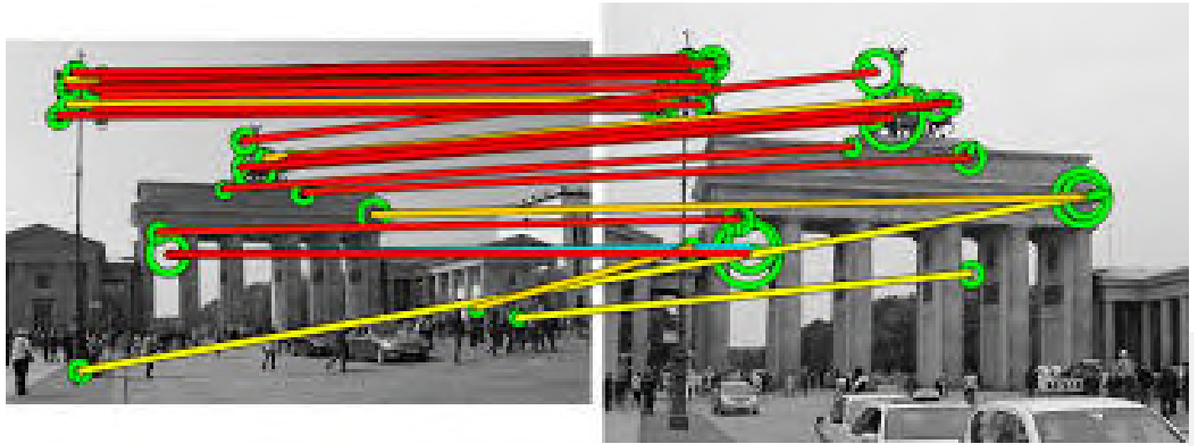
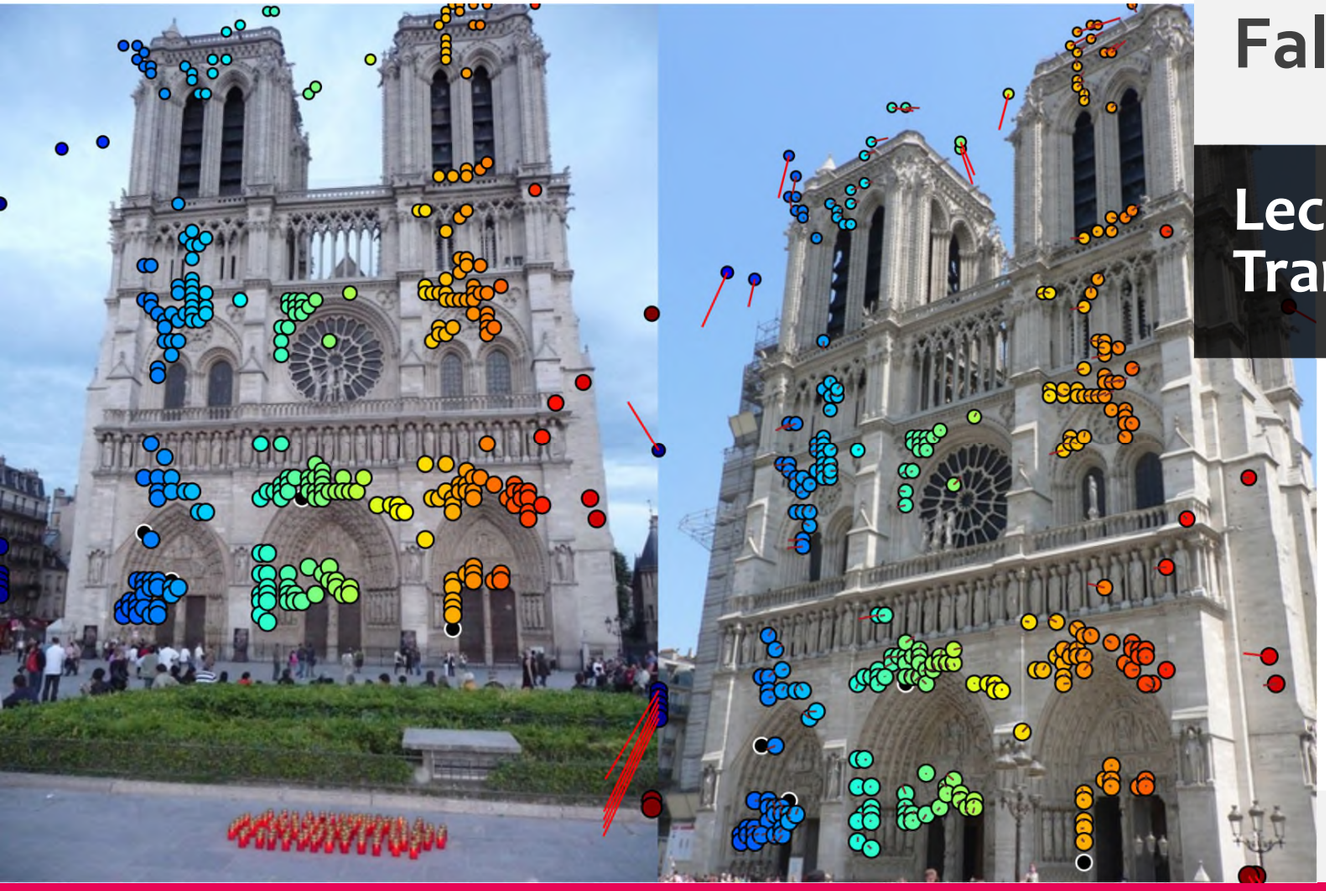




CS867 Computer Vision
Fall 2021

Lecture 8 : Scale Invariant Feature Transform

Dr. Muhammad Moazam Fraz

Department of Computing, NUST SEECS

Web: http://seecs.nust.edu.pk/faculty/mmfraz.html

Lab : http://vision.seecs.edu.pk

VISION @ SEECS

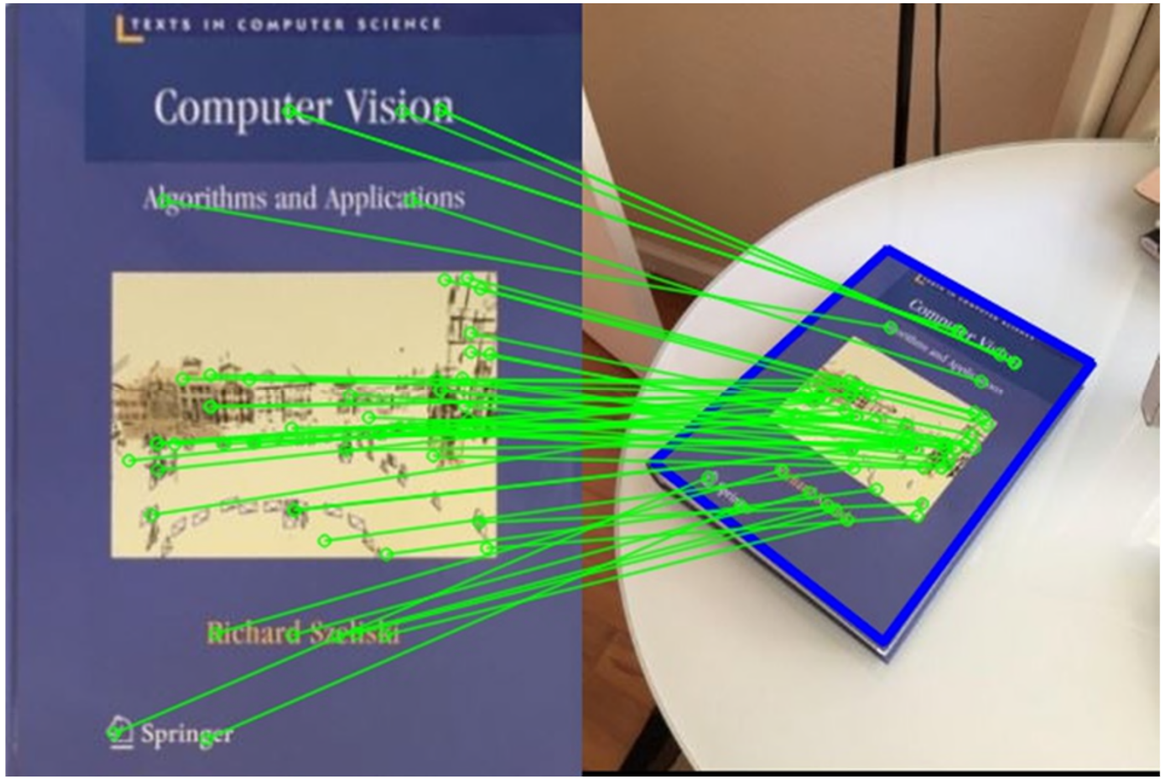
Last Class

- **Laplacian of Gaussian**
  - 2nd Order Image Derivative
  - Digital Approximation
  - Difference of Gaussian
  - Blob detection using LoG

- **Scale Space Image Processing**
  - Gaussian and Laplacian Pyramids

- **Local Image Features**
  - Corner Detection

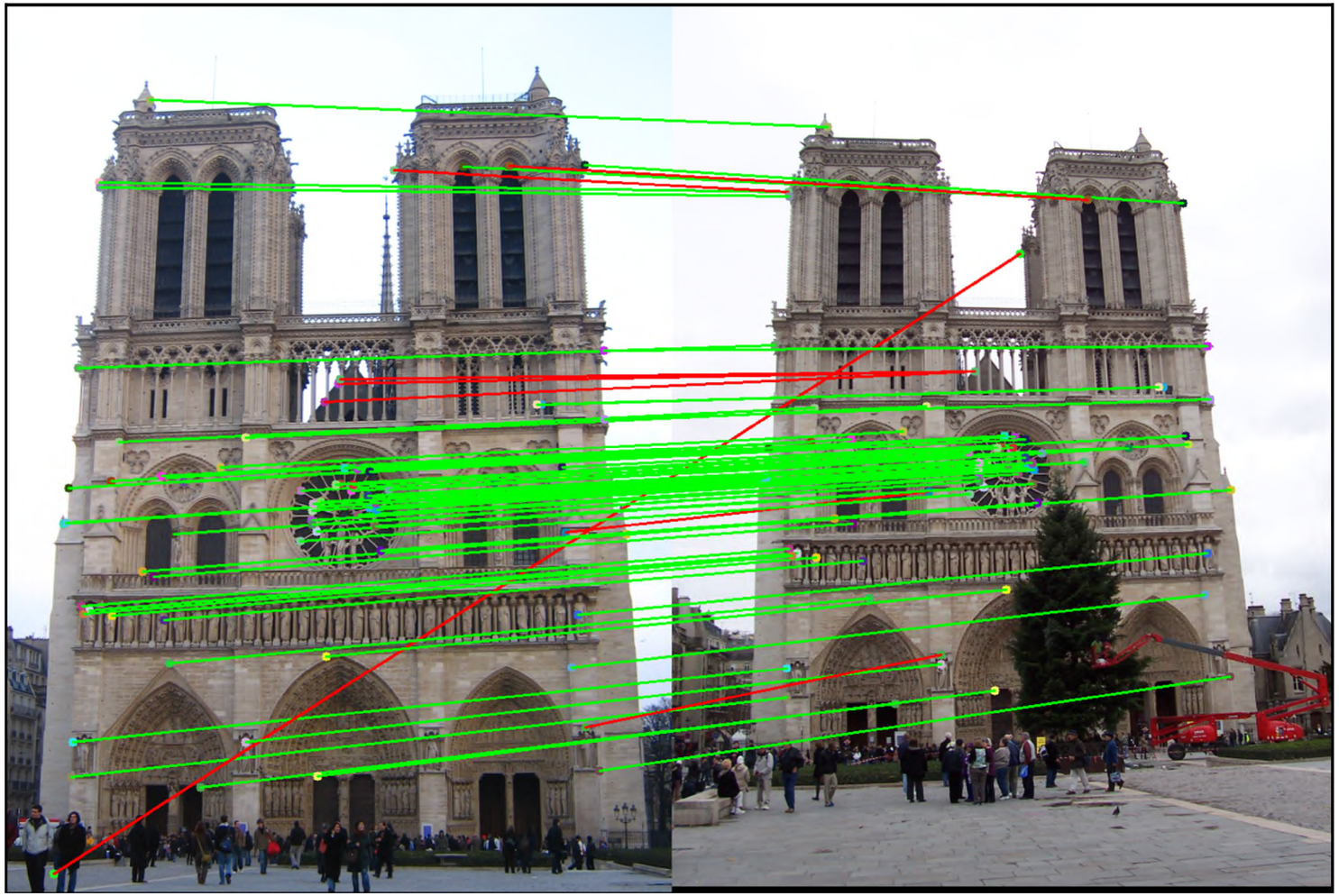
- Feature Descriptors



Learning Outcomes of This Class

- **Scale Invariant Feature Transform**
  - Workflow
  - Implementations
  - Demonstrations

- **Image Classification**
  - Histogram of Oriented Gradients
  - Bag of Visual Words

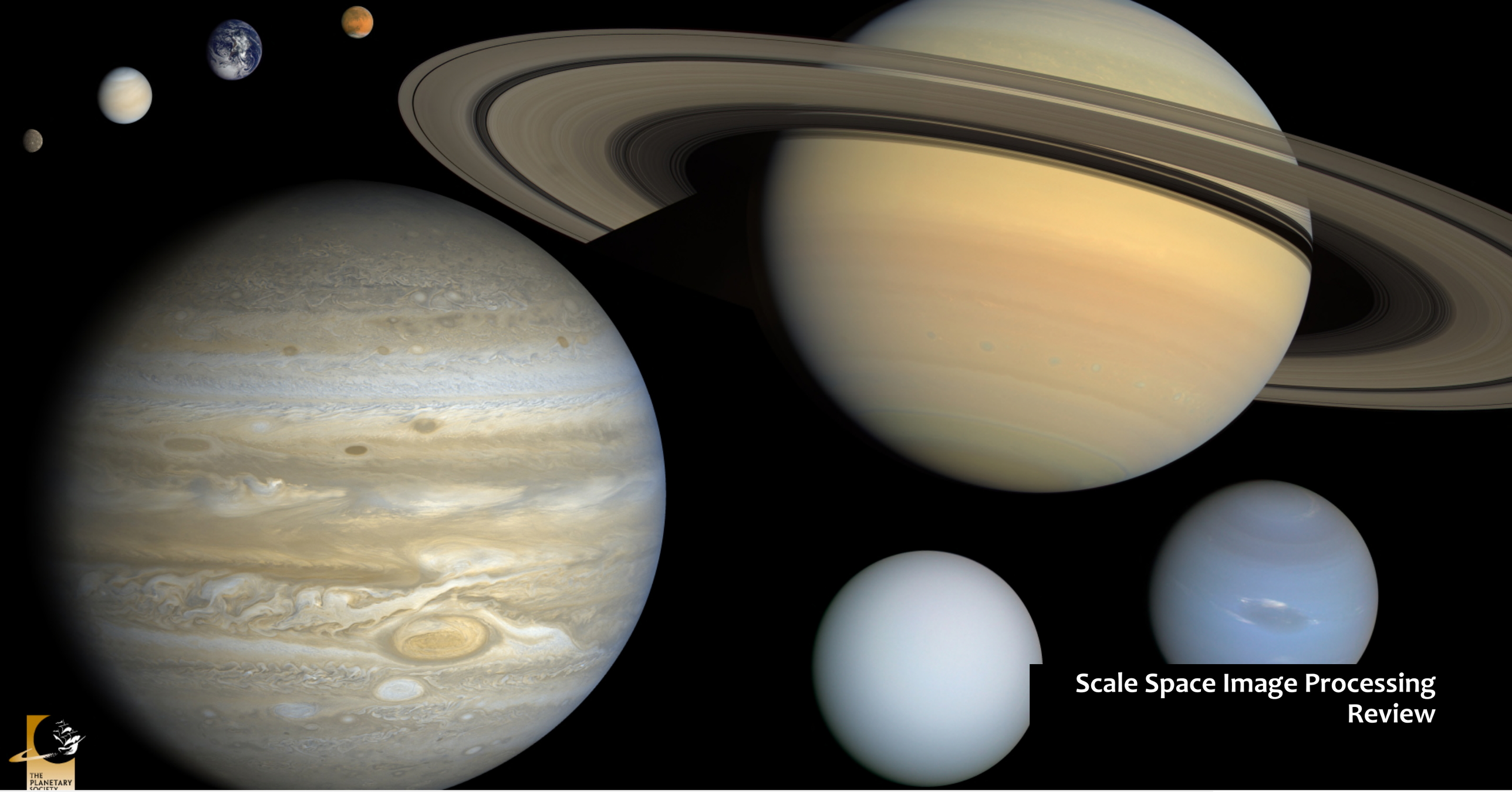


Goal : Local Image Features

- Extract distinctive invariant features
  - Correctly matched against a large database of features from many images

- Invariance to image scale and rotation

- Robustness to
  - Affine (rotation, scale, shear) distortion
  - Change in 3D viewpoint
  - Addition of noise
  - Change in illumination



Scale Space Image Processing Review

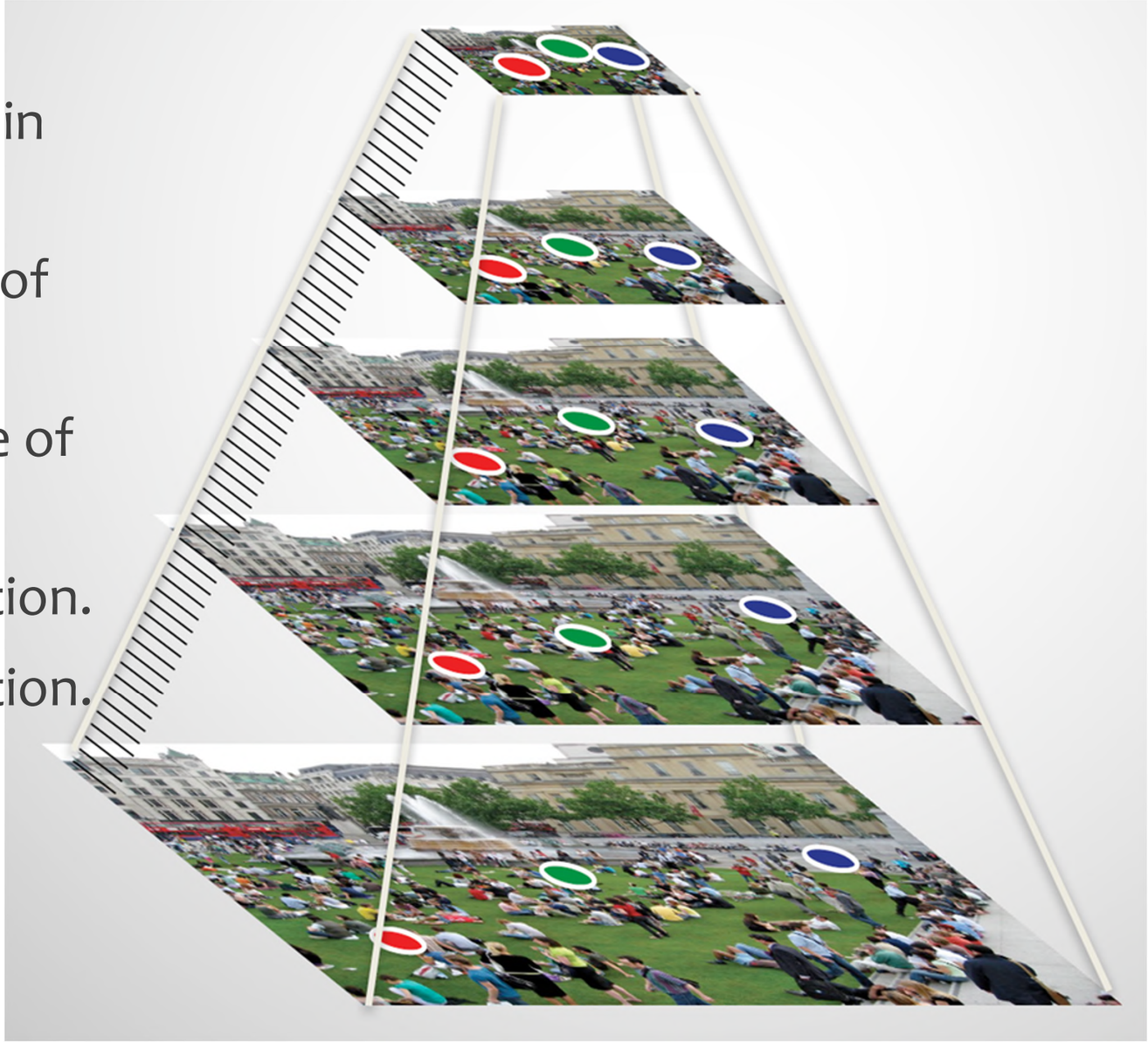
Scale Space Image Processing

- Corresponding image features can appear at different scales

- Like shift-invariance, ***scale-invariance*** of image processing algorithms is often desirable.
- Scale-space representation is useful to process an image in a manner that is both shift-invariant and scale-invariant

Multiscale Representation

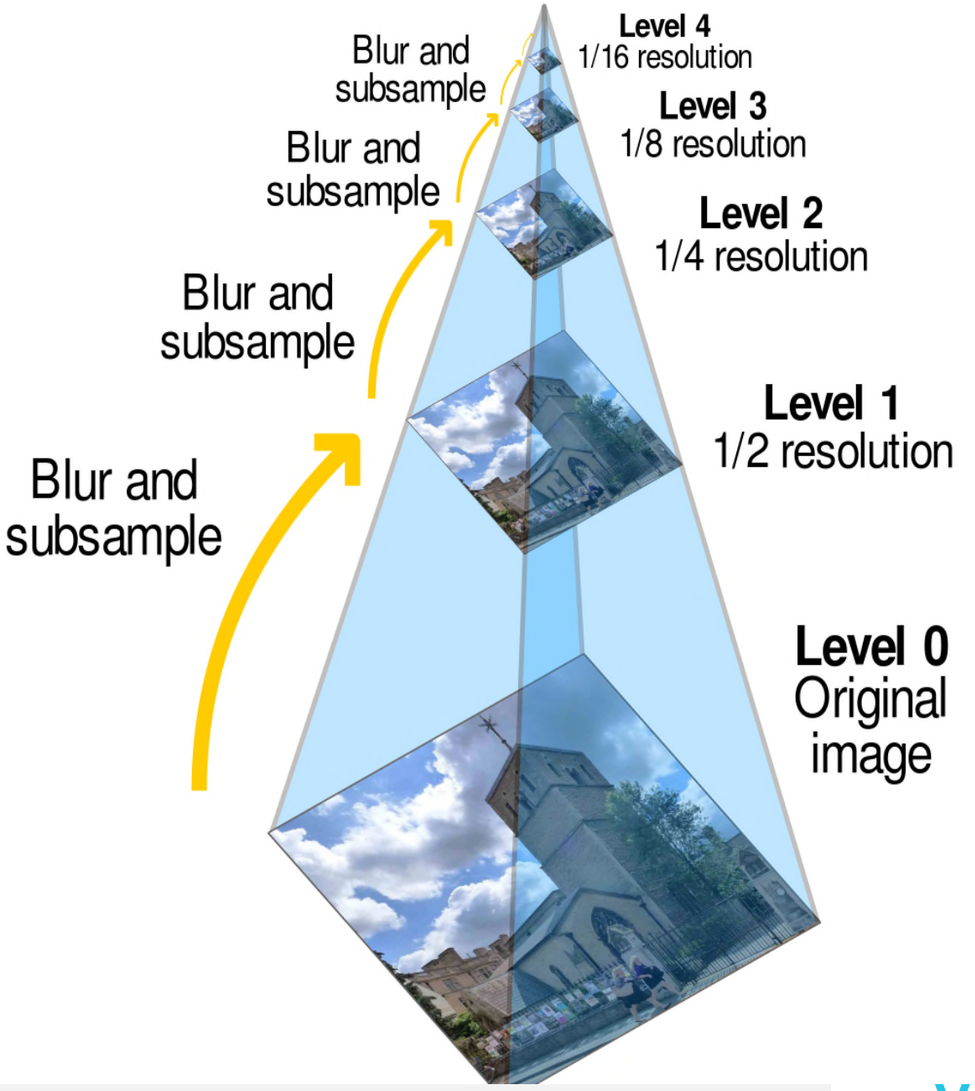
- Pyramid is **one way** to represent images in multi-scale
- Pyramid is built by using multiple copies of image.
- Each level in the pyramid is 1/2 of the size of previous level.
- The **lowest level** is of the highest resolution.
- The **highest level** is of the lowest resolution.



Pyramid can capture global and local features

Gaussian scale space representation

A **Gaussian scale space representation** of an image is the
set of images
that result from
convolving a Gaussian kernel
of
various sizes
with the original image.



Gaussian scale space representation

- What it is?
  - It is a formal theory for handling image structures at different scales,

- How?
  - By representing an image as a one-parameter family of smoothed images,
    - The scale-space representation, parametrized by the size of the smoothing kernel used for suppressing fine-scale structures

Gaussian
t : scale

$$g(x, y; t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t}$$

such that

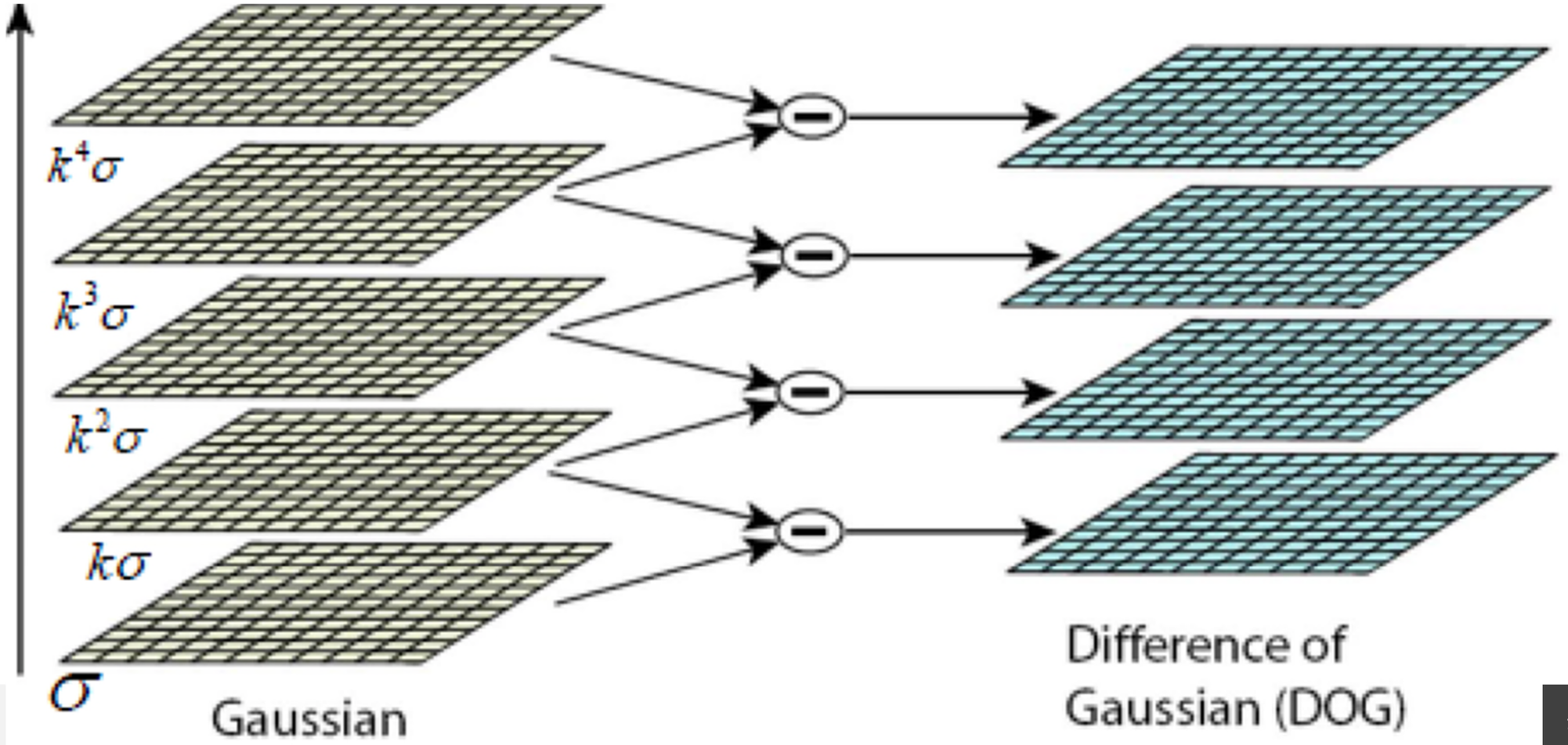
Gaussian Response

$$L(\cdot, \cdot; t) = g(\cdot, \cdot; t) * f(\cdot, \cdot),$$

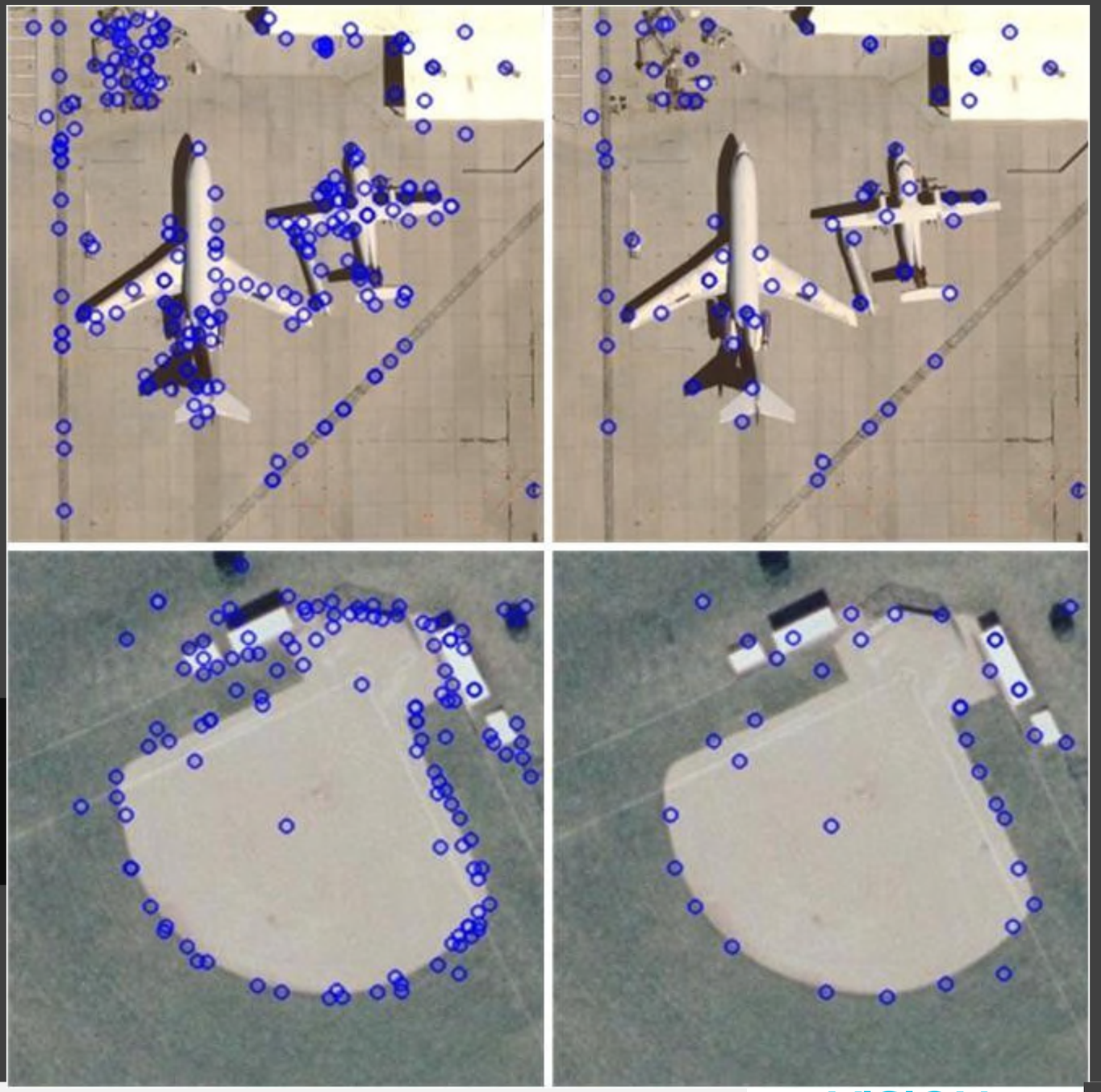
Original Image

Laplacian Pyramid Construction

- Laplacian : Edge detection
- Laplacian : Can be approximated by using Difference of Gaussian

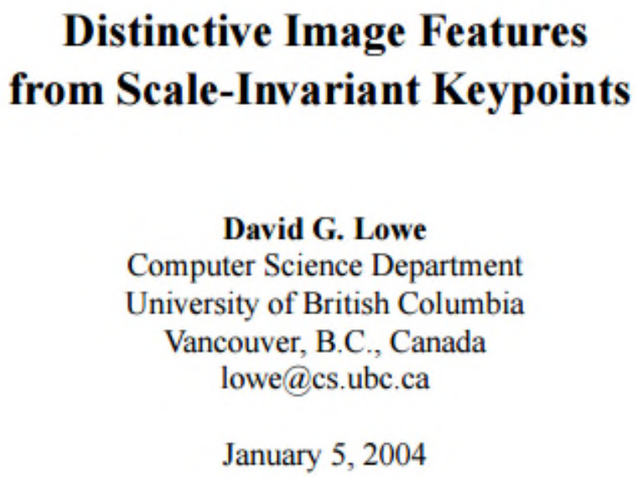


Scale Invariant Feature Transform



SIFT

- Stands for Scale Invariant Feature Transform
- Patented by university of British Columbia
- Similar to the one used in primate visual system (human, ape, monkey, etc.)
- **Transforms image data into scale invariant coordinates**



Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

January 5, 2004

http://www.cs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf

Lowe, D. "Distinctive image features from scale-invariant keypoints" International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

Revisited - Goal : Local Image Features

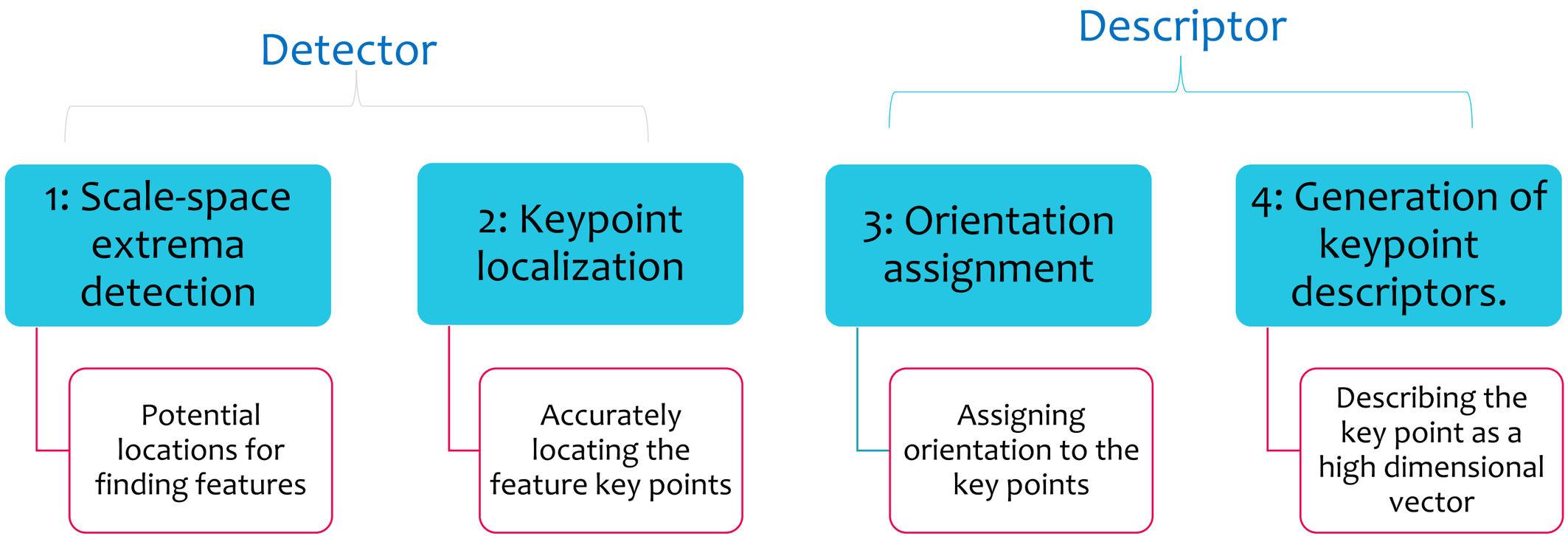
- Extract distinctive invariant features
  - Correctly matched against a large database of features from many images

- Invariance to image scale and rotation

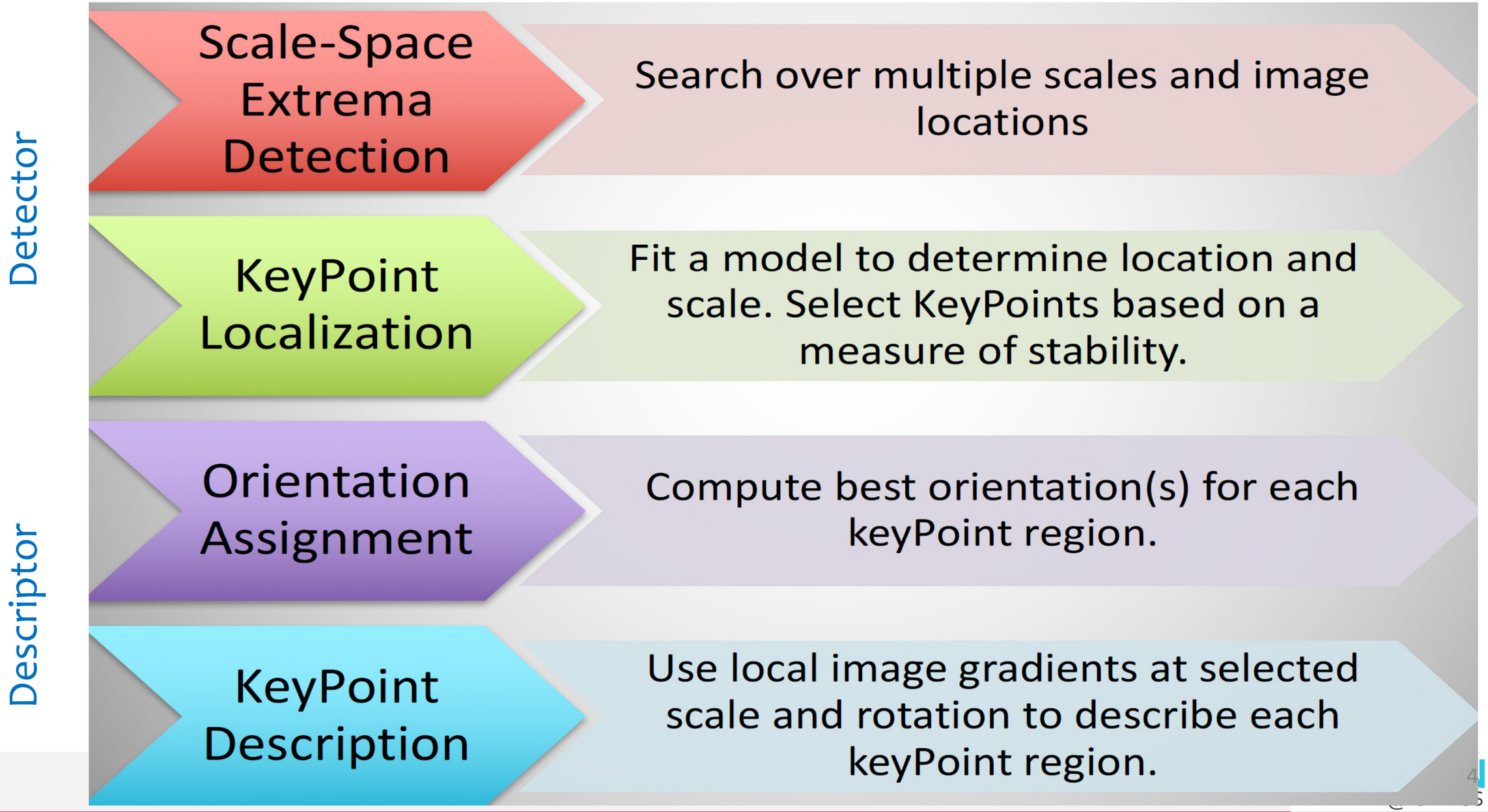
- Robustness to
  - Affine (rotation, scale, shear) distortion
  - Change in 3D viewpoint
  - Addition of noise
  - Change in illumination

Steps for Extracting Key Points (SIF Points)

Broadly speaking, the entire process can be divided into 4 parts:



SIFT : High Level Workflow



SIFT : High Level Workflow

Broadly speaking, the entire process can be divided into 4 parts:

- **Constructing a Scale Space:**
  - To make sure that features are scale-independent
- **Keypoint Localisation:**
  - Identifying the suitable features or keypoints
- **Orientation Assignment:**
  - Ensure the keypoints are rotation invariant
- **Keypoint Descriptor:**
  - Assign a unique fingerprint to each keypoint

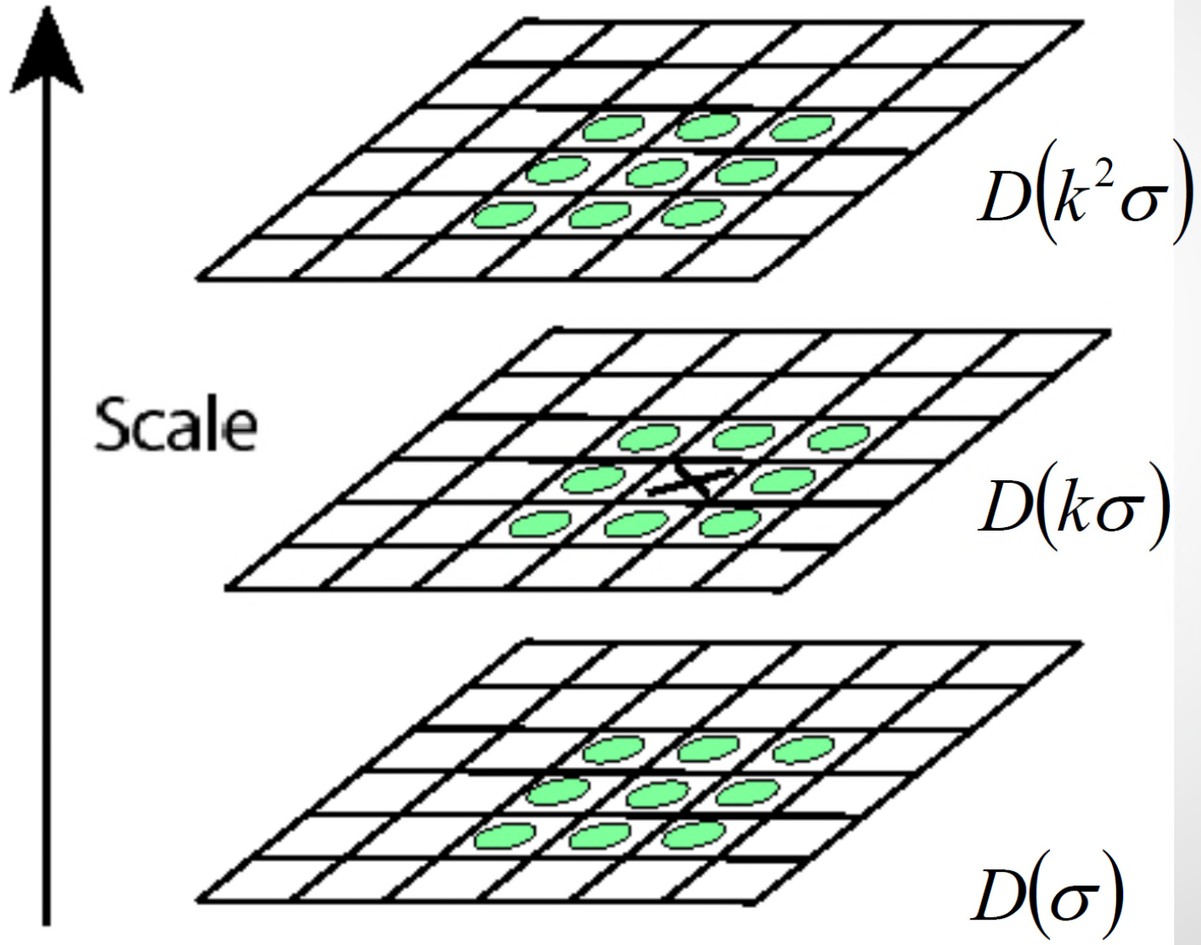
1: Scale Space Extrema Detection

- We want to find points that give us information about the objects in the image.

- The information about the objects lies is in the object's edges / corners / textures or other *interest points* (local features).

- We will represent the image in a way that gives us these *interest points* as this representation's extrema points.

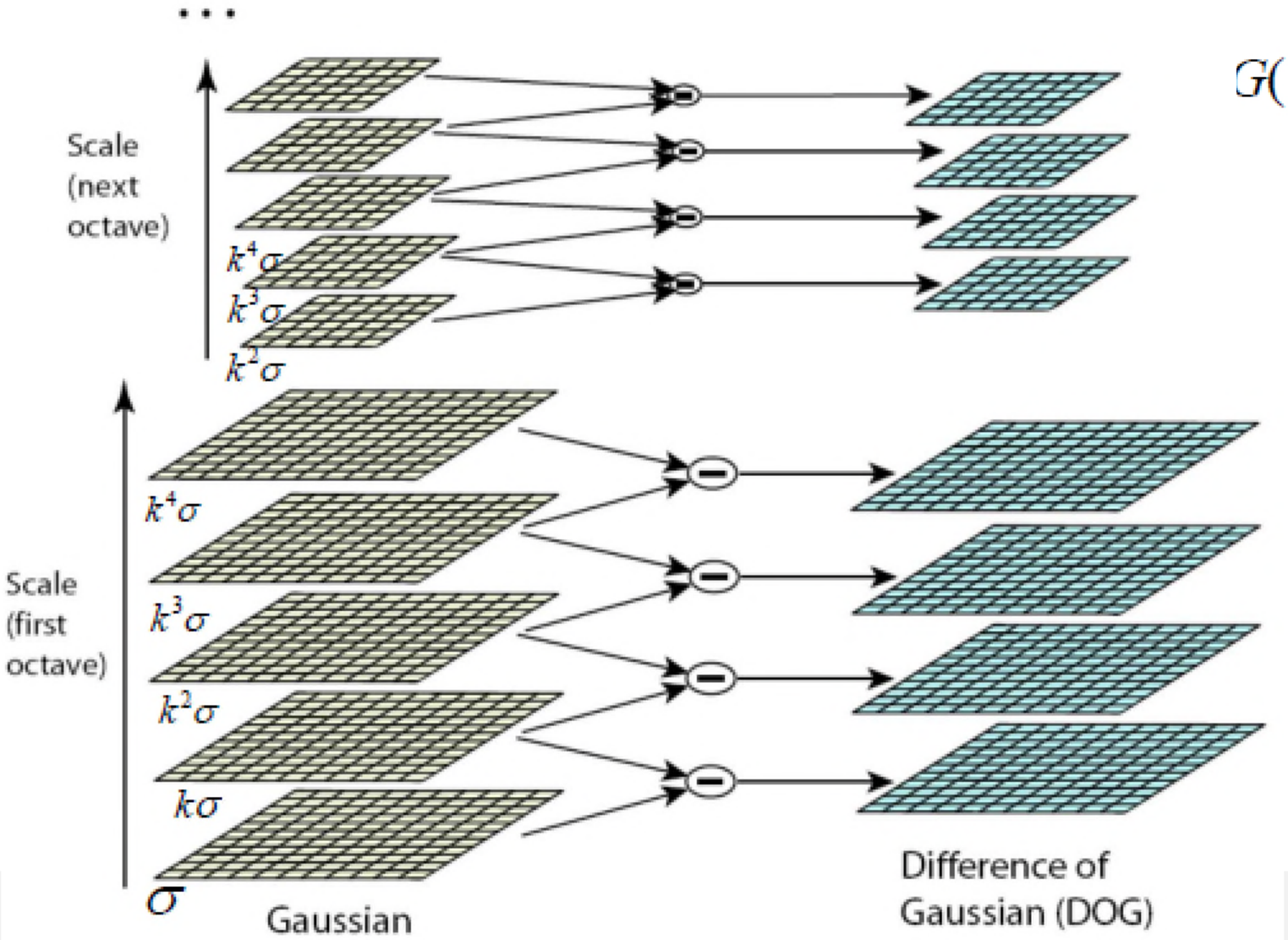
1: Scale Space Extrema Detection



- Extract local extrema (i.e., minima or maxima) in DoG pyramid.
  - -Compare each point to its 8 neighbors at the same level, 9 neighbors in the level above, and 9 neighbors in the level below (i.e., 26 total).

1: Scale-space extrema detection

- **Building a Scale Space**



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

The first step toward the detection of interest points is the convolution of the image with Gaussian filters at different scales, and the generation of difference-of Gaussian images from the difference of adjacent blurred images.

1: Scale-space extrema detection

	scale →				
octave	0.707107	1.000000	1.414214	2.000000	2.828427
	1.414214	2.000000	2.828427	4.000000	5.656854
	2.828427	4.000000	5.656854	8.000000	11.313708
	5.656854	8.000000	11.313708	16.000000	22.627417

$$\sigma = .707187.6;\ k = \sqrt{2}$$

1: Scale-space extrema detection

How many scales per octave?

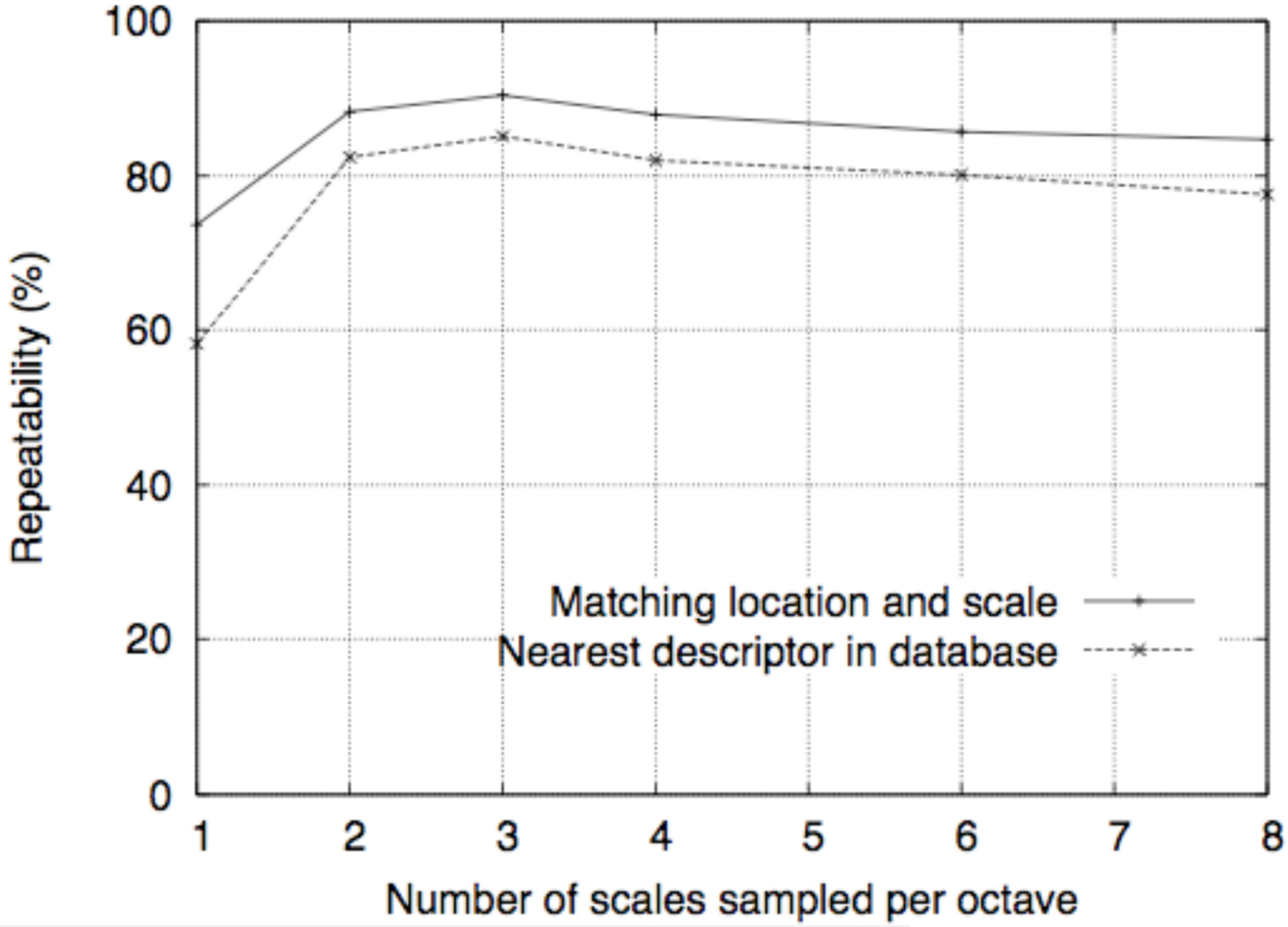
- A collection of 32 real images drawn from a diverse range, including
  - outdoor scenes, human faces, aerial photographs, and industrial

- Each image was then subject to a range of transformations:
  - rotation, scaling, affine stretch,
  - change in brightness and contrast
  - addition of image noise.

1: Scale-space extrema detection

How many scales per octave?

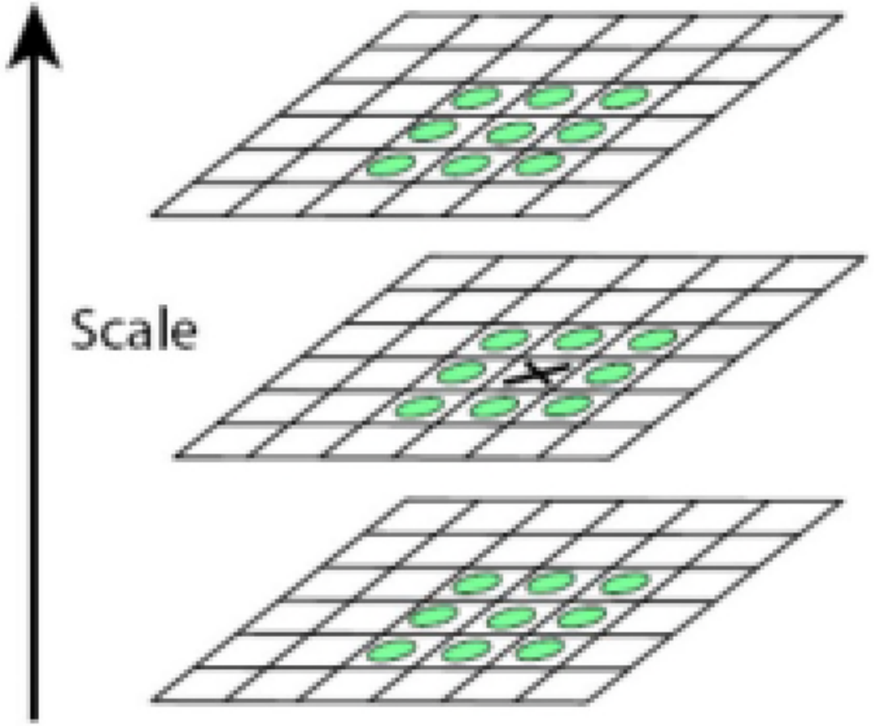
- Repeatability vs number of scales sampled per octave



1: Scale-space extrema detection

Scale Space Peak Detection

- Compare a pixel (**X**) with 26 pixels in current and adjacent scales (**Green Circles**)
- Select a pixel (**X**) if larger/smaller than all 26 pixels
- Large number of extrema, computationally expensive
  - Detect the most stable subset with a coarse sampling of scales

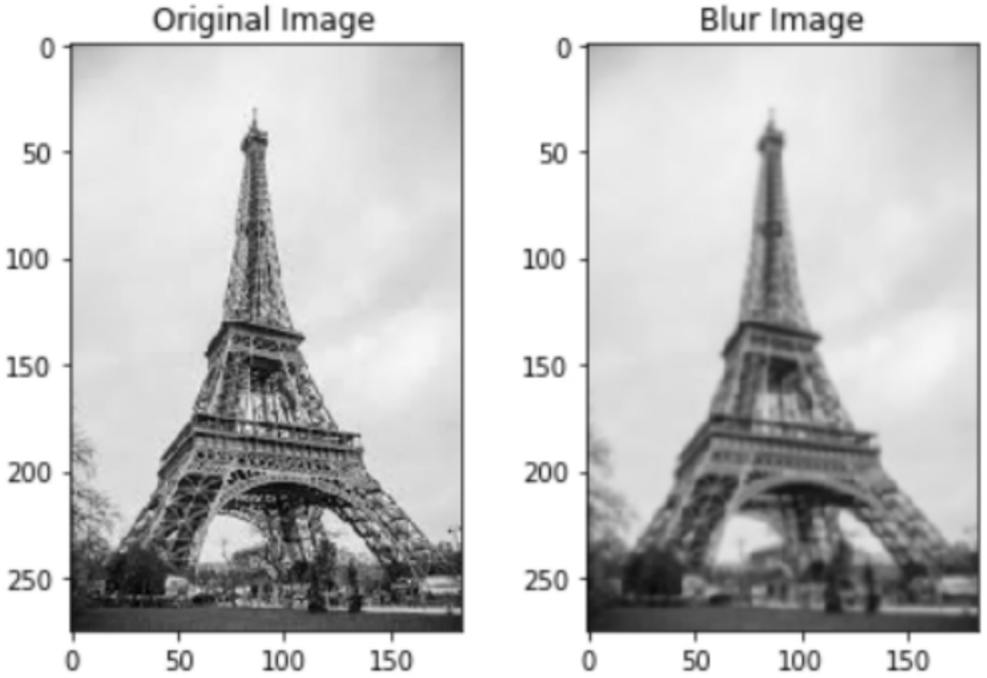


We ae not detecting the edges (zero crossings) rather interest points (LoG response is extrema)

Sample images

1: Constructing the Scale Space

- We need to identify the most distinct features in a given image while **ignoring any noise**.
- Additionally, we need to ensure that the features are **not scale-dependent**.
- These are critical concepts so let's revise about them one-by-one.
- *We use the **Gaussian Blurring technique** to reduce the noise in an image.*



1: Constructing the Scale Space

- ***Scale space*** *is a collection of images having different scales, generated from a single image.*

- Hence, these blur images are created for multiple scales.
  - To create a new set of images of different scales, we will take the original image and reduce the scale by half.
  - For each new image, we will create blur versions.

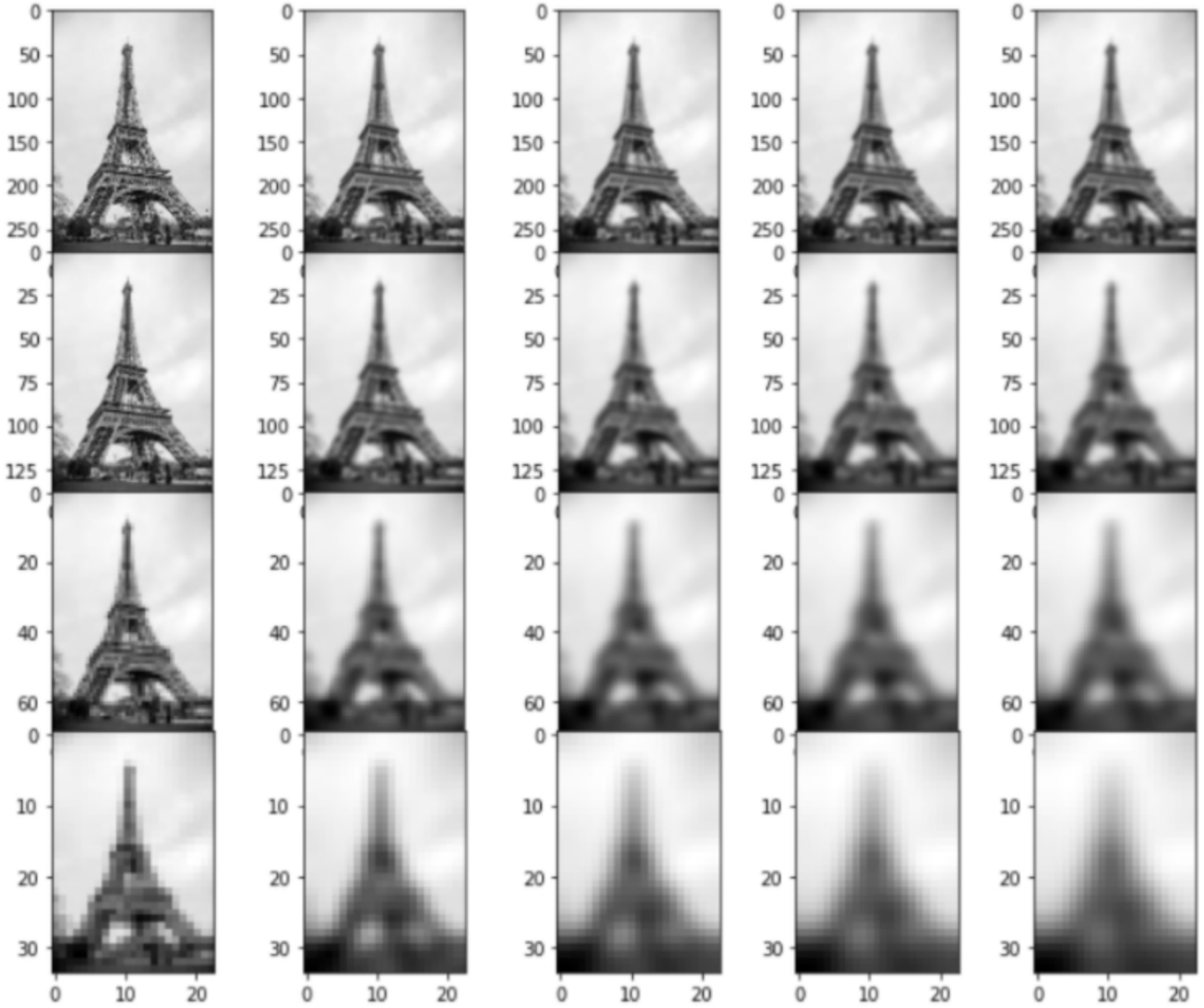
- How many times do we need to scale the image and how many subsequent blur images need to be created for each scaled image?
  - **The ideal number of octaves should be four**, and for each octave, the number of blur images should be five.

First Octave

Second Octave

Third Octave

Fourth Octave

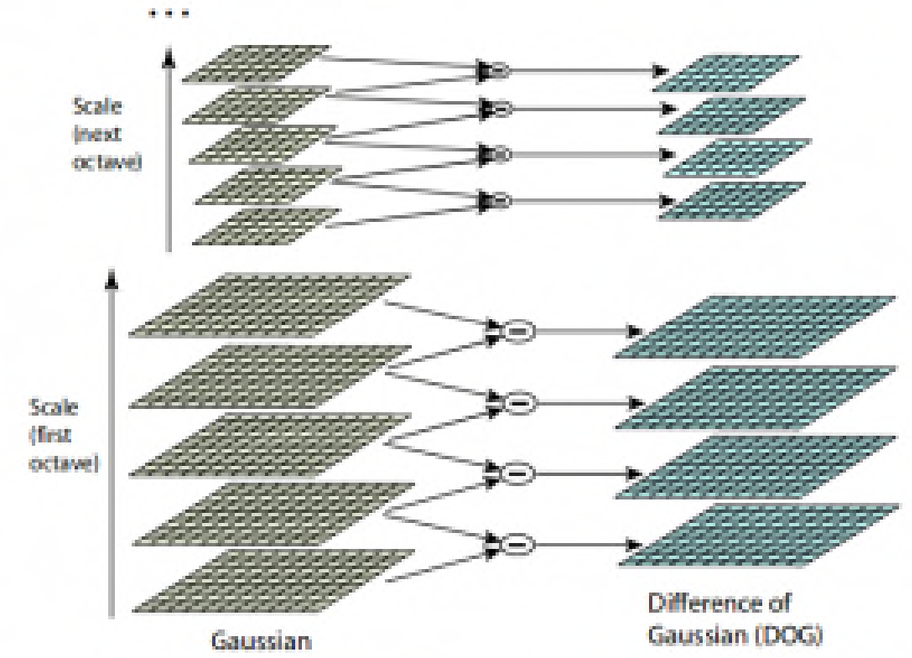


1: Constructing the Scale Space

- ***Difference of Gaussian*** is a feature enhanceme algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original.

- We have 5 images, all from the first octave (thus having the same scale).

- Each subsequent image is created by applying the Gaussian blur over the previous image.

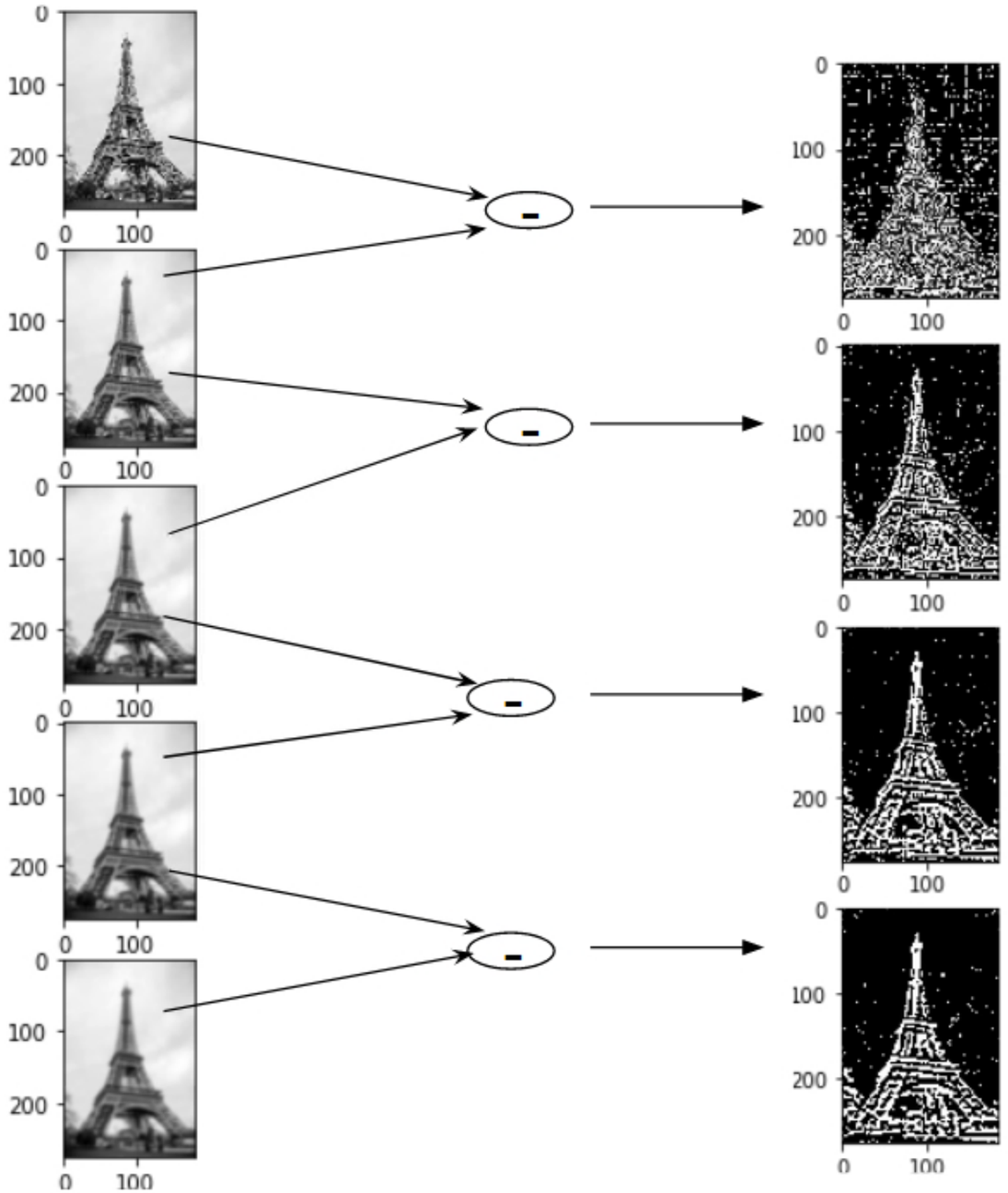


1: Constructing the Scale Space

- On the right, we have four images generated by subtracting the consecutive Gaussians.
- The results are jaw-dropping!

- We have enhanced features for each of these images.

- Note that here I am demonstrating it only for the first octave but the same process happens for all the octaves.

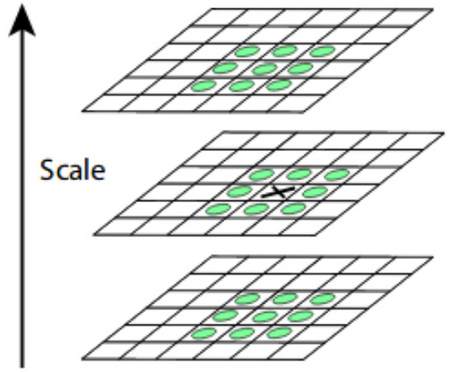


2: Keypoint Localization

- Once the images have been created, the next step is to find the important keypoints from the image that can be used for feature matching.

- **The idea is to find the local maxima and minima for the images.** This part is divided into two steps:
  1. Find the local maxima and minima
  2. Remove low contrast keypoints (keypoint selection)

To locate the local maxima and minima, we go through every pixel in the image and compare it with its neighboring pixels.



We now have potential keypoints that represent the images and are scale-invariant

2: Keypoint Localization

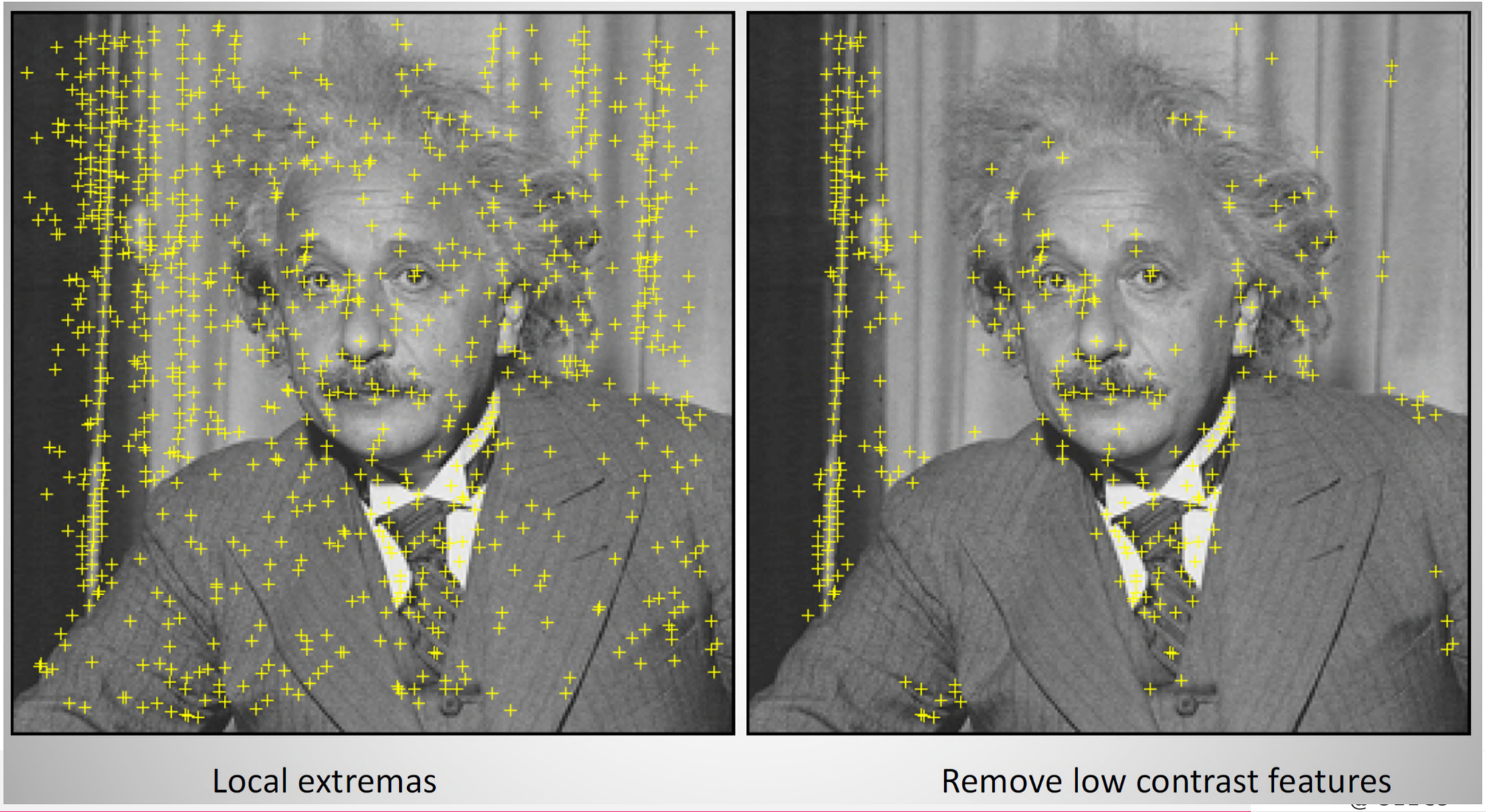
- So far we have successfully generated scale-invariant keypoints.
- But some of these keypoints may not be robust to noise.
- This is why we need to perform a final check to make sure that we have the *most accurate keypoints to represent the image features.*
- **Hence, we will eliminate the keypoints that have low contrast, or lie very close to the edge.**

  - To deal with the **low contrast keypoints**, a second-order Taylor expansion is computed for each keypoint. If the resulting value is less than 0.03 (in magnitude), we reject the keypoint.

  - Next, we perform a check to identify the poorly located keypoints. These are the keypoints that are **close to the edge and have a high edge response** but may not be robust to a small amount of noise.
    - A **second-order Hessian matrix** is used to identify such keypoints.

$$H_x = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

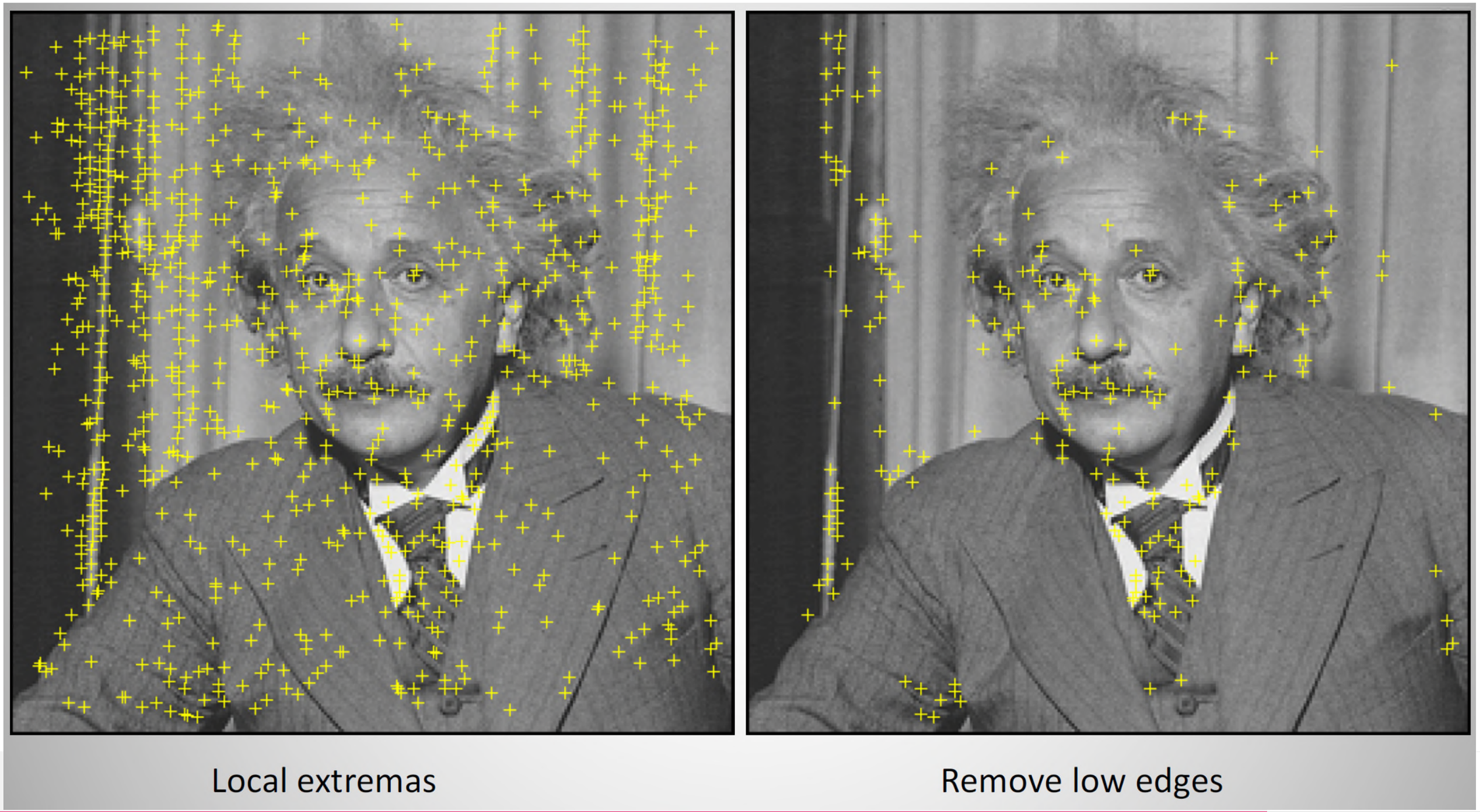
2: Keypoint Localization



Local extremas

Remove low contrast features

2: Keypoint Localization



Local extremas

Remove low edges

Steps for Extracting Key Points (SIF Points)

Ideal Descriptors

- Robust to:
  - Affine transformation
  - Lighting
  - Noise
- Distinctive
- Fast to match
  - Not too large
  - Usually L1 or L2 matching

Detector

1: Scale-space extrema detection

Potential locations for finding features

2: Keypoint localization

Accurately locating the feature key points

Descriptor

3: Orientation assignment

Assigning orientation to the key points

4: Generation of keypoint descriptors.

Describing the key point as a high dimensional vector

3: Orientation Assignment

Now that we have performed both the contrast test and the edge test to reject the unstable keypoints, we will now assign an orientation value for each keypoint to make the rotation invariant.

At this stage, we have a set of stable keypoints for the images. We will now assign an orientation to each of these keypoints so that they are **invariant to rotation**.

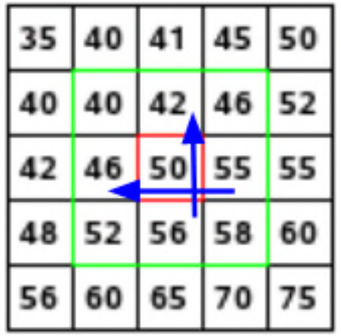
We can again divide this step into two smaller steps:

1. Calculate the magnitude and orientation
2. Create a histogram for magnitude and orientation

3: Orientation Assignment

Let's say we want to find the magnitude and orientation for the pixel value in red.

For this, we will calculate the gradients in x and y directions by taking the difference between 55 & 46 and 56 & 42.
This comes out to be Gx = 9 and Gy = 14 respectively.



Once we have the gradients, we can find the magnitude and orientation using the following formulas:

$$\text{Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]} = 16.64$$

$$\Phi = \text{atan}(Gy / Gx) = \text{atan}(1.55) = 57.17$$

The magnitude represents the intensity of the pixel and the orientation gives the direction for the same.

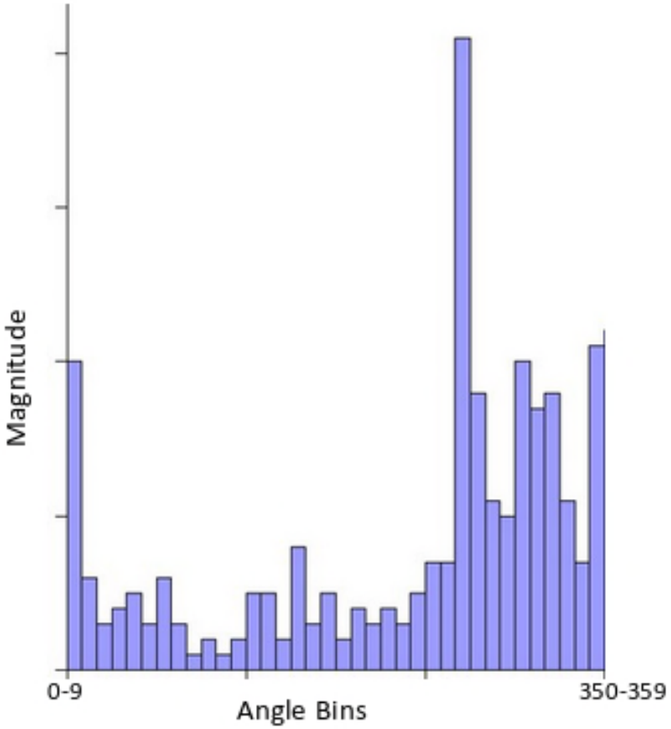
3: Orientation Assignment

3.2: Creating a Histogram for Magnitude and Orientation

On the x-axis, we will have bins for angle values, like 0-9, 10 – 19, 20-29, up to 360.

- Since our angle value is 57, it will fall in the 6th bin.
- The 6th bin value will be in proportion to (multiply with) the magnitude of the pixel, i.e. 16.64.

We will do this for all the pixels around the keypoint.



This histogram would peak at some point. **The bin at which we see the peak will be the orientation for the keypoint**.

Additionally, if there is another significant peak (seen between 80 – 100%), then another keypoint is generated with the magnitude and scale the same as the keypoint used to generate the histogram.
And the angle or orientation will be equal to the new bin that has the peak.

Effectively at this point, we can say that there can be a small increase in the number of keypoints.

4: Keypoint Descriptor

So far, we have stable keypoints that are scale-invariant and rotation invariant

We will use the neighboring pixels, their orientations, and magnitude, to generate a unique fingerprint for this keypoint called a 'descriptor'.

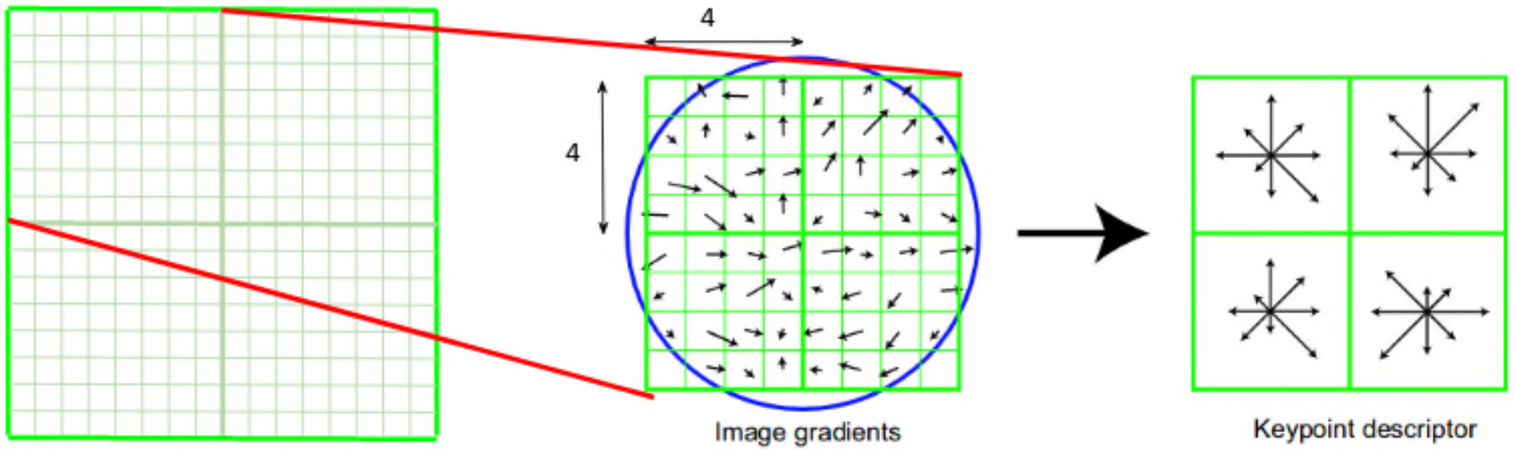
Additionally, since we use the surrounding pixels, the descriptors will be partially invariant to illumination or brightness of the images.

What we will do

We will first take a **16×16 neighborhood** around the **keypoint**. This 16×16 block is further divided into 4×4 sub-blocks

For each of these sub-blocks, we generate the histogram using magnitude and orientation.

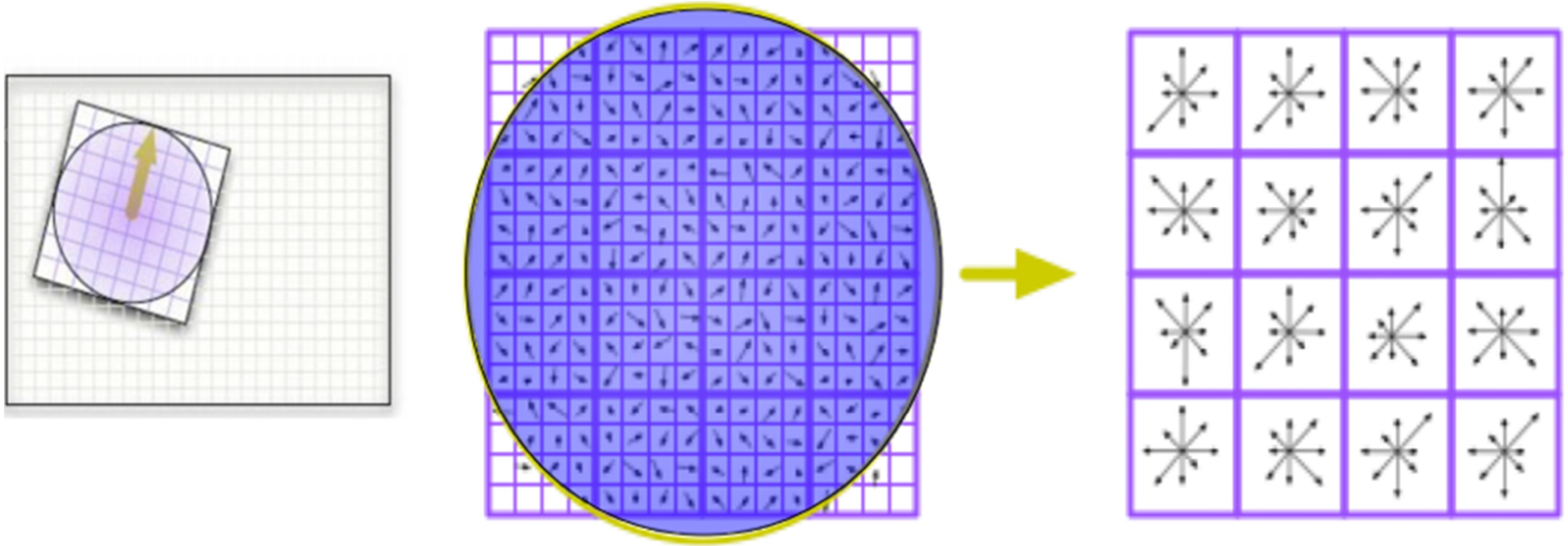
4: Keypoint Descriptor



At this stage, the bin size is increased and we take only 8 bins (not 36).
Each of these arrows represents the 8 bins and the length of the arrows define the magnitude.
So, we will have a total of 128 bin values for every keypoint.

4: Keypoint Descriptor

- Compute relative orientation and magnitude in a 16x16 neighbourhood at **keypoint**
  - Form weighted **histogram** (8 bin) for 4x4 regions
  - Weight by magnitude and spatial Gaussian
  - Concatenate 16 histograms in one long vector of 128 dimensions



4: SIFT Descriptor (summary)

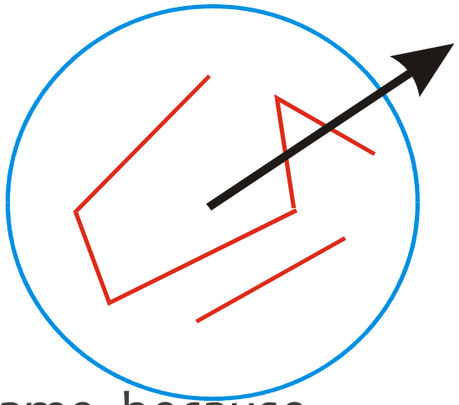
- Once a key point orientation has been selected, the feature descriptor is computed as a set of orientation histograms on 4×4 pixel neighbourhoods.
- The orientation histograms are relative to the keypoint orientation, the orientation data comes from the Gaussian image closest in scale to the keypoint's scale.
- Just like before, the contribution of each pixel is weighted by the gradient magnitude, and by a Gaussian with σ x 1.5 times the scale of the keypoint.
- Histograms contain 8 bins each, and each descriptor contains an array of 4 histograms around the keypoint.
- This leads to a SIFT feature vector with 4 × 4 × 8 = 128 elements.

Becoming rotation invariant

- We are given a keypoint and its scale from DoG

- We will select a characteristic orientation for the keypoint (based on the most prominent gradient there)

- We will describe all features **relative** to this orientation



- Causes features to be rotation invariant!
  - If the keypoint appears rotated in another image, the features will be the same, because they're **relative** to the characteristic orientation

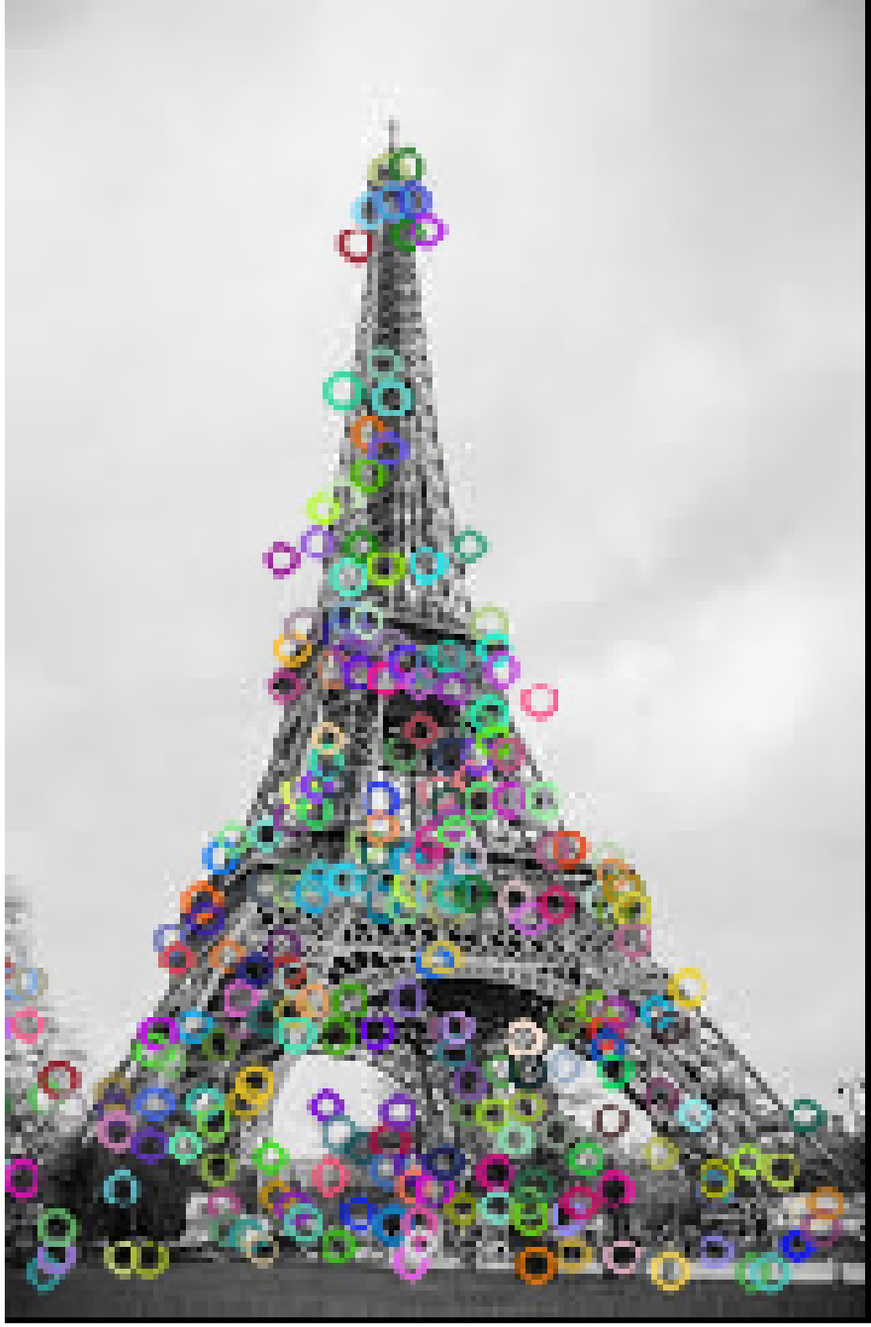
Change of Illumination

- **Change of brightness =>** doesn't effect gradients (difference of pixels value).
- **Change of contrast =>** doesn't effect gradients (up to normalization).
- **Saturation (non-linear change of illumination) =>** affects magnitudes much more than orientation.

  => Threshold gradient magnitudes to 0.2 and renormalize.

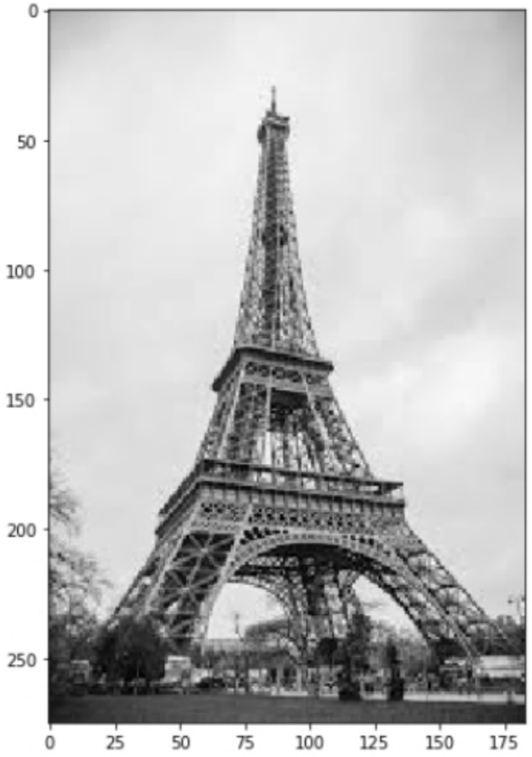
How to do hands On?

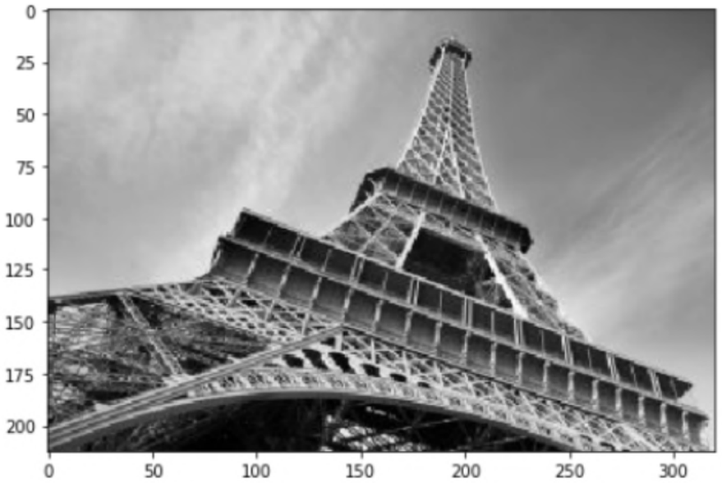
```
1   import cv2
2   import matplotlib.pyplot as plt
3   %matplotlib inline
4
5   #reading image
6   img1 = cv2.imread('eiffel_2.jpeg')
7   gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
8
9   #keypoints
10  sift = cv2.xfeatures2d.SIFT_create()
11  keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
12
13  img_1 = cv2.drawKeypoints(gray1,keypoints_1,img1)
14  plt.imshow(img_1)
```



SIFT keypoints

SIFT keypoints

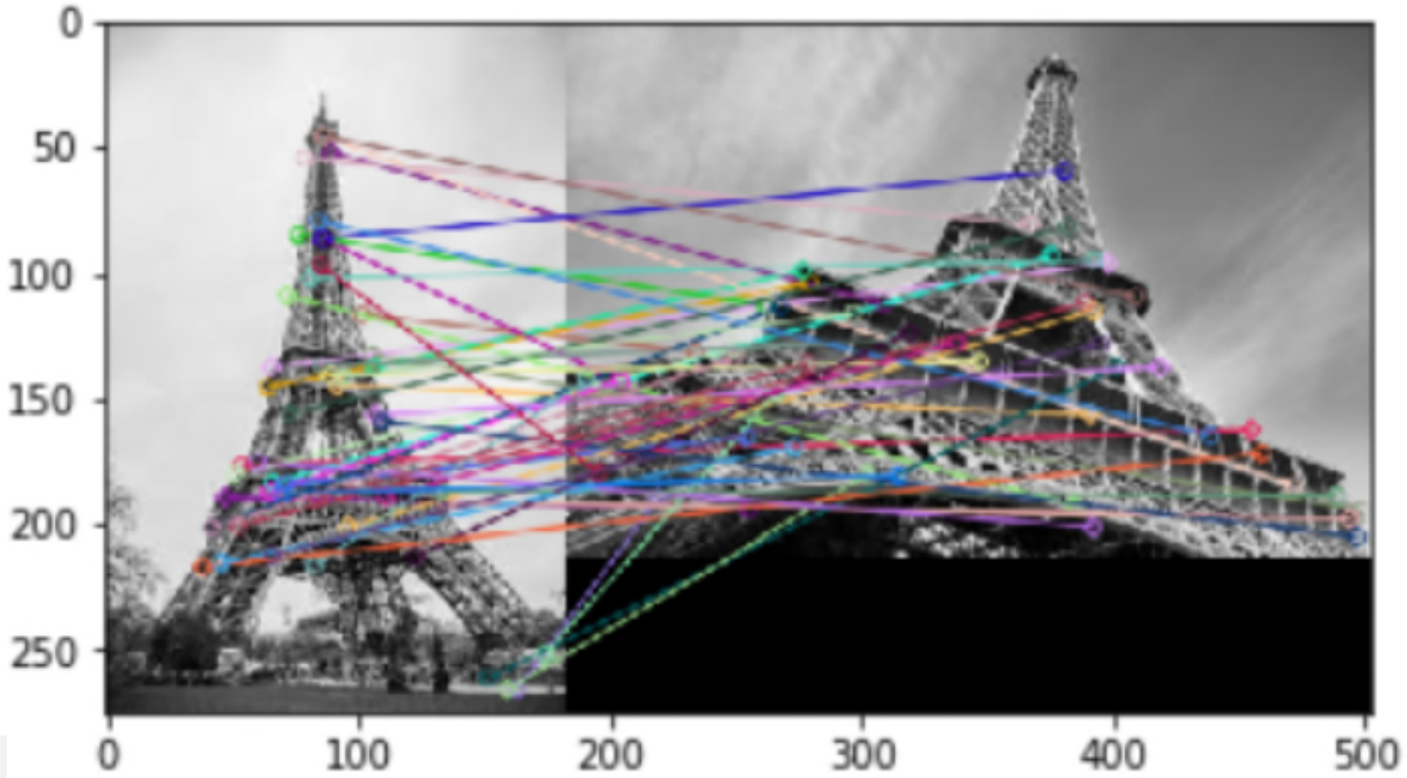




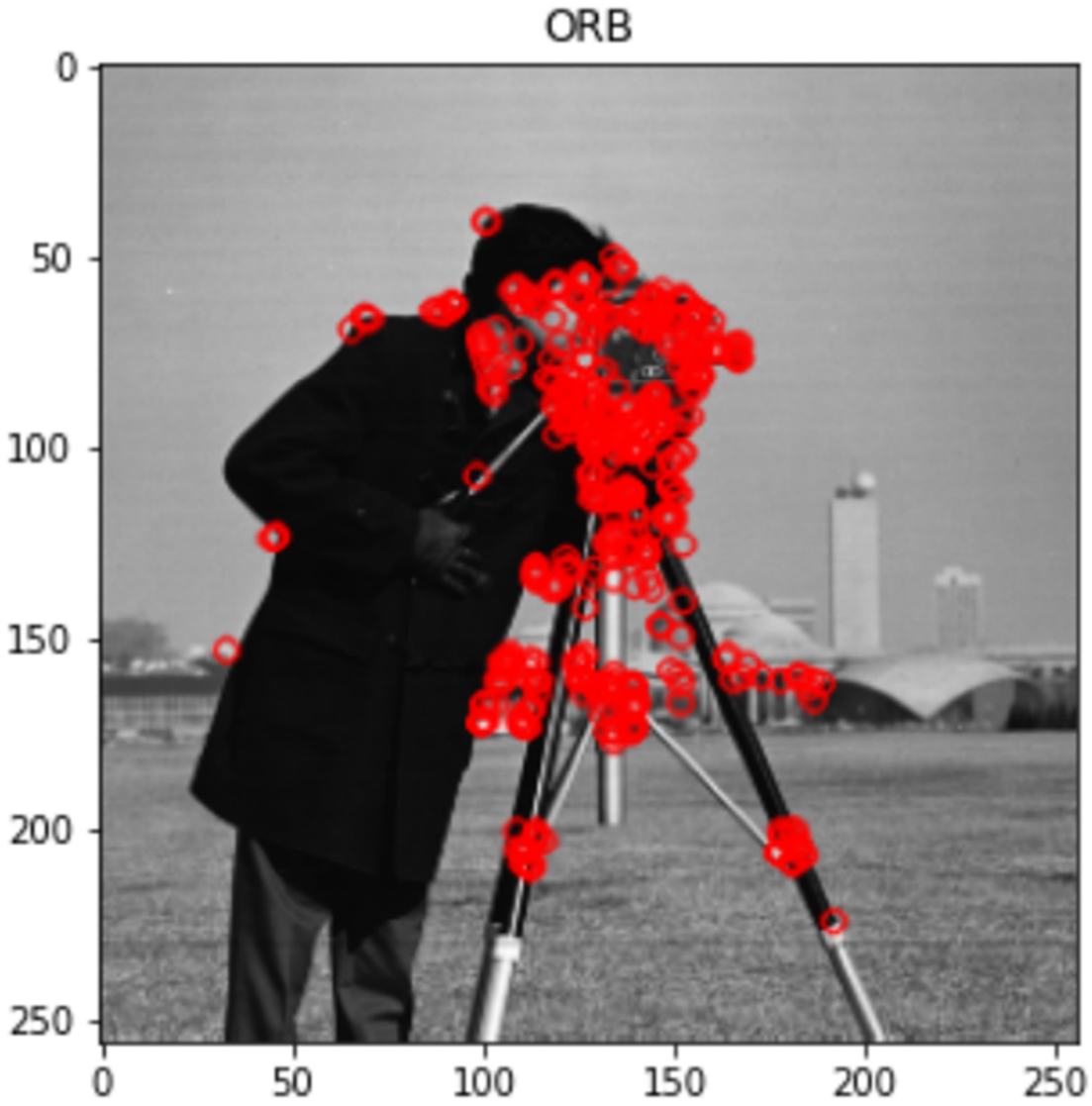

```
1   import cv2
2   import matplotlib.pyplot as plt
3   %matplotlib inline
4
5   # read images
6   img1 = cv2.imread('eiffel_2.jpeg')
7   img2 = cv2.imread('eiffel_1.jpg')
8
9   img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
10  img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
11
12  figure, ax = plt.subplots(1, 2, figsize=(16, 8))
13
14  ax[0].imshow(img1, cmap='gray')
15  ax[1].imshow(img2, cmap='gray')
```

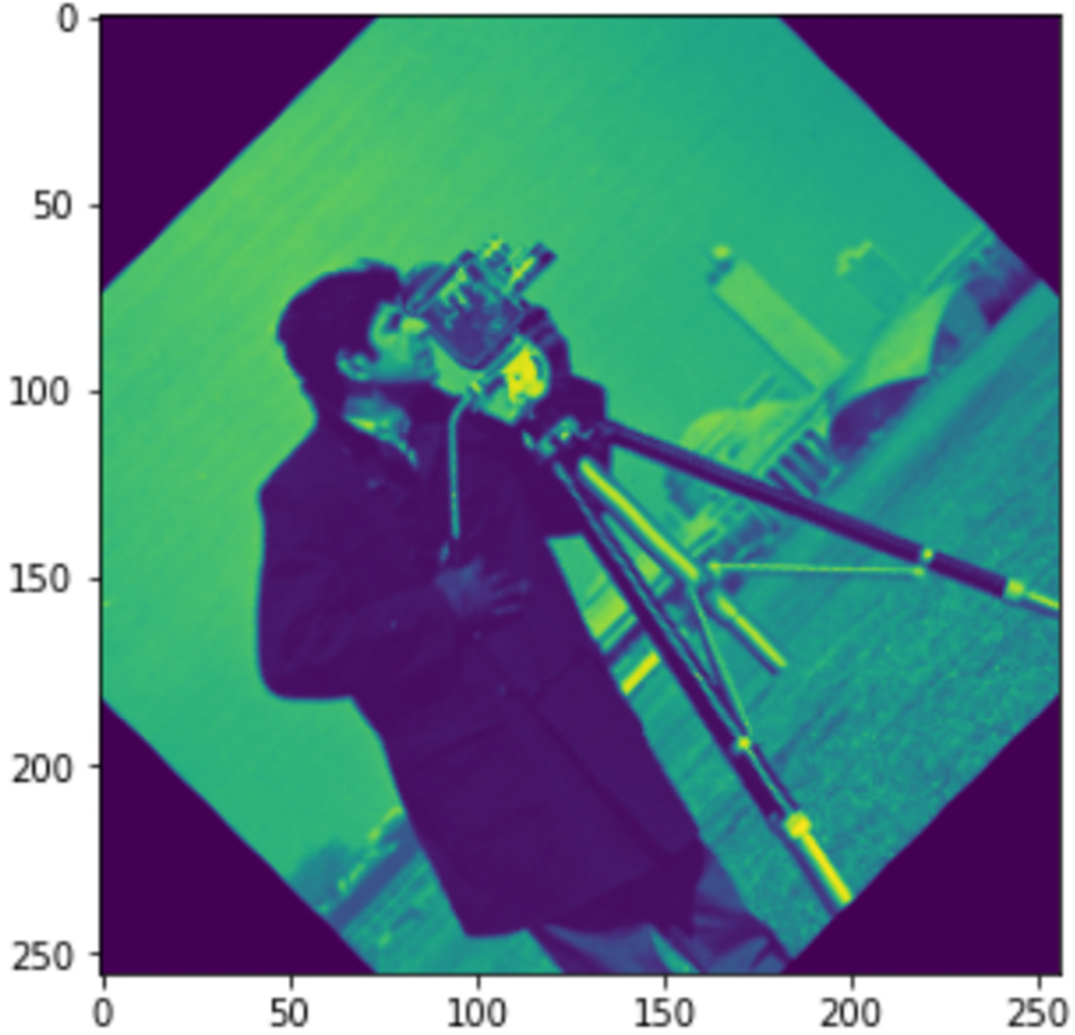
```python
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# read images
img1 = cv2.imread('eiffel_2.jpeg')
img2 = cv2.imread('eiffel_1.jpg')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

#sift
sift = cv2.xfeatures2d.SIFT_create()

keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)

len(keypoints_1), len(keypoints_2)
```

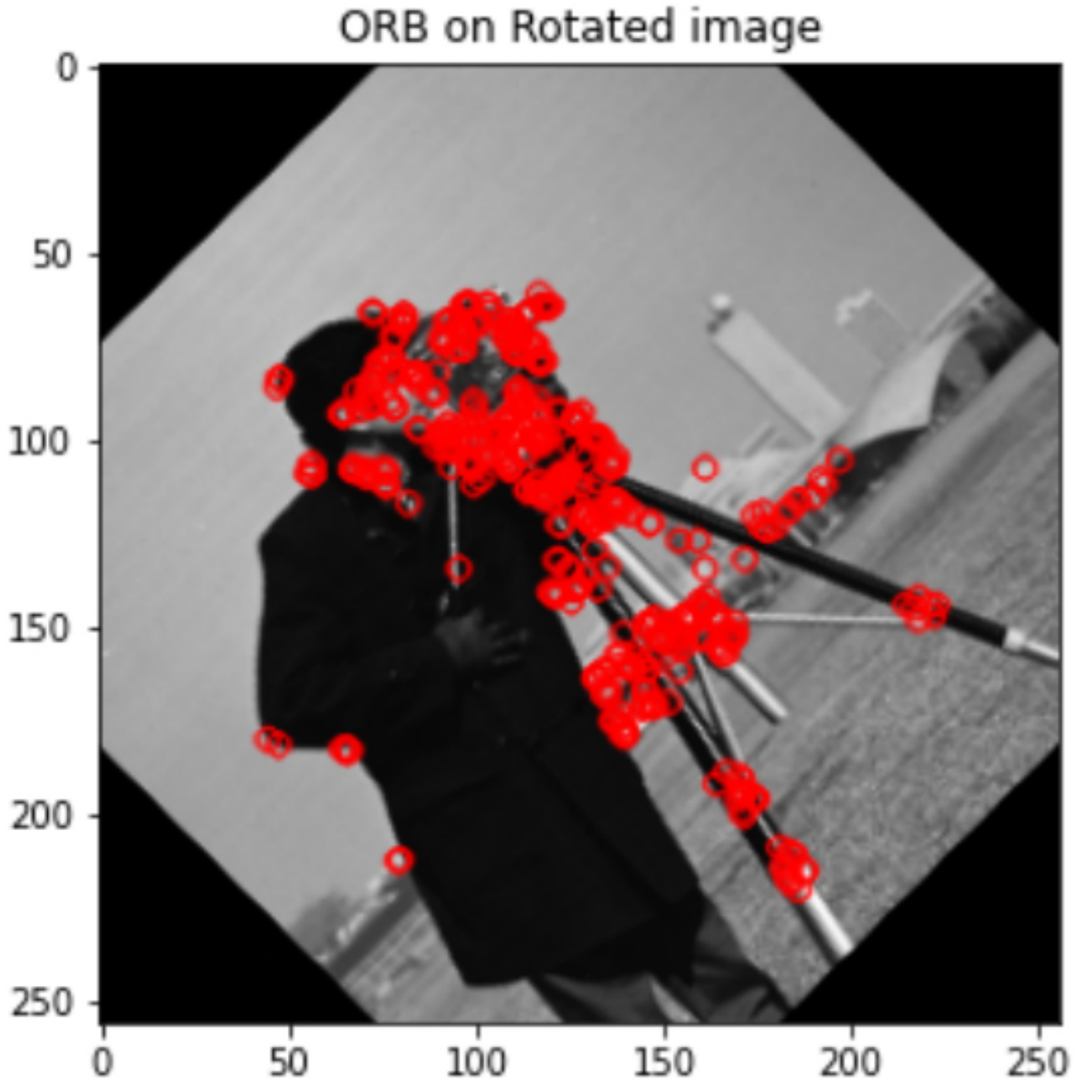
```python
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)

img3 = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, matches[:50], img2, flags=2)
plt.imshow(img3),plt.show()
```



```
img = cv2.imread('cameraman.png',0)
print(img.shape)

# Initiate STAR detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with ORB
kp_orb, des_orb = orb.detectAndCompute(img, None)

# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img, kp_orb, None, color=(255,0,0))
plt.imshow(img2)
plt.title("ORB")
plt.show()
```




0
50
100
150
200
250
0
50
100
150
200
250


ORB on Rotated image
0
50
100
150
200
250
0
50
100
150
200
250

Brute Force Matcher

- It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

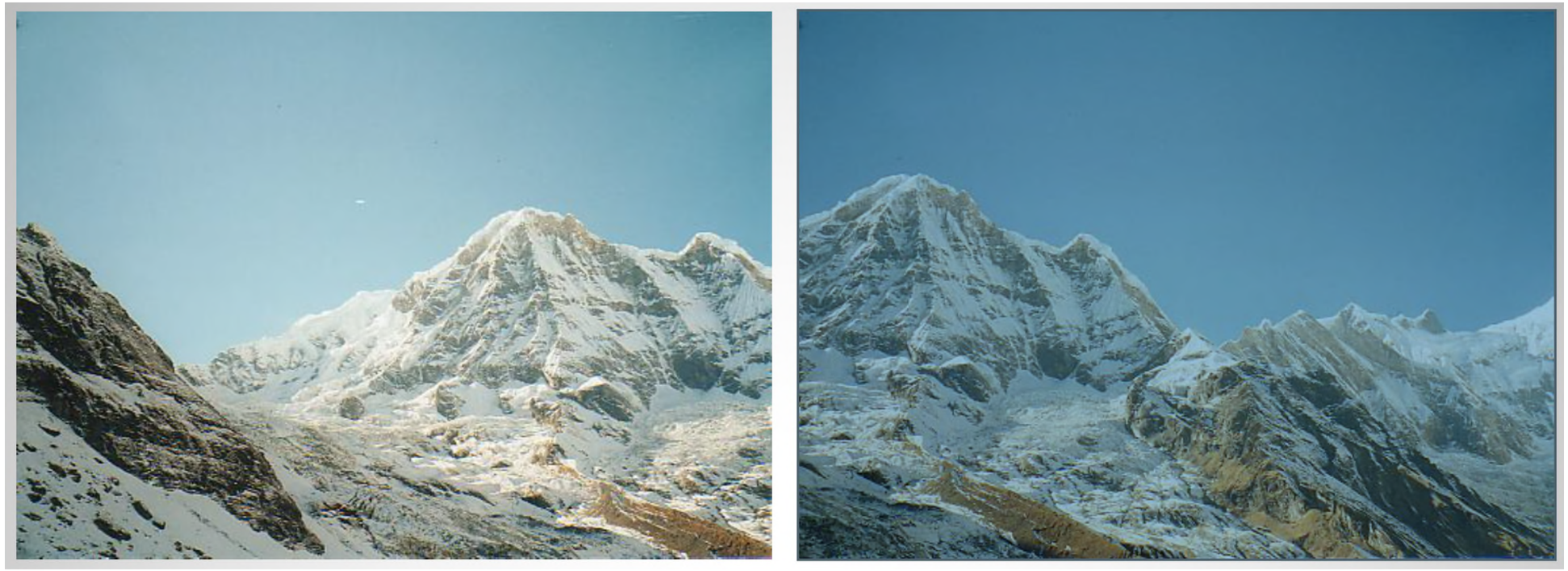
```
# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf.match(des_orb, des_orb_rotated)
matches = sorted(matches, key = lambda x:x.distance)
print(len(matches))
#lets show top 10 features matched
image_with_feature_matched = cv2.drawMatches(img, kp_orb, img_rotated, kp_orb_rotated, matches[:10], None, flags=2)
plt.imshow(image_with_feature_matched)
plt.title("Demonstration of top 10 features matched")
plt.show()
```

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html


Demonstration of top 10 features matched
0
50
100
150
200
250
0
100
200
300
400
500

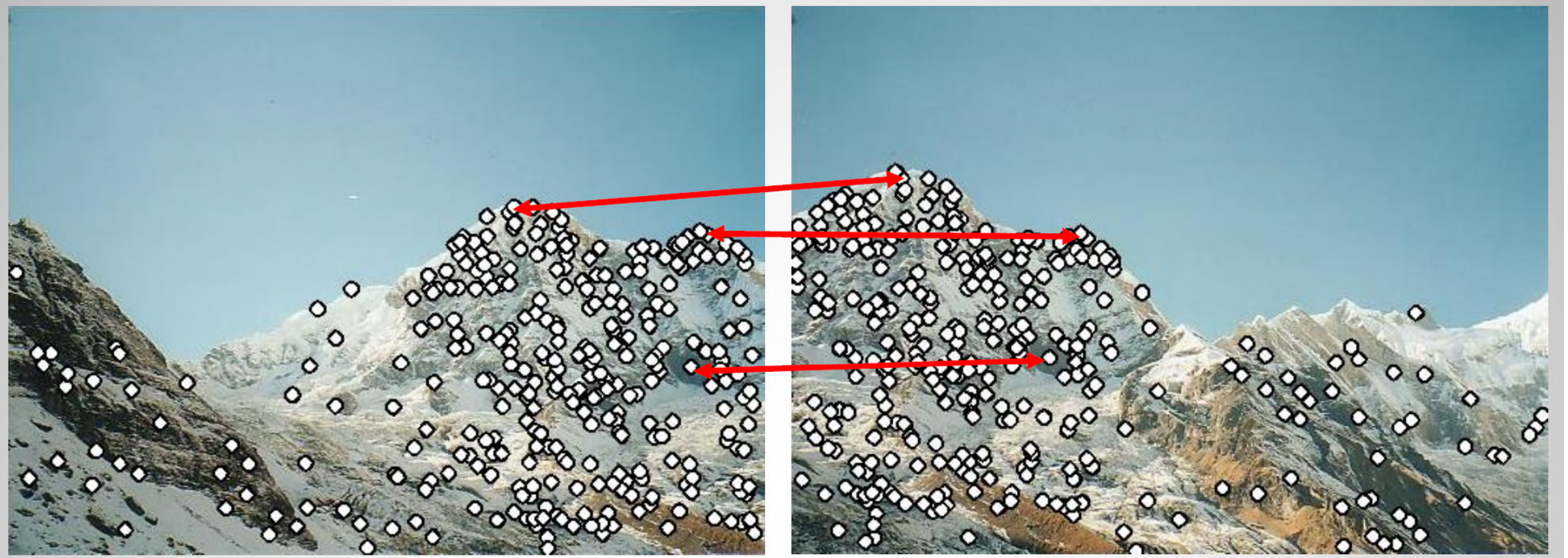
Applications of SIFT

- Object recognition
- Object categorization
- Location recognition
- Robot localization
- Image retrieval
- Image panoramas

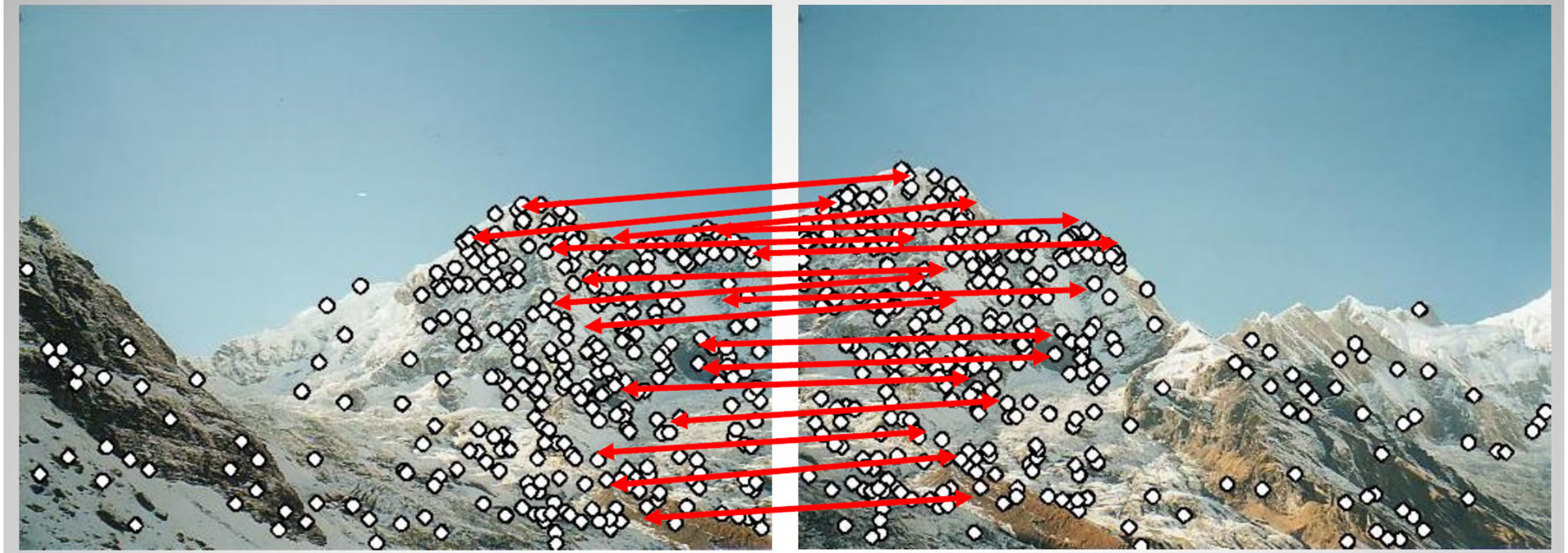


- Extract features

- Extract features
- Compute *putative matches*



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

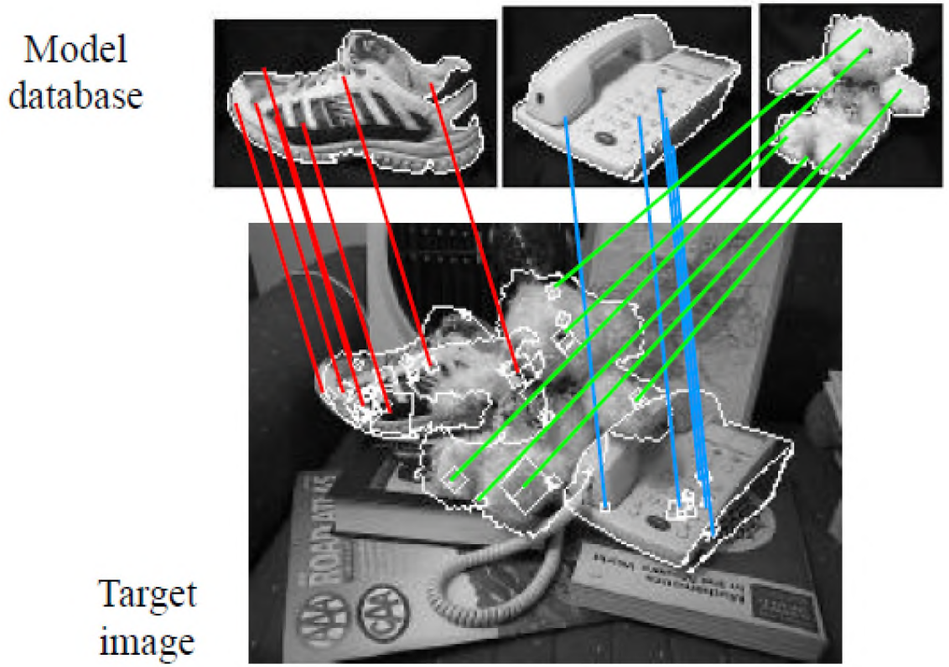
- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Object Recognition

- **For training images:**
  - Extracting keypoints by SIFT.
  - Creating descriptors database.
- **For query images:**
  - Extracting keypoints by SIFT.
  - For each descriptor - finding nearest neighbor in DB.
  - Finding cluster of at-least 3 keypoints.
  - Performing detailed geometric fit check for each cluster.

Object recognition (David Lowe)

Establish correspondence between the target image and (multiple) images in the model database



Object recognition (David Lowe)

- Establish correspondence between the query image and all images from the database depicting the same object or scene

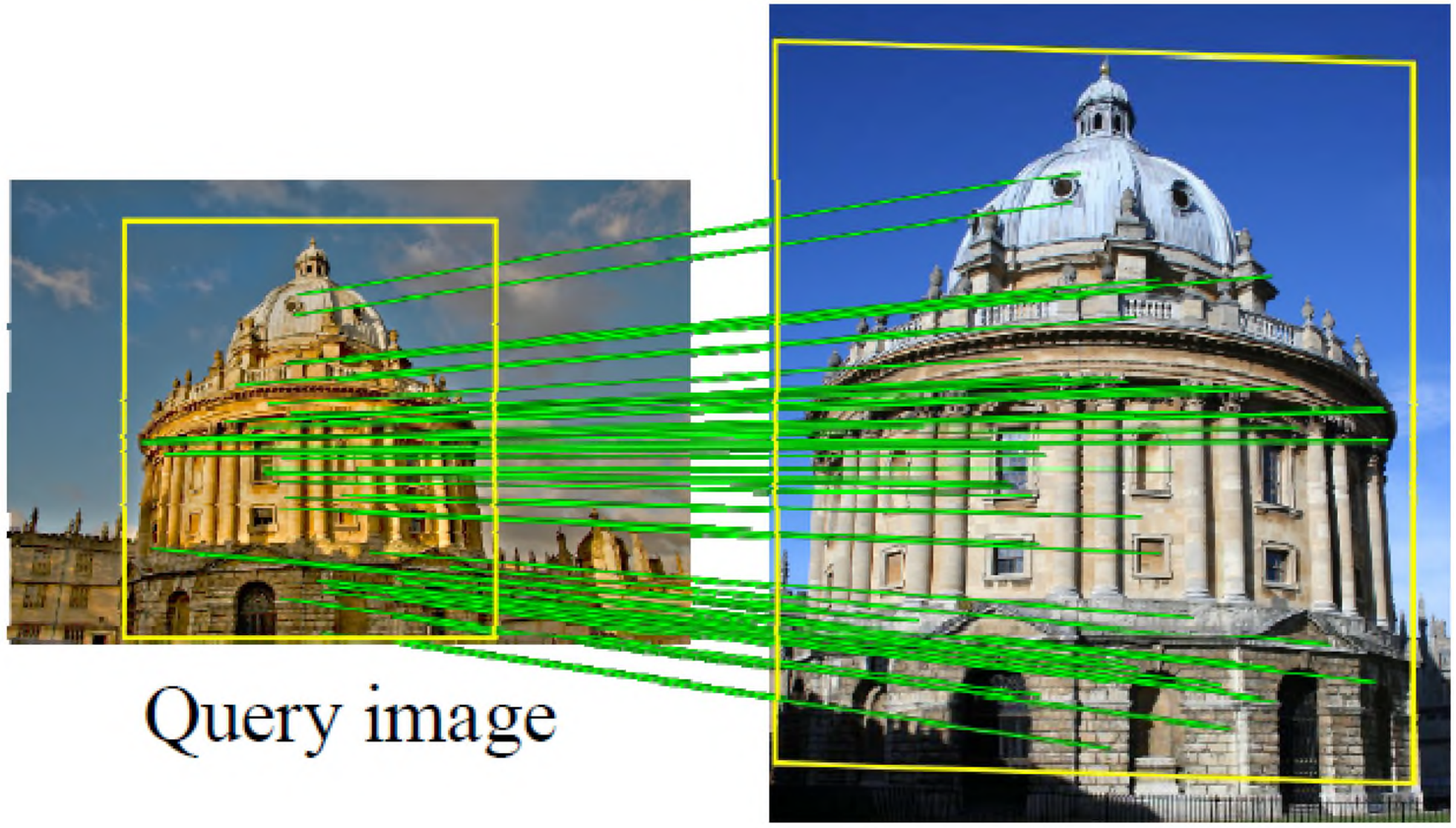


Image Registration

Automatic mosaicing

http://www.cs.ubc.ca/~mbrown/autosOtch/autosOtch.html

Wide baseline stereo

Performance

- Very robust
  - 80% Repeatability at:
    - 10% image noise
    - 45° viewing angle
    - 1k-100k keypoints in database
- Best descriptor in [Mikolajczyk & Schmid 2005]'s extensive survey
- 3670+ citations on Google Scholar

Other scale invariant feature detectors and descriptors

- **SURF** (Speeded-Up Robust Features)
- **FAST** (Features from Accelerated Segment Test)
- **BRIEF** (Binary Robust Independent Elementary Features)
- **ORB** (Oriented FAST and Rotated BRIEF)

Variations of SIFT features

- **PCA-SIFT**
  - http://www.cs.cmu.edu/~yke/pcasift/
- **SURF**
  - http://www.vision.ee.ethz.ch/~surf/eccv06.pdf
  - https://courses.cs.washington.edu/courses/cse576/13sp/projects/project1/artifacts/woodrc/index.htm
- **GLOH**
  - http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/10/slide_corso/A32_keypoint_detectors.pdf

Reading Material

- **Online tutorial:**
  - hmp://www.aishack.in/2010/05/sit-scaleinvariant-feature-transform/
  - Instructor: Dr. Mubarak Shah (University of Central Florida)
    - https://www.youtube.com/watch?v=NPcMS49V5hg
    - https://www.youtube.com/watch?v=KvapdyOERqE
  - **How to do SIFT for Object Recognition**
    - https://www.youtube.com/watch?v=PU6BKIh1y40
  - **Wikipedia:**
    - http://en.wikipedia.org/wiki/Scaleinvariant_feature_transform

- https://www.kaggle.com/wesamelshamy/tutorial-image-feature-extraction-and-matching

- **https://github.com/visionatseecs/opencv-starter/blob/main/06.ipynb**
  - NBVIEWER Link (https://nbviewer.jupyter.org/github/visionatseecs/opencv-starter/blob/main/06.ipynb)

References

- **References**
  - http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MURRAY/SIFT.html
  - http://robots.stanford.edu/cs223b04/project9.html
- **Lots of Implementations available**
  - VLFeat toolbox: http://www.vlfeat.org/
  - http://www.cs.ubc.ca/~lowe/keypoints/

Nice Read

A Performance Evaluation of Local Descriptors

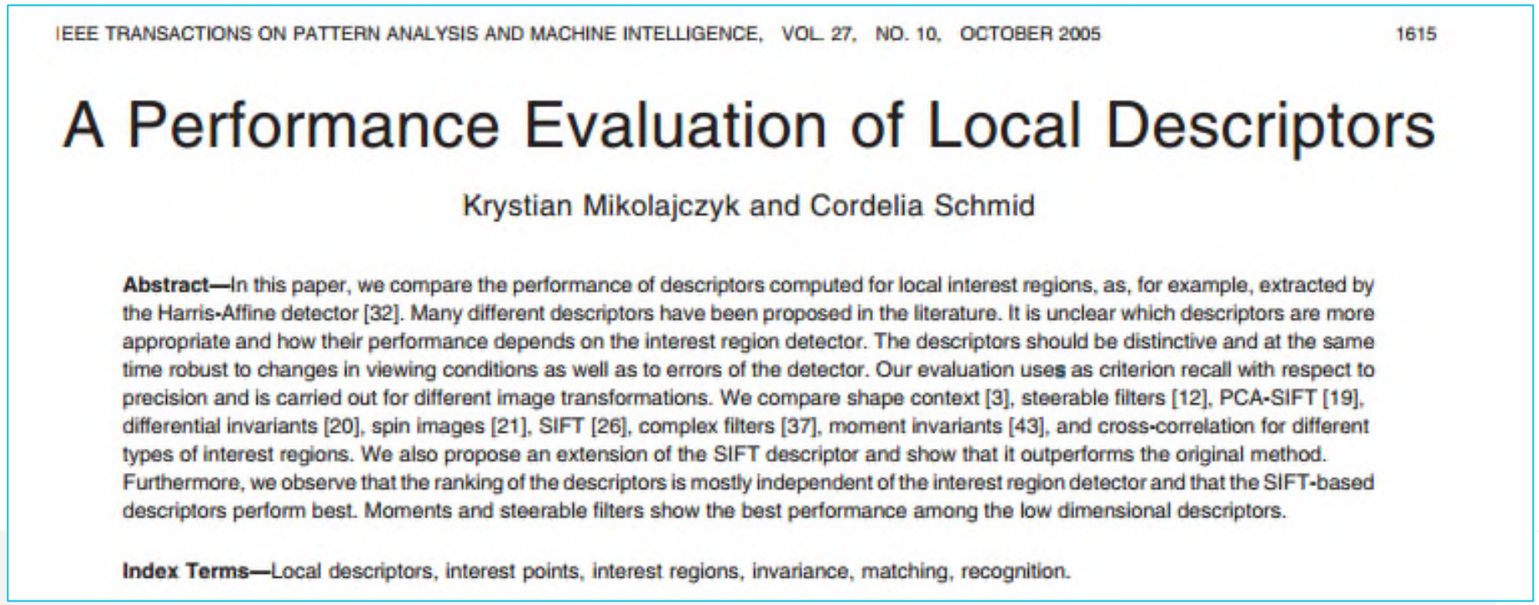
by

Krystian Mikolajczyk and Cordelia Schmid

published in

IEEE Transactions on Pattern Analysis And Machine Intelligence, Vol. 27, No. 10, October 2005

https://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/mikolajczyk_pami2004.pdf



A Performance Evaluation of Local Descriptors

Krystian Mikolajczyk and Cordelia Schmid

Abstract—In this paper, we compare the performance of descriptors computed for local interest regions, as, for example, extracted by the Harris-Affine detector [32]. Many different descriptors have been proposed in the literature. It is unclear which descriptors are more appropriate and how their performance depends on the interest region detector. The descriptors should be distinctive and at the same time robust to changes in viewing conditions as well as to errors of the detector. Our evaluation uses as criterion recall with respect to precision and is carried out for different image transformations. We compare shape context [3], steerable filters [12], PCA-SIFT [19], differential invariants [20], spin images [21], SIFT [26], complex filters [37], moment invariants [43], and cross-correlation for different types of interest regions. We also propose an extension of the SIFT descriptor and show that it outperforms the original method. Furthermore, we observe that the ranking of the descriptors is mostly independent of the interest region detector and that the SIFT-based descriptors perform best. Moments and steerable filters show the best performance among the low dimensional descriptors.

Index Terms—Local descriptors, interest points, interest regions, invariance, matching, recognition.

Acknowledgements

Various contents in this presentation have been taken from different books, lecture notes and the web. These solely belong to their owners, and are here used only for clarifying various educational concepts.
Any copyright infringement is *not* intended.