L7 (Scale Space Representation)

So, Harris corner detector is shift-rotation invariant, but it fails when the scale changes.
Scale invariance is a desirable property for a feature.
Scale space representation allows to systematically blur the image using a **Gaussian filter** with different standard deviations (σ), producing a **family of images** ranging from fine details to coarse structure (large σ).

$$s[x, y, \sigma] = h_G[x, y, \sigma] * s[x, y]$$

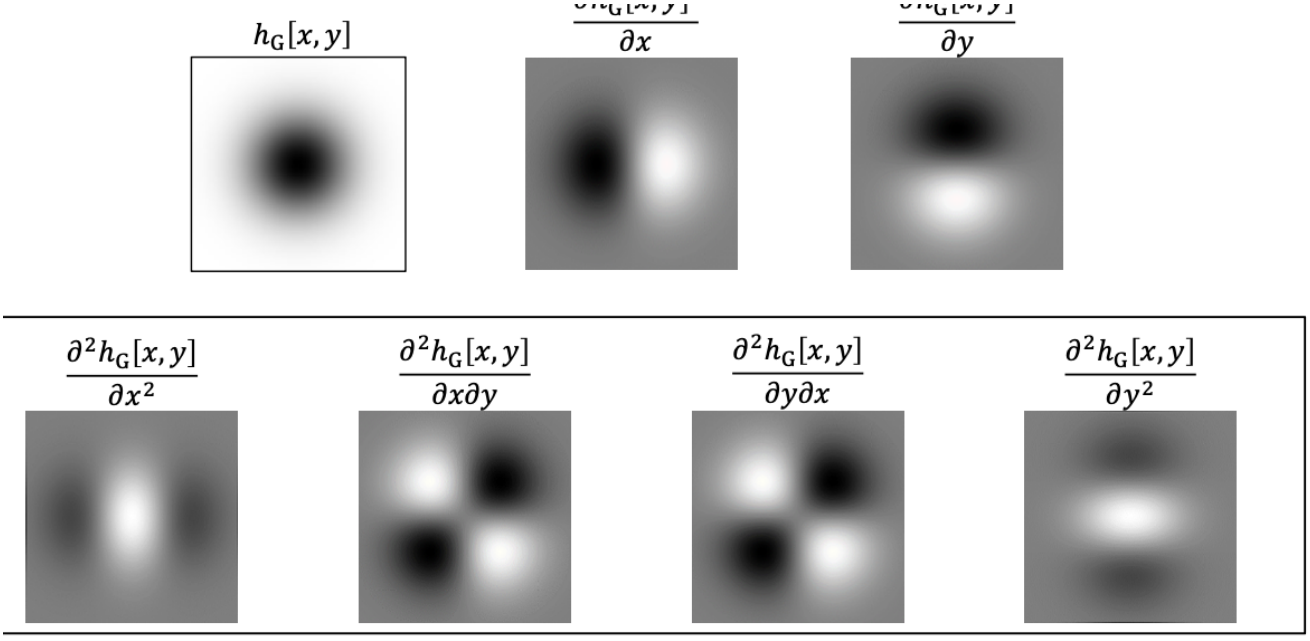
where

$$h_G[x, y, \sigma] = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2 + y^2}{2\sigma^2})$$

Gaussian Derivatives

Gaussian filter is always positive and it doesn't highlight any distinctive features like edges or corners. Since the Gaussian is symmetric and smooths, applying it alone does not help detect keypoints (corners or blobs).

We need to use Gaussian derivatives to detect meaningful features:



Hessian of Gaussian

Let's instead use Hessian of Gaussian. This is Hessian for reference:

$$H = \begin{bmatrix} \frac{\delta^2 s[x,y]}{\delta x^2} & \frac{\delta^2 s[x,y]}{\delta x \delta y} \\ \frac{\delta^2 s[x,y]}{\delta x \delta y} & \frac{\delta^2 s[x,y]}{\delta y^2} \end{bmatrix}$$

Laplacian of Gaussian

Laplacian of Gaussian(LoG) is the trace of Hessian of Gaussian:

$$\text{LoG} = \frac{\delta^2 h_G[x,y]}{\delta x^2} + \frac{\delta^2 h_G[x,y]}{\delta y^2}$$

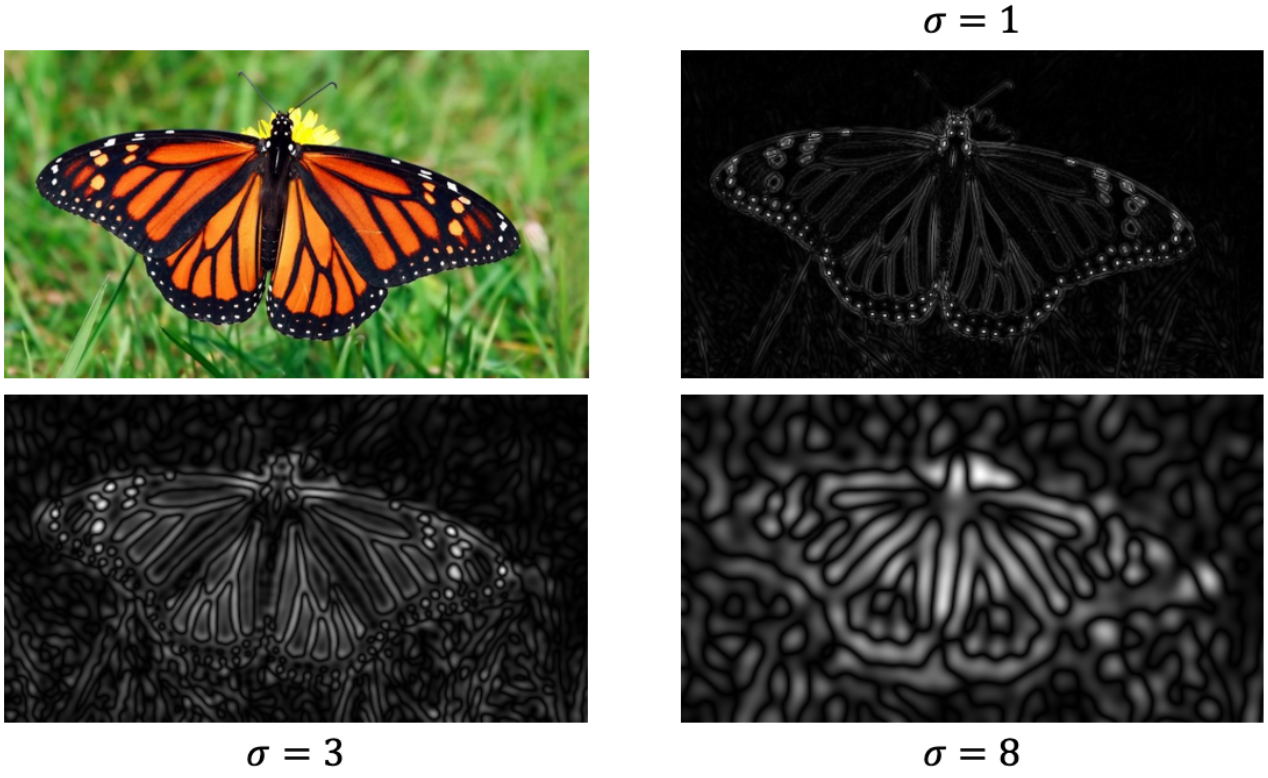
- zero-crossings indicate edges
- LoG responds strongly to circular and blob-like structures
- maximum response if blob "fits" **completely** under LoG window

Scale-normalized LoG

LoG response is **not scale-invariant**. Gaussian blur smooths contours, lowering gradient. We have to multiple by σ^2 to compensate for gradient decrease after Gaussian blur:

$$\sigma^2 \nabla^2 h_G[x,y,\sigma]$$

For small scales, LoG magnitude highlights small details. For large scales, LoG magnitude highlights large details:



Difference of Gaussian

DoG is an approximation to LoG and it's computationally cheaper and faster:

$$\text{DoG} = h_G[x, y, k\sigma] - h_G[x, y, \sigma] \approx (k-1)\sigma^2 \nabla^2 h_G[x, y, \sigma]$$

DoG already has scale normalization by σ^2

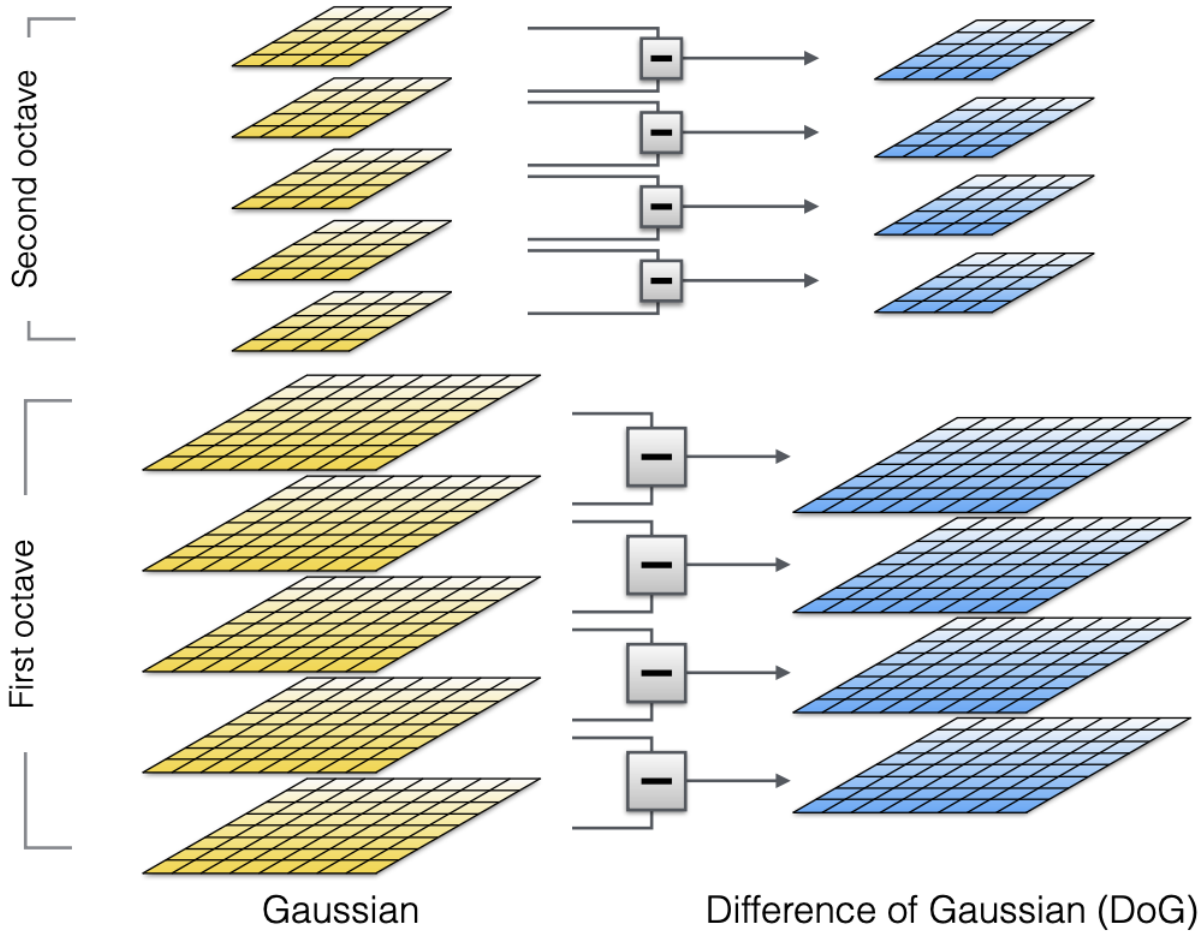
SIFT (Scale Invariant Feature Transform)

- invariant to scale and rotation
- good for affine distortions, noise, illumination changes
- good for matching, recognition

Consists of **feature detector** and **feature descriptor** that are independent.
Stages:

- scale space extrema detection
- keypoint localization
- orientation assignment
- keypoint descriptor

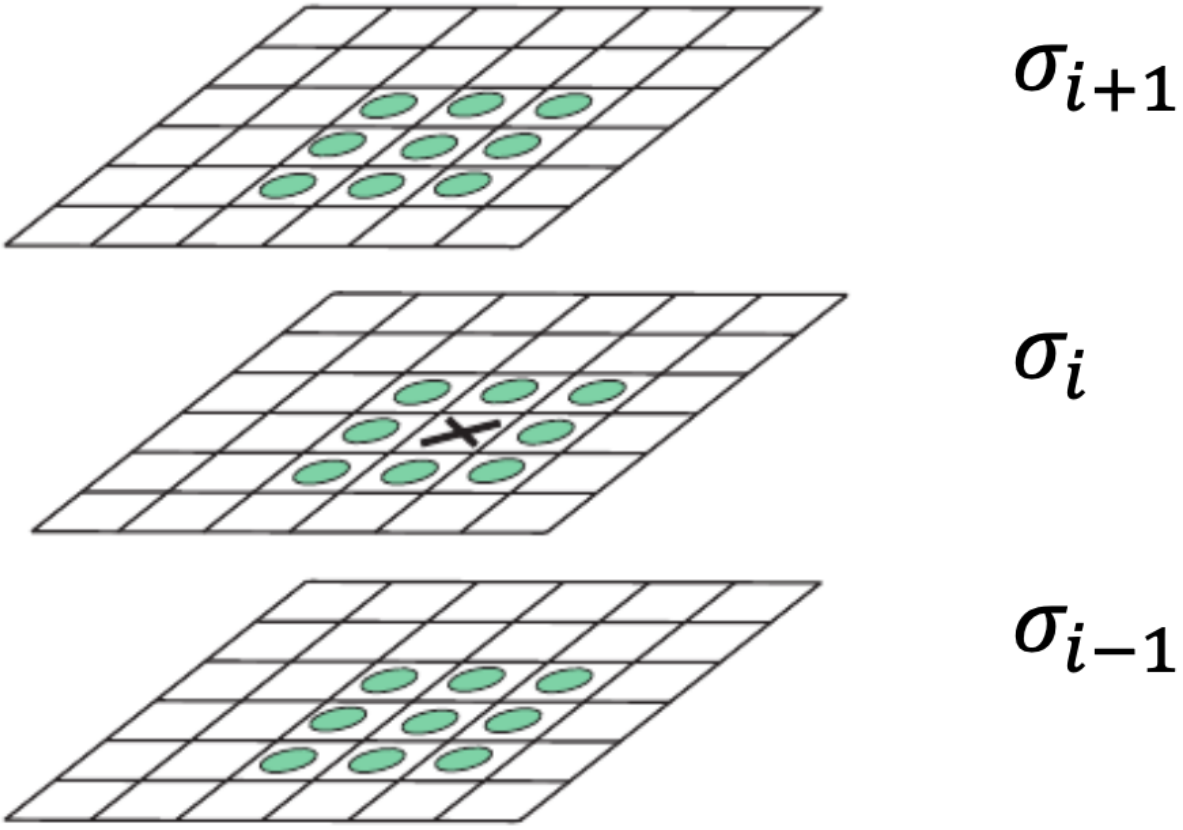
Scale-space extrema detection



DoG is calculated by subtracting two consecutive Gaussian images where $k = \sqrt{(2)}$ is the scaling factor.

Each neighbor in the DoG image is compared to its **26 neighbors**:

- 8 neighbors in the same scale
- 9 neighbors in the scale above
- 9 neighbors in the scale below.
  A pixel is considered **a keypoint** if it is a local maximum or minimum across these 26 locations:



Keypoint Localization

The goal here is to refine the detected keypoints by filtering out **low-contrast** points and unstable edge responses.

Refinement by Taylor Expansion

Given a detected extrema at (x, y, σ), the DoG function is approximated using second-order Taylor expansion:

$$D(x) = D + \frac{\delta D^T}{\delta x}x + \frac{1}{2}x^T\frac{\delta^2 D}{\delta x^2}x$$

The extremum location is refined by solving:

$$\hat{x} = -(\frac{\delta^2 D}{\delta x^2})^{-1}\frac{\delta D}{\delta x}$$

This step adjust the keypoint position to subpixel accuracy.

Removing low-contrast keypoints

If the intensity of a keypoint is too small, we ignore it by thresholding:

$$D(\hat{x}) < 0.03$$

Eliminating edge responses

DoG also responds to edges, not only keypoints. Keypoints located along edges tend to be **unstable** because small changes in the image can shift their position significantly. To filter out, use Hessian matrix:

$$H = \begin{bmatrix} \frac{\delta^2 s[x,y]}{\delta x^2} & \frac{\delta^2 s[x,y]}{\delta x \delta y} \\ \frac{\delta^2 s[x,y]}{\delta x \delta y} & \frac{\delta^2 s[x,y]}{\delta y^2} \end{bmatrix}$$

The ratio of eigenvalues of H is used for elimination:

$$r = \frac{\lambda_1}{\lambda_2}$$

$r \neq 1$ for edges and $r \sim 1$ for keypoints. But eigenvalues are **expensive** to compute. Instead use trace and determinant:

$$\text{ratio} = \frac{Tr(H)}{det(H)} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1\lambda_2} < \frac{(r+1)^2}{r}$$

If this inequality holds, we retain the **keypoints**; otherwise, they are discarded. Edges have large eigenvalue ratio, while corners have balanced eigenvalues.

Orientation Assignment

This returns a tuple $[x, y, \sigma, \theta]$ for a keypoint with θ for orientation. Orientation assignment is done via voting based on proportionality on gradient magnitude, distance from keypoint

Keypoint Descriptor

- compute relative orientation and magnitude in 16×16 neighborhood around keypoint
- form weighted histograms(8 bins) for 4×4 regions

SIFT creates a robust keypoint descriptor by building a **128-dimensional vector** from **16 local orientation histograms.** The vector undergoes **two normalizations** with a clipping step in between, making it resistant to changes in **illumination** and **contrast**. This process emphasizes the distribution of gradient orientations, which is more reliable for matching features than raw gradient magnitudes. The final result is the SIFT descriptor.

SURF (Speeded Up Robust Features)

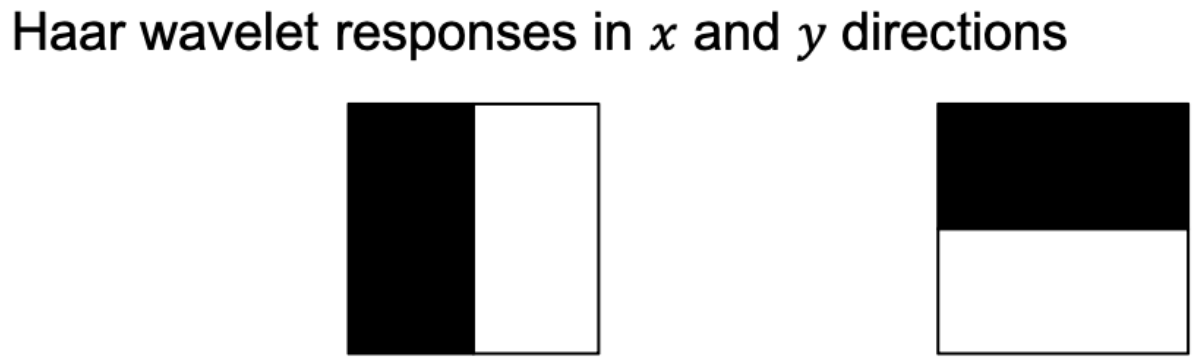
- detector and descriptor setup
- simplify SIFT
  SURF is faster because of the following:
- Box filters and integral images instead of Gaussian filters
- A simpler approximation of the Hessian matrix
- Fewer feature dimensions (64 vs SIFT's 128)

Feature detection:

- SURF uses Hessian matrix determinant approximation with box filters to look for blob-like structures: $det(H_{approx} \approx s_{xx}s_{yy} - (0.9s_{xy})^2$

Feature description:

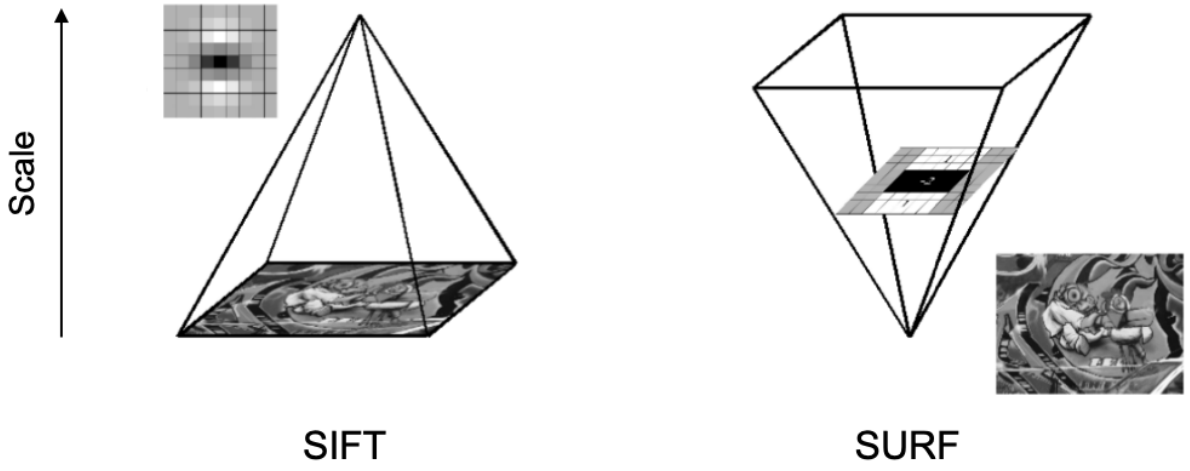
- SUFT; 64-dimensional vector based on Haar wavelet responses



Scale Space:

- SIFT creates a full Gaussian pyramid
- SURF uses box filters of different sizes on the original image.(image size is fixed)

- SURF's approach is faster because it doesn't need to create multiple image scales.



Upright SURF(U-SURF)

In many cases, cameras do not rotate much so it skips orientation assignment to achieve faster compute. It is robust to plus minus 15 degrees of tilt.