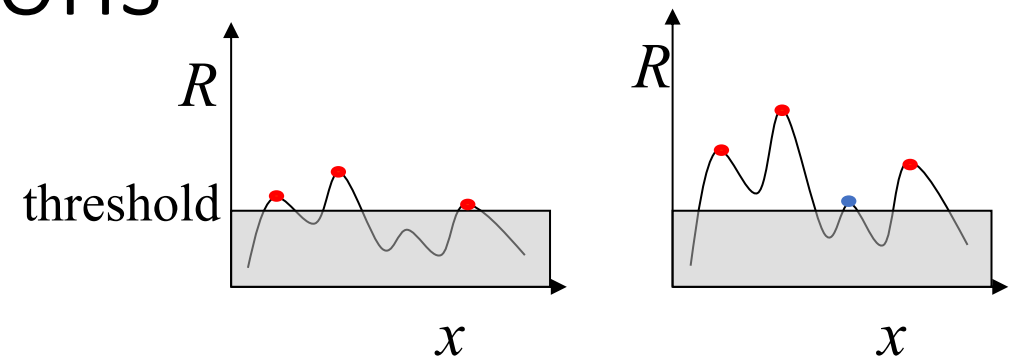
Intensity change



How should *photometric transformations* affect corner detection?

- Corner location:
  - Should *not* change!
- Probability of detection
  - Should *not* change

Harris corner detector invariance to photometric transformations



- Let us assume affine intensity change
- $I' = aI + b$
- What happens to image derivatives?
  - $I'_x = aI_x$
  - $I'_y = aI_y$
- What happens to the second moment matrix?
  - $M' = a^2 M$
- What happens to eigenvalues & cornerness response function?
  - $\lambda'_i = a^2\lambda_i, R' = a^2 R$
- Does this change probability of detecting a corner?
  - Yes, because of thresholding (last step)!
- What happens if no scaling (i.e., a = 1)?

Harris corner detector invariance to photometric transformations

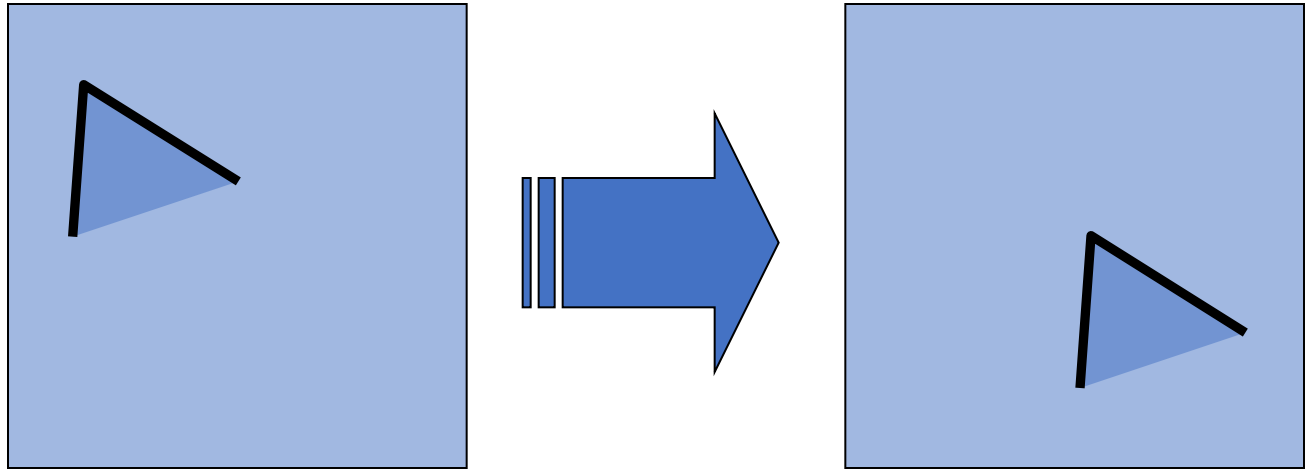
- Harris corner detector (both locations and probability of corner detection) is invariant to *additive* changes in intensity
  - changes in overall “Brightness”
- It is *not invariant* to scaling of intensity
  - Changes in “Contrast”

How should *geometric transformations* affect corner detection?

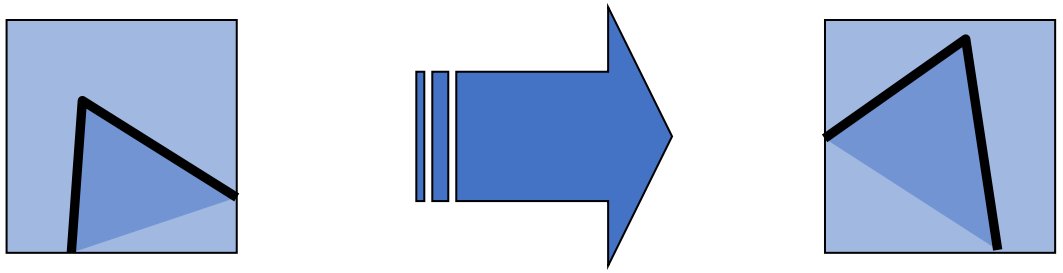
- Corner *location* should move as the underlying pixel moves
  - Thus corner location should be *equivariant* to geometric transformations
- Corner detection probability should be unaffected by geometric transformations
  - Thus *invariant* to geometric transformations

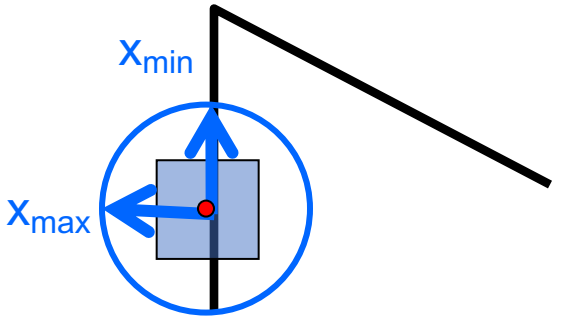
Harris corner detection and translation

- What happens if image is translated?
- Derivatives, second moment matrix obtained through convolution, which is *translation equivariant*
- Eigenvalues based only on derivatives so cornerness is *invariant*
- Thus Harris corner detection location is *equivariant to translation,* and response is *invariant to translation*



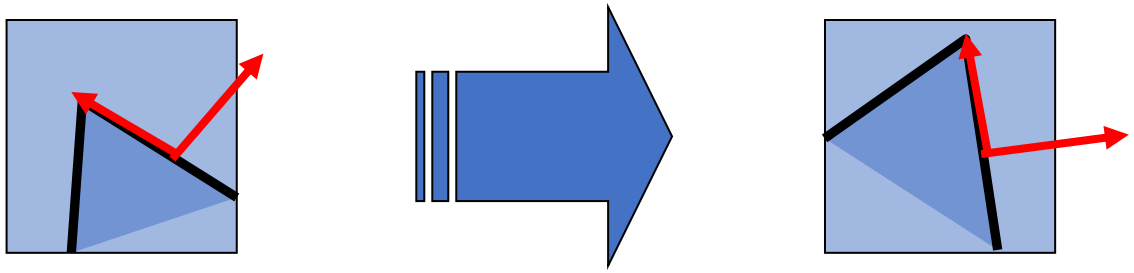
What about rotation?





- Now every patch is rotated, so problem?
- Recall properties of second moment matrix
- Eigenvalues and eigenvectors of M
  - Define shift directions with the smallest and largest change in error
  - x_{max}= direction of largest increase in E (across the edge)
  - λ_{max} = amount of increase in direction x_{max}
  - x_{min}= direction of smallest increase in E (along the edge)
  - λ_{min} = amount of increase in direction x_{min}

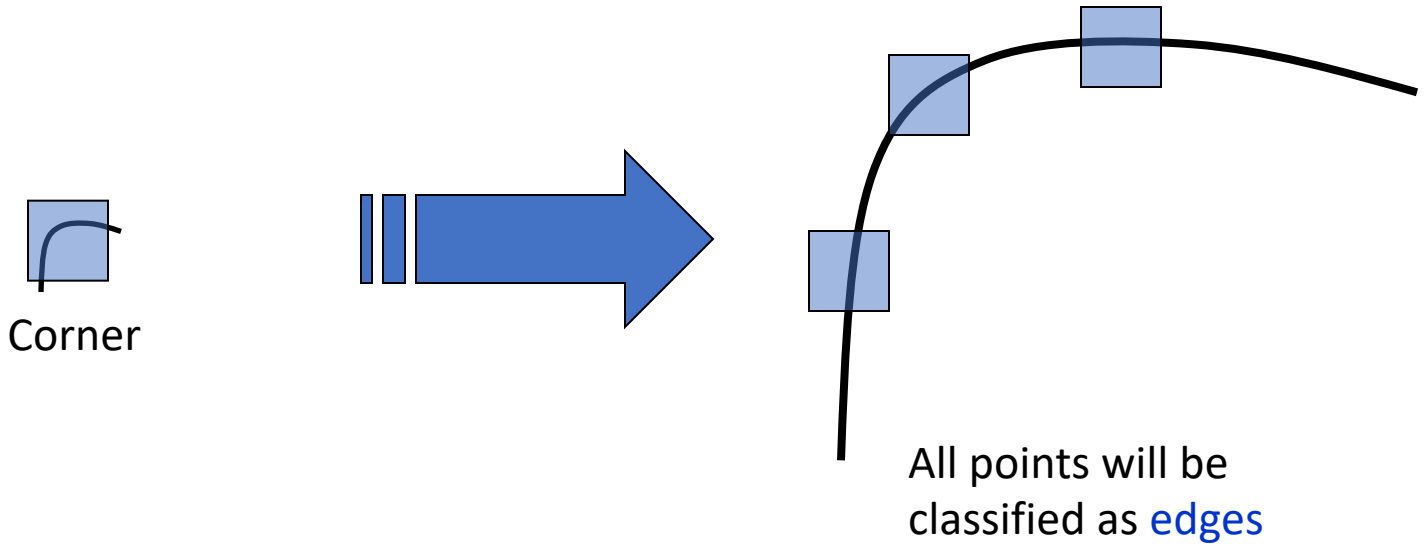
What about rotation?



- What happens to eigenvalues and eigenvectors when a patch rotates?
- Eigenvectors represent the *direction* of maximum / minimum change in appearance, so they rotate *with the patch*
- Eigenvalues represent the corresponding *magnitude* of maximum/minimum change so they *stay constant*
- Corner response is only dependent on the eigenvalues so is *invariant to* rotation
- Corner location is as before equivariant to rotation.

What about scaling?

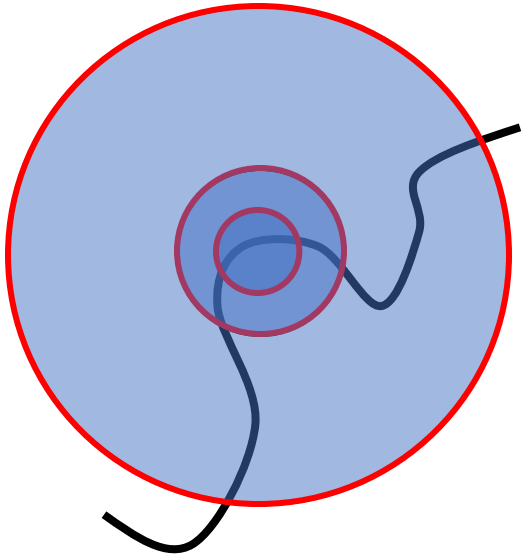
- What was one patch earlier is now many



Not invariant to scaling

Scale invariant detection

Suppose you're looking for corners

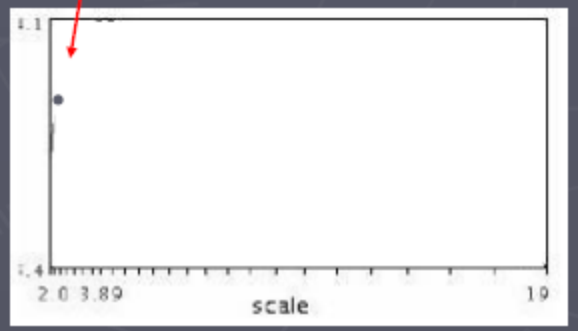


Key idea: find scale that gives local maximum of *cornerness*

- in both position and scale
- One definition of *cornerness*: the Harris operator

Automatic scale selection

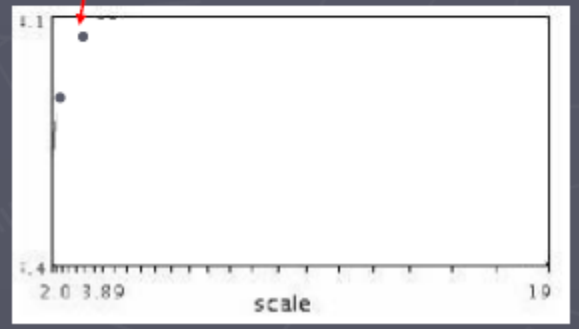
Lindeberg et al., 1996



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

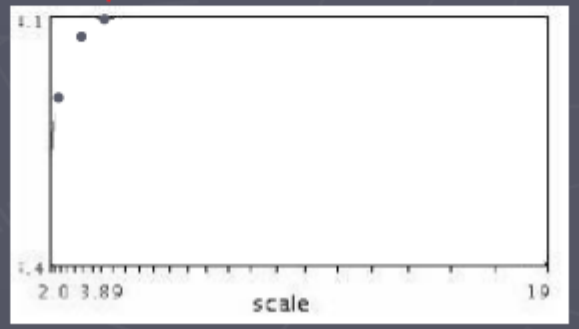
Slide from Tinne Tuytelaars

Automatic scale selection



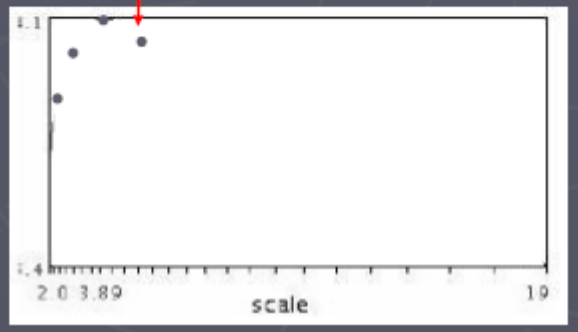
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



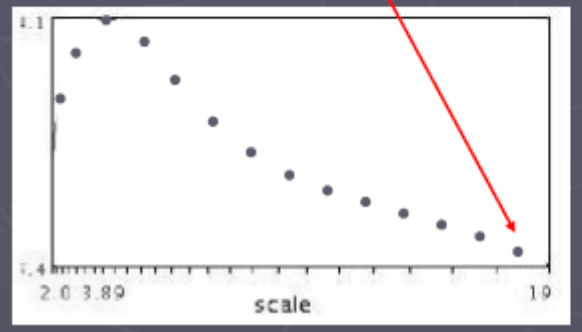
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



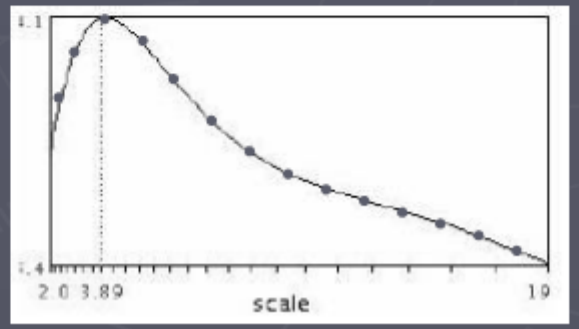
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



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

Automatic scale selection



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

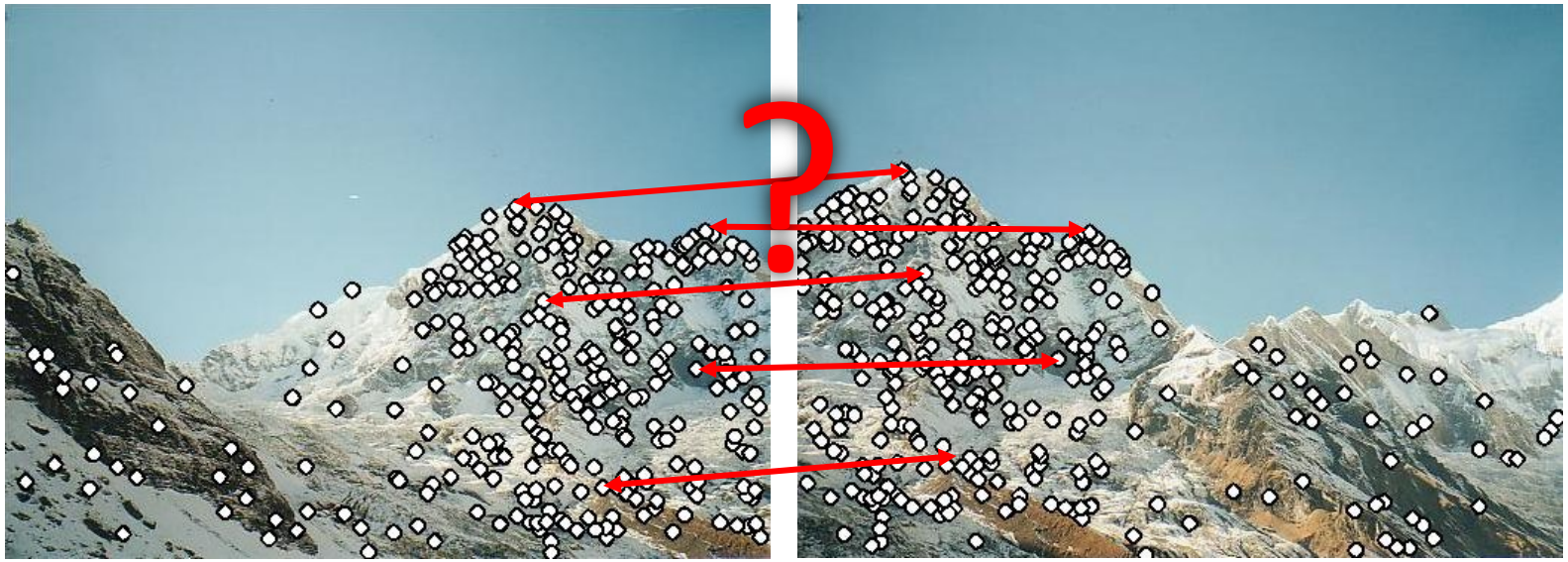
Implementation

- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid

Feature descriptors

Matching feature points

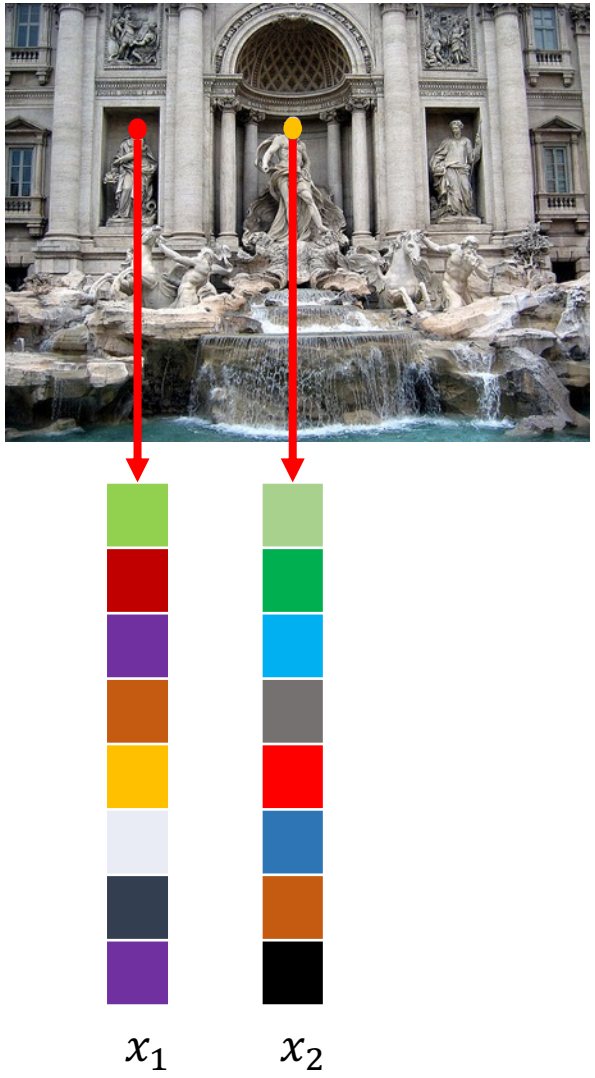
We know how to detect good points
Next question: **How to match them?**

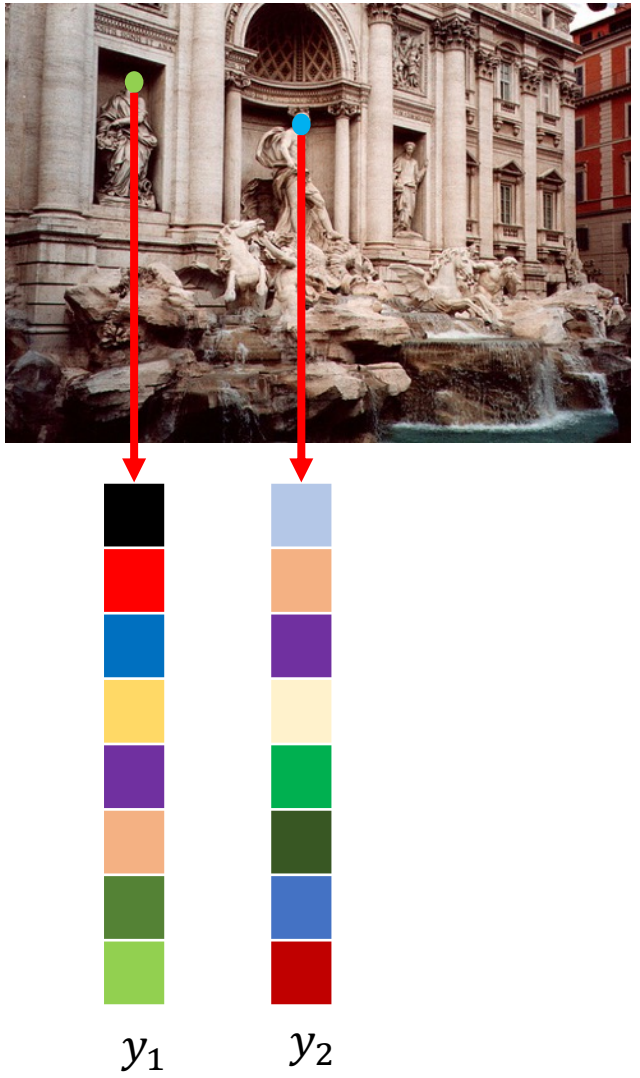


Two interrelated questions:

1. How do we *describe* each feature point?
2. How do we *match* descriptions?

Feature descriptor





Feature matching

- Measure the distance between (or similarity between) every pair of descriptors

	y_1	y_2
x_1	$d(x_1, y_1)$	$d(x_1, y_2)$
x_2	$d(x_2, y_1)$	$d(x_2, y_2)$

Invariance vs. discriminability

- Invariance:
  - Distance between descriptors of corresponding points should be small even if image is transformed

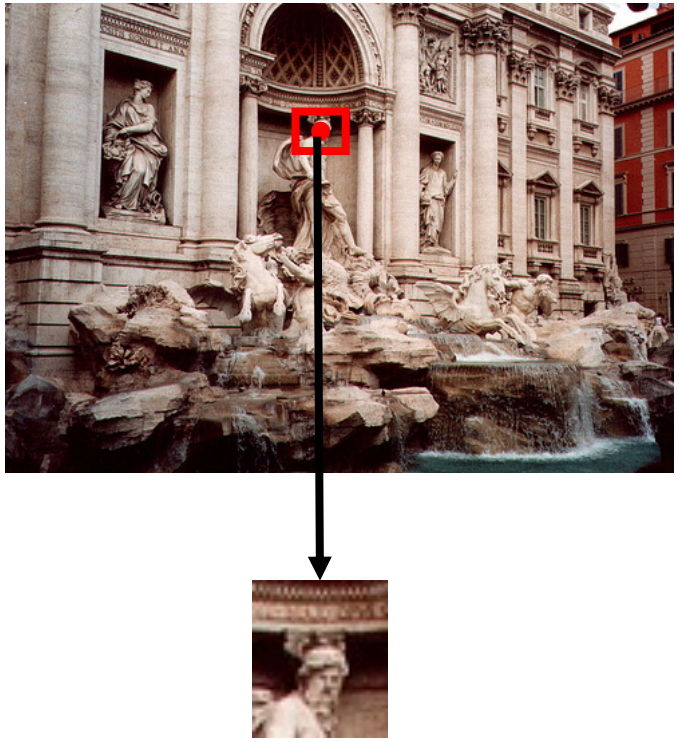
- Discriminability:
  - Descriptor for a point should be highly unique for each point (far away from other points in the image)

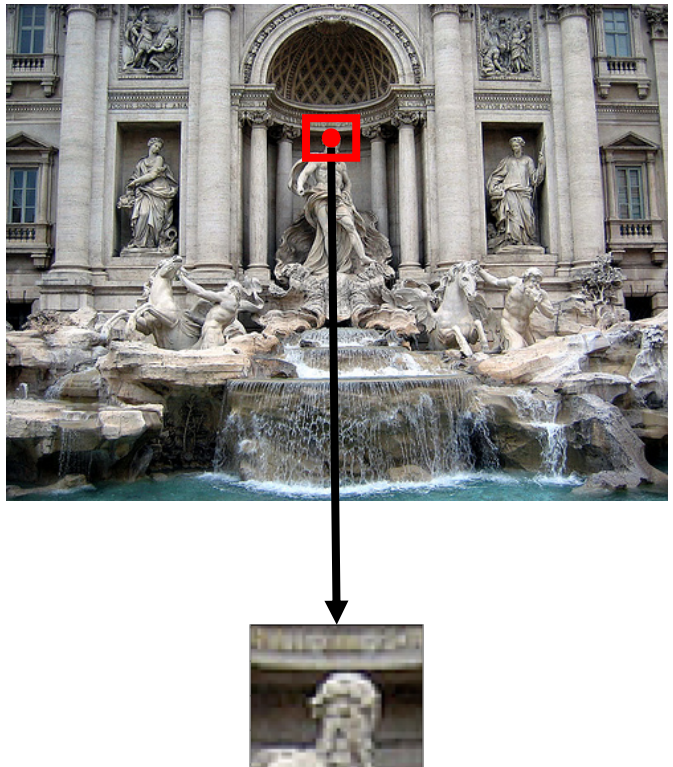
Simple baseline descriptors

- Simplest descriptor: a constant 0
  - What's this invariant to?
  - Is this discriminative?
- Next simplest descriptor: the color of the pixel
  - What's this invariant to?
  - Is this discriminative?

The “Patch” descriptor

- Take a window around the corner
- Write out colors of pixels in the window as a descriptor
- How does this affect discriminability? Invariance?



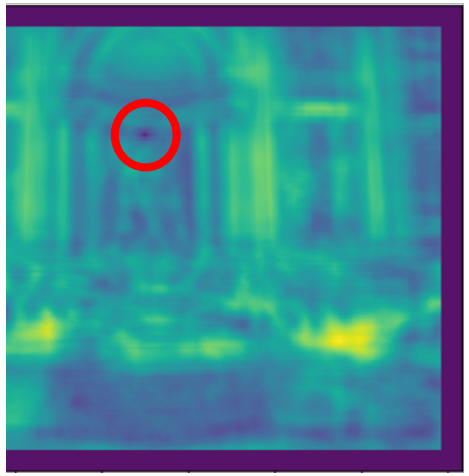


Matching “Patch descriptors” using SSD

- Match descriptors using euclidean distance
- $d(x, y) = ||x - y||^2$

Patch around corner

Second image

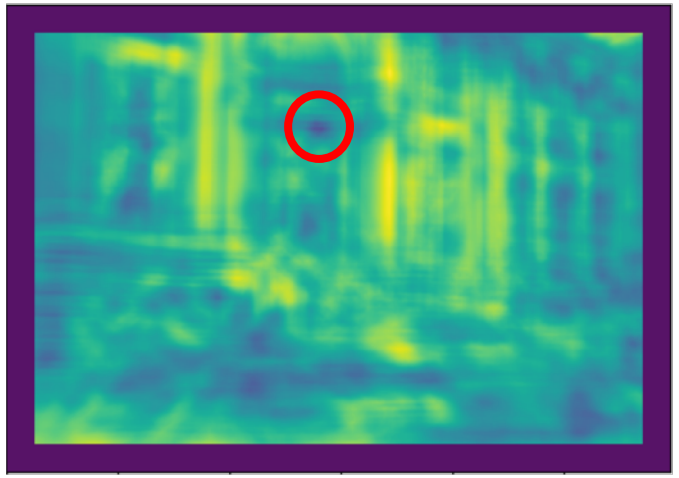


Descriptor distance. Blue is low, yellow is high

SSD on a slightly harder example

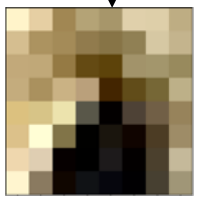
Patch around corner

Second image



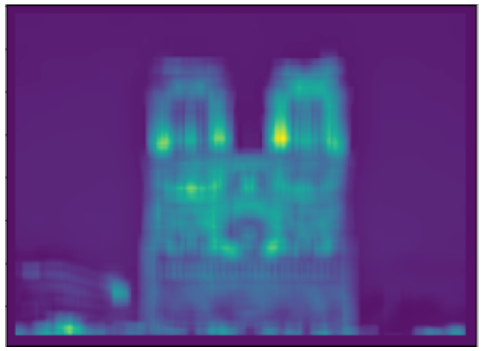
Descriptor distance. Blue is low, yellow is high

SSD on an even harder example



Patch around corner

Second image

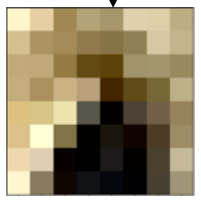


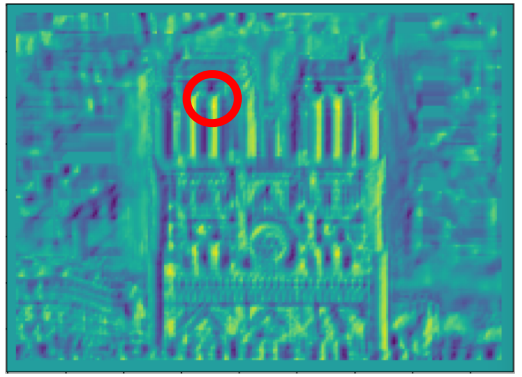
Descriptor distance. Blue is low, yellow is high

NCC - Normalized Cross Correlation

- Lighting and color change pixel intensities
- Example: increase brightness / contrast
- $I' = \alpha I + \beta$
- Subtract patch mean: invariance to β
- Divide by norm of vector: invariance to α
- $x' = x - < x >$
- $x'' = \frac{x'}{||x'||}$
- *similarity* = $x'' \cdot y''$

NCC - Normalized cross correlation





Basic correspondence

- Image patch as descriptor, NCC as similarity
- Invariant to?
  - Photometric transformations?
  - Translation?
  - Rotation?

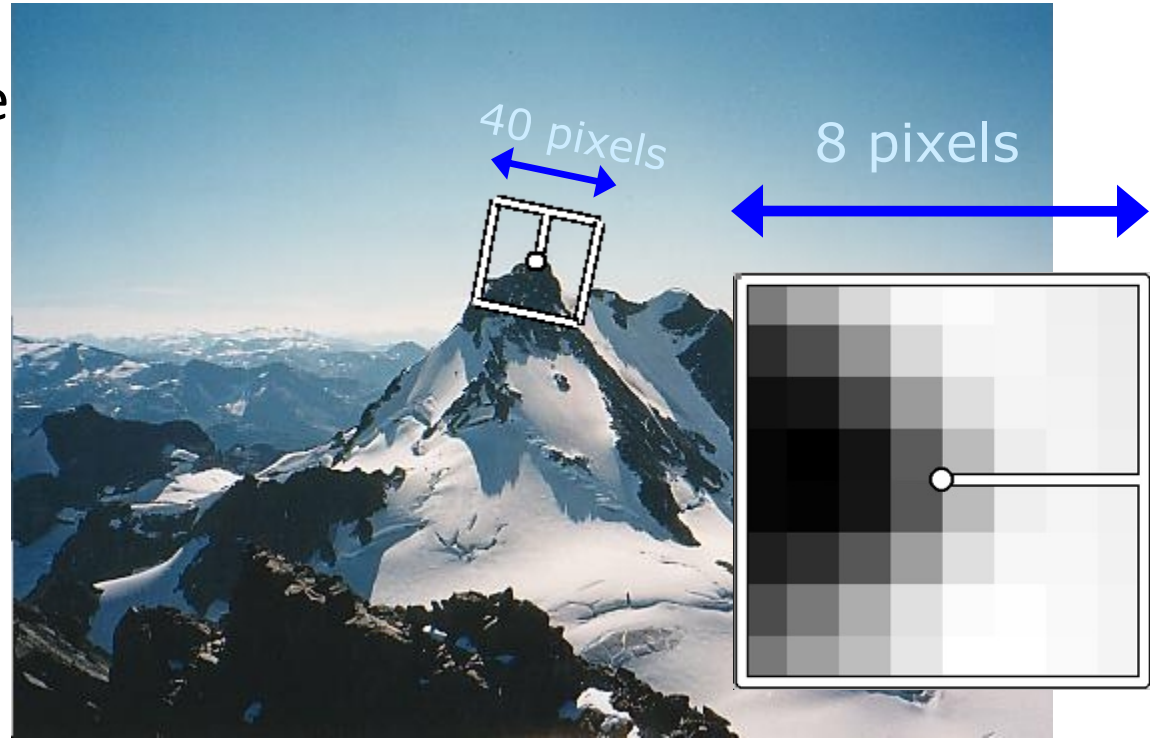
Rotation invariance for feature descriptors

- Find dominant orientation of the image patch
  - This is given by $\mathbf{x}_{max}$, the eigenvector of $\mathbf{M}$ corresponding to λ_{max} (the *larger* eigenvalue)
  - Rotate the patch according to this angle
  - Figure: line represents x_{max}, box represents patch we take as feature descriptor

Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

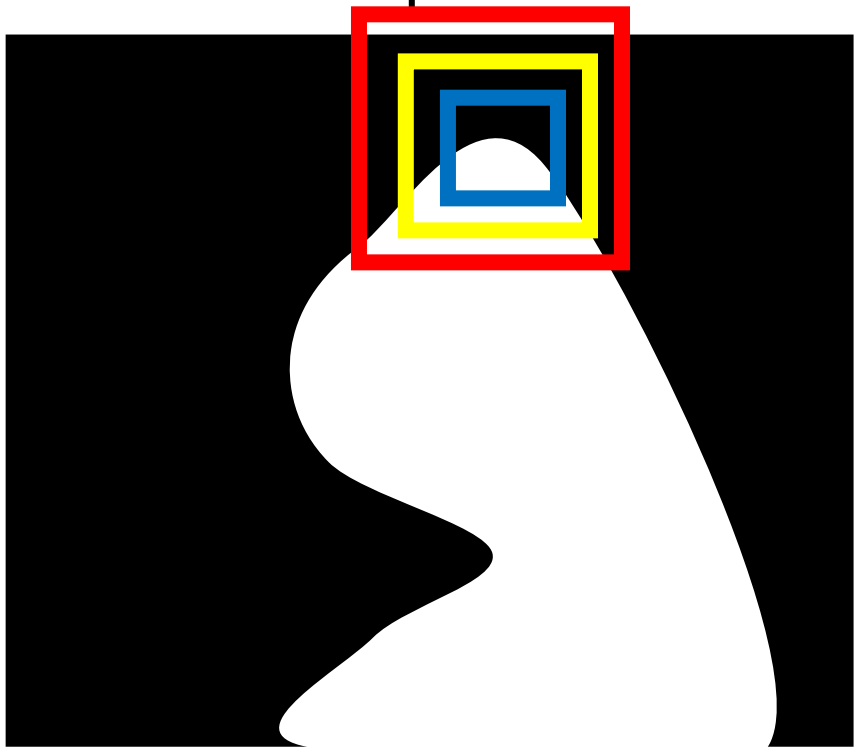
- Take 40x40 *oriented* square window around detected feature
- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window

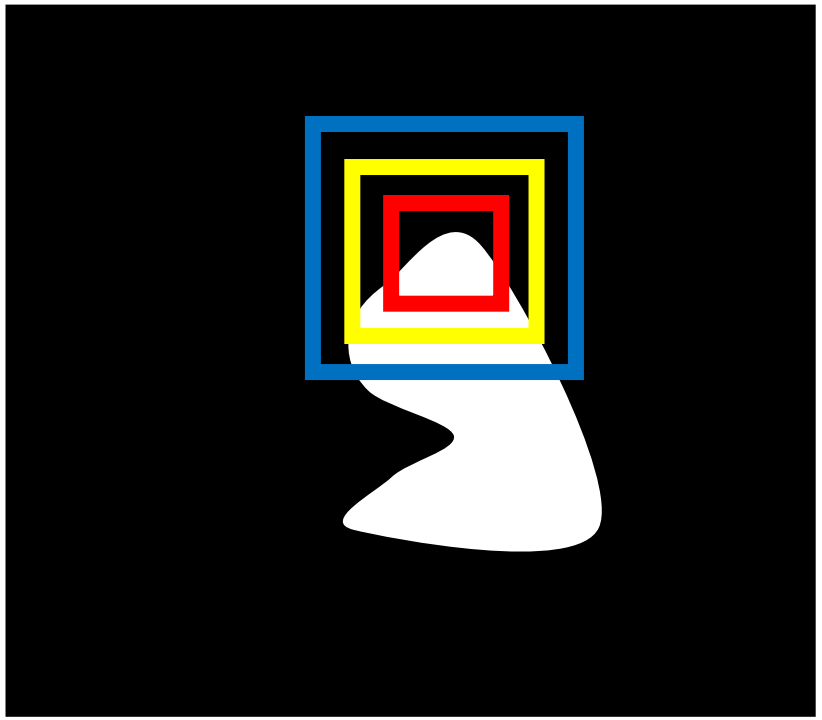


Adapted from slide by Matthew Brown

Scale invariance for feature descriptors

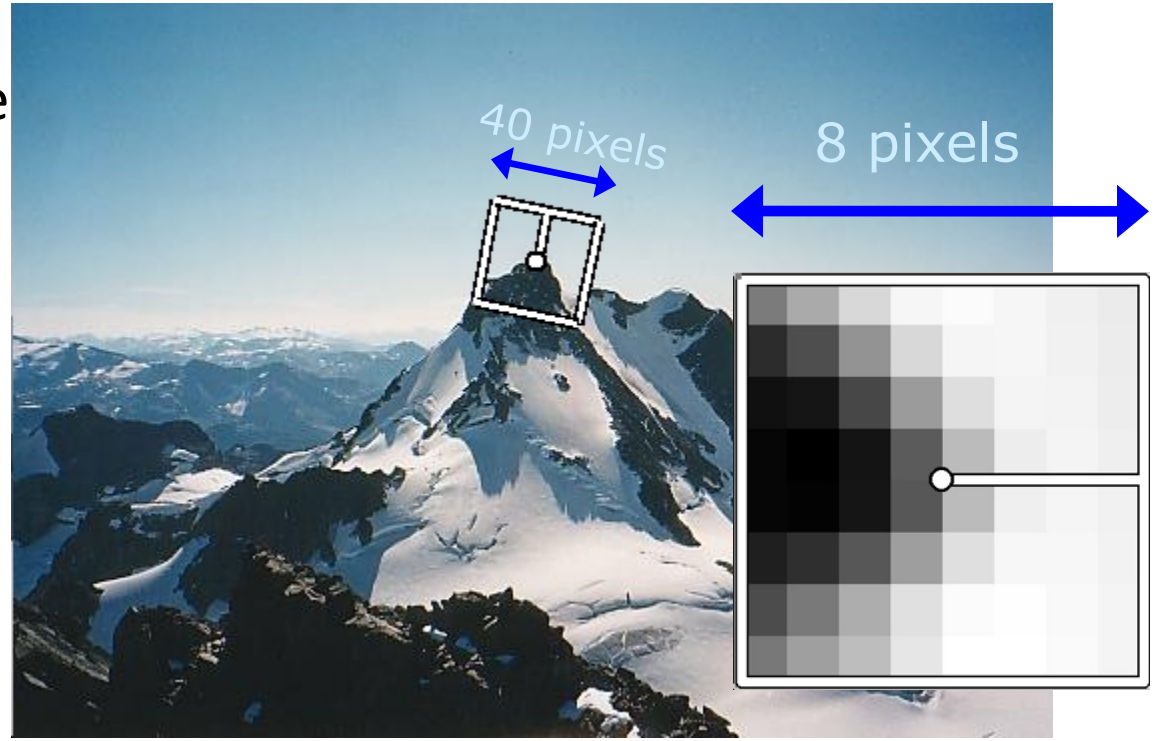
- Recall that corner detector searches over scales for maximum response: record scale which gives maximum response
- Use a patch of the same scale, then resize it to a fixed size





Multiscale Oriented PatcheS descriptor

- Take 40x40 *oriented* square window around detected feature at *appropriate scale*
- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window



Adapted from slide by Matthew Brown

Detections at multiple scales

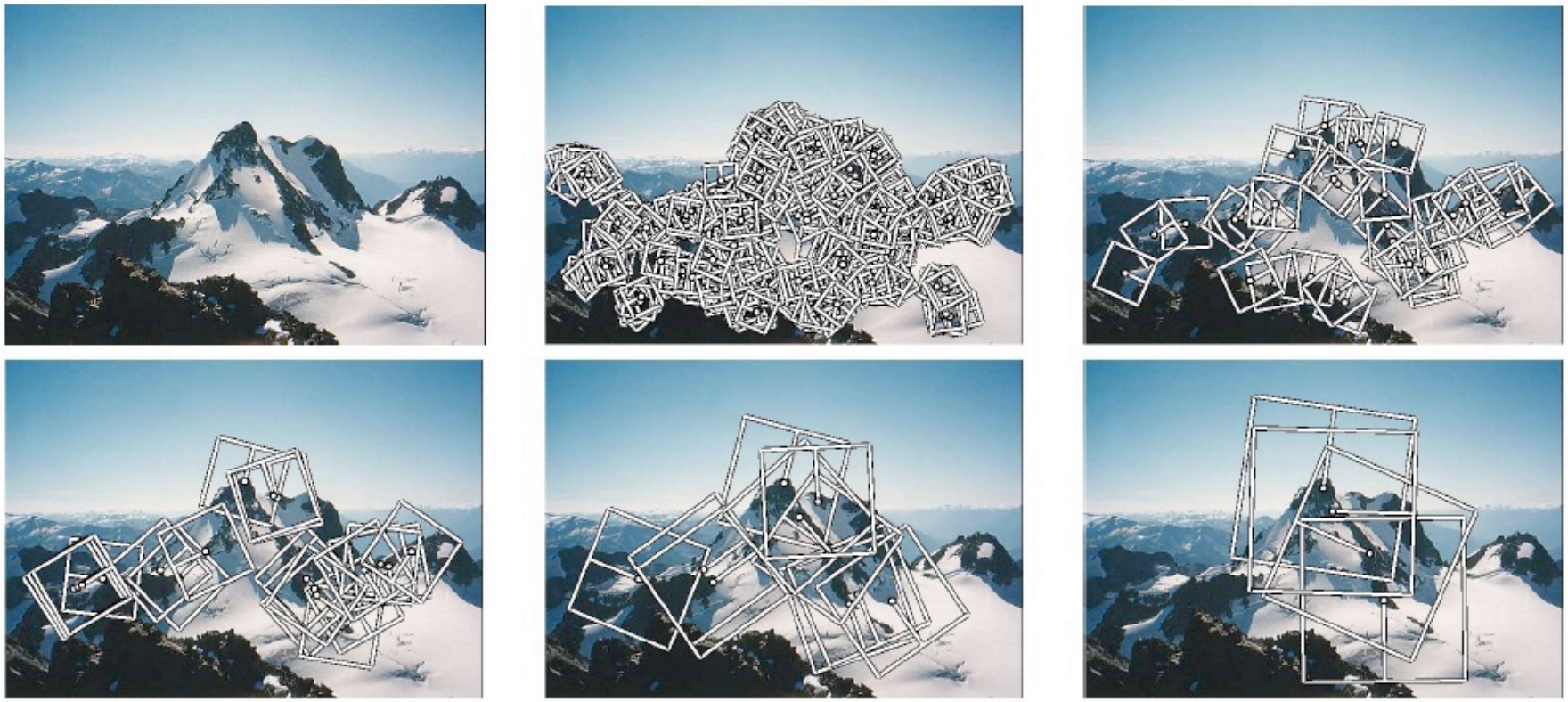


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

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