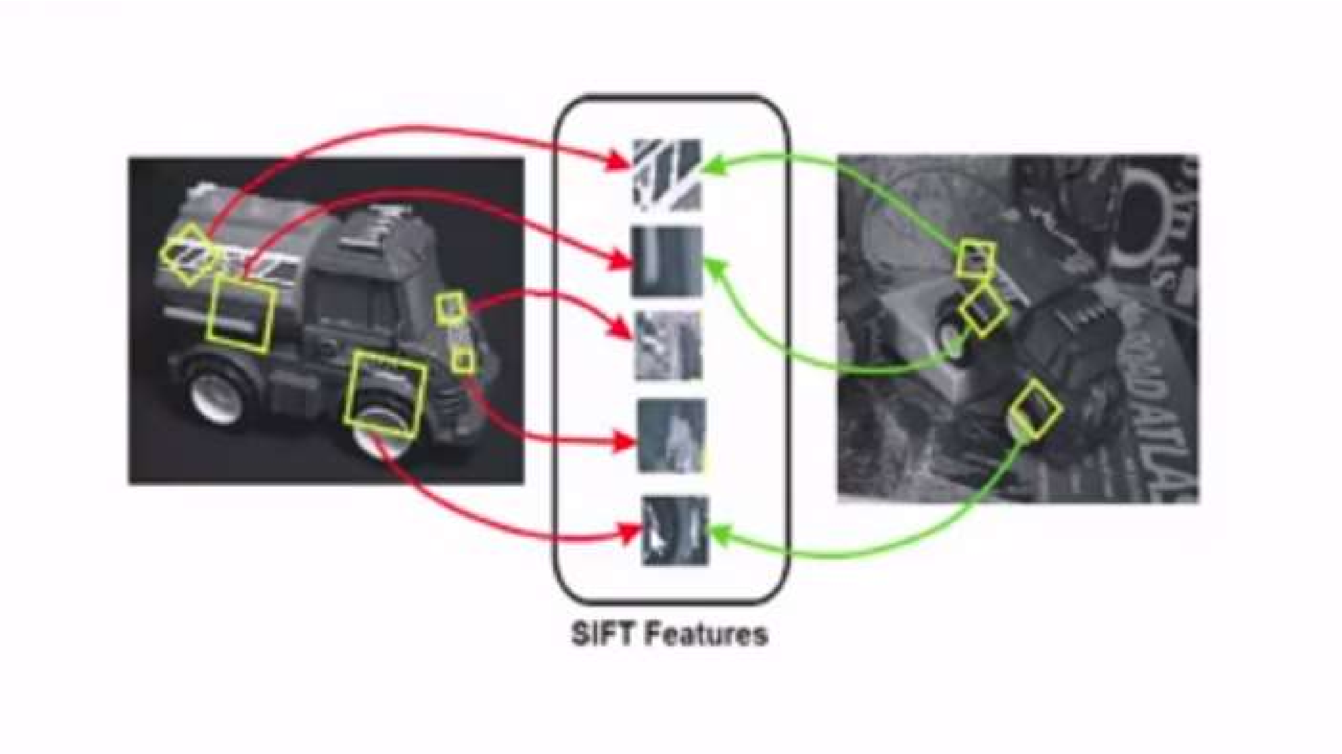
Introduction to SIFT(Scale Invariant Feature Transform)

Deepanshu Tyagi
Mar 16, 2019 · 7 min read

SIFT stands for Scale-Invariant Feature Transform and was first presented in 2004, by **D.Lowe**, University of British Columbia. SIFT is invariance to image scale and rotation. This algorithm is patented, so this algorithm is included in the Non-free module in OpenCV.



Major advantages of SIFT are

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects

- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to a wide range of different feature types, with each adding robustness

This is part of a 7-series Feature Detection and Matching. Other articles included

- Introduction To Feature Detection And Matching
- Introduction to Harris Corner Detector
- Introduction to SURF (Speeded-Up Robust Features)
- Introduction to FAST (Features from Accelerated Segment Test)
- Introduction to BRIEF (Binary Robust Independent Elementary Features)
- Introduction to ORB (Oriented FAST and Rotated BRIEF)

The algorithm

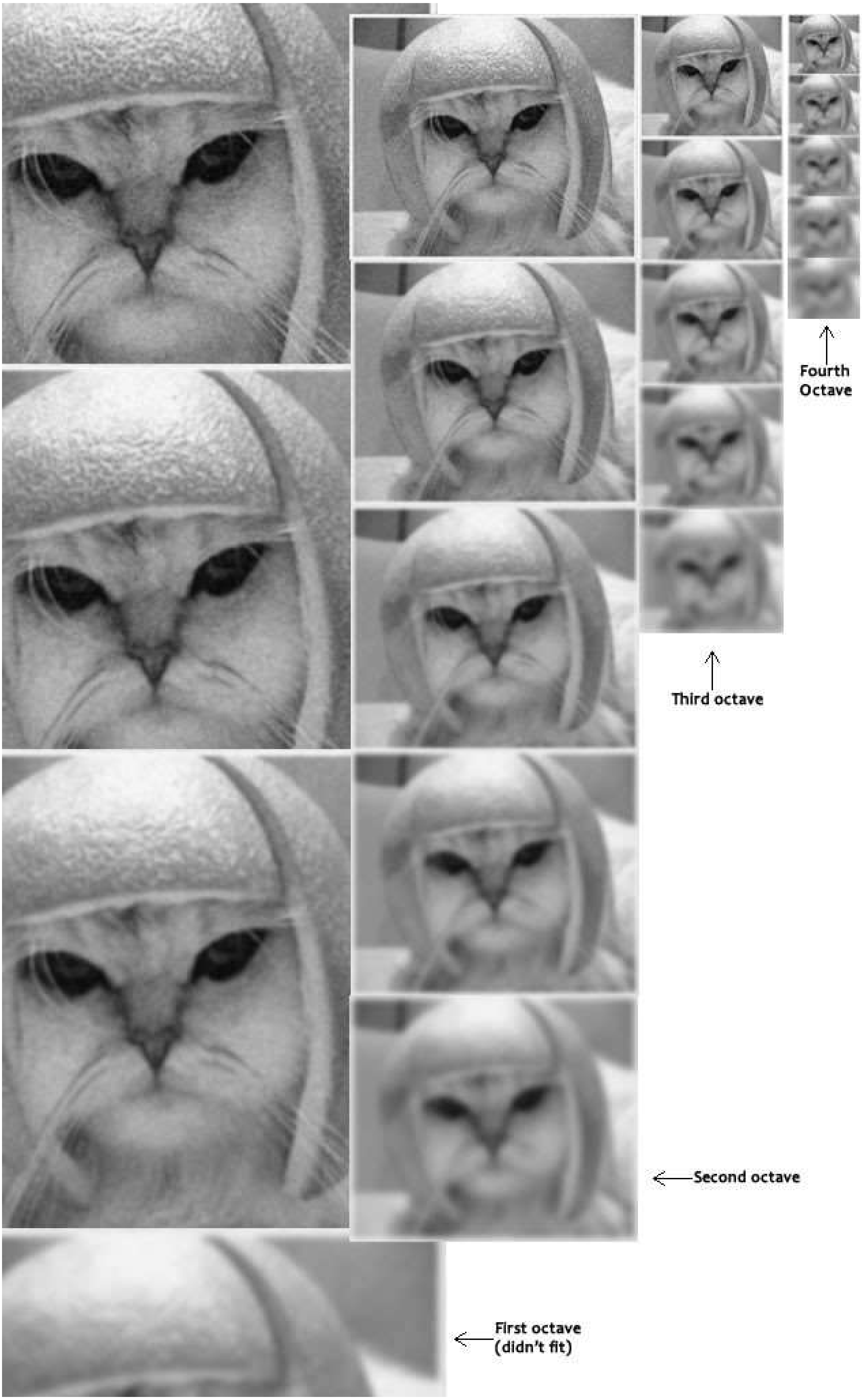
SIFT is quite an involved algorithm. There are mainly four steps involved in the SIFT algorithm. We will see them one-by-one.

- **Scale-space peak selection:** Potential location for finding features.
- **Keypoint Localization:** Accurately locating the feature keypoints.
- **Orientation Assignment:** Assigning orientation to keypoints.
- **Keypoint descriptor:** Describing the keypoints as a high dimensional vector.
- **Keypoint Matching**

Scale-space peak Selection

Scale-space

Real world objects are meaningful only at a certain scale. You might see a sugar cube perfectly on a table. But if looking at the entire milky way, then it simply does not exist. This multi-scale nature of objects is quite common in nature. And a scale space attempts to replicate this concept on digital images.


Fourth Octave
Third octave
Second octave
First octave (didn't fit)

The scale space of an image is a function L(x,y,σ) that is produced from the convolution of a Gaussian kernel(Blurring) at different scales with the input image. Scale-space is separated into octaves and the number of octaves and scale depends on the size of the original image. So we generate several octaves of the original image. Each octave's image size is half the previous one.

Blurring

Within an octave, images are progressively blurred using the Gaussian Blur operator. Mathematically, "blurring" is referred to as the convolution of the Gaussian operator and the image. Gaussian blur has a particular expression or "operator" that is applied to each pixel. What results is the blurred image.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Blurred image

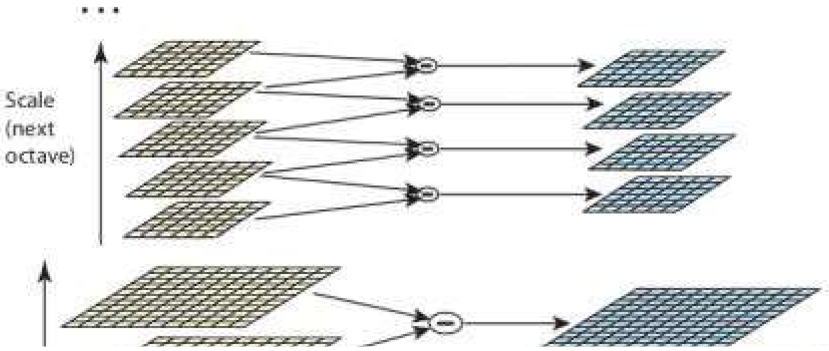
G is the Gaussian Blur operator and I is an image. While x,y are the location coordinates and σ is the "scale" parameter. Think of it as the amount of blur. Greater the value, greater the blur.

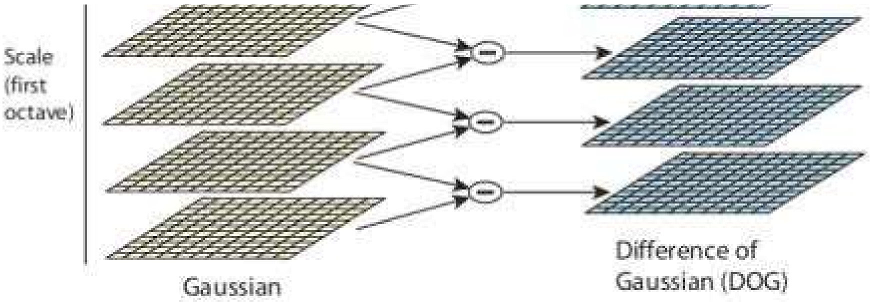
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Gaussian Blur operator

DOG(Difference of Gaussian kernel)

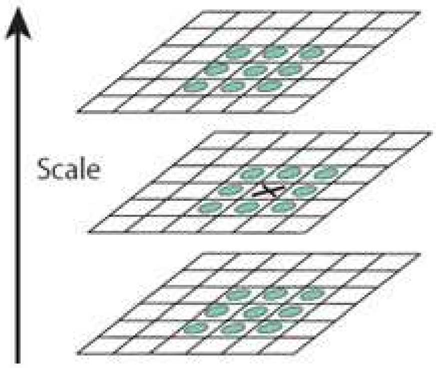
Now we use those blurred images to generate another set of images, the Difference of Gaussians (DoG). These DoG images are great for finding out interesting keypoints in the image. The difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and *kσ*. This process is done for different octaves of the image in the Gaussian Pyramid. It is represented in below image:





Finding keypoints

Up till now, we have generated a scale space and used the scale space to calculate the Difference of Gaussians. Those are then used to calculate Laplacian of Gaussian approximations that are scale invariant.

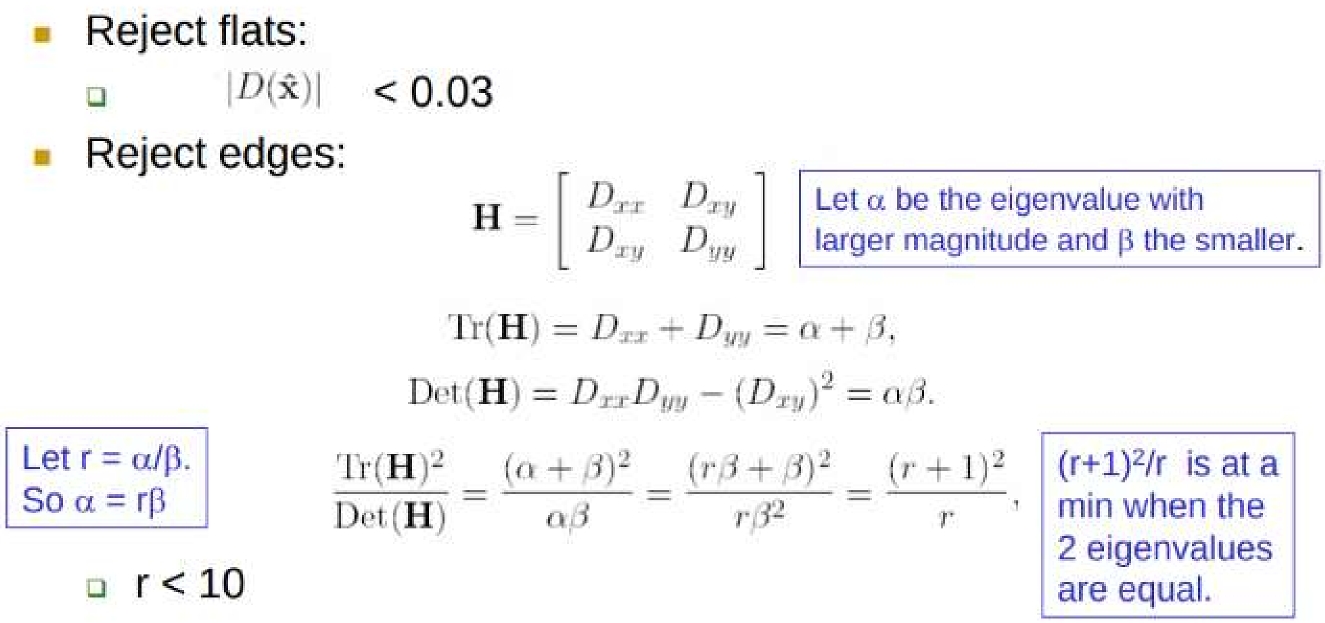


One pixel in an image is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales. This way, a total of 26 checks are made. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale.

Keypoint Localization

Key0points generated in the previous step produce a lot of keypoints. Some of them lie along an edge, or they don't have enough contrast. In both cases, they are not as useful as features. So we get rid of them. The approach is similar to the one used in the Harris Corner Detector for removing edge features. For low contrast features, we simply check their intensities.

They used Taylor series expansion of scale space to get a more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. DoG has a higher response for edges, so edges also need to be removed. They used a 2x2 Hessian matrix (H) to compute the principal curvature.



- Reject flats:
  - □ $|D(\hat{\mathbf{x}})|$ < 0.03
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$
$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let r = α/β.
So α = rβ

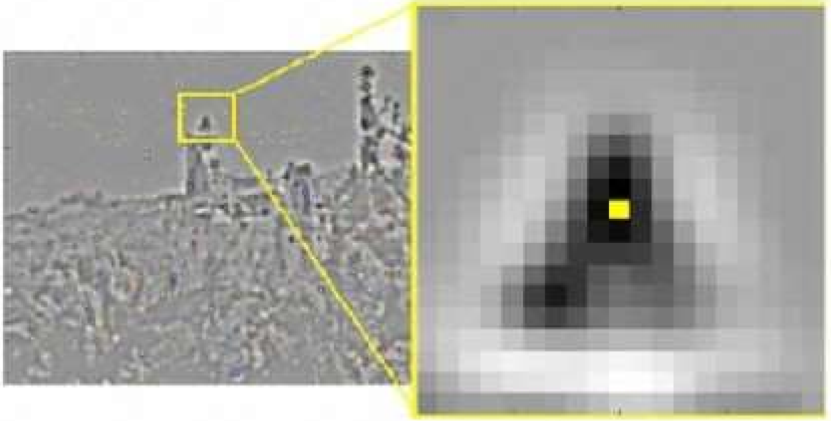
$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

(r+1)²/r is at a min when the 2 eigenvalues are equal.

- □ r < 10

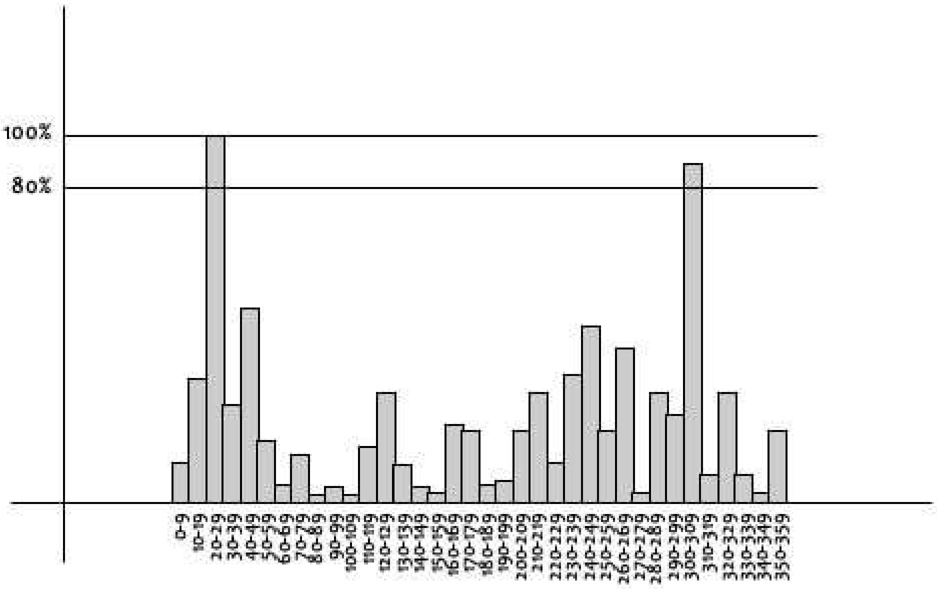
Orientation Assignment

Now we have legitimate keypoints. They've been tested to be stable. We already know the scale at which the keypoint was detected (it's the same as the scale of the blurred image). So we have scale invariance. The next thing is to assign an orientation to each keypoint to make it rotation invariance.



A keypoint

A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. Let's say the gradient direction at a certain point (in the "orientation collection region") 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 the gradient at that point. Once you've done this for all pixels around the keypoint, the histogram will have a peak at some point.

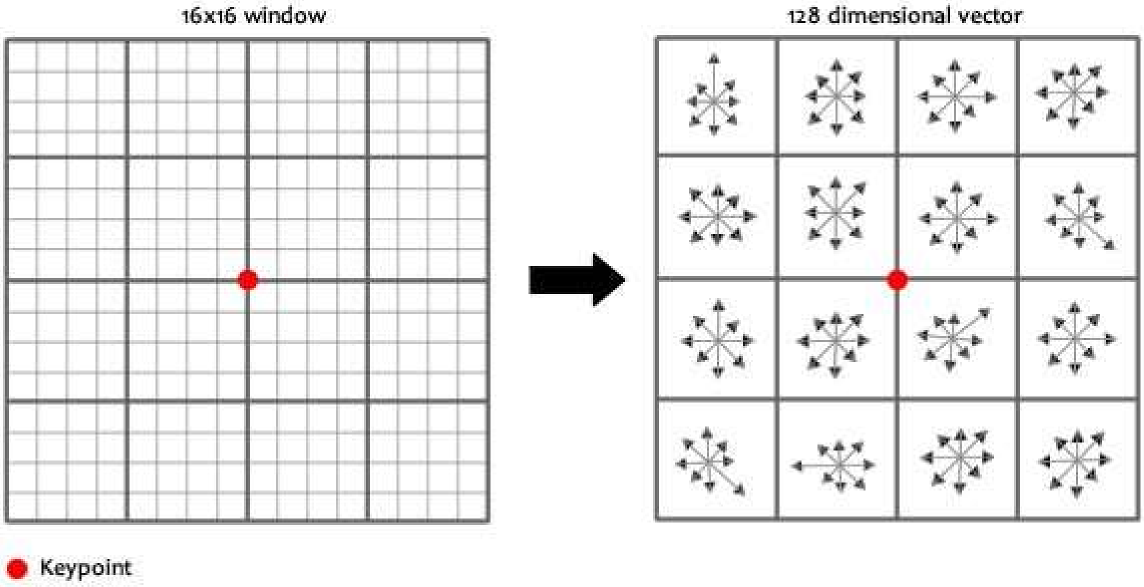


The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contributes to the stability of matching.

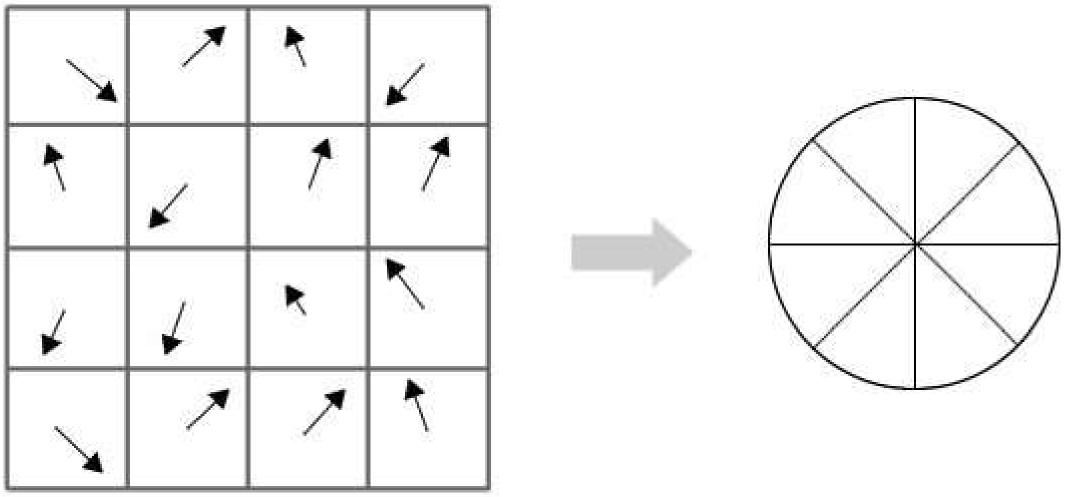
Keypoint descriptor

At this point, each keypoint has a location, scale, orientation. Next is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in viewpoint and illumination.

To do this, 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. It is represented as a feature vector to form keypoint descriptor. This feature vector introduces a few complications. We need to get rid of them before finalizing the fingerprint.

1. **Rotation dependence** The feature vector uses gradient orientations. Clearly, if you rotate the image, everything changes. All gradient orientations also change. To achieve rotation independence, the keypoint's rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint's orientation.

2. **Illumination dependence** If we threshold numbers that are big, we can achieve illumination independence. So, any number (of the 128) greater than 0.2 is changed to 0.2. This resultant feature vector is normalized again. And now you have an illumination independent feature vector!

Keypoint Matching

Keypoints between two images are matched by identifying their nearest neighbors. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, the ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper.

Implementation

I was able to implement sift using OpenCV(3.4). Here's how I did it:

SIFT (Scale-Invariant Feature Transform)

Import resources and display image

In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Load the image
image1 = cv2.imread('./images/face1.jpeg')

# Convert the training image to RGB
training_image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

# Convert the training image to gray scale
training_gray = cv2.cvtColor(training_image, cv2.COLOR_RGB2GRAY)

# Create test image by adding Scale Invariance and Rotational Inva
riance
test_image = cv2.pyrDown(training_image)
test_image = cv2.pyrDown(test_image)
num_rows, num_cols = test_image.shape[:2]
```

sift.ipynb hosted with ♡ by GitHub view raw

Github link for the code: https://github.com/deepanshut041/feature-detection/tree/master/sift

References

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- https://in.udacity.com/course/computer-vision-nanodegree--nd891
- http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

Thanks for reading! If you enjoyed it, hit that clap button below and follow Data Breach for more updates