Feature descriptors

Computer Vision lecture 3

Szeliski Ch 7.1, 8.1

Today's lecture

- From last time:
  - Image features give us a useful representation of an image
  - Indicate image locations that are of interest: corners, edges, "blobs"
  - Invariant to rotation, brightness (sort of), scale (sort of)

- Today:
  - Feature descriptors: how to describe an image's contents?
  - Both locally (around features) and globally

Local vs global descriptors

- **Global** image descriptors summarise or describe the contents of an entire image
  - Typically output a vector “signature” representing the image
  - Examples: “tiny image” (Gaussian pyramid top level)
  - GIST (100s of elements),
  - Histogram of Oriented Gradients (HOG, 1000s of elements)
  - CNN models (millions of elements)

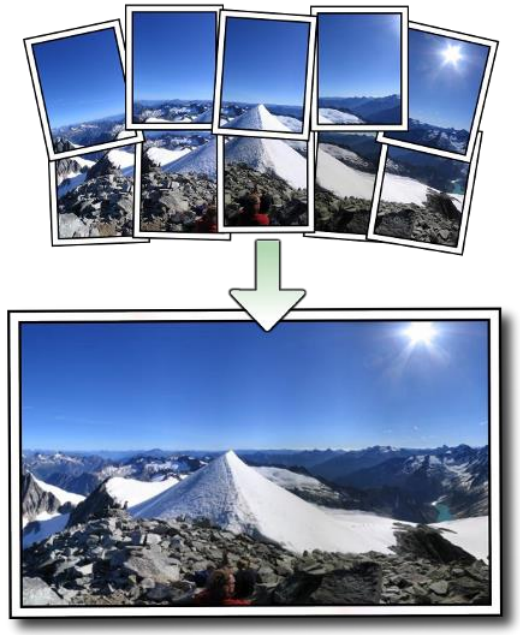
- Global image descriptors are used for tasks like:
  - Image classification
  - Scene recognition
  - Image search and retrieval
  - Just 1 descriptor per image

Local vs global descriptors

- **Local** feature descriptors describe a patch around a feature location
  - Also as a vector or “signature”

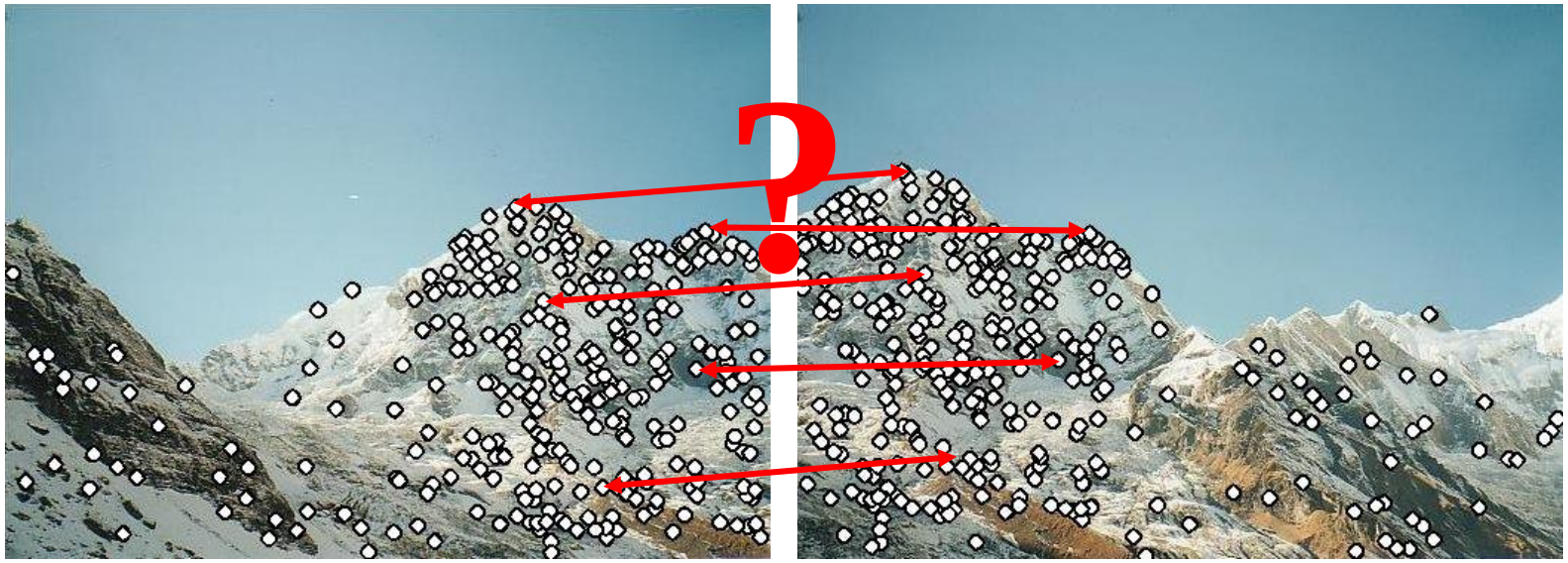
- Local descriptors are usually used for tasks based on objects within images
  - Object matching and recognition
  - Object motion tracking
  - 3D reconstruction from stereo images
  - Image warping and panorama stitching
  - A more detailed description of image contents

- Local and global descriptors are based on similar principles
  - In fact, can build a global descriptor from local descriptors
- We will focus on local descriptors today



Why feature descriptors?

- We know **where** the good features are
- How can we match them?



The simplest descriptor

- Just use the pixel values of the patch (a.k.a template matching)

- Can work in limited cases…
  - What are some problems?

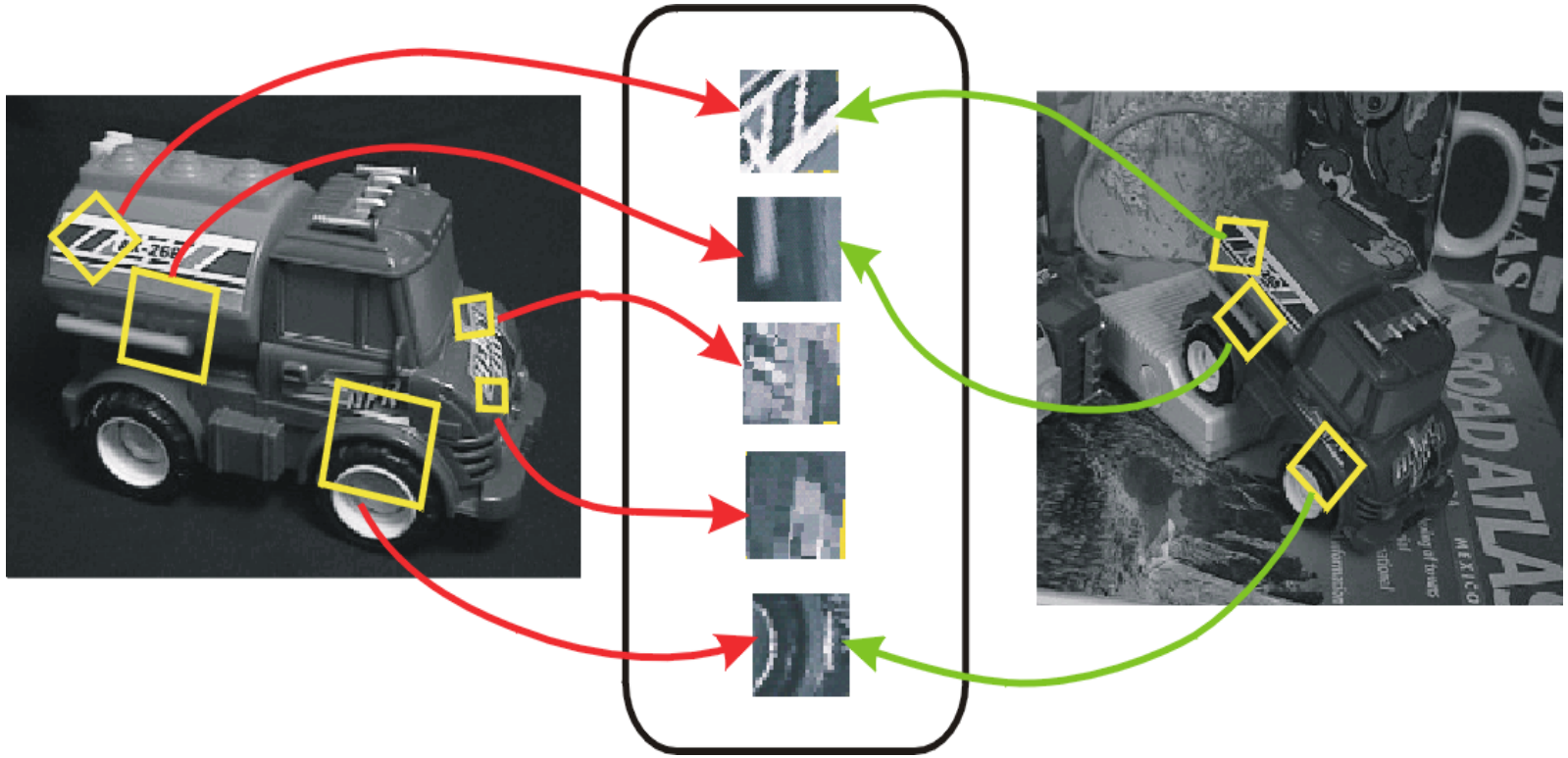
What makes a good descriptor?

by Diva Sian

by scgbt

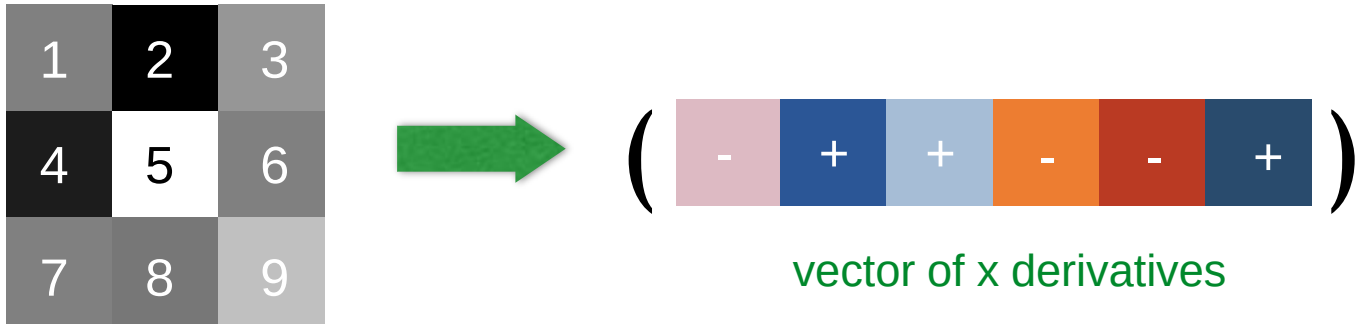
What makes a good descriptor?

- Geometric invariance: translation, rotation, scale
- Photometric invariance: brightness, exposure, tone …
- Unique within image



The gradient descriptor

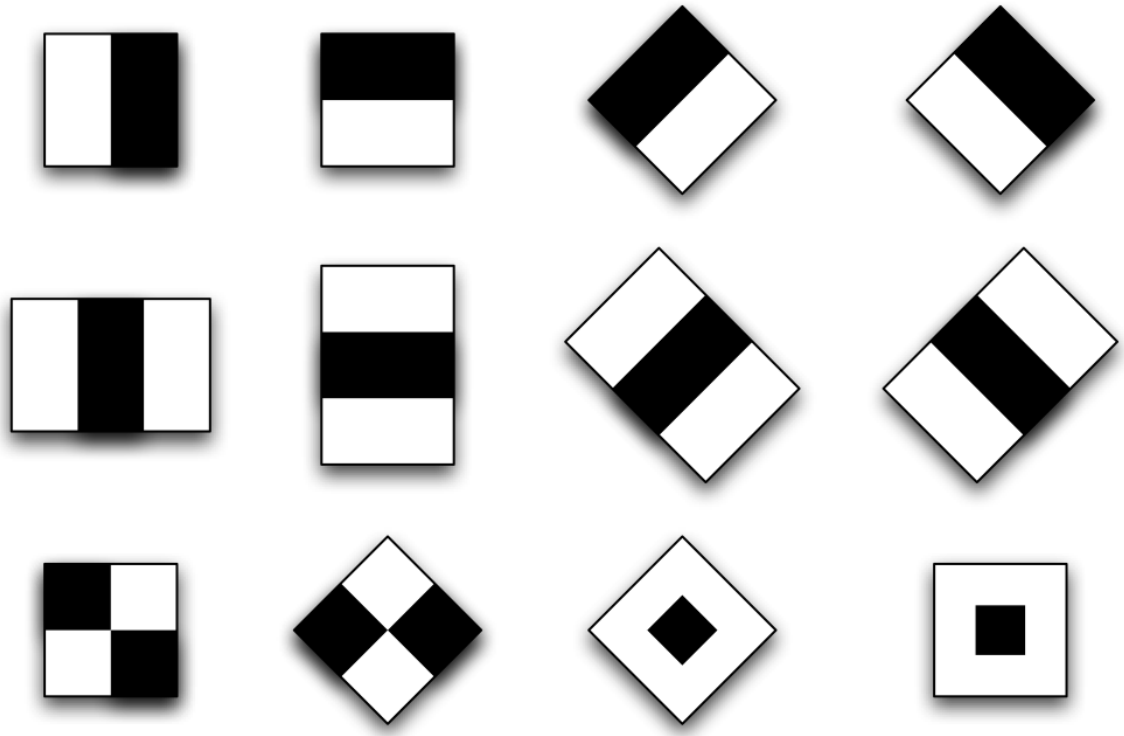
- Use x or y derivatives instead of pixel values:



- What have we gained?
- Why binary values?

Haar features

- Convolve image patch with a set of binary “Haar-like” filters
- Outputs form a vector which is the descriptor
- Filters are 1 where white, -1 where black
- Represents differences between adjacent image regions
- Very efficient to compute using integral images

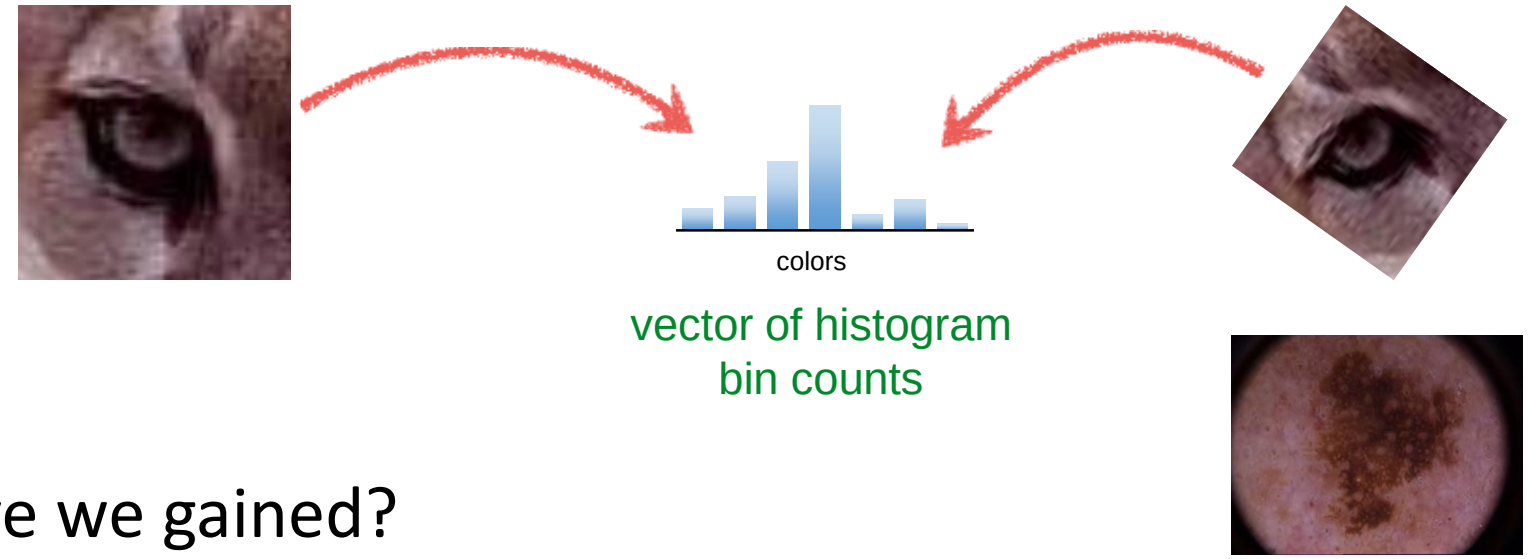


- What are the pros and cons?

Example:
Multi-Image Matching using Multi-Scale Oriented Patches.
M. Brown, R. Szeliski and S. Winder. (CVPR2005).

A colour histogram

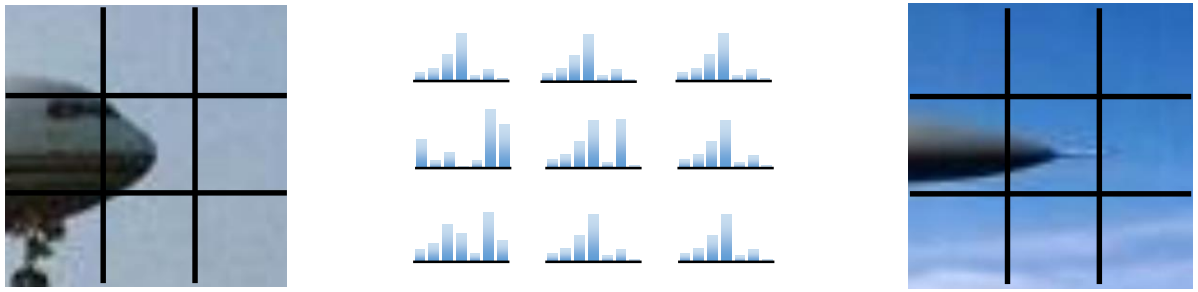
- Store counts of pixel values in a histogram



- What have we gained?
- What have we lost?

Spatial histograms

- Compute several colour histograms over spatial cells

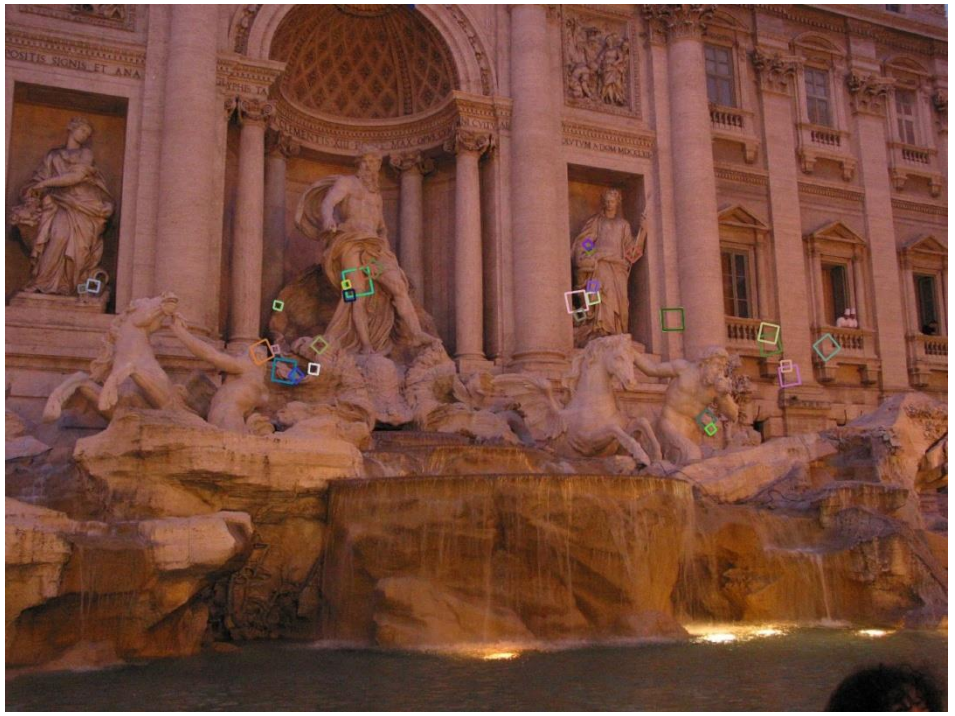


- What have we gained?
- What have we lost?

Scale Invariant Feature Transform (SIFT)

David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. Int. J. Comput. Vision 60, 2

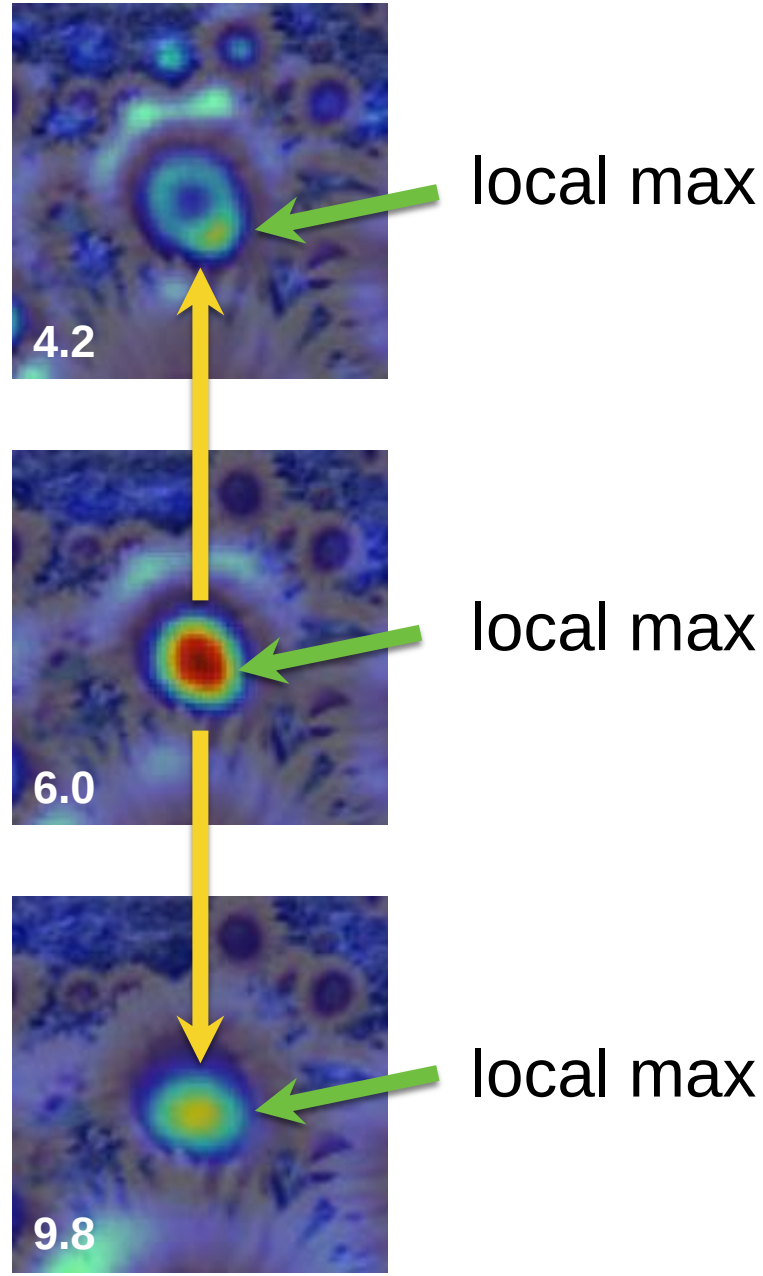
- SIFT method is a feature detector and descriptor
  - Can handle changes in viewpoint up to about 60 degrees
  - Can handle significant changes in illumination



Remember this?

```
For each level of the Gaussian pyramid
   Compute feature response
    (e.g. Harris, Laplacian)
```

```
For each level of the Gaussian pyramid
     if local maximum and cross-scale
     save scale and location of feature (x, y, s)
```



Rotation and scale invariant features

- Compute gradient values within the scale invariant feature patch
  - Recall gradient has a direction and a magnitude

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right]$$

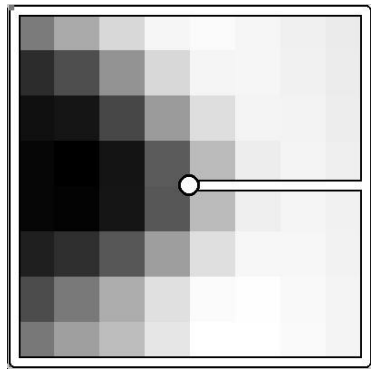
gradient

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

direction

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

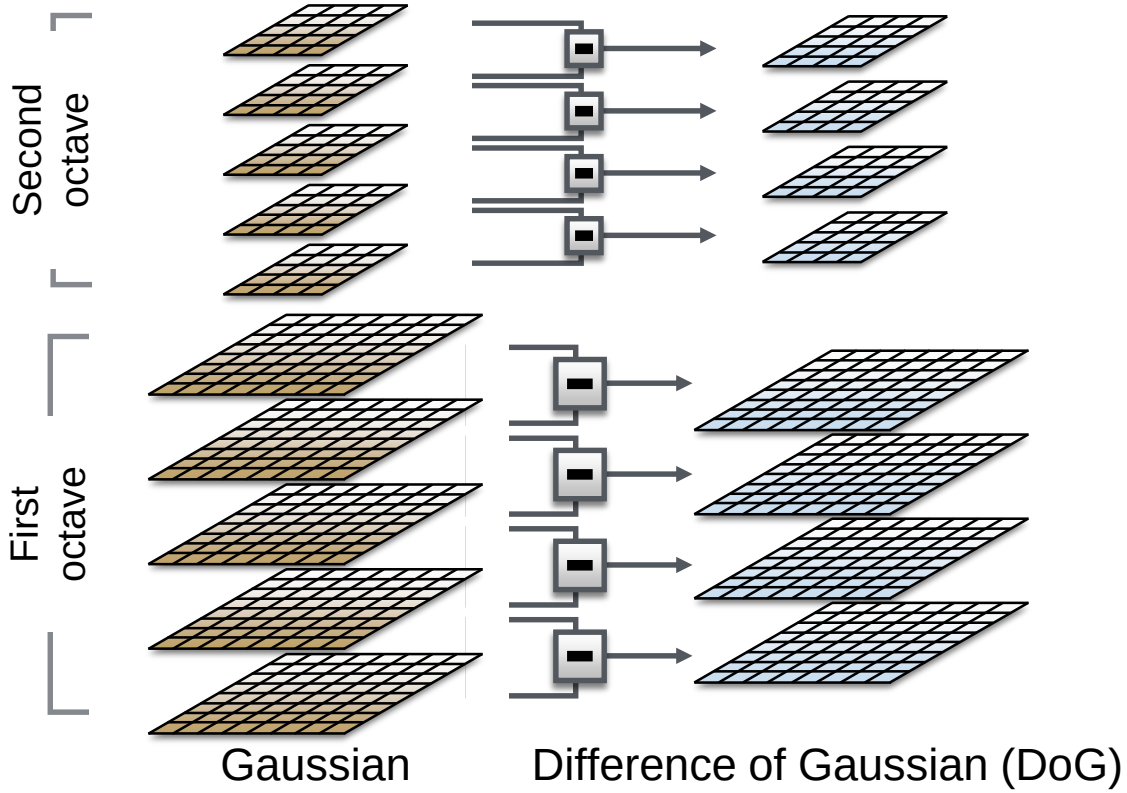
amplitude

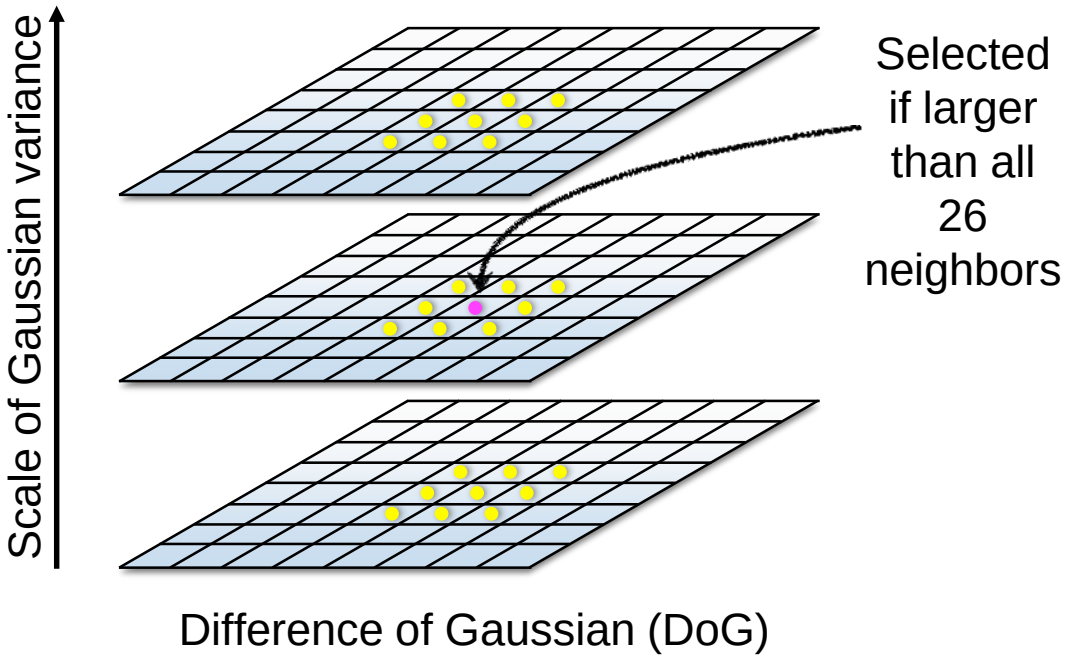


Save the ”dominant” orientation angle θ along with (x, y, s)

SIFT feature detector

- 1. Create image pyramid and select local maxima
  - Above a minimum response threshold

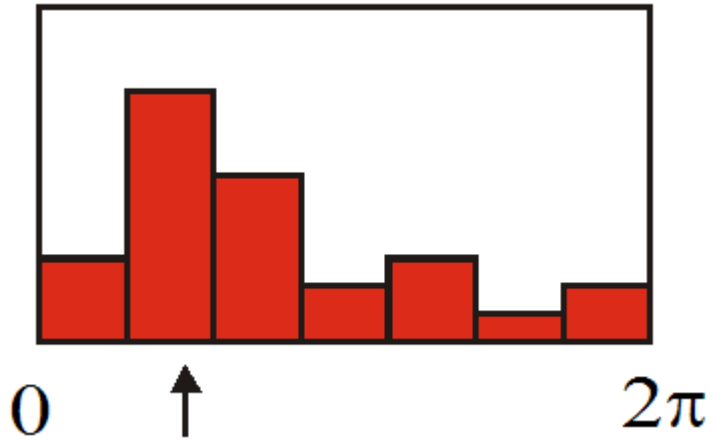




SIFT feature detector

- 2. Compute the dominant gradient direction at this scale
  - For rotation invariance

```
For each pixel in a window W about the feature:
    Calculate gradient magnitude and orientation m, t
    Add  to m to bin t in histogram of gradients
Feature orientation t_f = mode of histogram
```



SIFT feature descriptor

- The SIFT descriptor is a spatial histogram of gradients:

```
For each of the 4x4 subwindows surrounding the feature:
    For each pixel in the subwindow:
        Calculate gradient orient relative to feature orient
        Add to 8 bin histogram of gradients for the subwindow
```

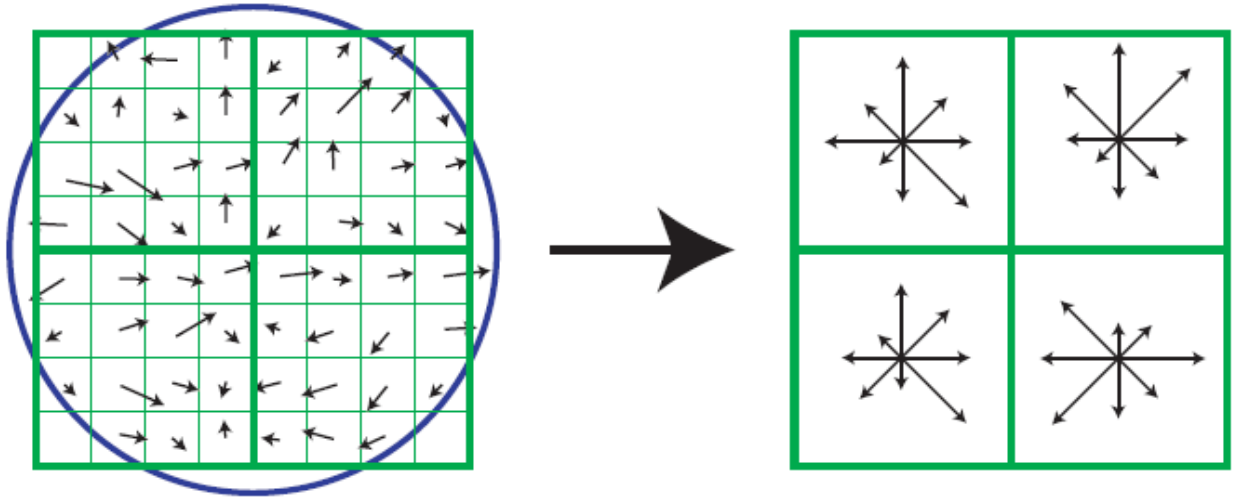


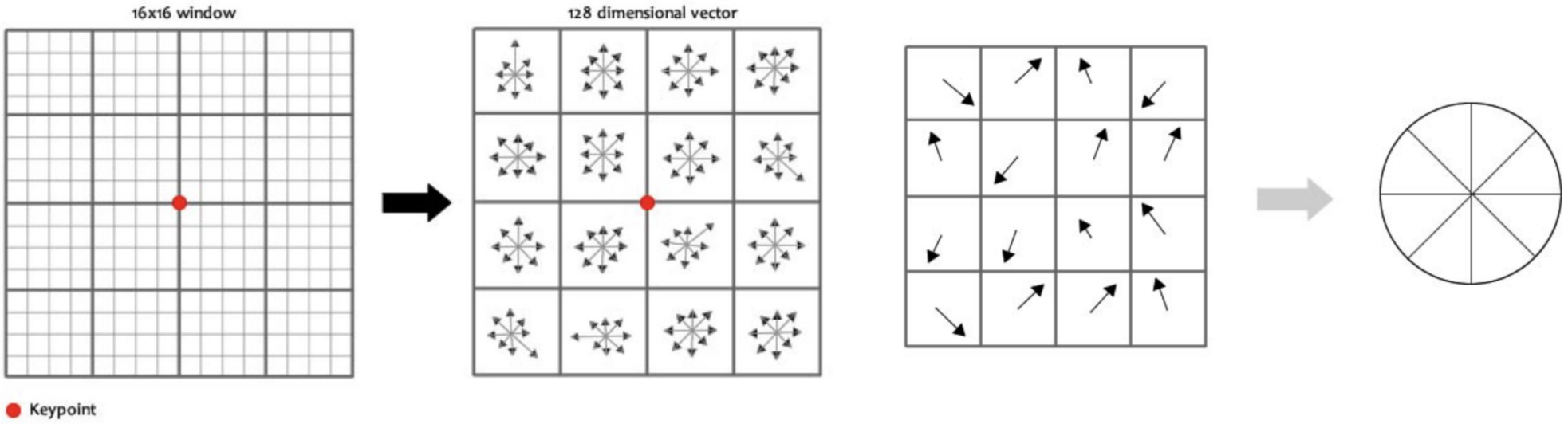
Image gradients

Keypoint descriptor

- These histograms form the SIFT descriptor vector

SIFT Descriptor

- a 16x16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size.
- For each sub-block, 8 bin orientation histogram is created.
- So 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 X 8 directions give 128 bin values



Why does it work?

- Feature descriptor is calculated relative to locally dominant scale and rotation
  - Giving it some invariance to geometric transformations
- Entirely based on image gradients
  - Giving it some invariance to photometric transformations
- Several other detector/descriptor methods based on the same principles
  - SURF, BRIEF, ORB, AKAZE, GLOH, DAISY etc

SIFT in OpenCV

```
import numpy as np
import cv2 as cv

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

sift = cv.SIFT_create()
kp = sift.detect(gray,None)

img=cv.drawKeypoints(gray,kp,img)

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

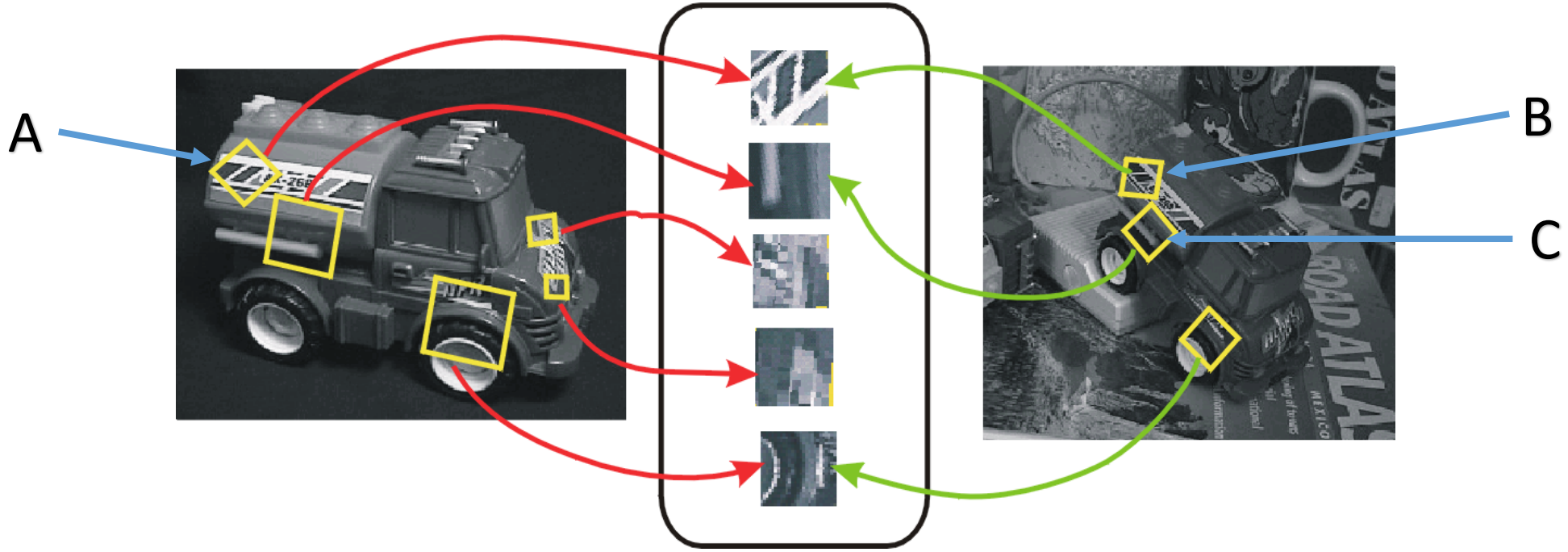
image

```
sift = cv.SIFT_create()
kp, des = sift.detectAndCompute(gray,None)
```

Here kp will be a list of keypoints and des is a numpy array of shape $(\text{Number of Keypoints}) \times 128$.

Deep Learning based Descriptors

- What's our expectation of descriptors?
  - Generate similar features for the matched patches (AB)
  - Generate dissimilar features for the mismatched patches (AC)



Deep Learning based Descriptors

- HardNet
  - Network
    - It takes an image patch as input, and it outputs a feature vector

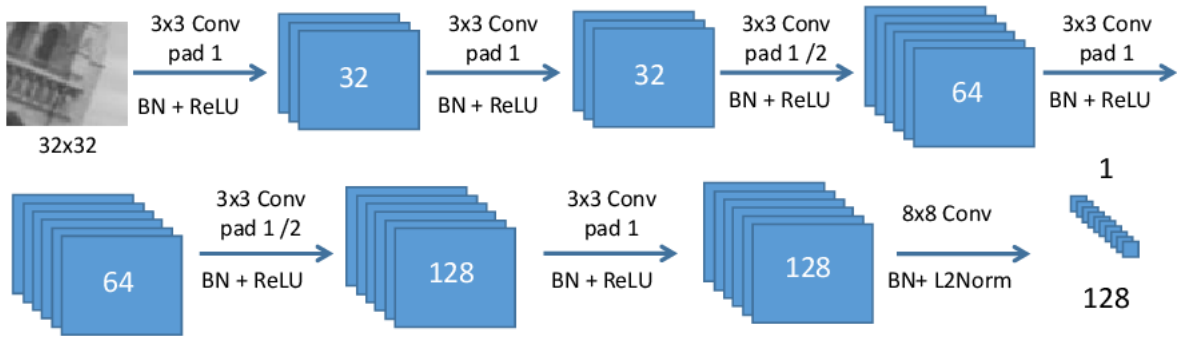


Figure 3.1: HardNet descriptor architecture. Image credit: [39].

  - Training data
    - For each patch, it has a list of matched patches and mismatched patches.

Deep Learning based Descriptors

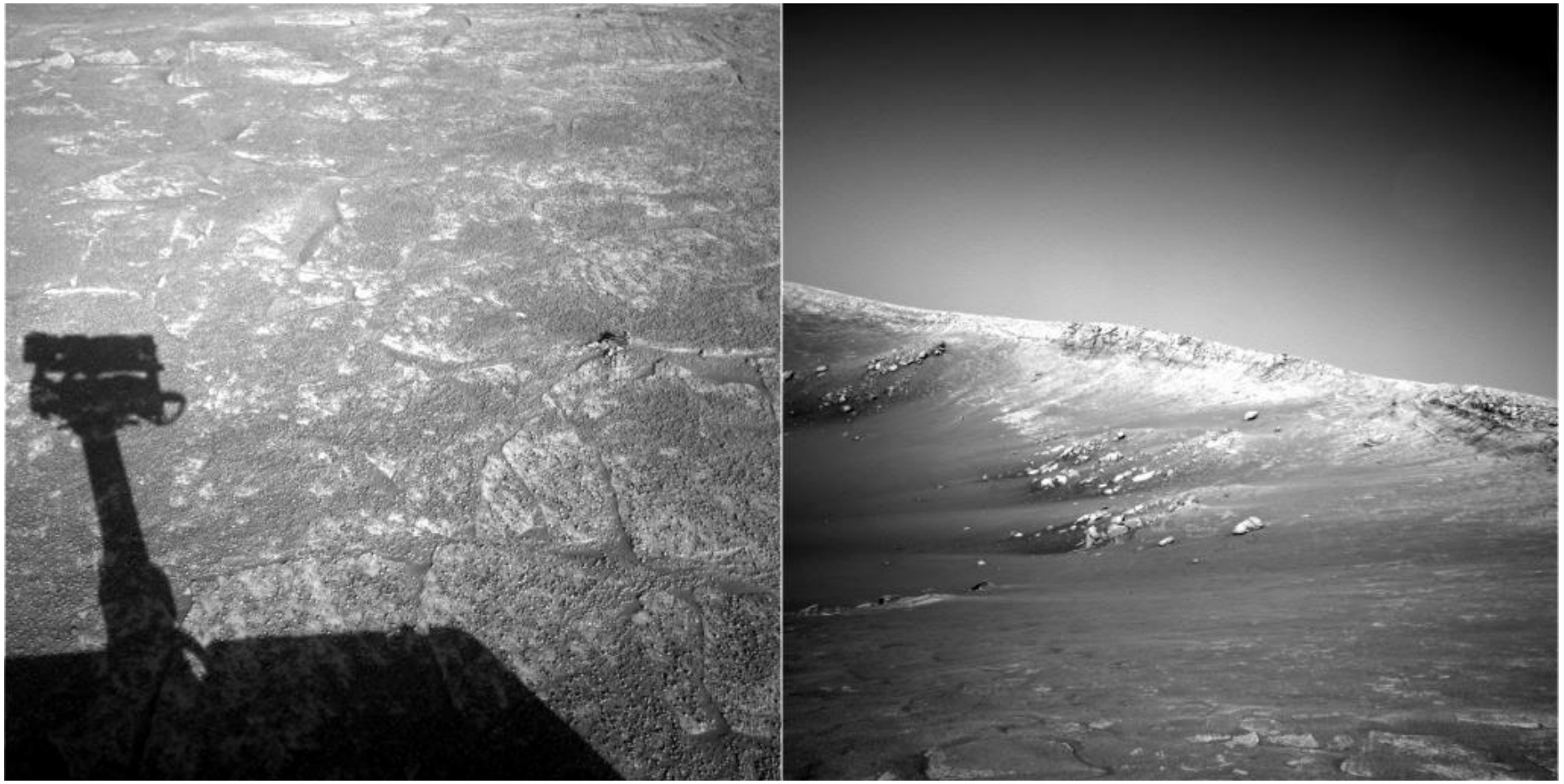
- HardNet
  - A training sample
    - One patch A, a matched patch B, and a mismatched patch C
  - Triplet Loss
    - $L = \min(0, m + d(f(A), f(B)) - d(f(A), f(C)))$
    - Here, m is margin, a constant value, e.g., m = 1
    - d(a,b) means the distance between (a,b)
    - f(a) means the feature of (a)

Summary

- Feature descriptors describe a local patch at a feature location
- Used for mainly for applications that require matching features
- Many options, with trade offs between
  - Complexity to compute, store and compare
  - Invariance to rotation and scale changes
  - Invariance to photometric changes (lighting, contrast etc)

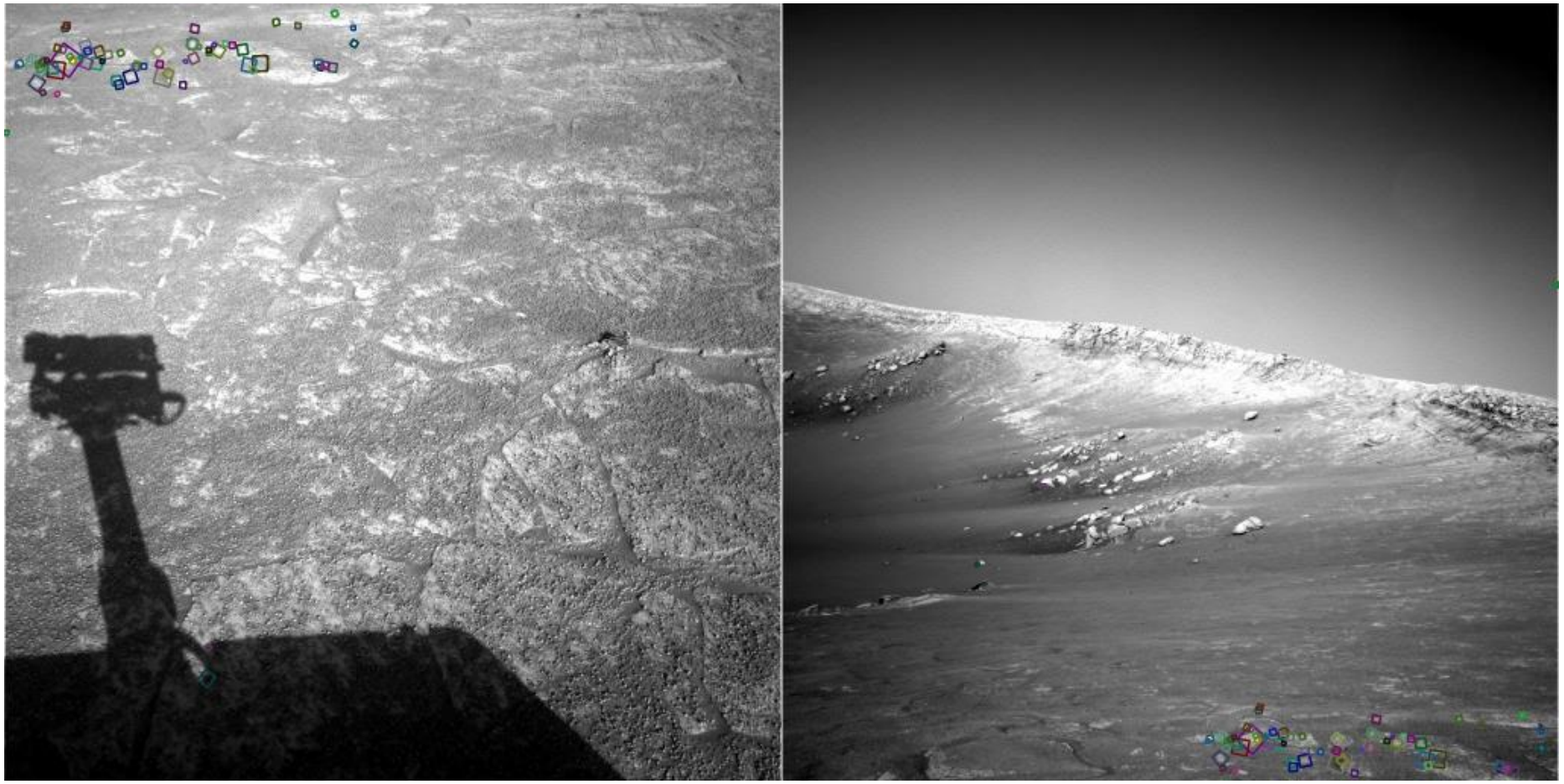
Feature matching

Can you spot the matches?



NASA Mars Rover images

Answer below (look for tiny colored squares…)

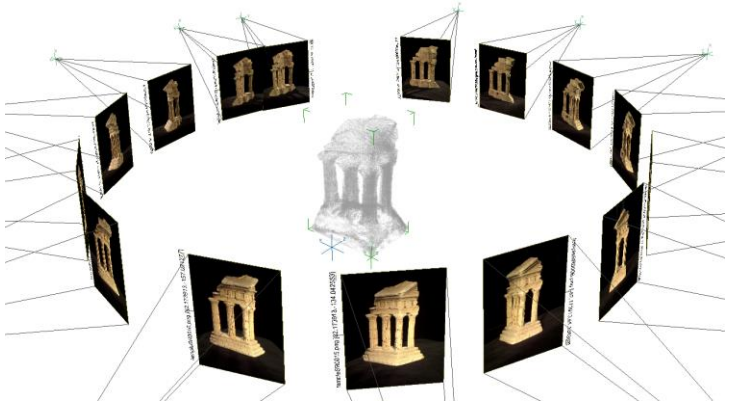


NASA Mars Rover images
with SIFT feature matches
Figure by Noah Snavely

Feature matching

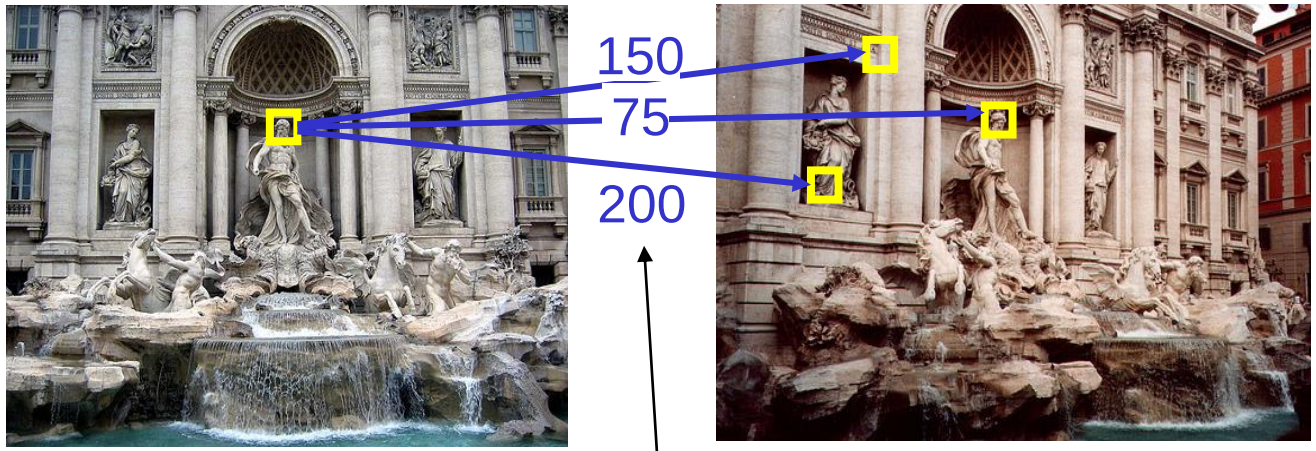
- Earlier: feature detectors and descriptors
- Detect “useful” locations in an image, then describe the area around them
  - For example, the SIFT feature, histograms of gradients

- Today: feature matching, a fundamental step in many applications
  - including image warping (panoramas),
  - object tracking,
  - object retrieval
  - 3D reconstruction from stereo



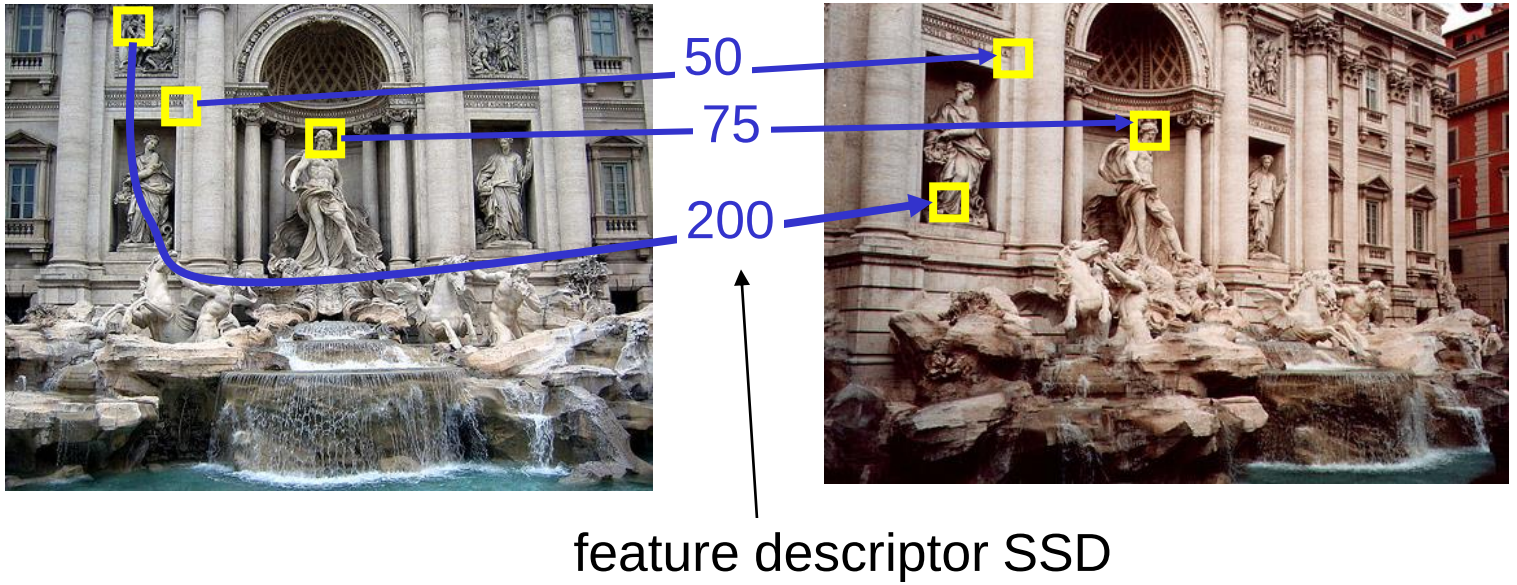
Feature matching

- Now we've done the hard work creating a descriptor, matching is simple! (maybe)
- SIFT descriptor is a vector of length 128 (16 * 8 bin histograms)
- Given feature F_1 in image 1, what feature F_2 in image 2 is best match?
  - F_2 that minimises sum squared difference SSD(F_1, F_2), i.e. $||F_2 - F_1||^2$



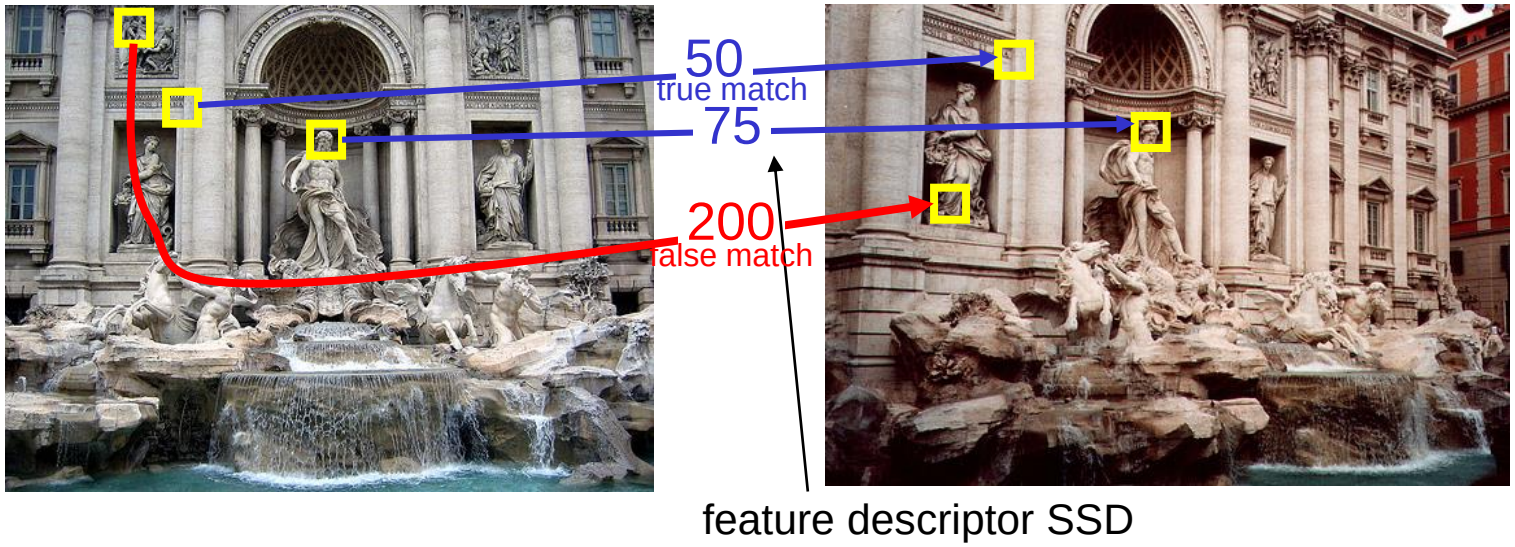
Feature matching

- Keep all matches where the SSD is less than a threshold
- How do we set the threshold?
  - Not to mention other parameters
  - Need to measure overall matching performance



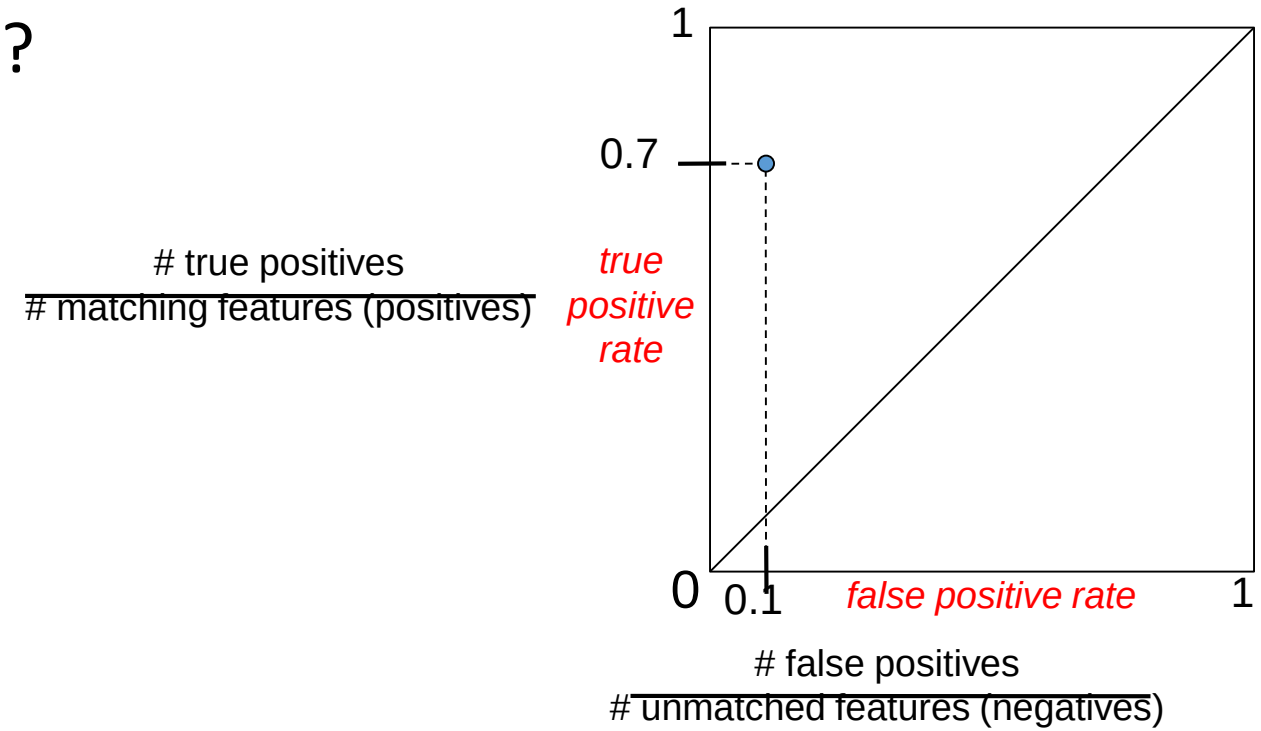
True and false positives

- **True positive** = a correct match
  - How could we maximise the number of true positives?
- **False positive** = an incorrect match
  - How could we minimise the number of false positives?



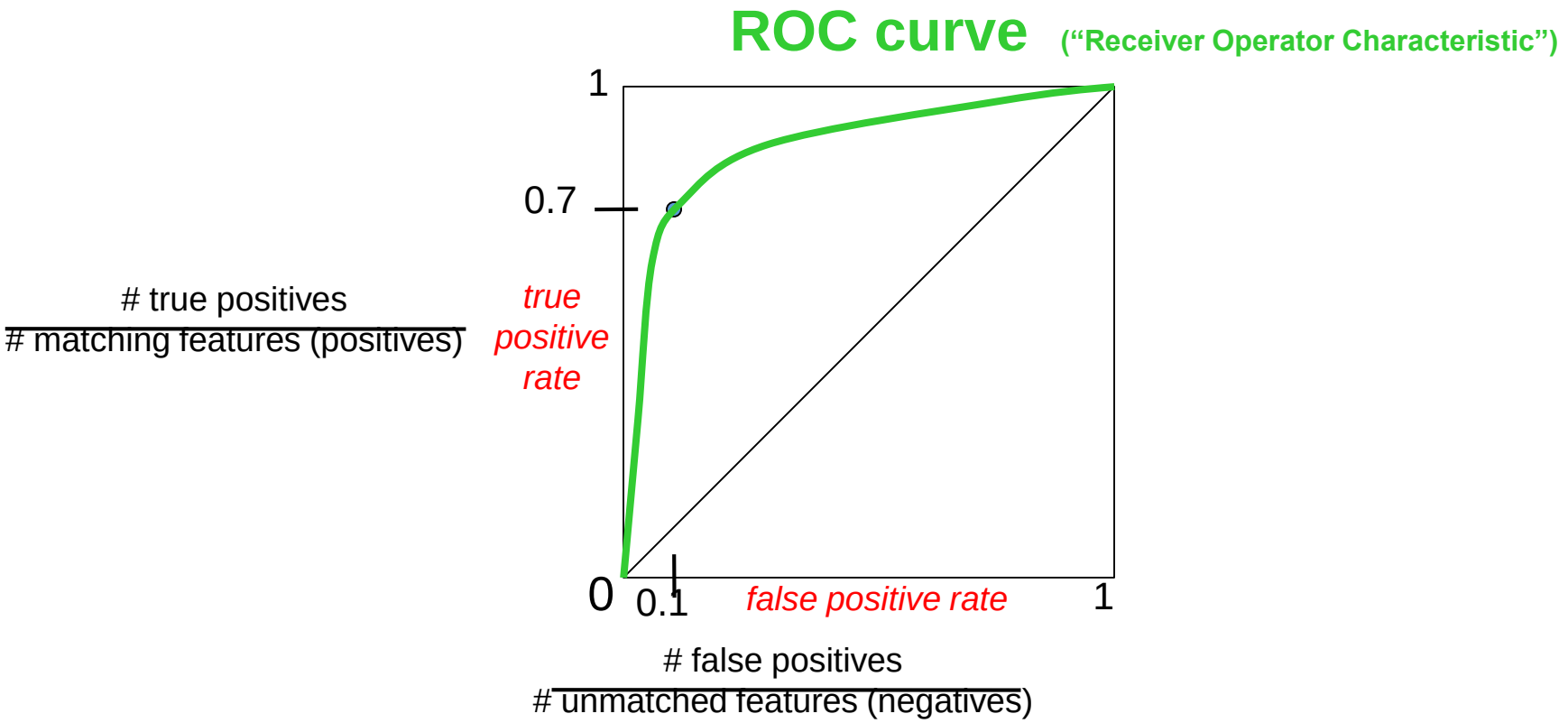
True and false positive rate

- **True positive rate**: what fraction of the actual matches did we correctly find?
- **False positive rate**: what fraction of features did we wrongly assign a match to?



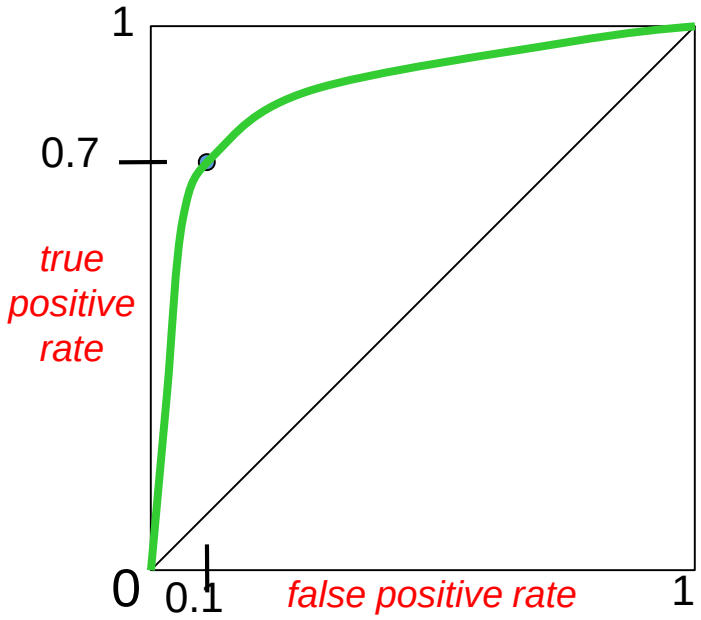
Evaluating the result

- ROC (“Receiver Operator Characteristic”) curve measures this tradeoff, for varying threshold or parameter setting
- Lets us evaluate matching performance independent of these settings



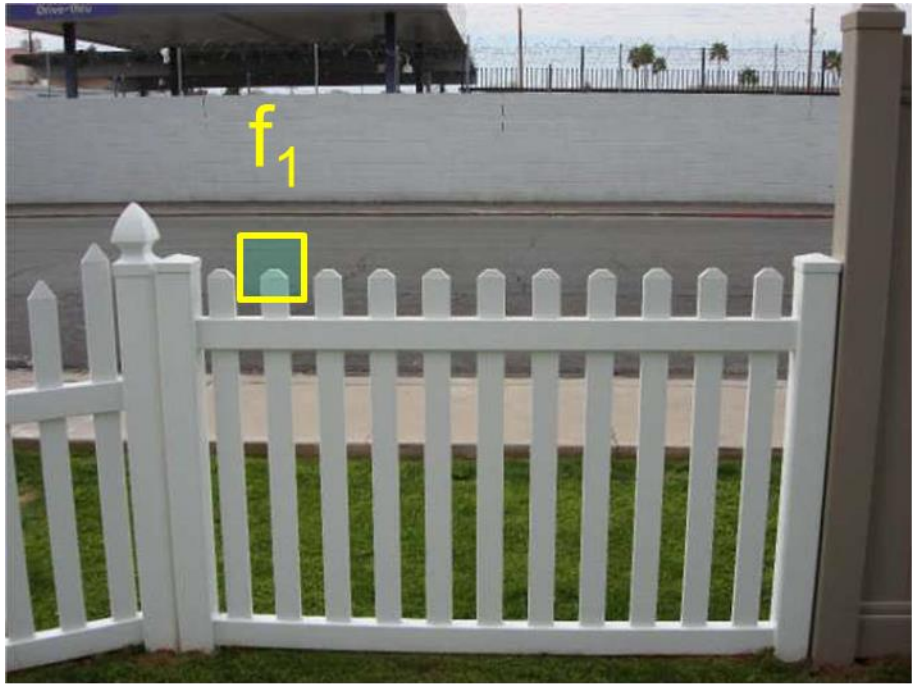
Evaluating the result

- The ROC curve:
  - Generated by counting # correct and incorrect matches, for varying threshold/parameter
  - want to maximise the area under the curve (AUC) – what is the ideal value?
  - Useful for comparing different feature detectors, descriptors, matchers
    - Somewhat independent of specific settings
  - Also used to set threshold value
    - Set tradeoff between true and false error

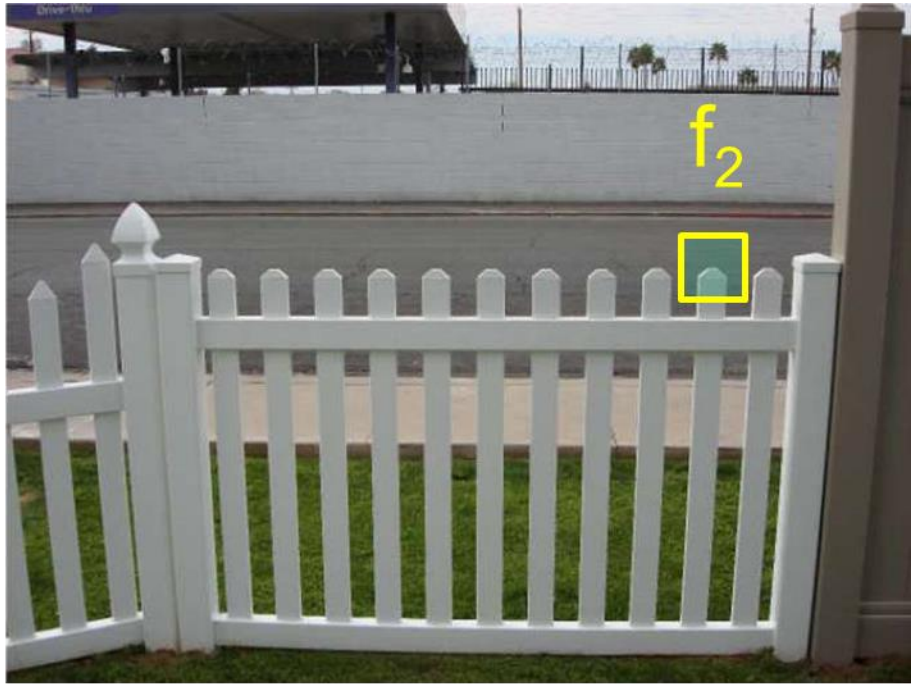


Second nearest neighbour

- Sometimes the SSD can be very small for quite bad (ambiguous) matches:



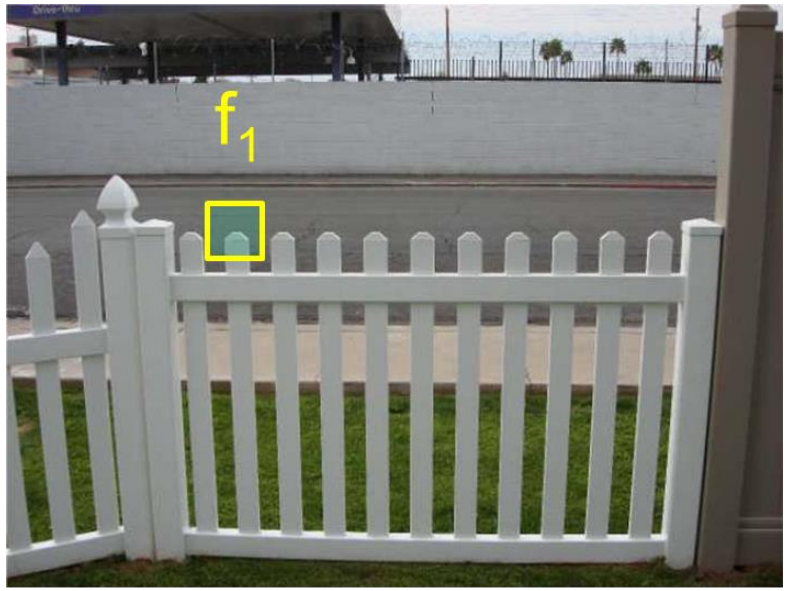
I_1



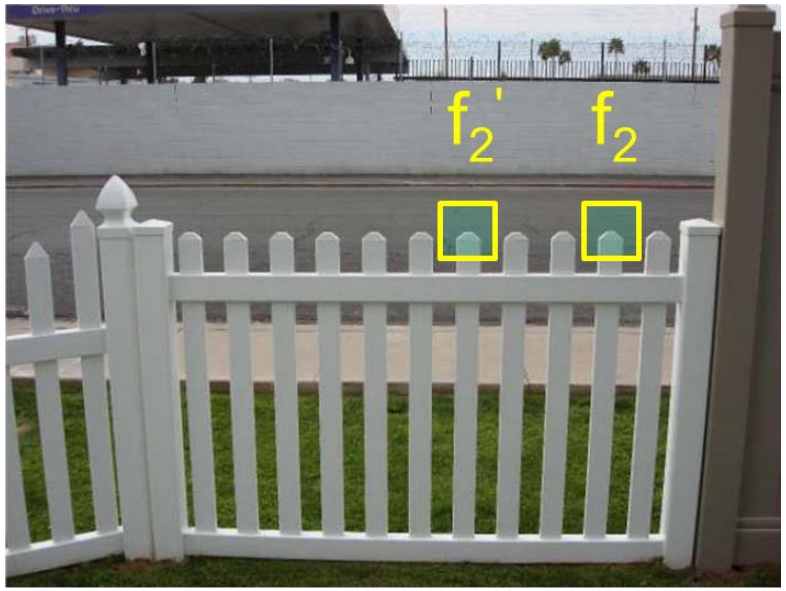
I_2

Second nearest neighbour

- How can we avoid this?
  - Compute the distance ratio SSD(f_1, f_2) / SSD(f_1, f_2')
  - Where f_2' is the second best match to f_1 in image 2
  - For ambiguous matches, this will be small



I_1



I_2

Feature matching

- So far we have just used the matching strength of individual features
  - Threshold based on SSD
  - Second nearest neighbour threshold

- For a static scene, matching point position should change consistently between images
  - All shift to the left, all rotate etc
  - Can we use this to improve matching?

Simple example

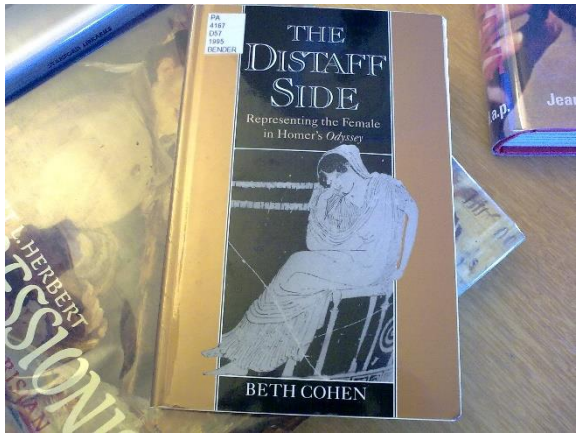
- If one image is just a shifted version of the other:
  - Take our best match
  - Calculate the shift between the feature locations, $T = X_2 - X_1$
  - Only keep matches that agree with this shift

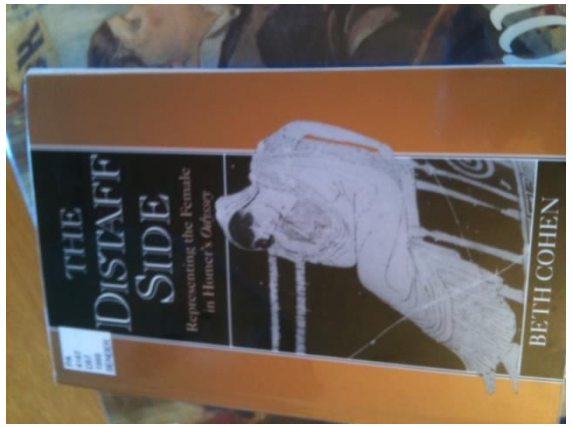
Simple example, part 1

- Take all matches that are over threshold
- Estimate translation between images (e.g. average feature disp)
- Then only keep matches within threshold of this
- Then what if there is a rotation and a scale?
- Linear least squares
- Estimating a warp that aligns the images (useful for panorama)
- But that is not robust due to square error...

Problems with simple example

- What if our strongest match happens to be wrong?
- This problem gets worse for more complex transformations
- E.g. affine transformation, $X_1 = A*X_2 + T$, requires 3 correct matches to estimate
  - We will return to this later…





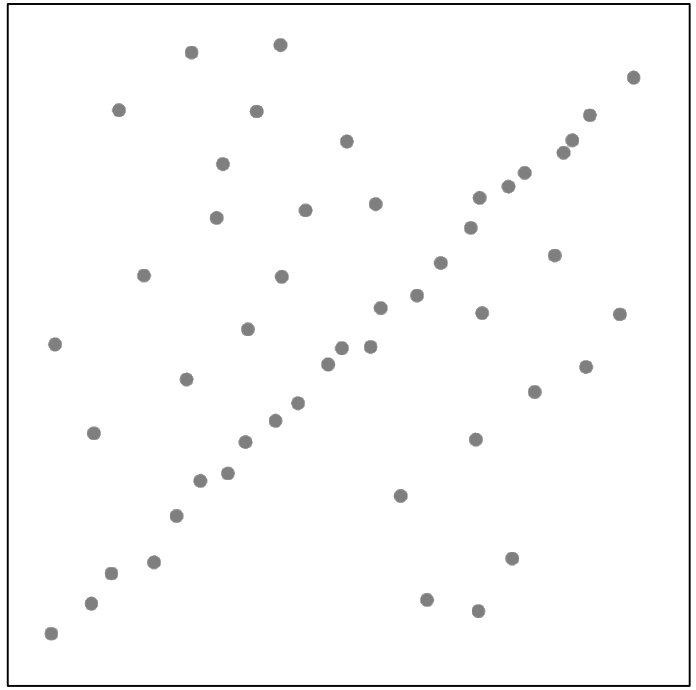
Consensus methods

- What if our estimated transformation happens to be wrong?
- Probably, not many other matches will agree with it.

- Idea:
  - Pick multiple sets of matches and estimate transform for each
  - Count how many other matches agree with them
  - Keep the matches with the highest number of agreement (consensus)

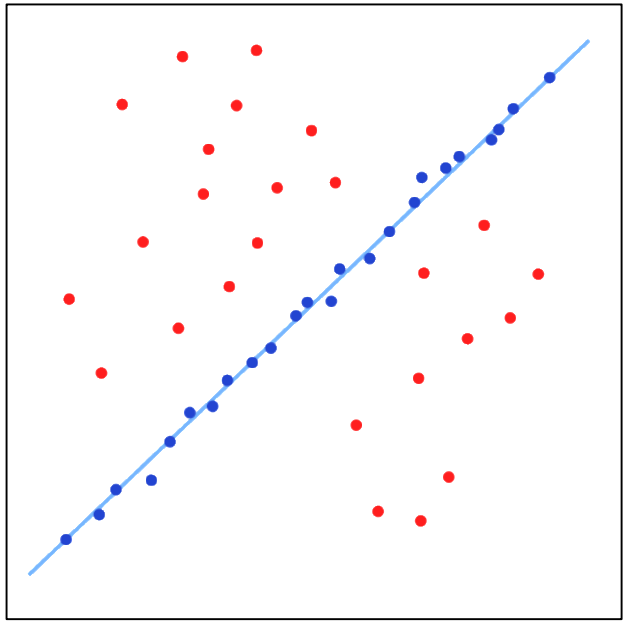
The simplest estimator

- We use all matches to solve a least square error fitting problem
  - Given a hypothesis, each match has a error, and our goal is to find a model that makes the total error is smallest
  - Similar to fit a linear model, e.g., y=ax+b
  - Challenges: there are outliers



RANSAC to remove outliers

- Idea:
  - Pick multiple sets of matches and estimate transform for each
  - Count how many other matches agree with them
  - Keep the matches with the highest number of agreement (consensus)
- Example for line-fitting
  - Pick 10 groups of point sets, and each group has 2 points
  - For each set, fit a, b
  - Count how many other points have y-(ax+b) < threshold
  - Choose the best model



RANSAC algorithm

- RANdom Sampling And Consensus
- Input:
  - Point matches
  - An estimated error rate for the matches (say 40%)
  - A threshold E specifying a fitting tolerance

- 1. Choose a random set of 3 matches
  - What is the probability all are correct?
  - $p = (1 - 0.4)^3 = 0.216$

- 2. Use this set of match to estimate affine transform between images
  - Count the number of matches M for which $|| Ax_1 + T - x_2||^2 < E$

RANSAC

- 3. Repeat steps 1 and 2 enough times to be 99% sure that at least one set of matches is correct
  - How many times is this?
  - Say we choose N sets of matches. The probability none of them contains all correct matches is $q = (1 - p)^N$
  - So the probability at least one is correct is 1- q
  - For 99% confidence we want N such that $q < 0.01$
  - So: $q = (1 - p)^N$

    So $\log(q) = \log((1-p)^N)$

    $= N*\log(1 - p)$

    So $N = \log(q) / \log(1 - p)$

  - In our example, N = 19 repetitions (at least)

- 4. Return the transform that maximises num. matches M, and the matches that agree with it

How to estimate the transform?

- We will return to this a bit later…
- For now, both Matlab and OpenCV have functions to estimate a transformation from matching point sets:

  MATLAB: estimateGeometricTransform(..)
  OpenCV: getAffineTransform(..)
  among others…

- If you provide more matches than required, the transform minimises the least square error
  - $\text{Sum}_i\ (||\ Ax^i_1 + T - x^i_2||^2\ )$
  - i.e. the sum of squared residuals

More problems…

- What is this transformation?
- And how can we estimate it from point matches?
- This is the topic of our next lecture…