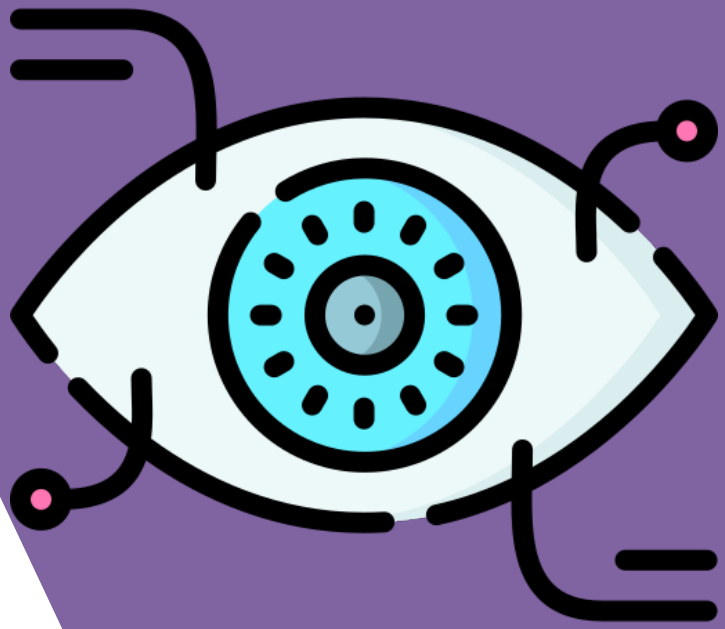
Computer Vision

A.A. 2024-20245

Lecture 6: Local Features – Scale Invariant Feature Transform



SAPIENZA
UNIVERSITÀ DI ROMA

ALCOR Lab

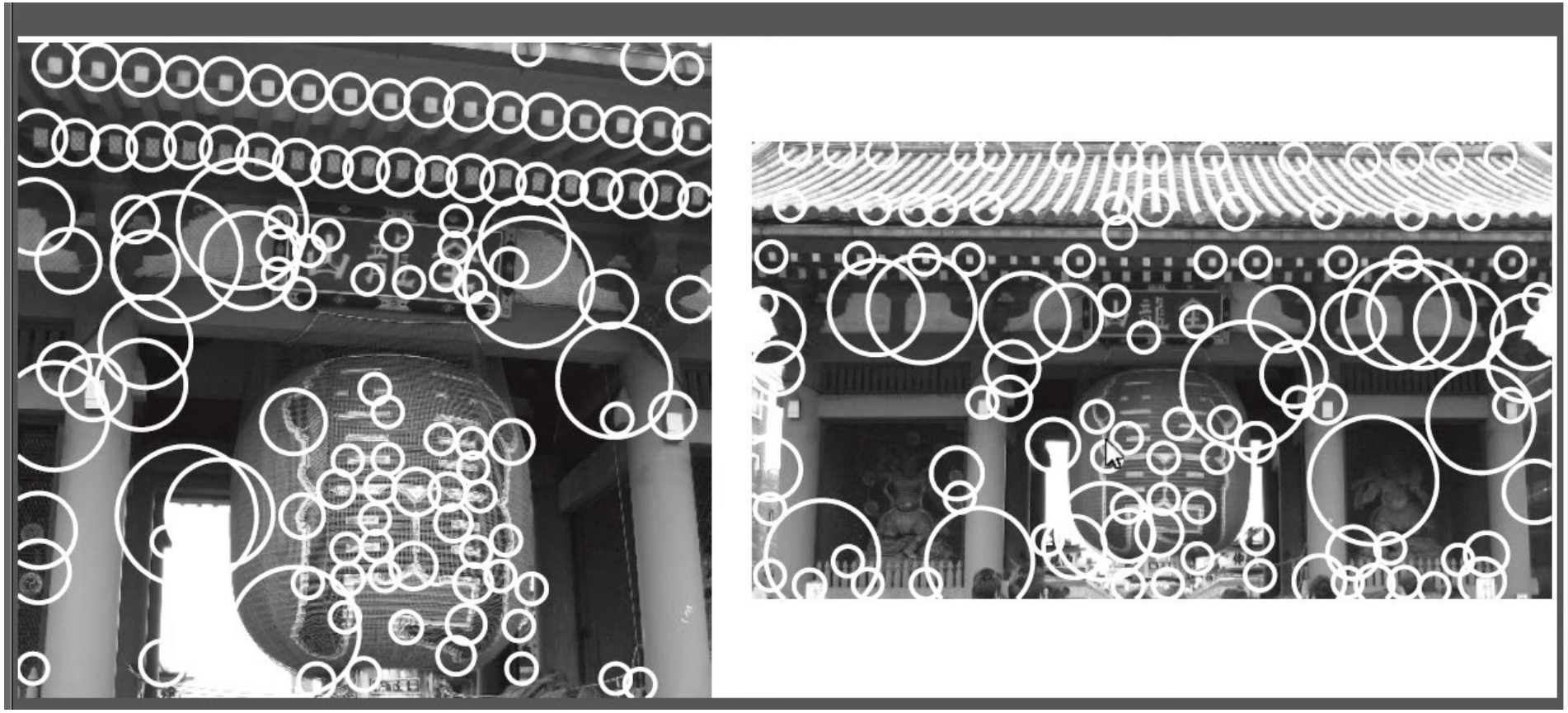
Reading

Szeliski: Chapter 7.1, 8.1, 8.2

Today’s class

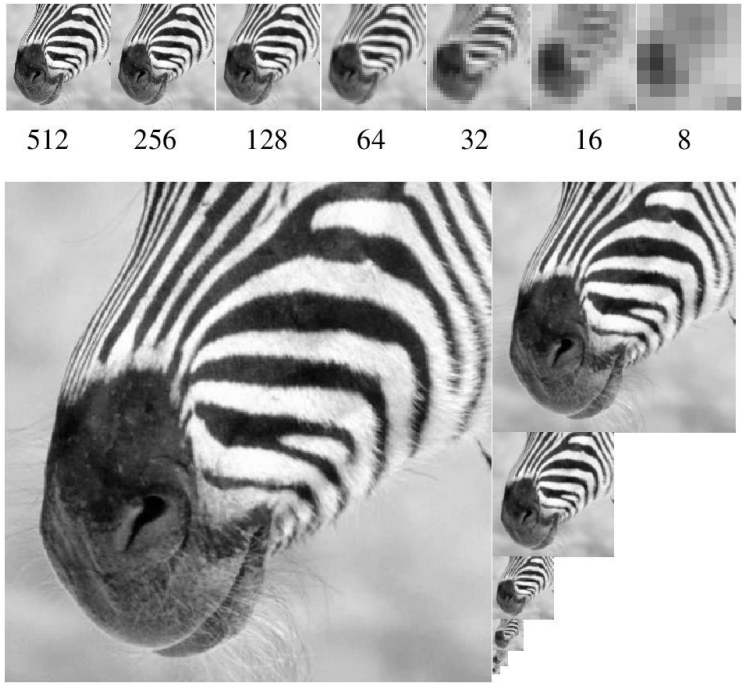
- SIFT detector
- SIFT descriptor
- Feature Matching
- Evaluating Results

Scale-space blob detector: Example



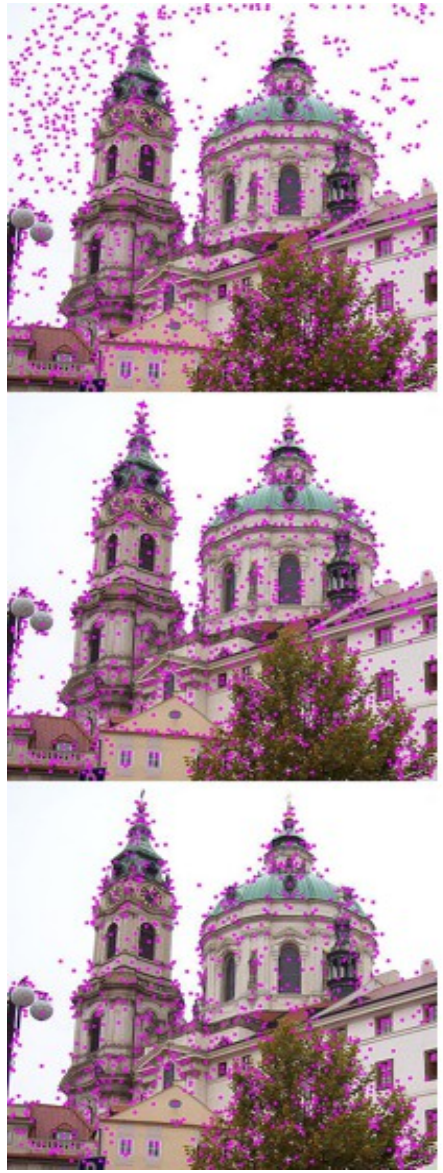
Scale Space

Proper scaling of objects in new image is unknown



Exploring features in different scales is helpful to recognize different objects.

- Blob detector-→ LoG σ acts as a scaling parameter.
- SIFT algorithm uses Difference of Gaussians which is an approximation of LoG.



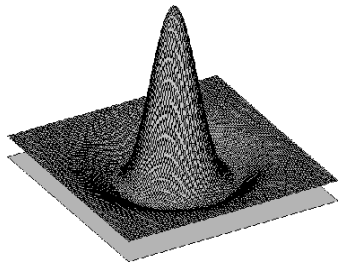
SIFT

(Scale Invariant Feature Transform)

SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

Find local maxima in 3D position-scale space



$L_{xx}(\sigma) + L_{yy}(\sigma)$

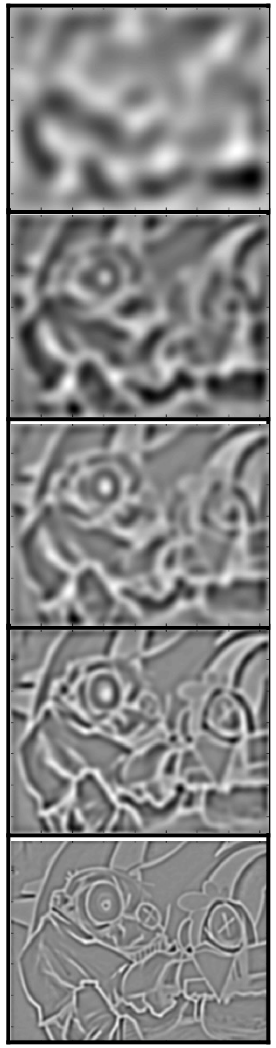
σ^5

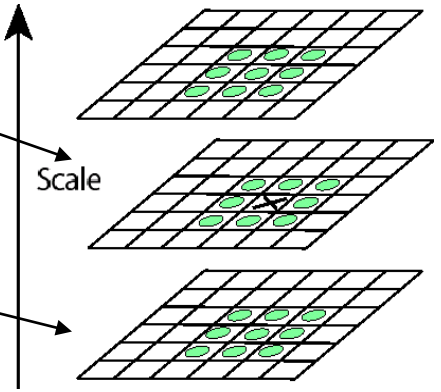
σ^4

σ^3

σ^2

σ





$\Rightarrow$ **List of**
(x, y, s)

Approximating Laplacian of Gaussian

- Functions for determining scale $f = \text{Kernel} * \text{Image}$

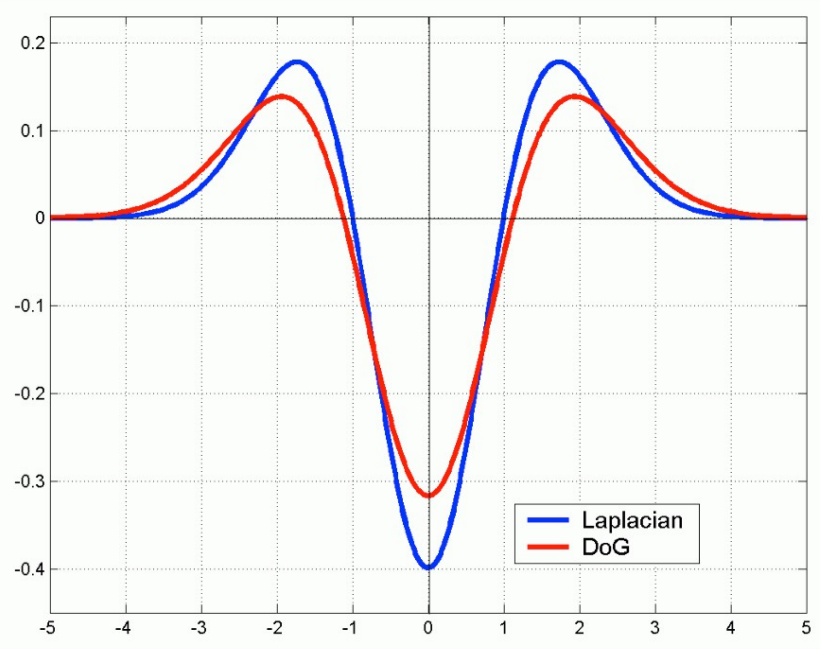
Kernels:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)



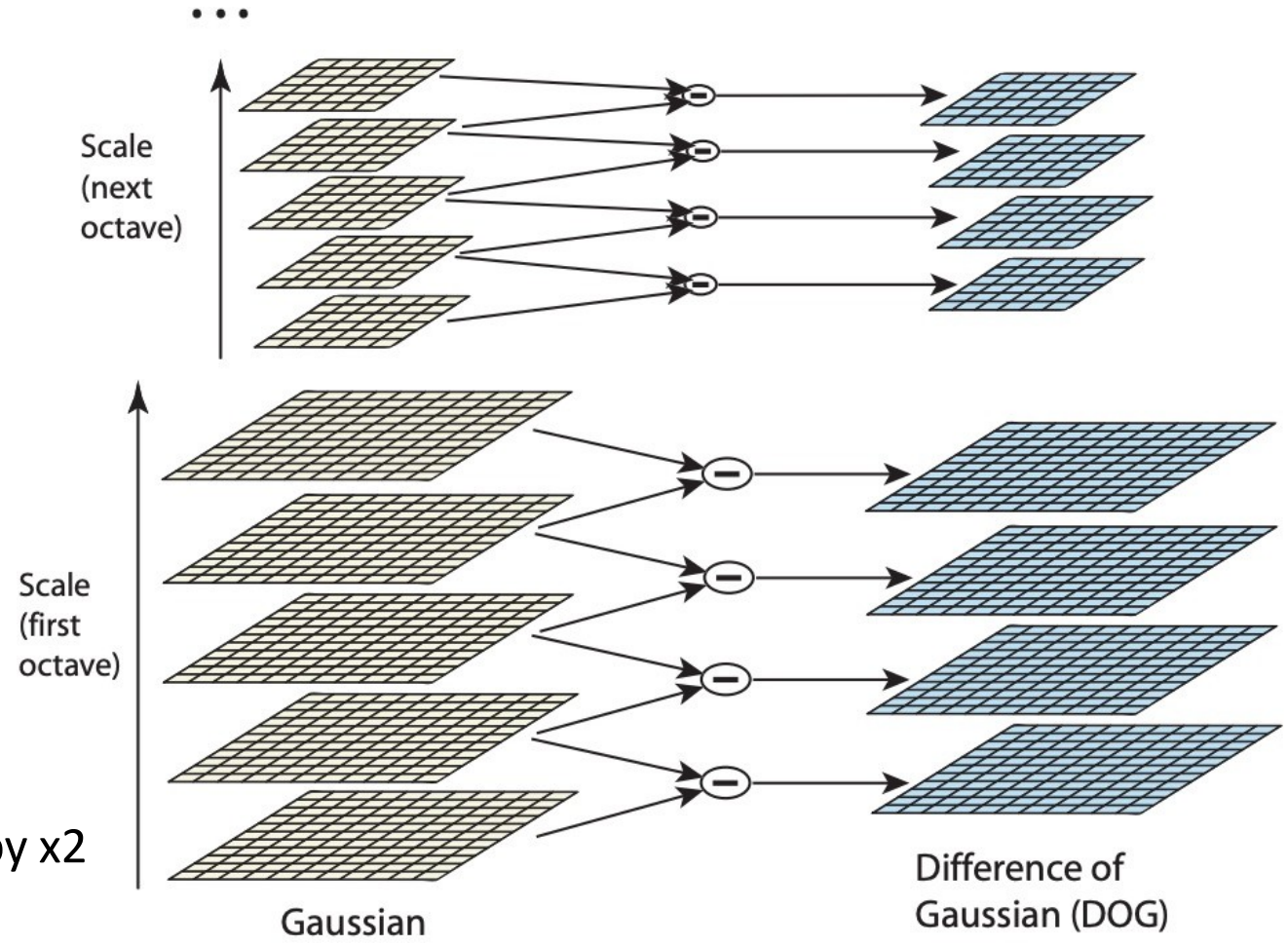
where Gaussian

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

Note: The LoG and DoG operators are both rotation equivariant

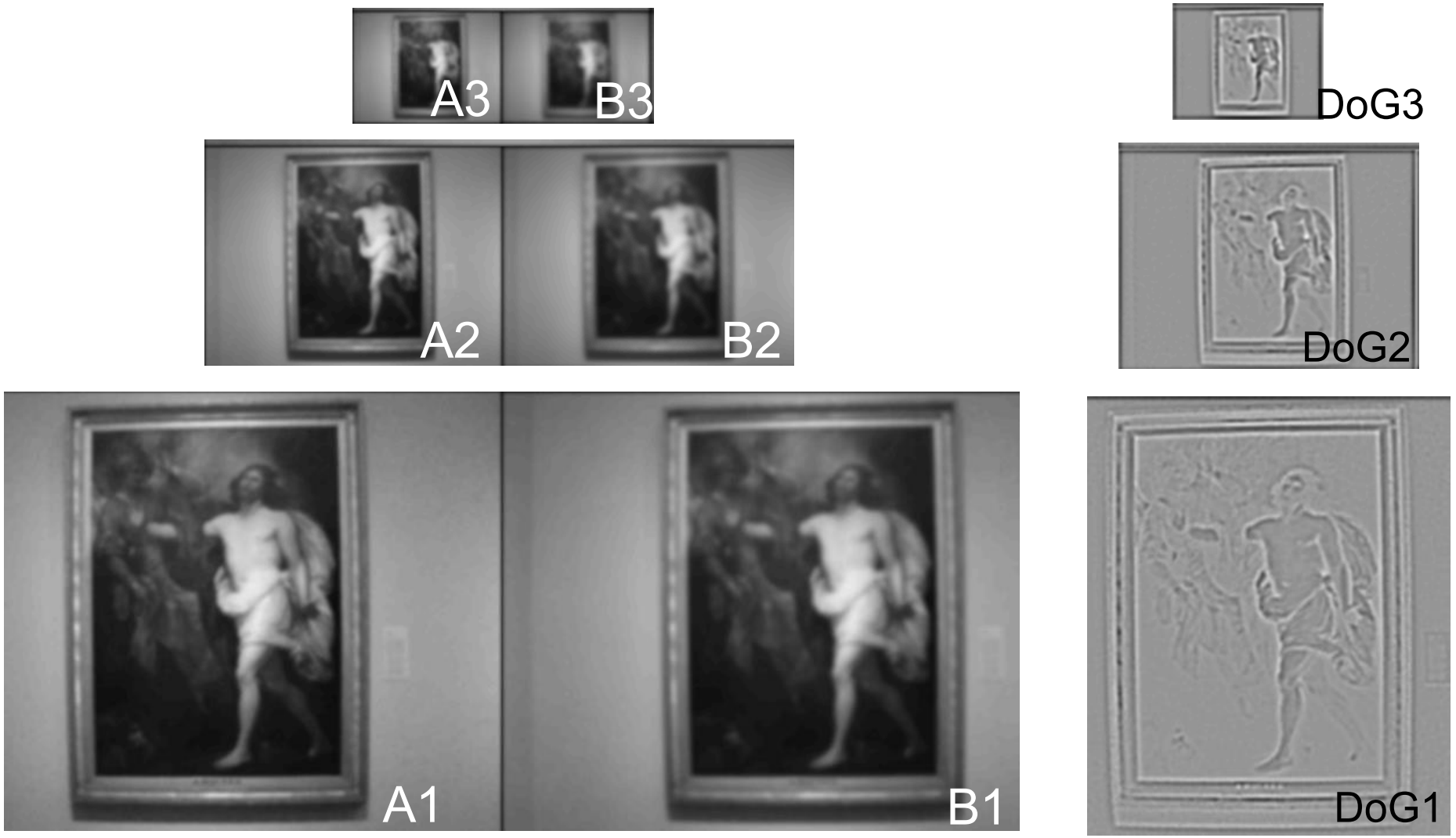
SIFT detector

- Approximate LoG with a *difference of Gaussians* (DoG)
  - Laplacian:
    $\sigma^2\left(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)\right)$
  - DoG:
    $G(x,y,k\sigma) - G(x,y,\sigma)$
- Compute DoG via an *image pyramid*



- In each Octave you progressively blur the image
- To go to next Octave you downsample the image by x2

DoG Example

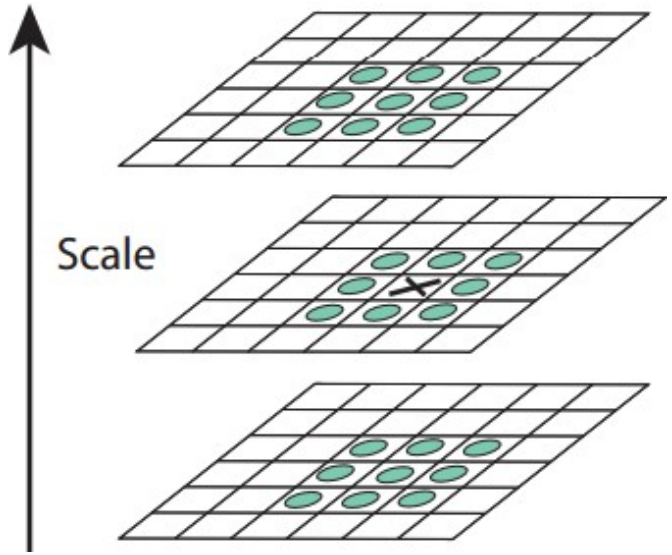


Ashley L. Kapron

Scale space extrema detection

Find maxima and minima of scale space:

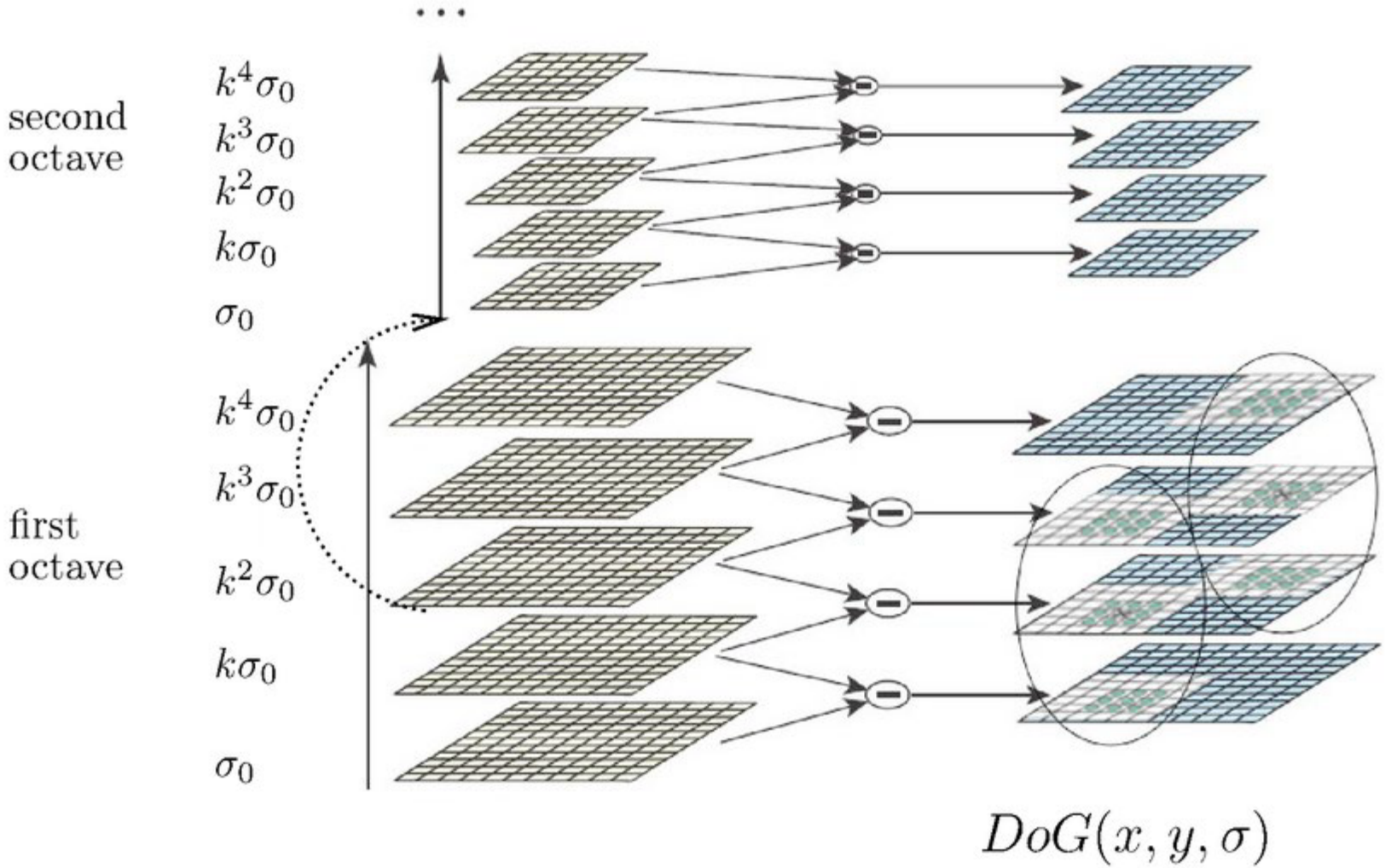
- For each point on a DOG level:
  - Compare to 26 neighbors at adjacent level
    (within the current image, the one above and below it)
- Repeat for each DOG level
- If the point is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale.
- The keypoint is represented as $\mathbf{x} = \{x, y, \sigma\}$
- We know the scale at which the keypoint was detected: scale invariance.



Regarding the different parameters: number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, $k = \sqrt{2}$ as optimal values.

David G. Lowe, IJCV 2004

Pyramid of DoG

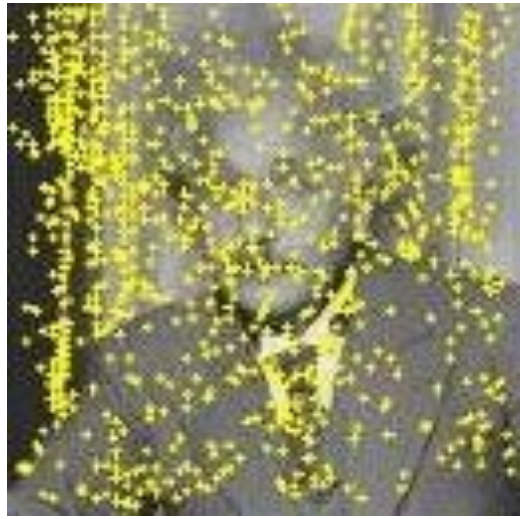


SIFT keypoint stability - Illumination

- Removing low contrast features:
  - If the magnitude of the intensity of the blurred image at the current pixel in the DoG is less than a certain value, it is rejected
    - Remove all keypoints with M_ij less than 0.1 times the max value

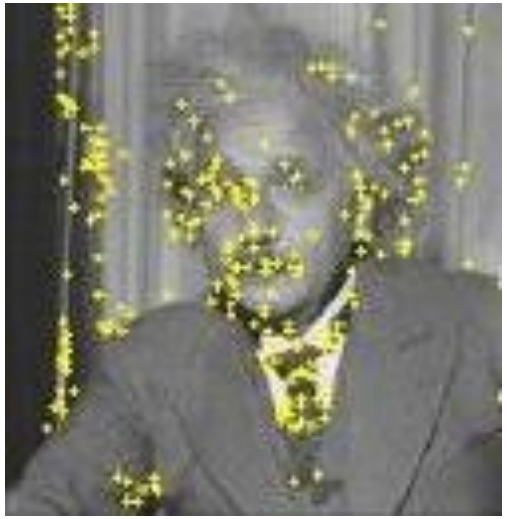
- Motivation: Low contrast is generally less reliable than high for feature points

Example of keypoints



Max/mins from DOG pyramid

Filter by illumination thresholding

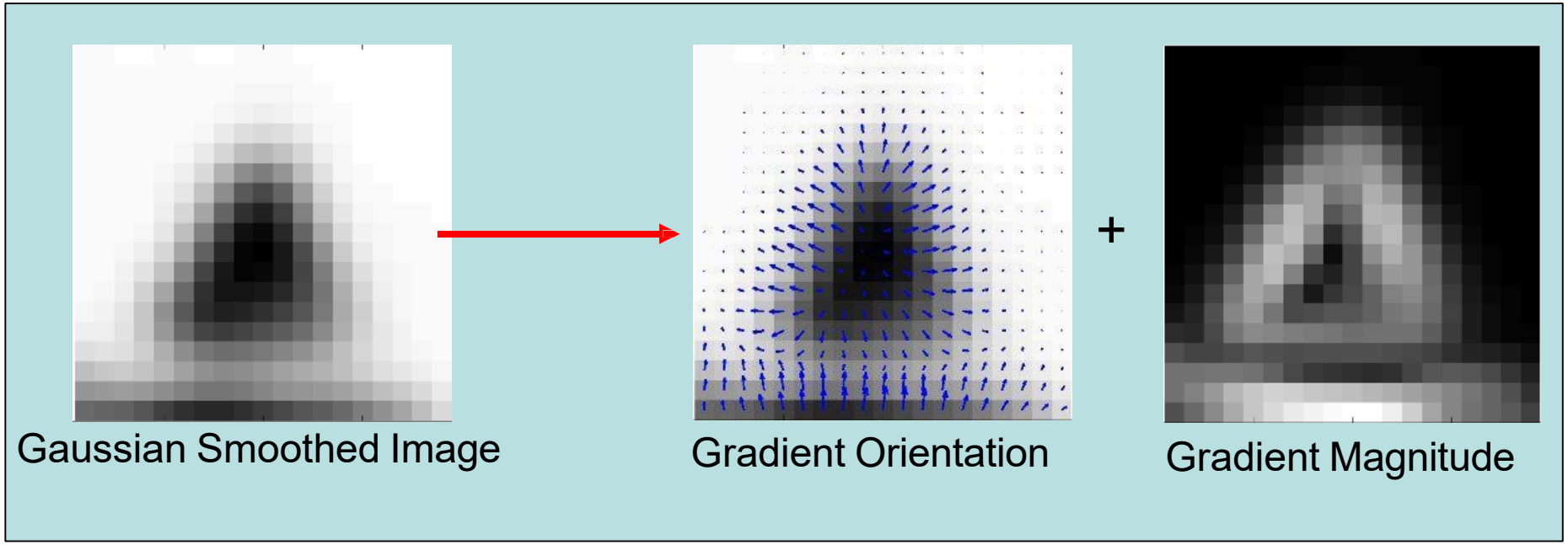


Removing edge (keep only corner)

Gradient-based filtering:
Keypoints located at low gradient regions or along weak edges are discarded.

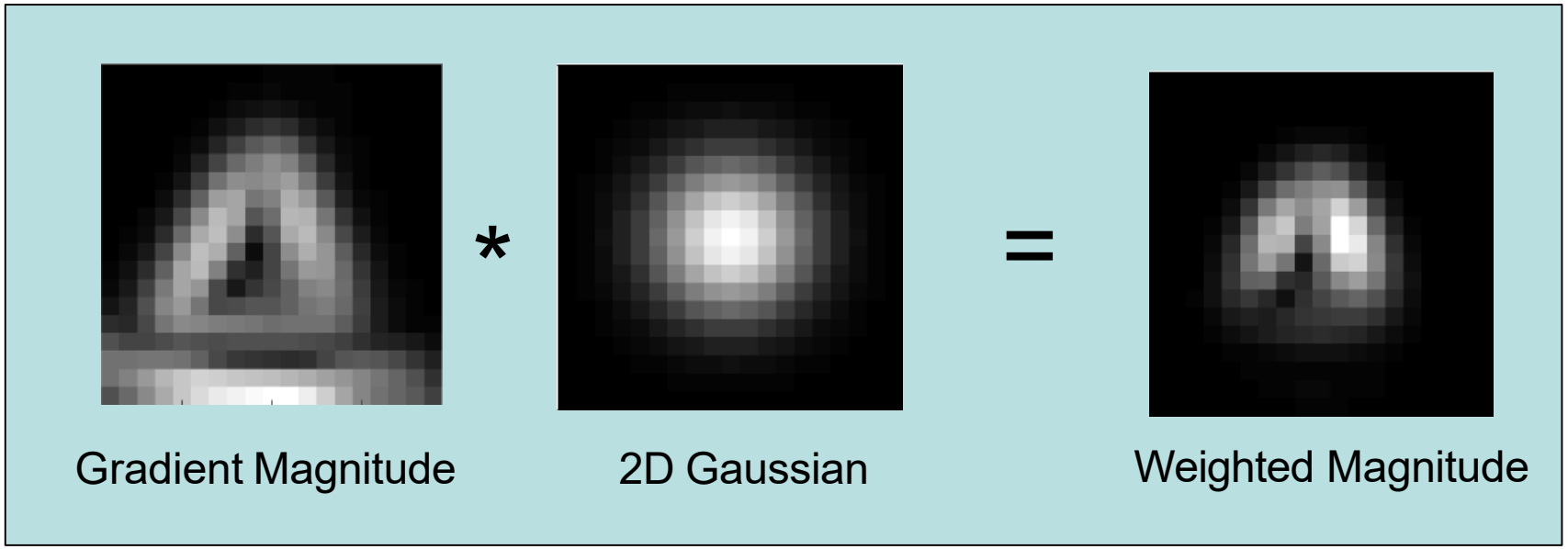
Keypoint Orientation

- The next thing is to assign an orientation to each keypoint.
- This orientation provides rotation invariance.
- For all levels, compute
  - Gradient magnitude and orientation (the size of the "orientation collection region" around the keypoint depends on its scale)



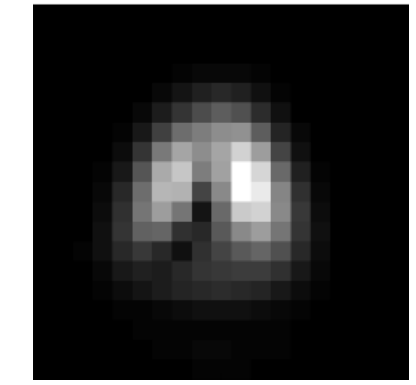
Keypoint Orientation

- Gradient magnitude weighted by 2D gaussian

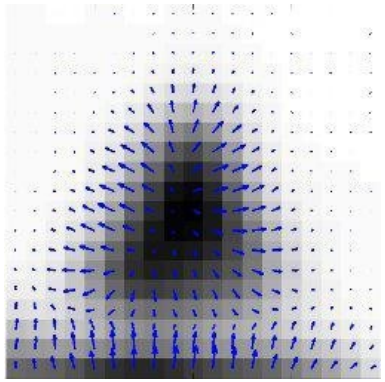


Keypoint Orientation

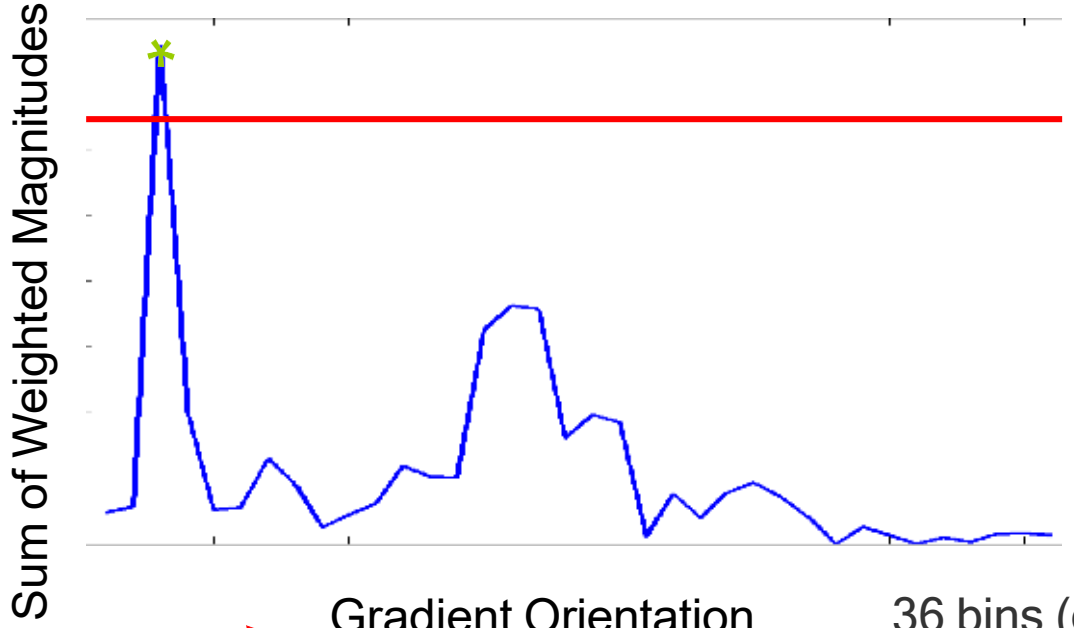
- Build a histogram of orientations in term of sum of weighted magnitude
  - the 360 degrees of orientation are broken into 36 bins (each 10 degrees)
- Identify peak and assign orientation and sum of magnitude to keypoint



Weighted Magnitude



Gradient Orientation



Ashley L. Kapron

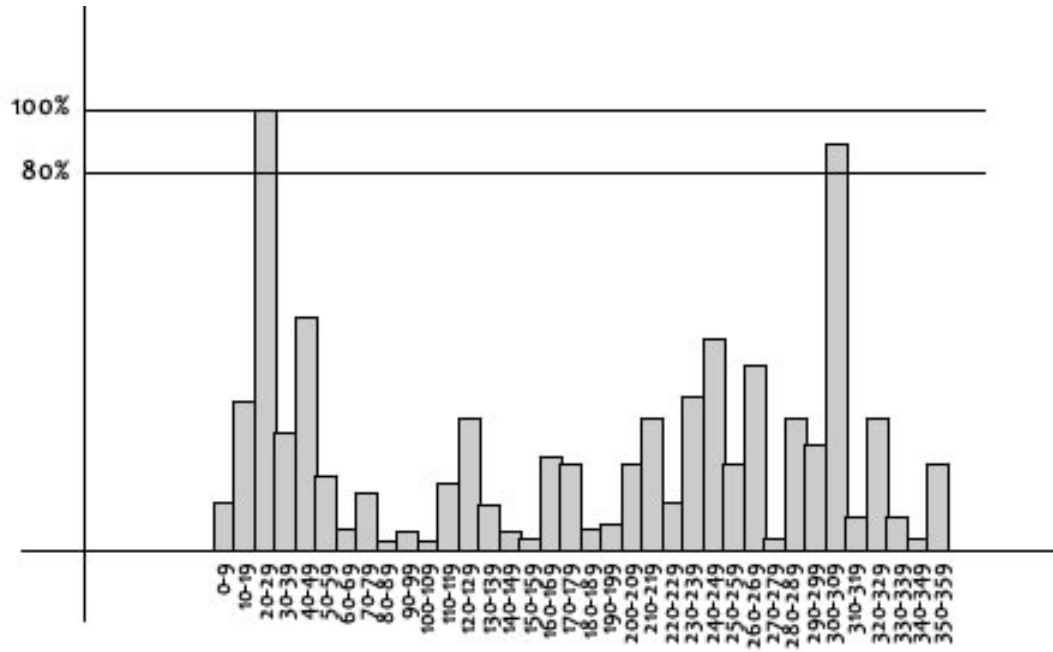
Keypoint Orientation (example)

Example: gradient direction is 18.759 degrees, then it will go into the 10-19 degree bin. And the "amount" that is added to the bin is proportional to the magnitude of gradient at that point.

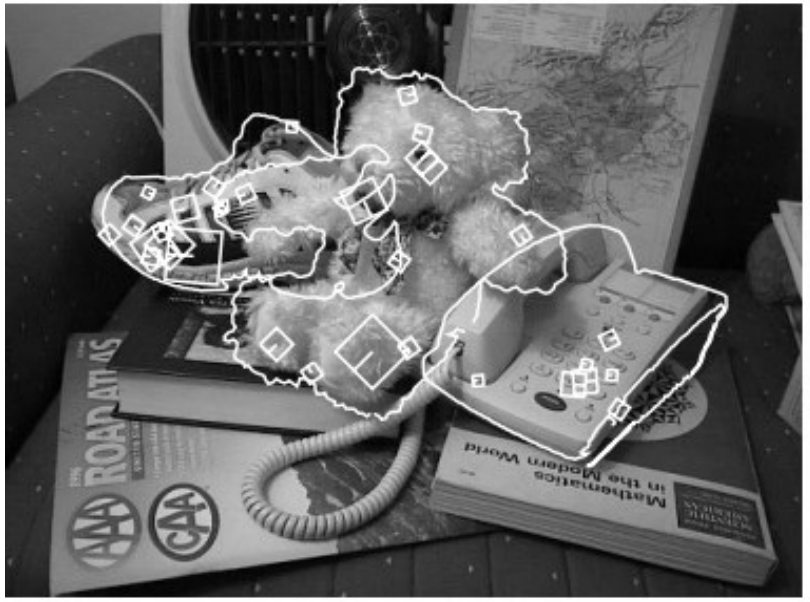
- Once you've done this for all pixels around the keypoint, the histogram will have a peak at some point.
- The histogram peaks is at 20-29 degrees. So, the keypoint is assigned orientation 3 (the third bin)

Any peaks above 80% of the highest peak are converted into a new keypoint.
This new keypoint has the same location and scale as the original. But its orientation is equal to the other peak.

So, orientation can split up one keypoint into multiple keypoints.



SIFT: Scale-invariant feature transform



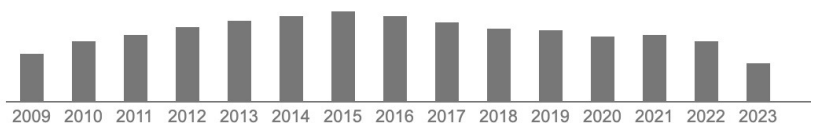
D. Lowe. Object recognition from local scale-invariant features. ICCV 1999

D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV* 60 (2), pp. 91-110, 2004

David Lowe

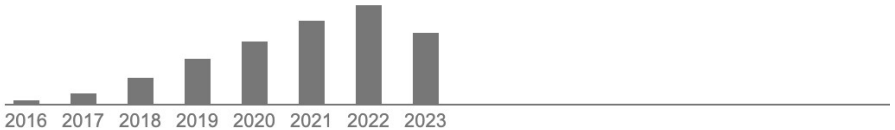
Distinctive image features from scale-invariant keypoints

Authors	David G Lowe
Publication date	2004/11/1
Journal	International journal of computer vision
Volume	60
Issue	2
Pages	91-110
Publisher	Springer Netherlands
Description	This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through …
Total citations	Cited by 71971



Deep Residual Learning for Image Recognition

Authors	Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Publication date	2016
Conference	Computer Vision and Pattern Recognition (CVPR), 2016
Description	Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers. The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.
Total citations	Cited by 185084



Today’s class

- SIFT detector
- SIFT descriptor
- Feature Matching
- Evaluating Results

Local Image Description

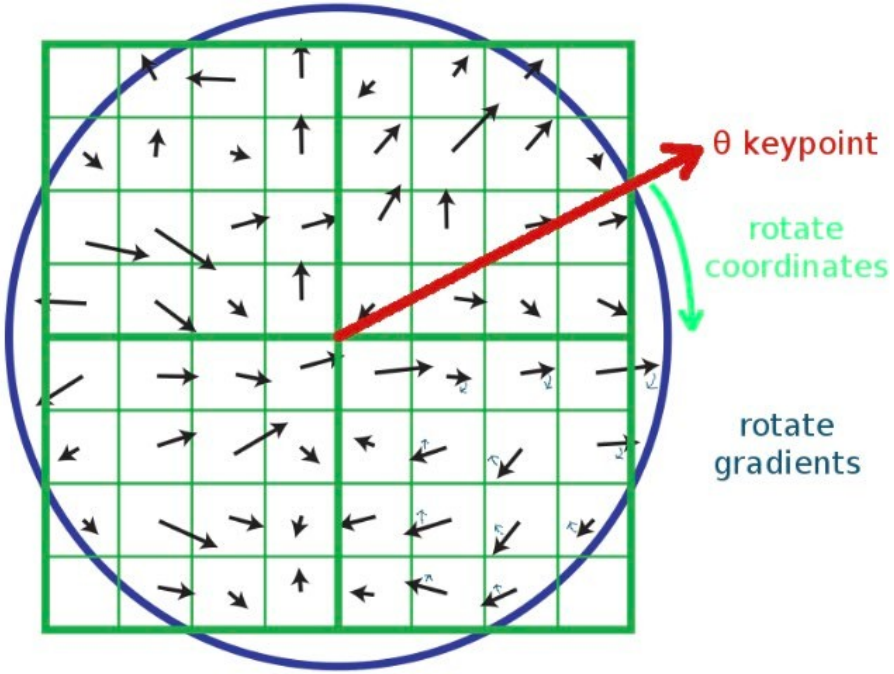
- For each detected keypoint is assigned:
  - Location
  - Scale (analogous to level it was detected)
  - Orientation (assigned in previous orientation step)
- Now: Describe local image region invariant to transformations

SIFT keypoint Example

SIFT descriptor

The steps of building the SIFT descriptor are as following:

1. Use the **Gaussian blurred image** associated with the keypoint's scale
2. Take **image gradients over a 16x16** square window around the detected feature
3. **Rotate the gradient** directions AND locations relative to the keypoint orientation (given by the dominant orientation)

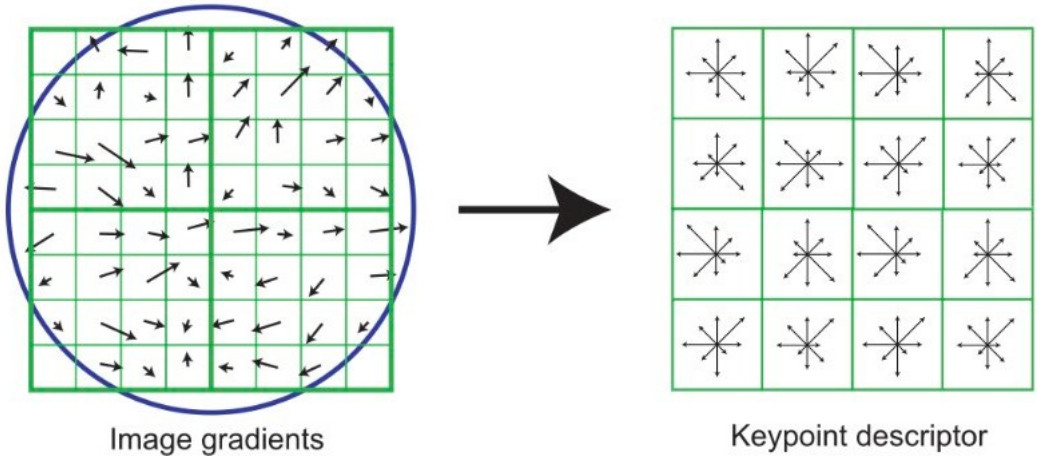


SIFT descriptor

4. Divide the 16x16 window into a 4x4 grid of cells

5. **Compute an orientation histogram** with 8 orientations bins for each cell bins (summing the weighted gradient magnitude)

5.



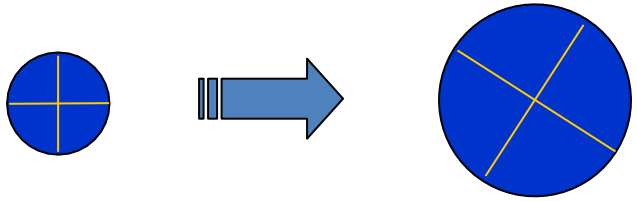
6. The resulting **SIFT descriptor is a length 128 vector** representing a 4x4 histogram array with 8 orientation bins per histogram.

SIFT descriptor properties

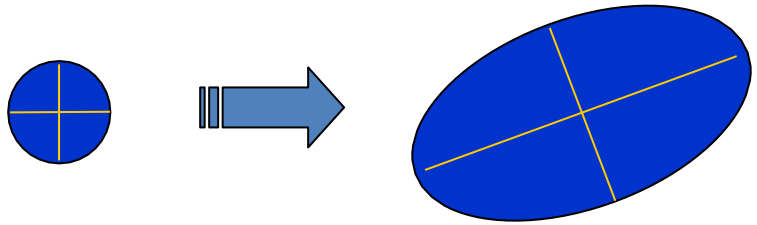
- invariant to rotation because we rotated the gradients: **we are assuming the rotated image will generate a key point at the same location as the original image**.
- Invariant to scale because we worked with the scaled image from DoG.
- Invariant/robustness to illumination variation since we worked with the orientation and we don't take in consideration the magnitude of the gradient.
- Slightly robustness to affine transformation and to noise (empirically found).

Affine Invariant Detection

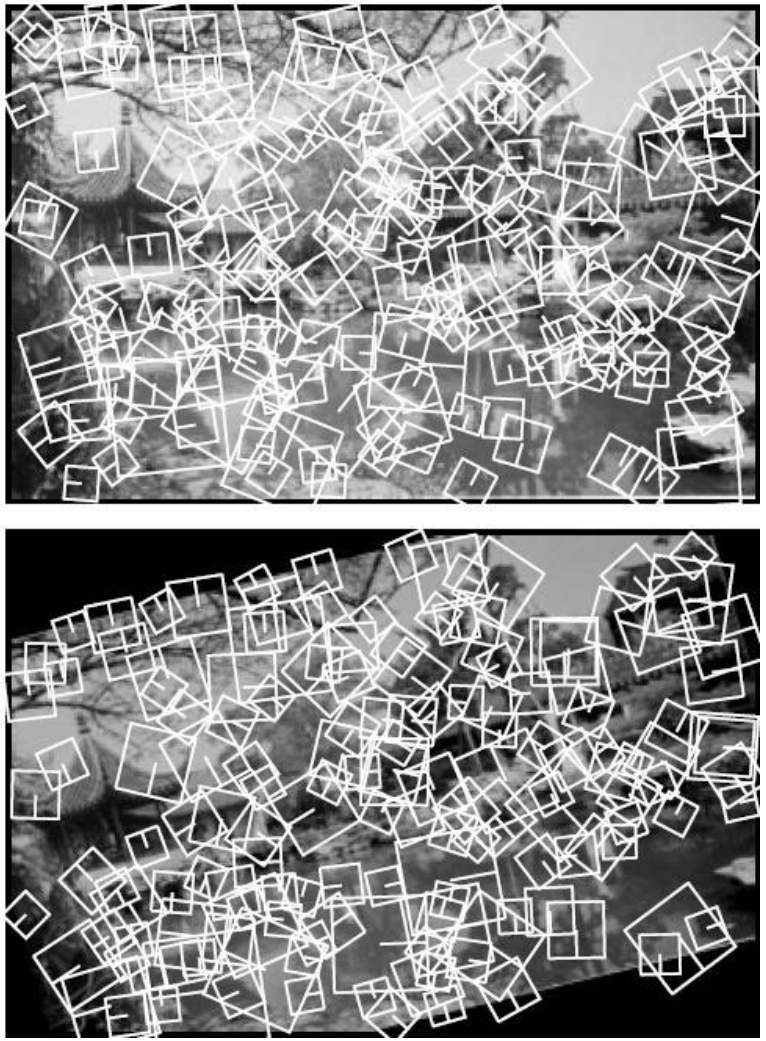
Similarity transform (rotation + uniform scale)



Affine transform (rotation + non-uniform scale)



Stability Test

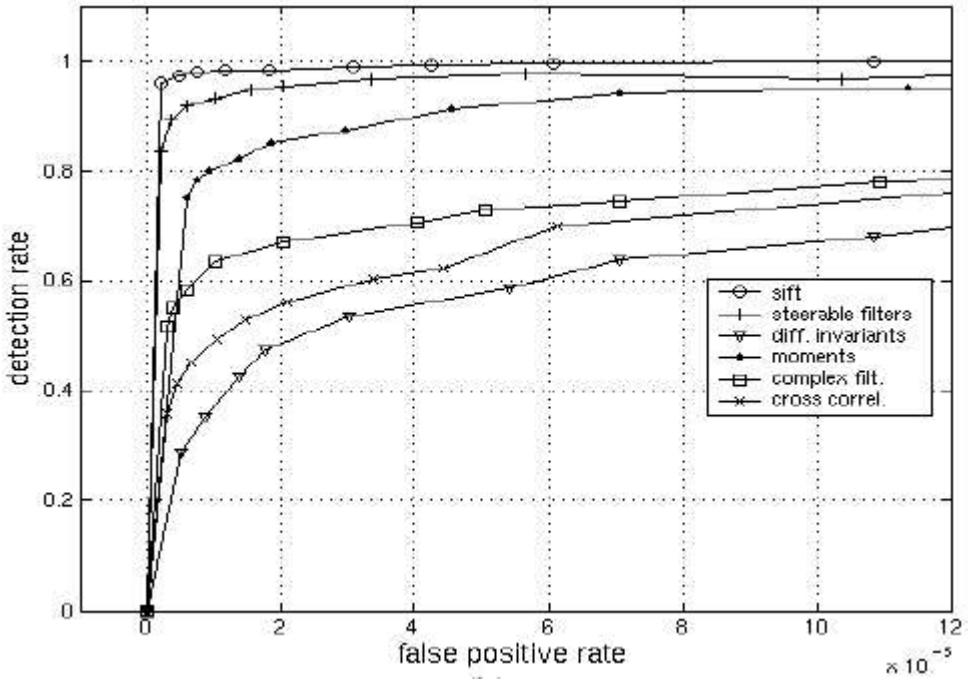


78% of the keypoints survive from rotation, scaling, stretching, change of brightness and contrast, and addition of pixel noise.

Stability Test

Empirically found[2] to show very good performance, invariant to *image rotation*, *scale*, *intensity change*, and to moderate *affine* transformations

Scale = 2.5
Rotation = 45^0

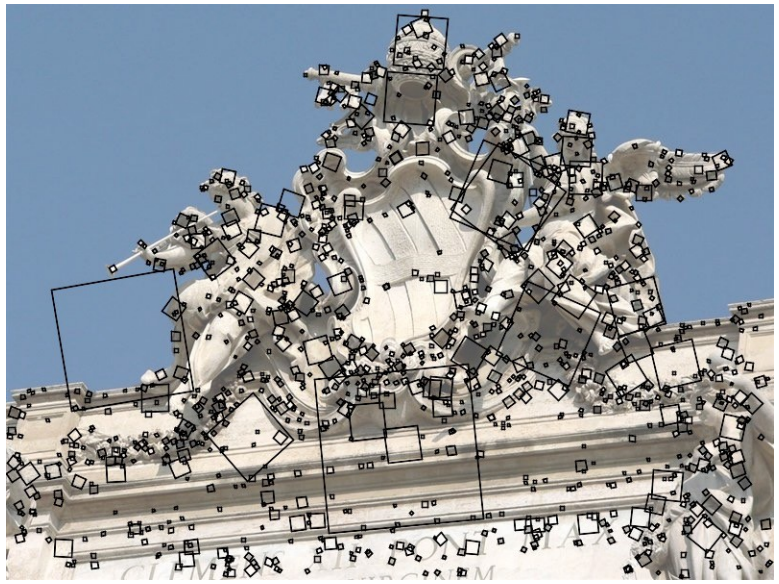


1 D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

[2] K.Mikolajczyk, C.Schmid. "A Performance Evaluation of Local Descriptors". CVPR 2003

SIFT detector: Example outputs

- Detected keypoints with characteristic scales and orientations:



SIFT for matching

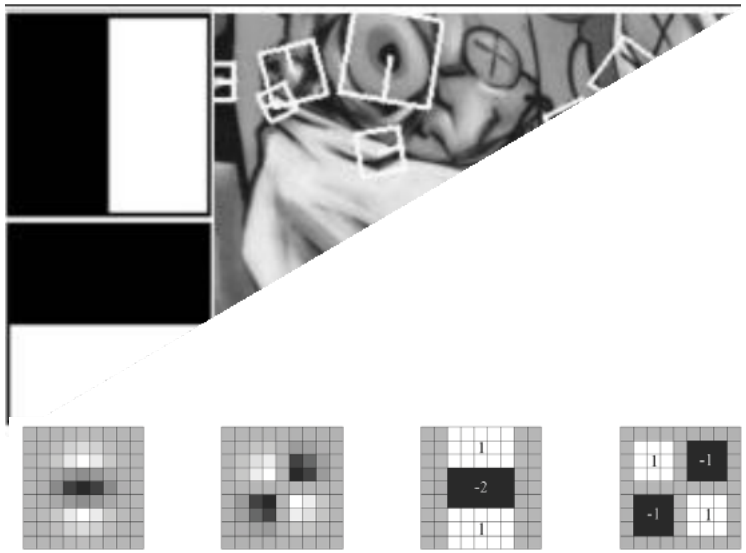
- Extraordinarily robust detection and description technique
  - Can handle changes in viewpoint
    - Up to about 60 degree out-of-plane rotation
  - Can handle significant changes in illumination
    - Sometimes even day vs. night
  - Fast and efficient—can run in real time
  - Lots of code available

Source: N. Snavely

- Many local feature detectors have executables available online:
  - http://www.robots.ox.ac.uk/~vgg/research/affine
  - http://www.cs.ubc.ca/~lowe/keypoints/
  - http://www.vision.ee.ethz.ch/~surf

- SIFT and alternative to SIFT and implemented in famous CV libraries such as OpenCV https://opencv.org/

Local descriptors: SURF



SURF: fast approximation of SIFT idea

Efficient computation by 2D box filters & integral images
$\Rightarrow$ 6 times faster than SIFT

Equivalent quality for object identification

GPU implementation available

Feature extraction @ 200Hz
(detector + descriptor, 640×480 img)

http://www.vision.ee.ethz.ch/~surf

[Bay, ECCV'06], [Cornelis, CVGPU'08]

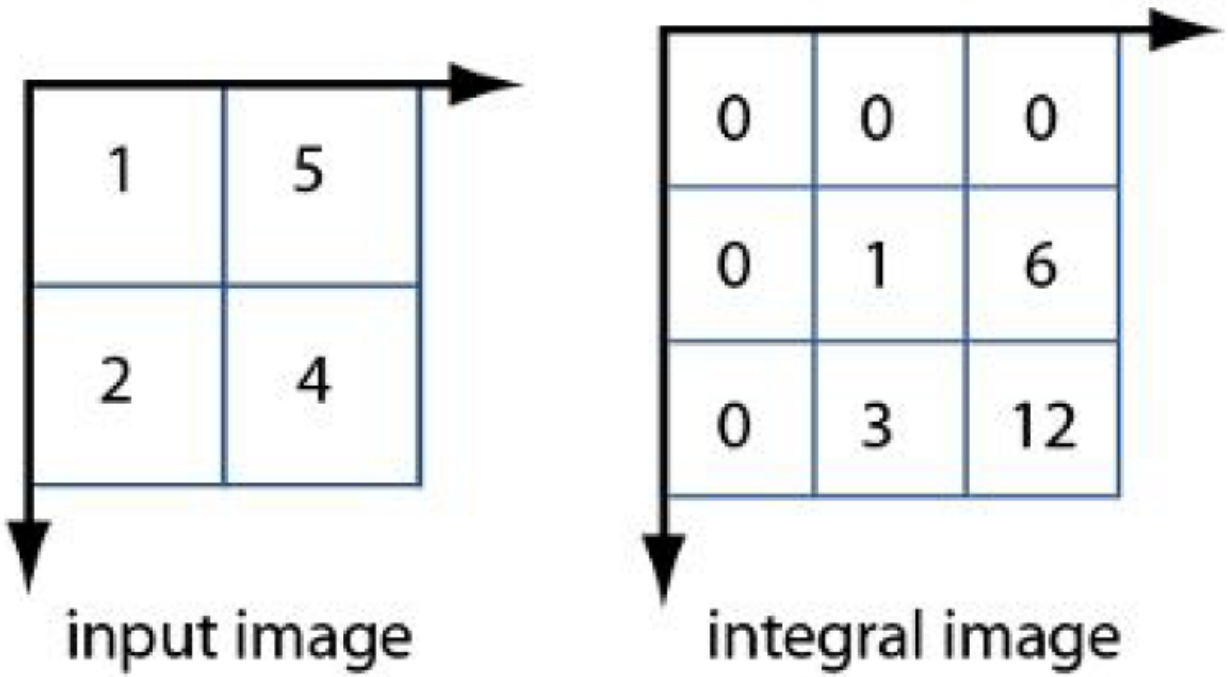
SURF

SURF (**S**peeded-**U**p **R**obust **F**eatures) is a feature detection framework introduced by Herbert Bay and his colleagues at ETH Zurich. SURF interest points are in-plane rotation-invariant, robust to noise, and overall, extremely fast to calculate. This procedure can be divided into three steps:

1. Interest Point Detection
2. Interest Point Description
3. Interest Point Matching

Integral images

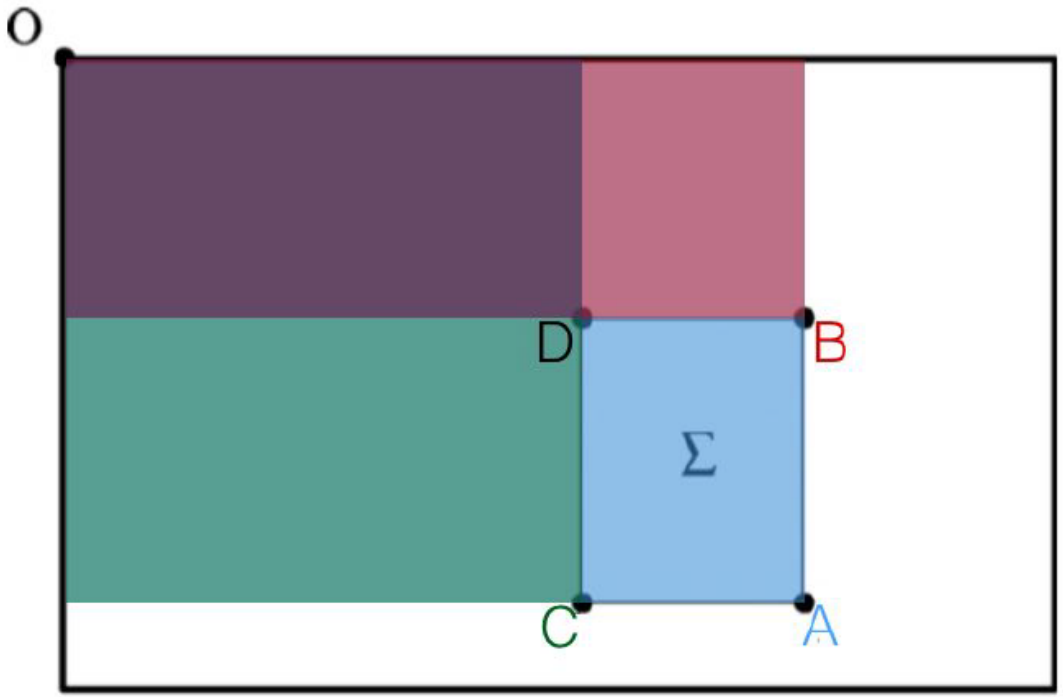
Integral images are an image transform such that any entry of an integral image $I_{\Sigma}(\mathbf{x})$ at a location $\mathbf{x} = (x, y)^T$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and $\mathbf{x}$.



$$I_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j)$$

Integral Images

Integral images are incredibly efficient. It is possible to characterize a region of the image using four memory accesses and three operations. This makes it very cheap to detect blobs.



$$\Sigma = A - B - C + D$$

Detection: Hessian-based Interest points

The detector detects blob-like structures at locations where the determinant of the Hessian is maximum.

The Hessian is defined as such: $$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

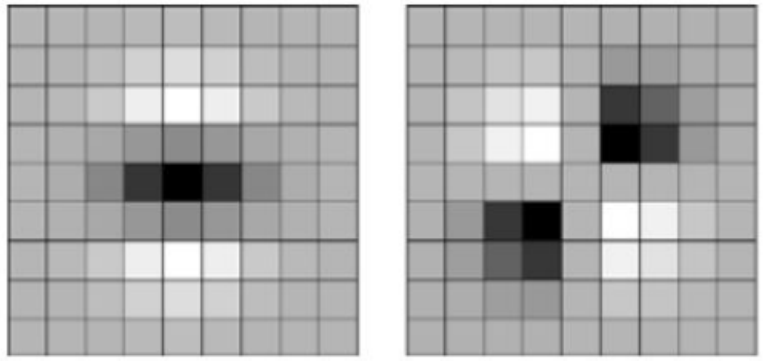
where $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image I in point $\boldsymbol{x}$.

Detection: Hessian approximation

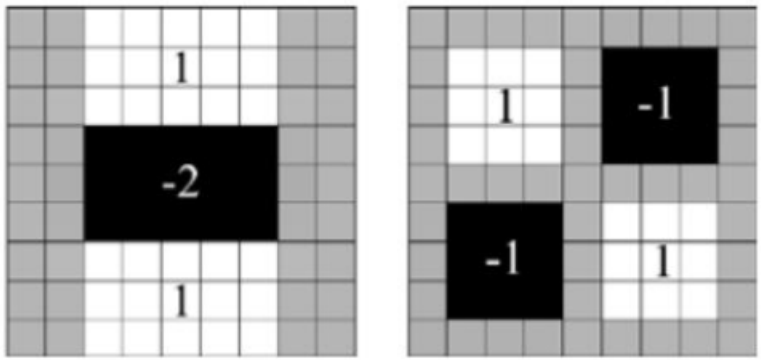
The actual computation of the Hessian matrix is expensive and slow. Instead, the Hessian can be approximated using box filters!

$$det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2$$

where D_{xx} is the approximation of the Gaussian second order partial derivative in the x-direction and $w = 0.9$.



The Gaussian second order partial derivative in y- and xy-direction.



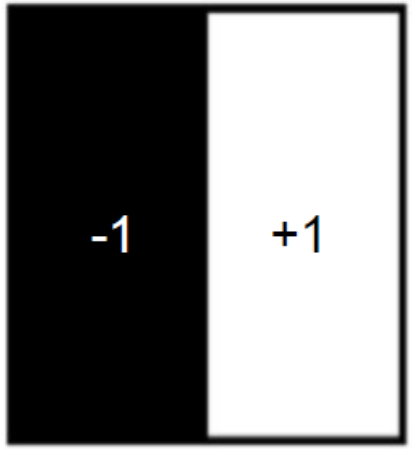
Box filter approximations of the Gaussian second order partial derivatives.

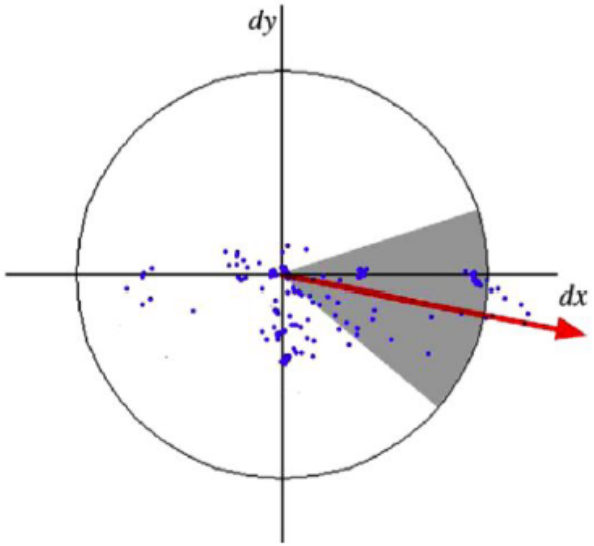
Detection: Scale-Space representation

To match interest points across different scales, a pyramidal scale space is built. Rather than serial downsampling, each successive level of the pyramid is built by upscaling the image in parallel. Each scale is defined as the the response of the image convolved with a box filter of a certain dimension (9x9, 15x15, 27x27 etc.). The scale space is further divided into octaves (sets of filter responses).

Descriptor: Orientation Assignment

The Haar wavelet responses in x- and y-direction within a circular neighborhood with radius 6*s* is calculated. Responses are weighted with a Gaussian ($\sigma = 2s$) centered at the interest point and then the directional strengths are plotted. These plots are then divided into sliding orientation windows and local orientation vectors are computed as the sum of the x and y responses within each window. The dominant orientation is the largest of all such vectors across all windows.





NB: s is the scale where the keypoint is detected

Descriptor: Feature Vector

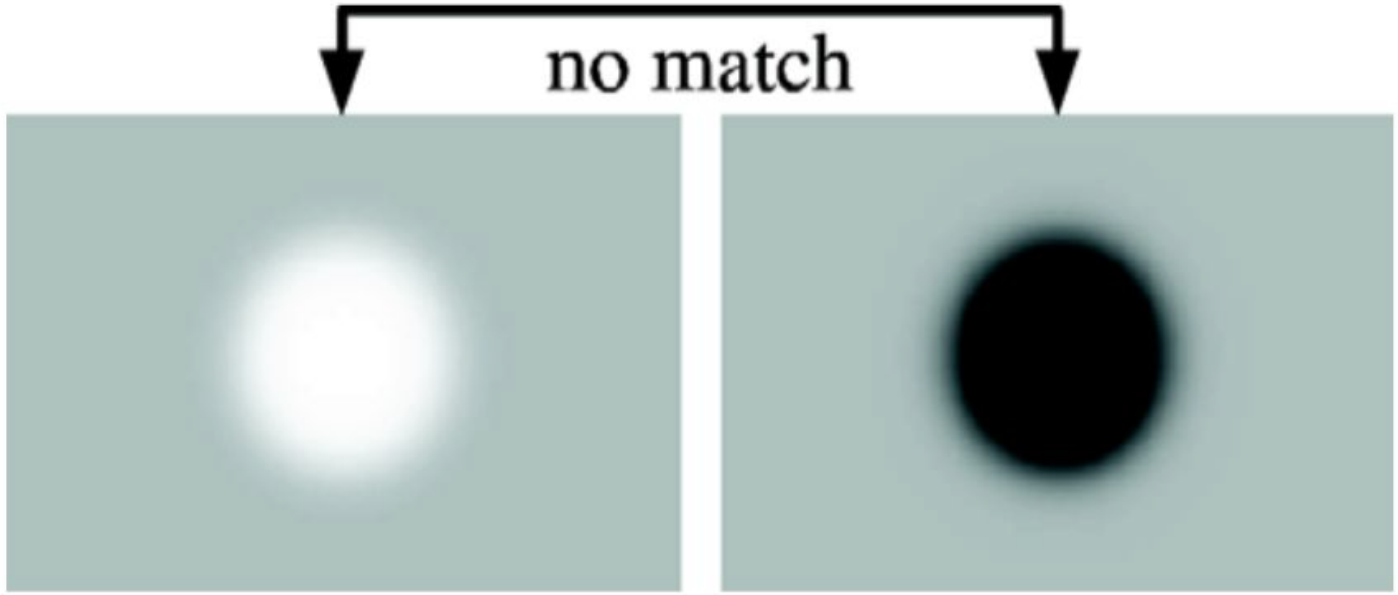
To extract features, an axis-orientated square window of size 20*s* and centered around the interest point is defined. This window is subdivided into a 4 x 4 grid. The "horizontal" and "vertical" Haar wavelet response is calculated over each subdivision and four metrics are extracted from each subdivision using 5 x 5 equally spaced points. These metrics are then summed to produce the local feature vector. These local feature vectors are concatenated to form a 64-element feature vector describing the interest point and surrounding neighborhood.

$$\mathbf{v} = \begin{bmatrix} \Sigma d_x \\ \Sigma d_y \\ \Sigma |d_x| \\ \Sigma |d_y| \end{bmatrix}$$

where d_x is the "horizontal" Haar wavelet response
d_y is the "vertical" Haar wavelet response

Matching: Laplacian Indexing

For fast indexing during the matching phase, the sign of the Laplacian (Tr(H)) for the underlying interest point is included in the discrimination cascade. The sign of the Laplacian distinguishes bright blobs on dark backgrounds from the opposite situation and serves as a meaningful metric to divide the set of all interest points.



Recent advances in interest points

Binary feature descriptors

- **BRIEF: Binary Robust Independent Elementary Features**, ECCV 10
- **ORB (Oriented FAST and Rotated BRIEF)**, CVPR 11
- BRISK: Binary robust invariant scalable keypoints, ICCV 11
- Freak: Fast retina keypoint, CVPR 12
- LIFT: Learned Invariant Feature Transform, ECCV 16

Binary descriptors

- Complex features such as SIFT work well and are gold standard
- SIFT is expensive to compute

- **Binary descriptors** aim at generating small binary strings that are easy to compute and compare

Key Idea of Binary Descriptors

Fairly simple strategy

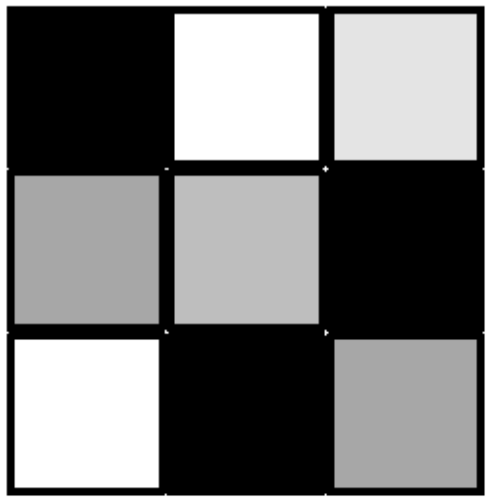
- Select a patch around a keypoint
- Select a set of pixel pairs in that patch
- For each pair, compare the intensities

$$b = \left\{ \begin{array}{ll} 1 & \text{if } I(s_1) < I(s_2) \\ 0 & \text{otherwise} \end{array} \right\}$$

- Concatenate all b's to a bit string

Example

image area



index (for pairs)

1	2	3
4	5	6
7	8	9

pairs: $\{(5,1), (5,9), (4,6), (8,2), (3,7)\}$

tests: $b=0 \quad b=0 \quad b=0 \quad b=1 \quad b=1$

result: $B = 00011$

Key Advantages of Binary Descriptors

- **Compact descriptor**
  The number of pairs gives the length in bits

- **Fast to compute**
  Simply intensity value comparisons

- **Trivial and fast to compare**
  Hamming distance

$$d_{\text{Hamming}}(B_1, B_2) = \text{sum}(\text{xor}(B_1, B_2))$$

Important Remark – Pairs

In order to compare descriptors among images, we must:

- Use the same pairs
- Maintain the same order in which the pairs are tested

Different descriptors once determine the way the pairs are chosen and fix it!

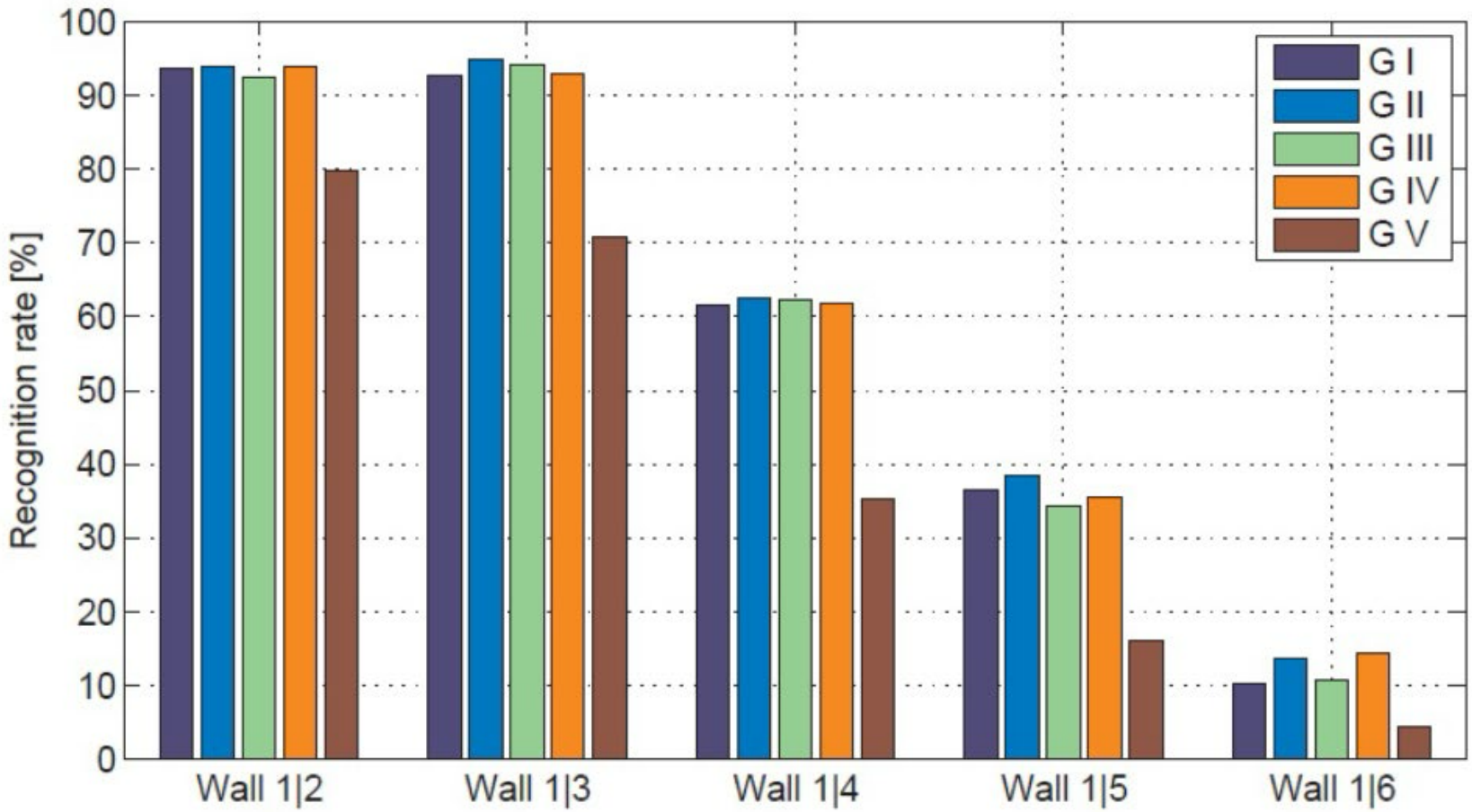
BRIEF : Binary robust independent elementary features

- First binary image descriptor
- Proposed in 2010
- 256 bit descriptor
- Provides five different geometries as sampling strategies
- Noise: operations performed on a **smoothed** image to deal with noise

BRIEF Sampling Pairs

- G I: Uniform random sampling
- G II: Gaussian sampling
- G III: s_1 Gaussian; s_2 Gaussian centered around s_1
- G IV: Discrete location from a coarse polar gird
- G V: s_1=(0,0) ; s_2 are all location from a coarse polar gird

Performance: G I – G IV are all good, G V less useful



ORB:
Oriented FAST Rotated BRIEF

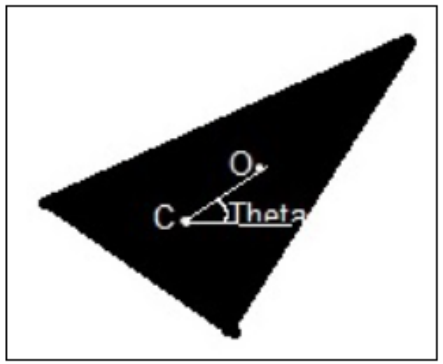
An extension to BRIEF that

- Adds rotation compensation
- Learns the optimal sampling pairs

ORB: Rotation Compensation

- Estimates the center of mass and the main orientation of the area/patch
- Image moment

$$m_{pq} = \sum_{x,y} x^p \, y^q \, I(x,y)$$



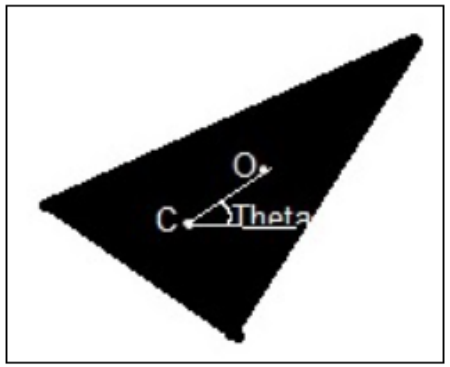
where *p* and *q* are non-negative integers representing the order of the moment

- Center of mass $C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$
- Orientation $\theta = \text{atan2}(m_{01}, m_{10})$

ORB: Rotation Compensation

- Given CoM and orientation C, θ, we can rotate the coordinates of all pairs by θ around C:

$$s' = T(C, \theta)\, s$$



- Use the transformed pixel coordinates for performing the test
- Invariance to rotation in the plane

ORB: Learning Sampling Pairs

Pairs should be / have

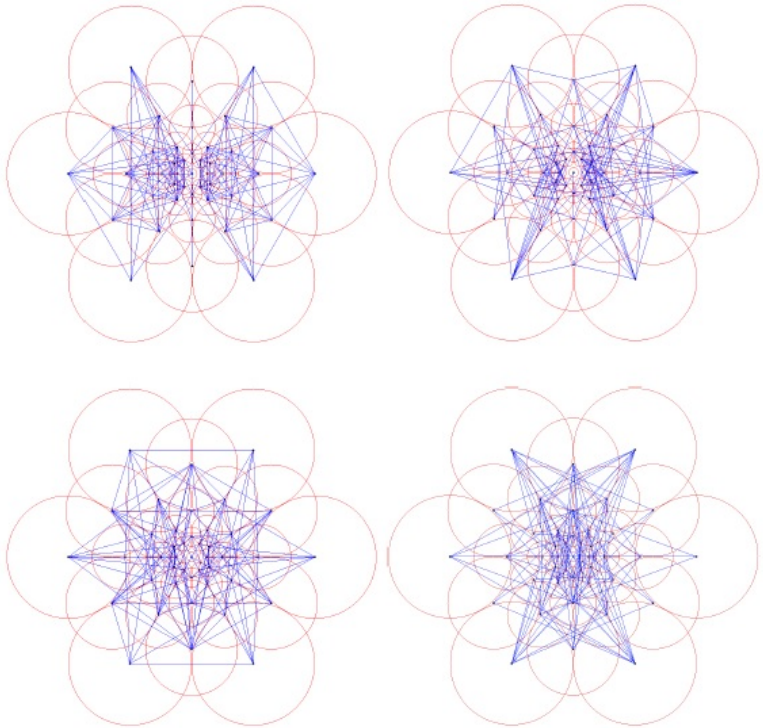
- **uncorrelated** – so that each new pair adds new information to the descriptor
- **high variance** – it makes a feature more discriminative

- ORB defines a strategy for selection 256 pairs optimizing for both properties using a training database

ORB vs. SIFT

- ORB is 100x faster than SIFT
- ORB: 256 bit vs. SIFT: 4096 bit
- ORB is not scale invariant
  (achievable via an image pyramid)
- ORB mainly in-plane rotation invariant
- ORB has a similar matching
  performance as SIFT (w/o scale)
- Several modern online systems
  (e.g. SLAM) use binary features

FREAK sampling



It focuses on capturing local image patterns using a retinal sampling strategy inspired by the human visual system.

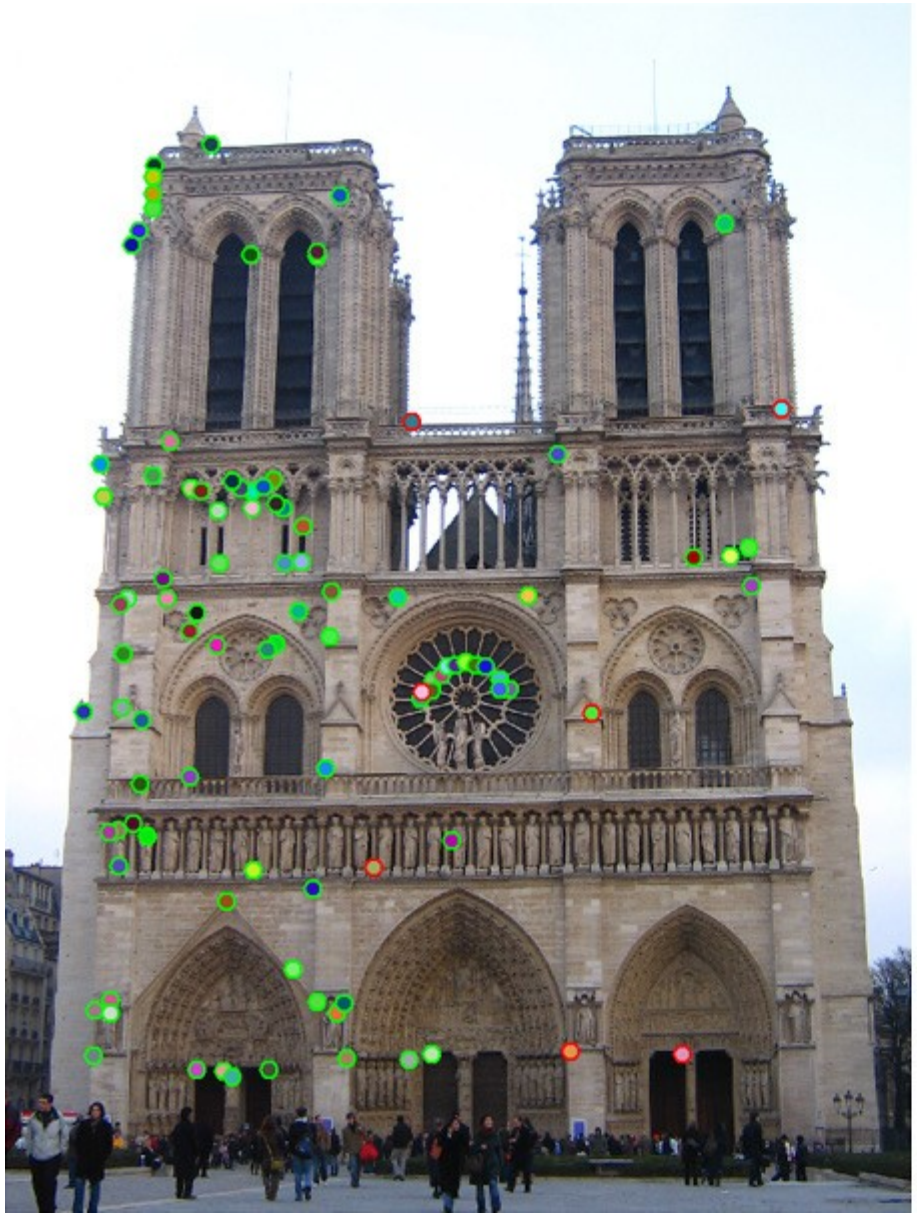
Summary

- Keypoints and descriptor together define common visual features
- Keypoint defines the location
- Descriptor describes the appearance
- Several descriptors operating on gradient histograms (SIFT, SURF, …)
- Binary descriptors for efficiency (BRIEF, ORB, ...)

Today’s class

- SIFT detector
- SIFT descriptor
- Feature Matching
- Evaluating Results

Which features match?



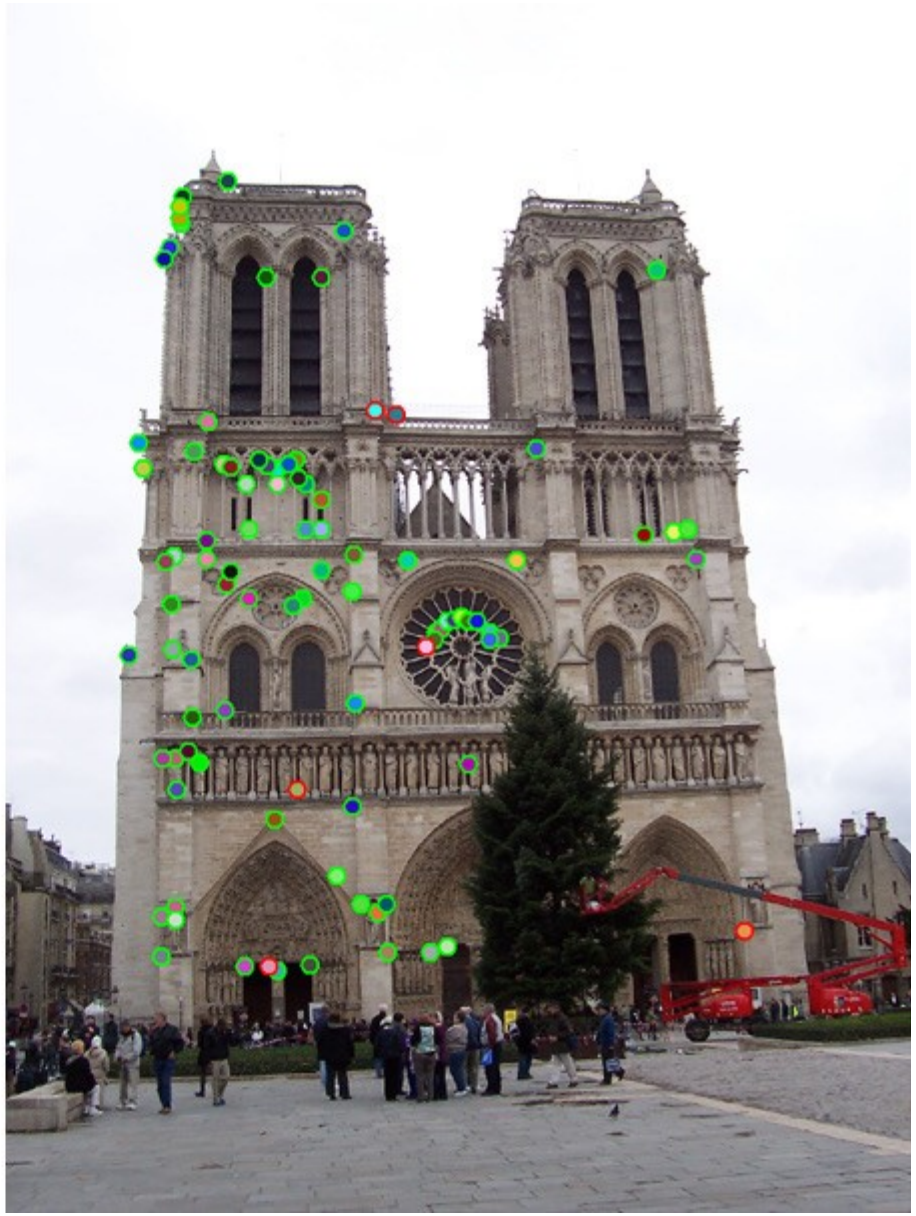


Image Matching

- Find all key points identified in a target image
  - Each keypoint will have 2D location, scale and orientation, as well as invariant descriptor vector ***(x,y,s,theta,d)***
- For each keypoint, search similar descriptor vectors in a reference image
  - Descriptor vector may match more than one reference descriptor vectors

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
   - Any distance metric, *d(f1,f2)*, would work: L2 (Mean Squared Error), L1 (Mean Absolute Error) loss are commonly used

2. Test all the features in I_2, find the one with min distance

(OR)

2. Test all the features in I_2, find top k matches

Feature Matching: example

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

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

I_1

I_2

Feature distance

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

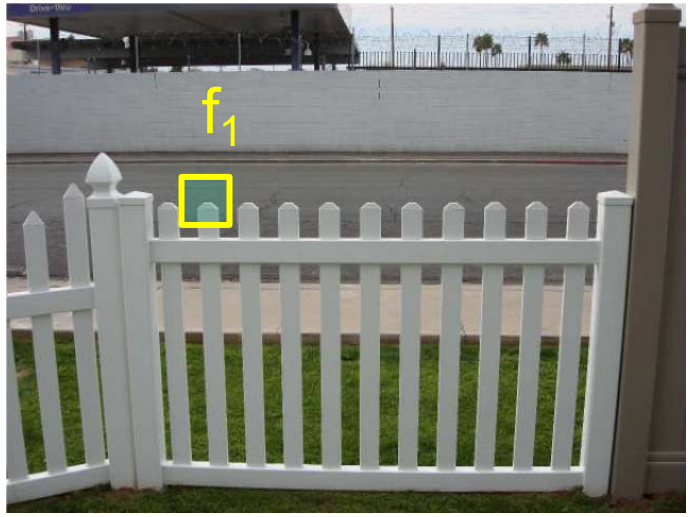
- Simple approach: L_2 distance, $||f_1 - f_2||$
  - But can give small distances for ambiguous (incorrect) matches

I_1 I_2

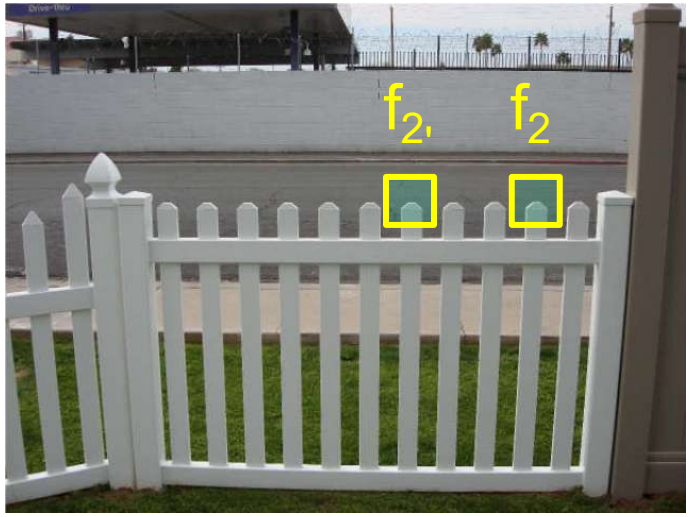
Feature distance: Ratio test

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

- Better approach: distance ratio = $||f_1 - f_2|| / ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - Sorting by this ratio puts matches in order of confidence.
  - Set the **distance ratio threshold (ρ)** to around 0.5, which means that we require our best match to be at least twice as close as our second best match to our initial features descriptor. Thus discarding our ambiguous matches and retaining the good ones



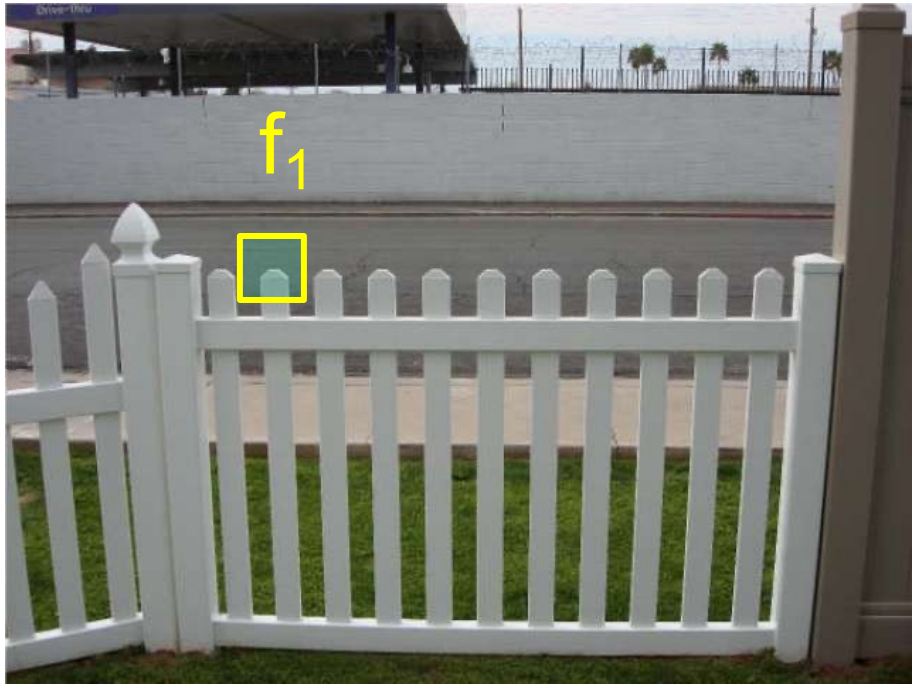
I_1



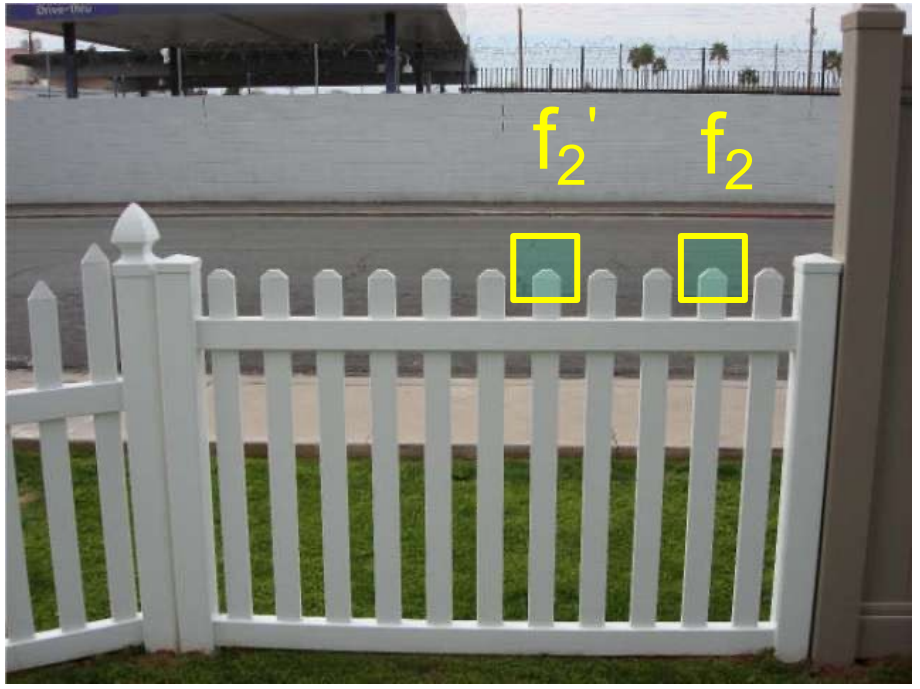
I_2

Feature distance: Ratio Test

- Often matches can be ambiguous. f1 can have similar distance to both f2 and f2’
- Ratio Test:
  - Keep top 2 match: f2, f2’
  - If d(f1,f2) < **0.75** * d(f1,f2’), then: match f1 with f2 and keep the point.
    - Else reject the match as ambiguous
  - If d(f1,f2) < Threshold, then: keep this as ‘strong’ match, else: ‘reject’



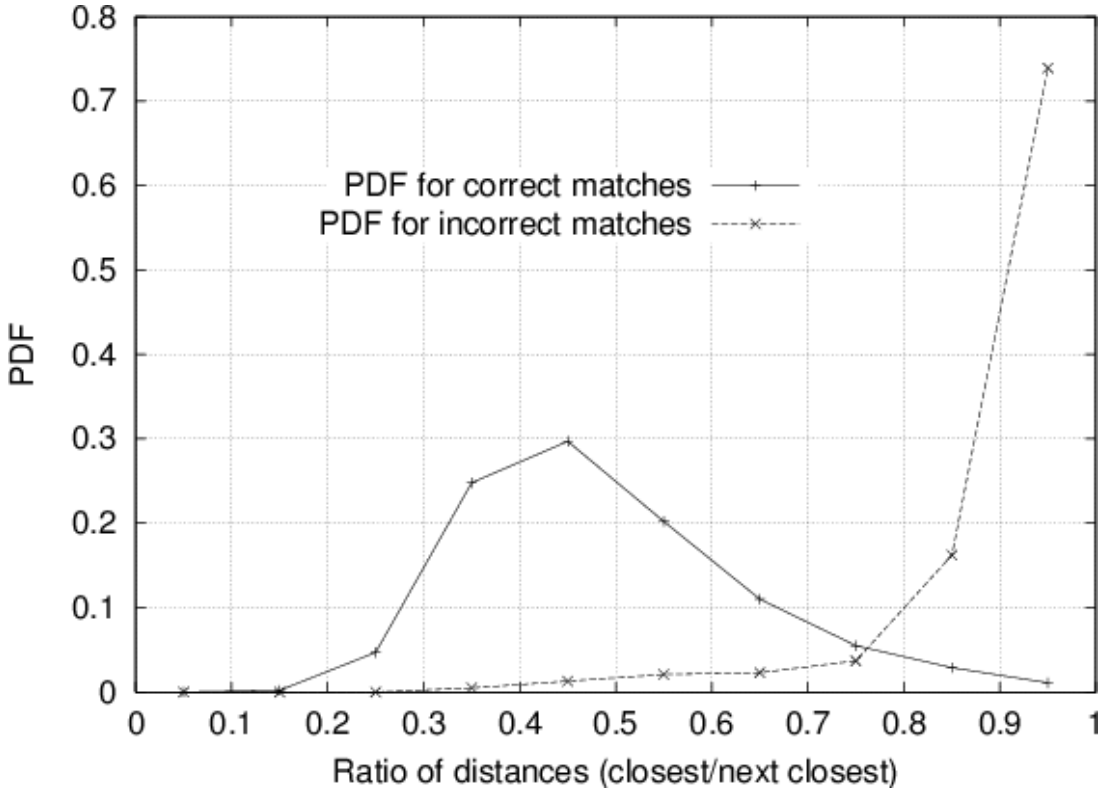
I_1



I_2

Matching SIFT features

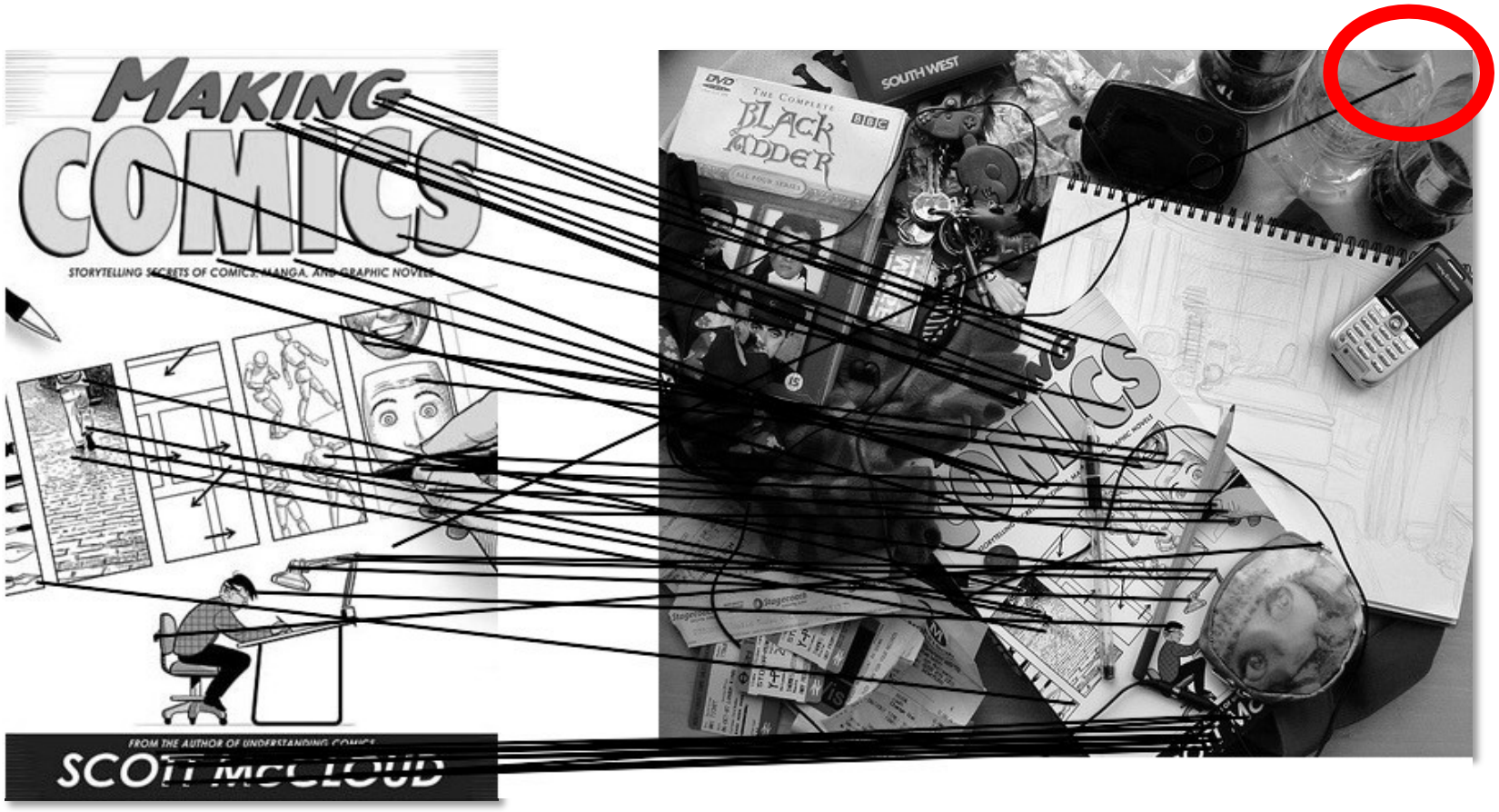
- Accept a match if SSD(f_1, f_2) / SSD(f_1, f_2') < t
- t=0.75,0.80 have given good results in object recognition.
- Eliminated 90% of false matches.
- Discarded less than 5% of correct matches



Feature matching example

We'll deal with **outliers** later

RANSAC next slides!



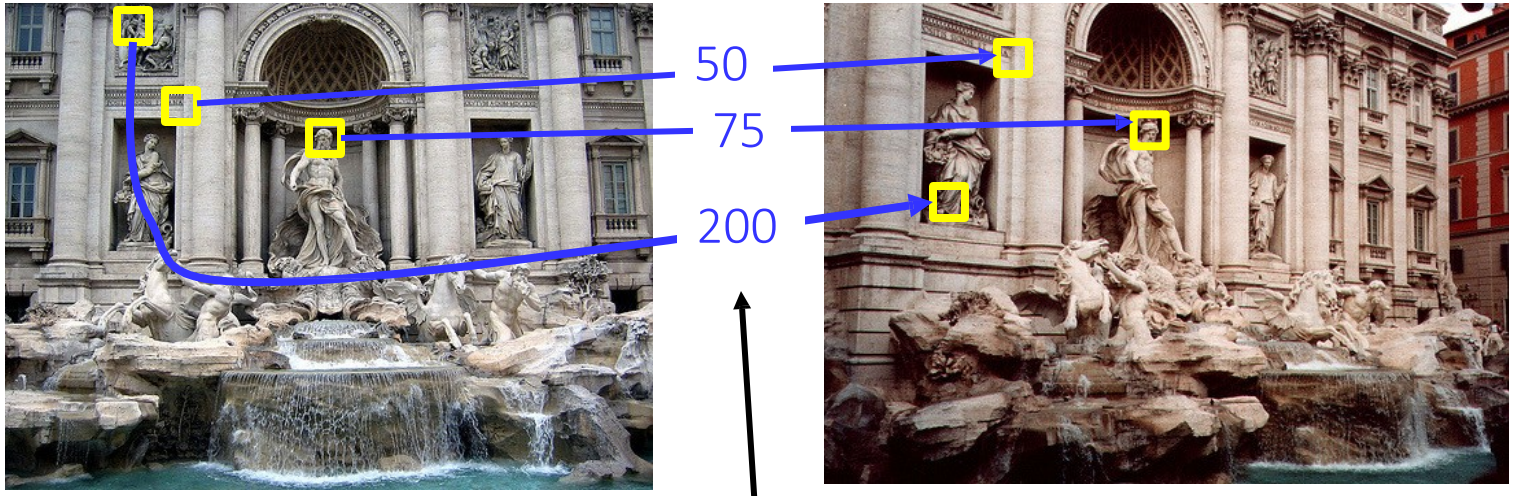
51 matches (thresholded by ratio score)

Today’s class

- SIFT detector
- SIFT descriptor
- Feature Matching
- Evaluating Results

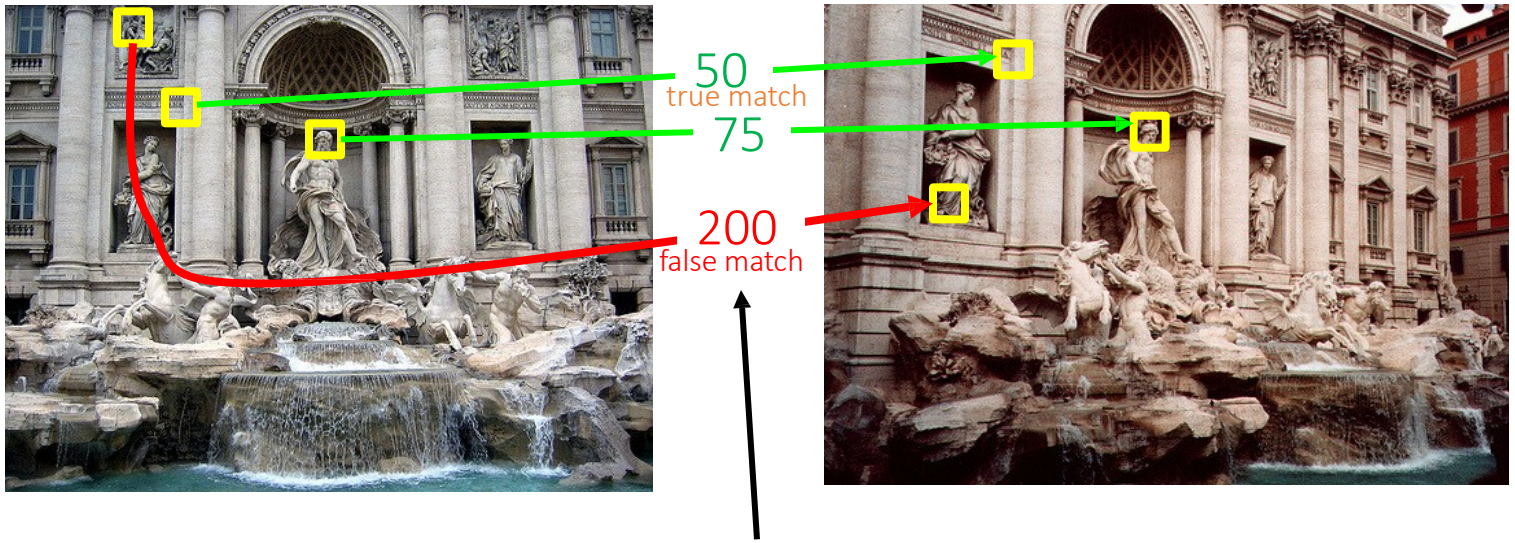
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

How can we measure the performance of a feature matcher?



We can choose distance threshold to decide if the match is ‘good’ or not.

The distance threshold affects performance

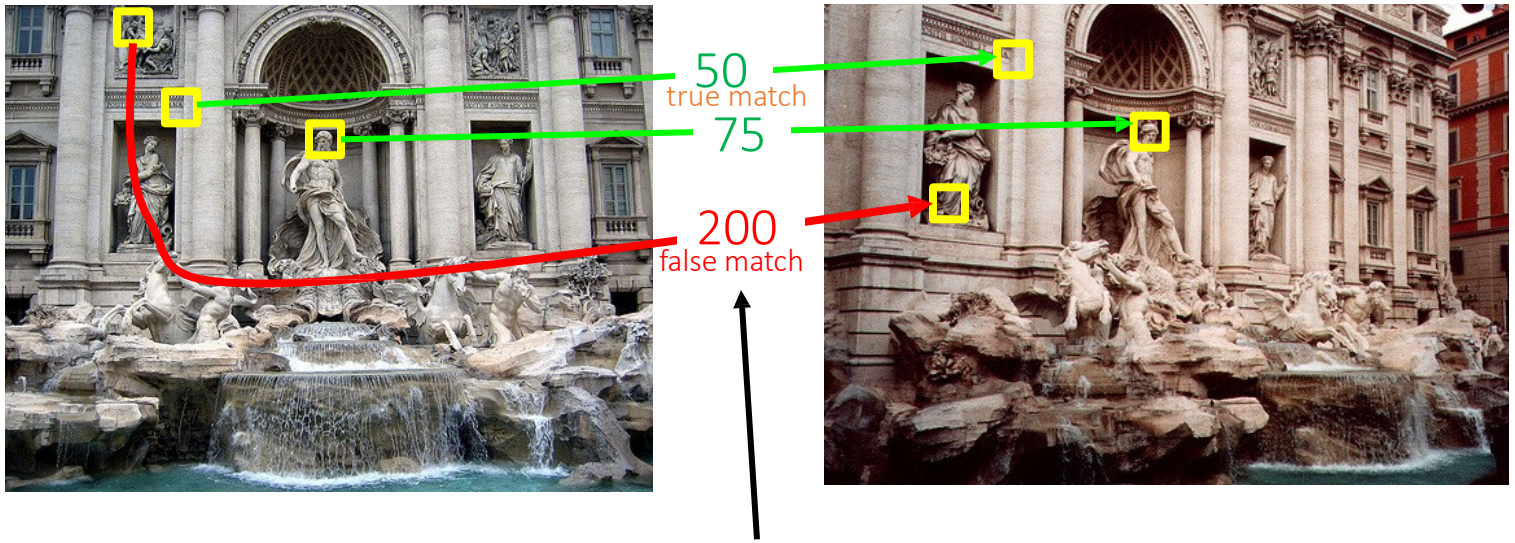
- True positives = # of detected matches that survive the threshold that are correct
- False positives = # of detected matches that survive the threshold that are incorrect

Example

- Suppose our matcher computes 1,000 matches between two images.
  - 800 are correct matches, 200 are incorrect (according to an oracle that gives us ground truth matches)
  - A given threshold (e.g., ratio distance = 0.6) gives us 600 correct matches and 100 incorrect matches that survive the threshold
  - True positives = # of detected matches that survive the threshold that are correct
  - False positives = # of detected matches that survive the threshold that are incorrect
  - True positive rate = 600 / 800 = ¾
  - False positive rate = 100 / 200 = ½

True/false positives

How can we measure the performance of a feature matcher?

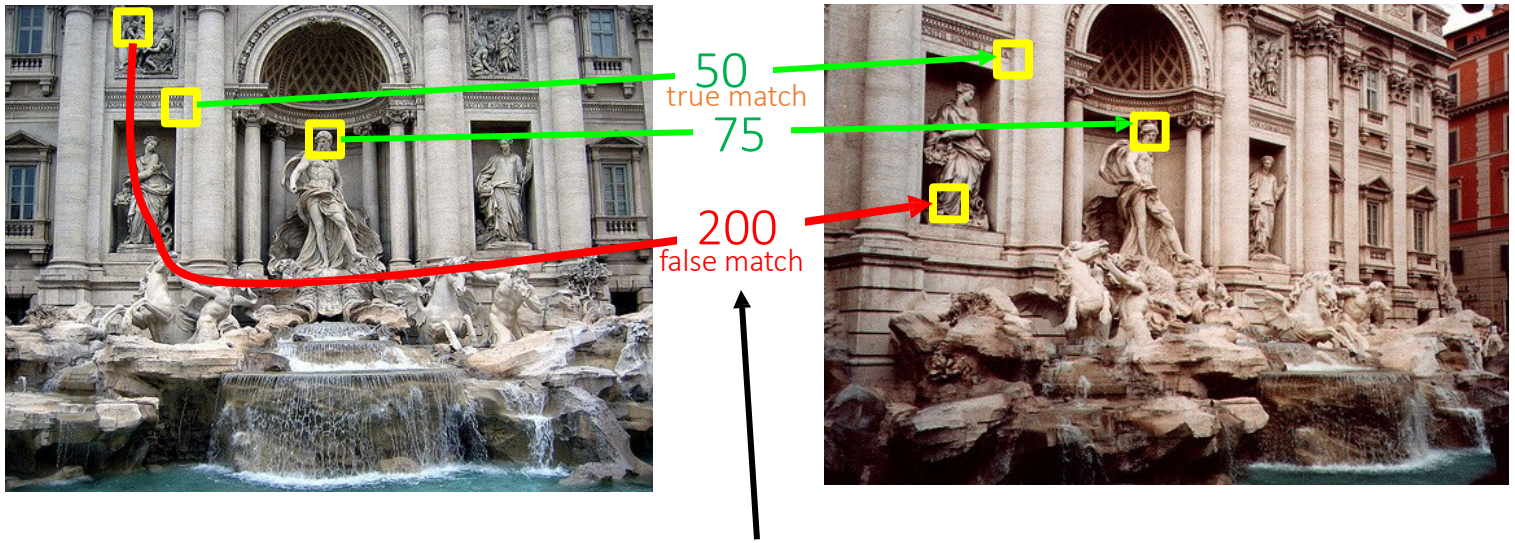


True positives = # of detected matches that survive the threshold that are correct
False positives = # of detected matches that survive the threshold that are incorrect

Suppose we want to maximize true positives. How do we set the threshold? (We keep all matches with distance below the threshold.)

True/false positives

How can we measure the performance of a feature matcher?



True positives = # of detected matches that survive the threshold that are correct
False positives = # of detected matches that survive the threshold that are incorrect

Suppose we want to minimize false positives. How do we set the threshold? (We keep all matches with distance below the threshold.)

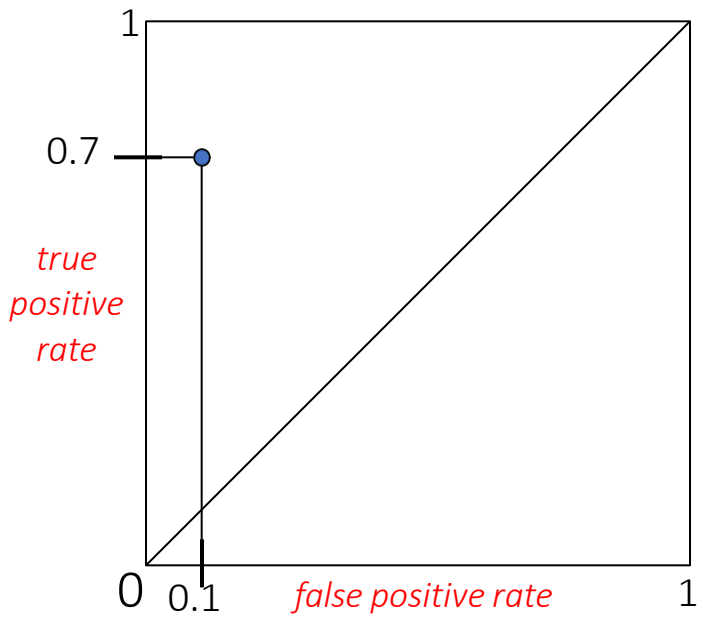
Evaluation - digression

Evaluating the results

How can we measure the performance of a feature matcher?

true positives surviving threshold
total correct matches (positives)

recall



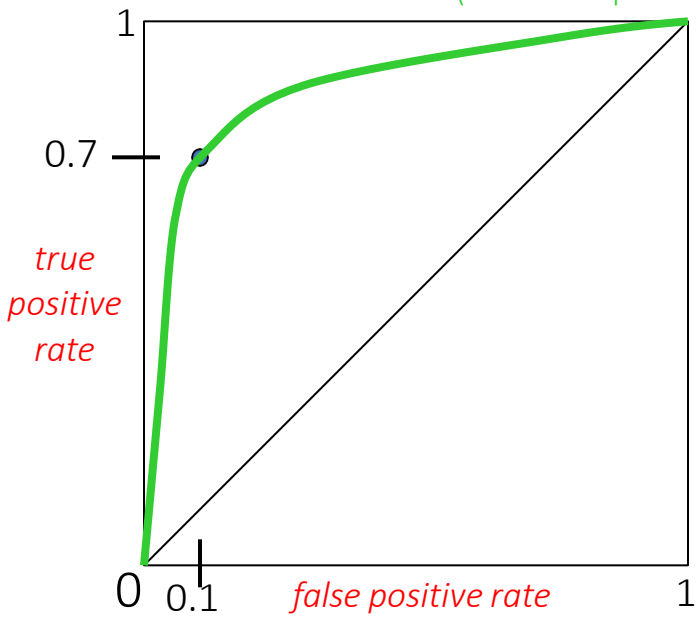
false positives surviving threshold
total incorrect matches (negatives)

1 - *specificity*

Evaluating the results

How can we measure the performance of a feature matcher?

ROC curve ("Receiver Operator Characteristic")



true positives surviving threshold
total correct matches (positives)

recall

false positives surviving threshold
total incorrect matches (negatives)

1 - *specificity*

Generated by choosing different threshold value

Single number: Area Under the Curve (AUC)

E.g. AUC = 0.87

1 is the best

ROC curves – summary

- By thresholding the match distances at different thresholds, we can generate sets of matches with different true/false positive rates
- ROC curve is generated by computing rates at a set of threshold values swept through the full range of possible threshold
- Area under the ROC curve (AUC) summarizes the performance of a feature pipeline (higher AUC is better)
- We will come back to this in binary classification, face verification, object detection, image retrieval, etc.

Be careful of “Accuracy”

The simplest measure of performance would be the fraction of items that are correctly classified, or the “accuracy” which is:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

It’s the **proportion of correct predictions among the total number of cases examined.**

But this measure is dominated by the larger set (of positives or negatives) and favors trivial classifiers.

e.g. if 5% of items are truly positive, then a classifier that always says “negative” is 95% accurate.

Precision and Recall

Precision *(How much we are precise in the detection)*

Of all patients where we classified $y = 1$, what fraction actually has the disease?

$$\frac{\text{True Positive}}{\text{\# Estimated Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Recall *(How much we are good at detecting)*

Of all patients that actually have the disease, what fraction did we correctly detect as having the disease?

$$\frac{\text{True Positive}}{\text{\# Actual Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Confusion matrix

Estiamted class \ Actual class	1 (p)	0 (n)
1 (Y)	True positive (TP)	False positive (FP)
0 (N)	False negative (FN)	True negative (TN)

Precision and Recall

Precision:

- Precision is a measure of the accuracy of the positive predictions made by a classifier. It answers the question: "Of all the instances predicted as positive, how many were truly positive?“
- Precision is calculated using the formula:
  Precision=True Positives/(True Positives + False Positives)
- A high precision indicates that the classifier has a low rate of false positives, meaning that when it predicts positive, it is likely to be correct.

Recall (Sensitivity or True Positive Rate):

- Recall measures the ability of a classifier to capture all the relevant positive instances. It answers the question: "Of all the true positive instances, how many were successfully predicted?"
- Recall is calculated using the formula:
  Recall=True Positives/(True Positives + False Negatives)
- A high recall indicates that the classifier is good at identifying positive instances, even if it means having a higher rate of false positives.

F1 score

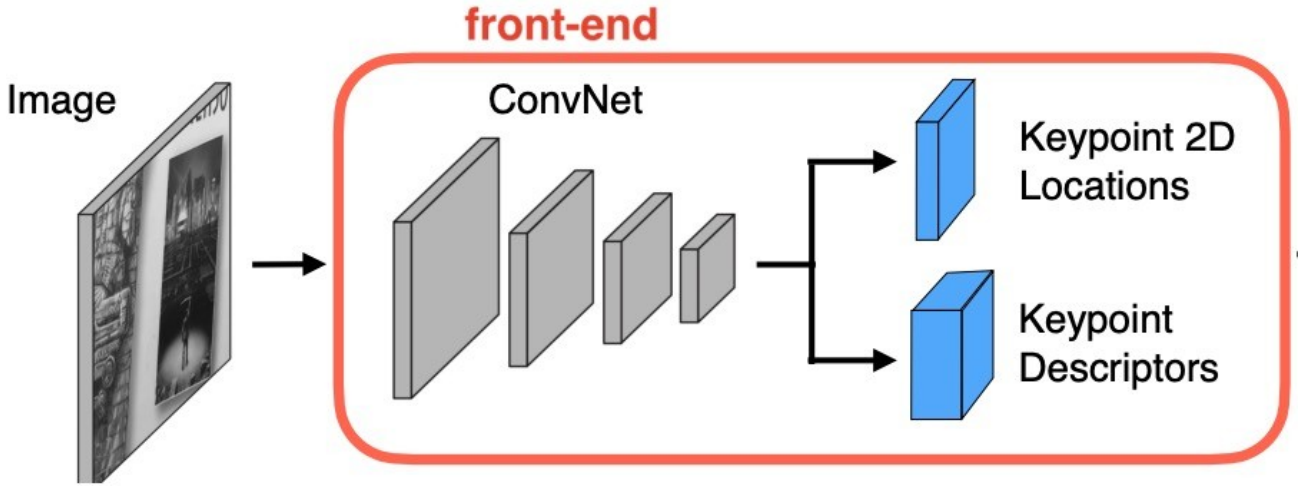
- **Precision** focuses on the accuracy of positive predictions, emphasizing how well the model avoids false positives.
- **Recall** focuses on the ability of the model to capture all relevant positive instances, emphasizing how well the model avoids false negatives.

- The F1 score, which is the harmonic mean of precision and recall, is a metric that combines both measures

$$2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

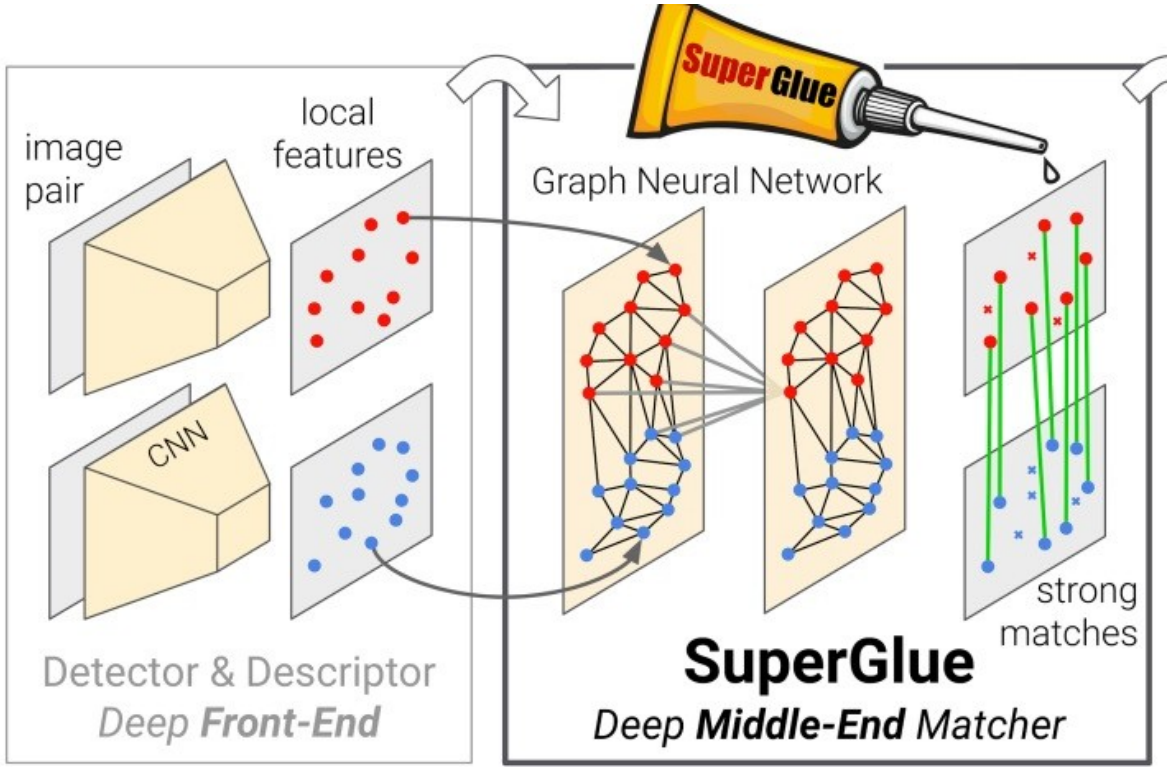
/ Evaluation - digression

Local features & matching in Deep Learning era



- Local Features = SuperPoint
- Train first on synthetic data
- Self-supervised learning on real data

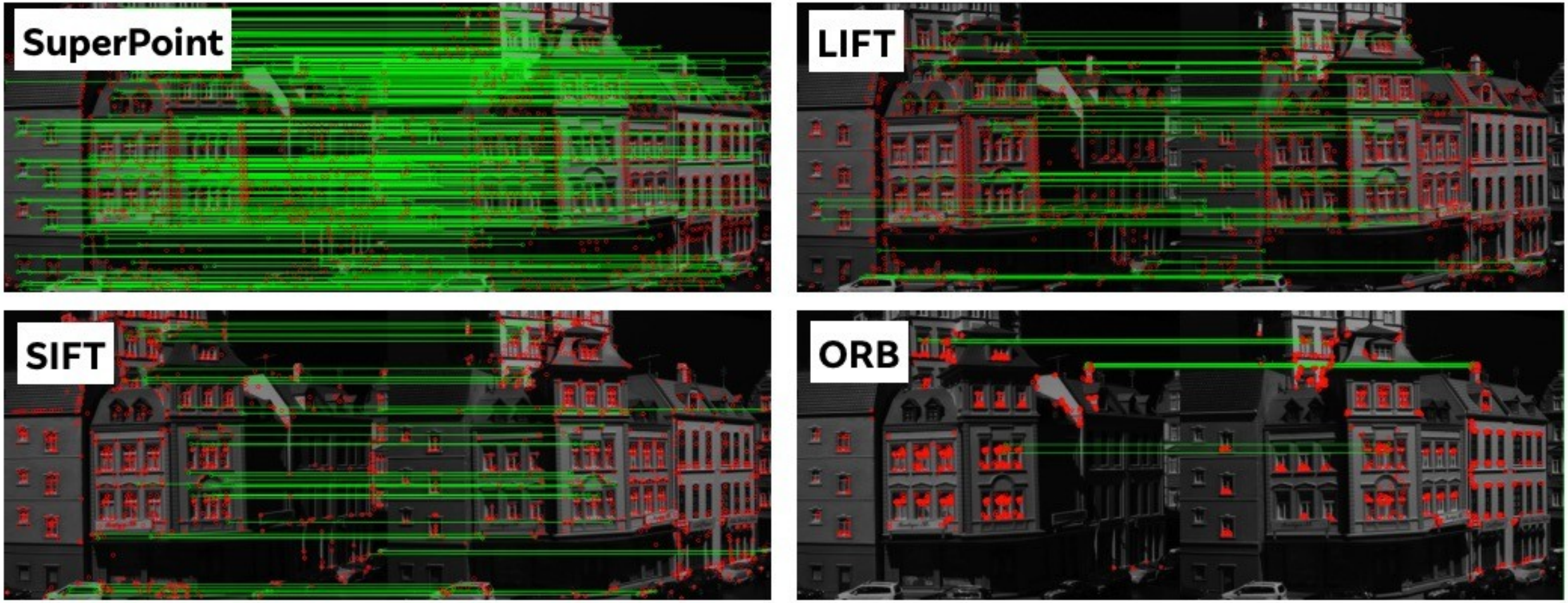
DeTone, Daniel & Malisiewicz, Tomasz & Rabinovich, Andrew. (2018). SuperPoint: Self-Supervised Interest Point Detection and Description.



- Feature Matching = SuperGlue

SuperGlue: Learning Feature Matching With Graph Neural Networks, Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020

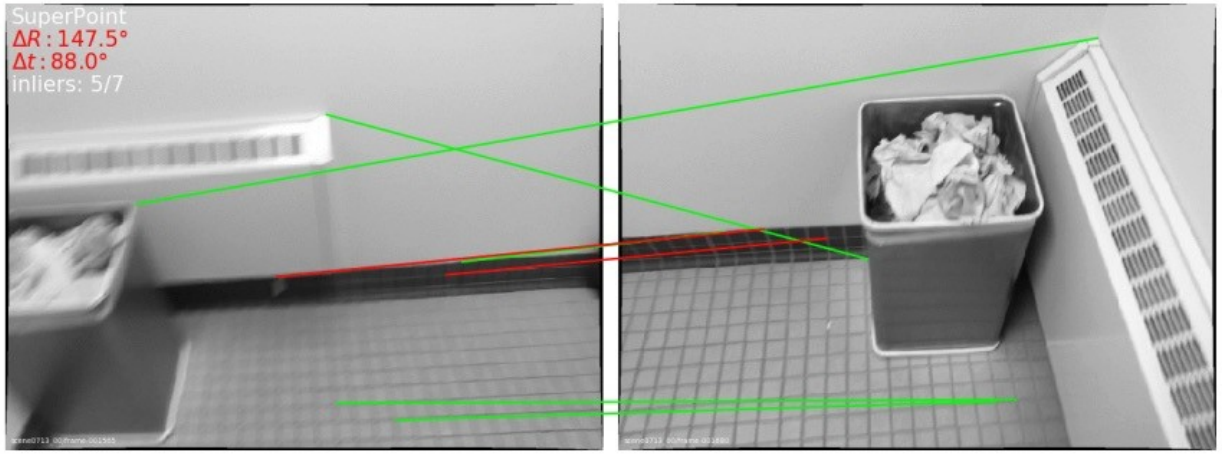
Performance of SuperPoint

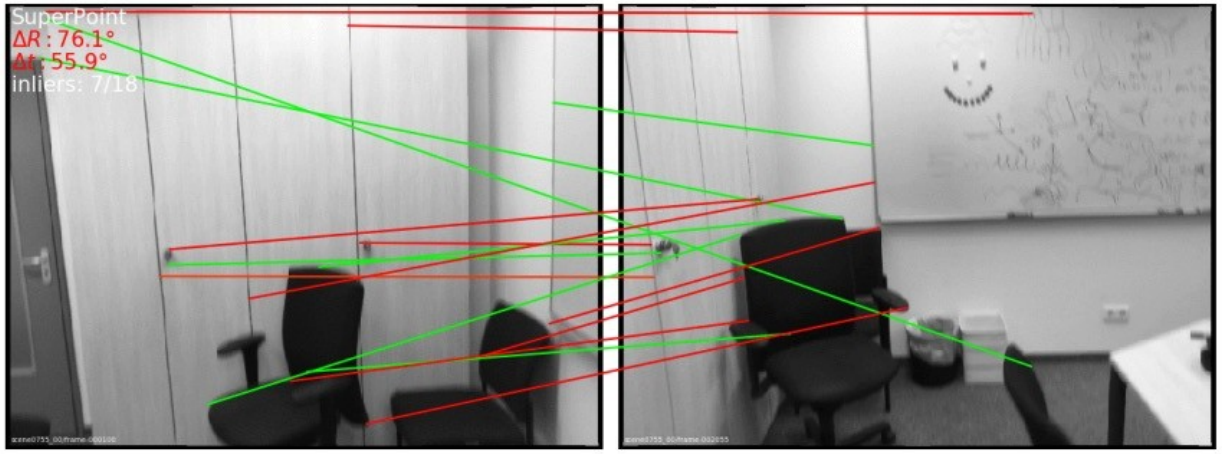


K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. "LIFT: Learned Invariant Feature Transform", European Conference on Computer Vision (ECCV), 2016.

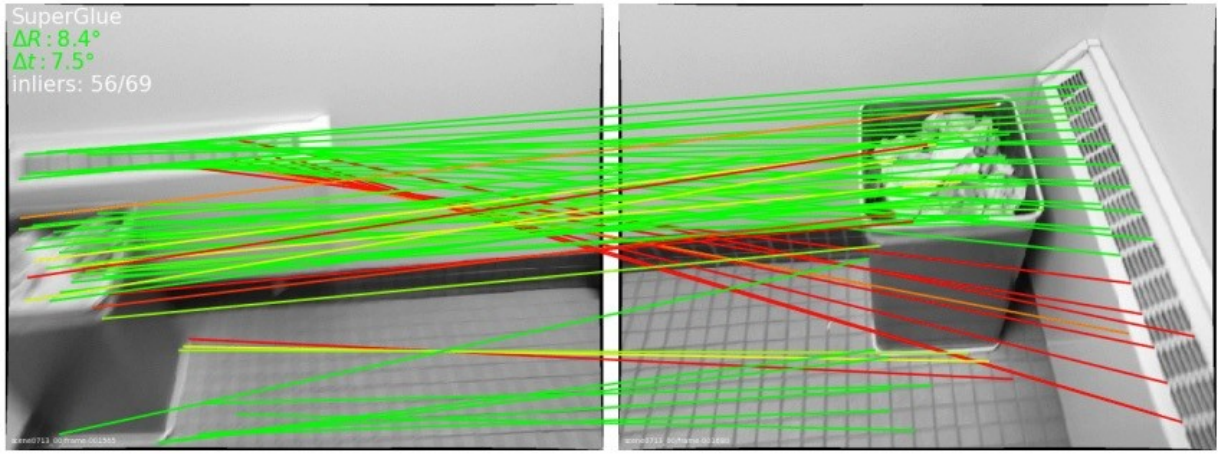
Performance of SuperGlue

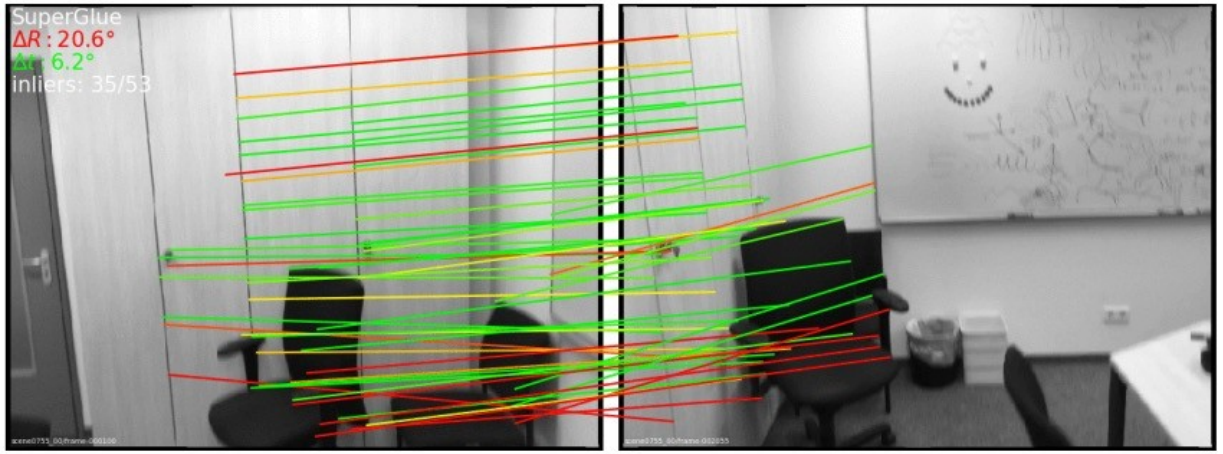
SuperPoint + NN + heuristics





SuperPoint + **SuperGlue**





SuperGlue: more **correct matches** and fewer **mismatches**

Slide Credits

- CS5670, Introduction to Computer Vision, Cornell Tech, by Noah Snavely.
- CS 194-26/294-26: Intro to Computer Vision and Computational Photography, UC Berkeley, by Alyosha Efros.
- Fall 2022 CS 543/ECE 549: Computer Vision, UIUC, by Svetlana Lazebnik.

Acknowledgements: some slides and material from Bernt Schiele, Mario Fritz, Michael Black, Bill Freeman, Fei-Fei Li, Justin Johnson, Serena Yeung, R. Szelisky, Ioannis Gkioulekas, Roni Sengupta, Andreas Geiger, Cyrill Stachniss, Gil Levi, A. Efros, J. Hayes, D. Lowe and S. Savarese