CMSC 491/691

Lecture 6

Image Features II

Recap: Harris Corner Detector

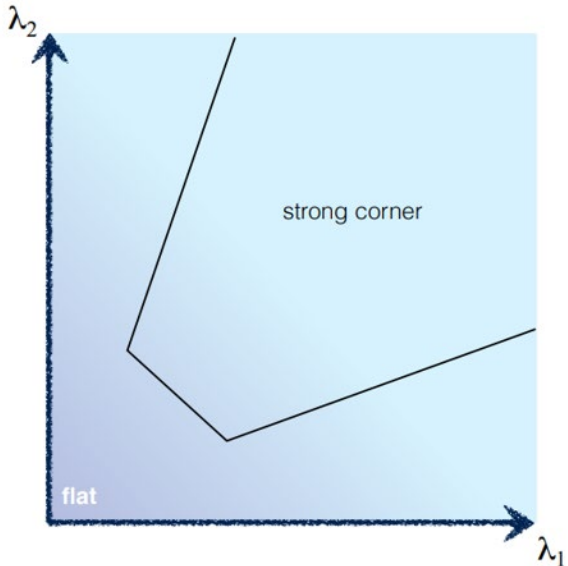
- **Key Idea:** Corners are good. Edges are OK. Flat regions are meh.
- Good feature ➔ high error:
  - But this is slow to compute

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

- Use Taylor Approximation:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

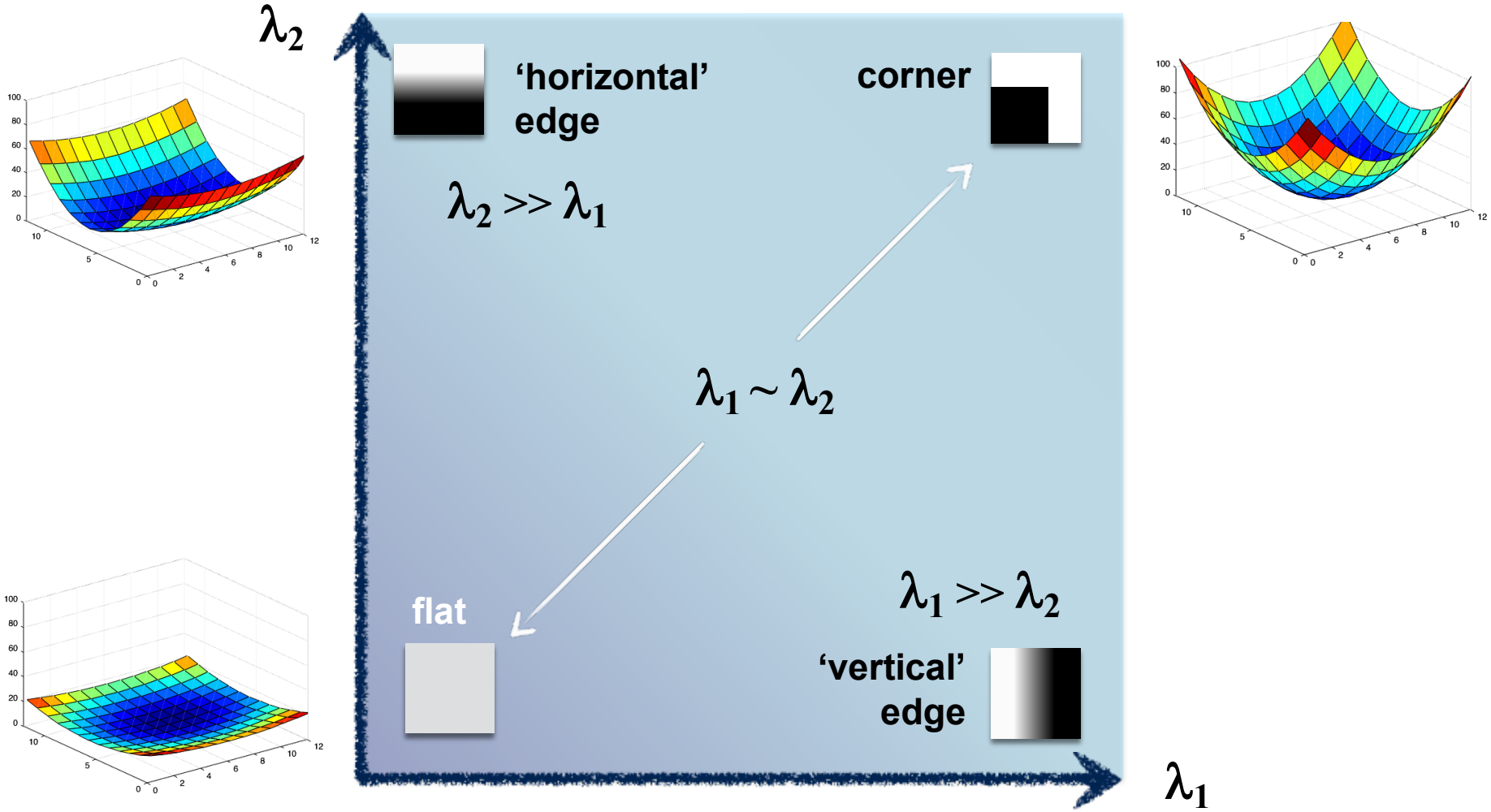
- Do thresholding in Eigenspace to select features



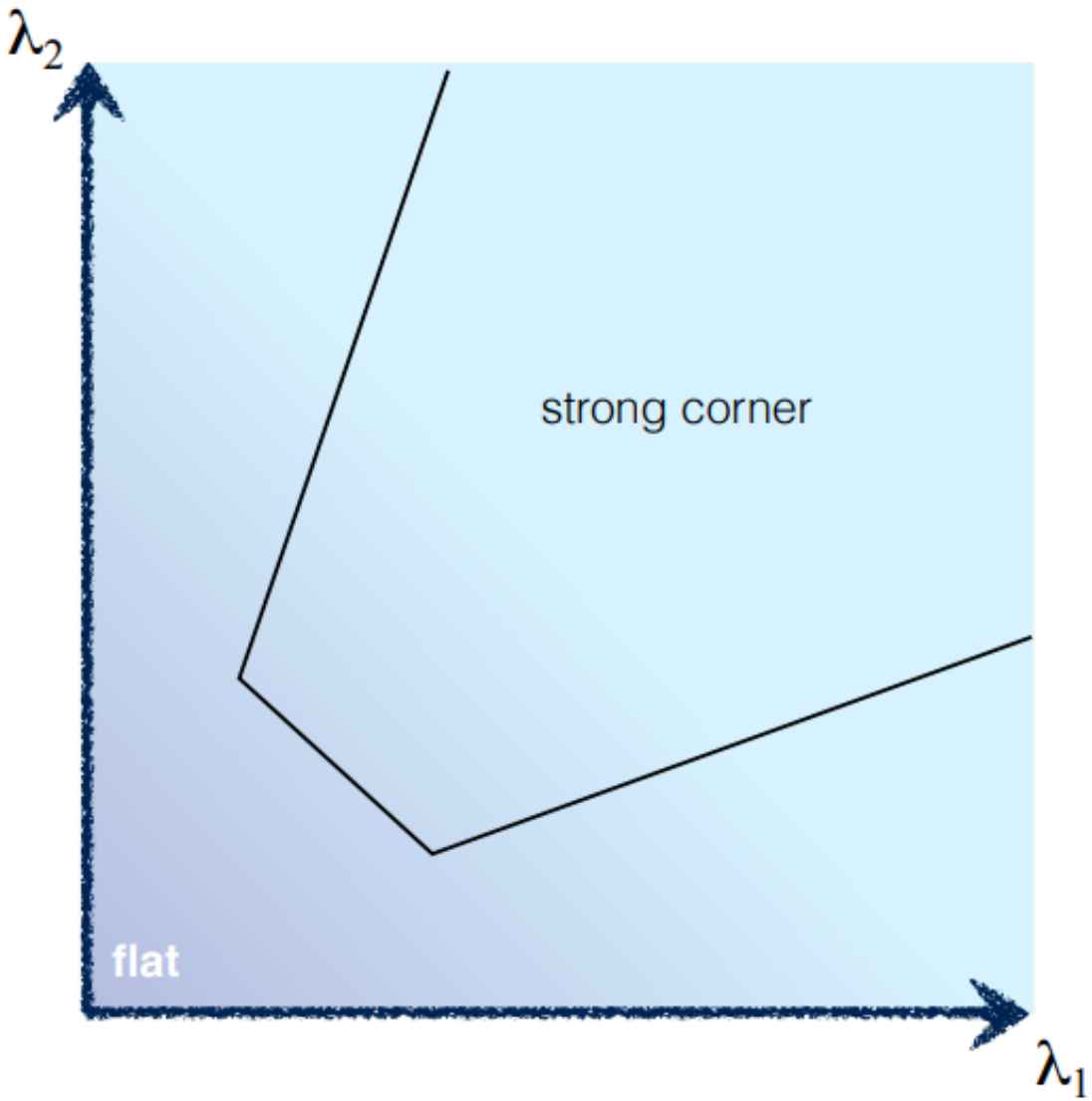
Recall …

Corner Detection in Eigenspace

interpreting eigenvalues



5. Use threshold on eigenvalues to detect corners



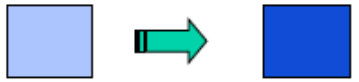
Think of a function to score ‘cornerness’

Properties of Harris Corners

Invariance and covariance

- We want corner locations to be *invariant* to photometric transformations and *covariant* to geometric transformations
  - **Invariance:** image is transformed and corner locations do not change
  - **Covariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations

Affine intensity change

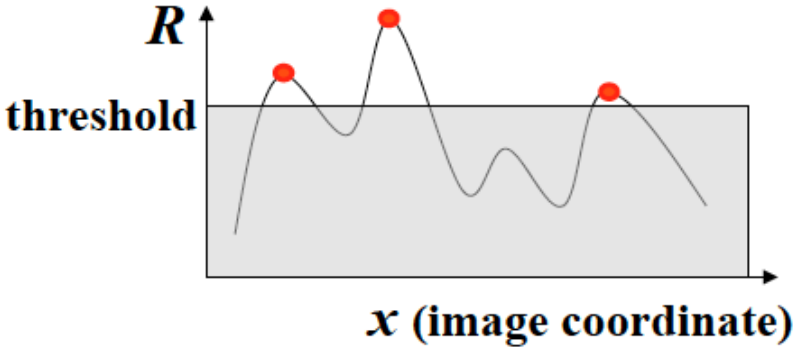


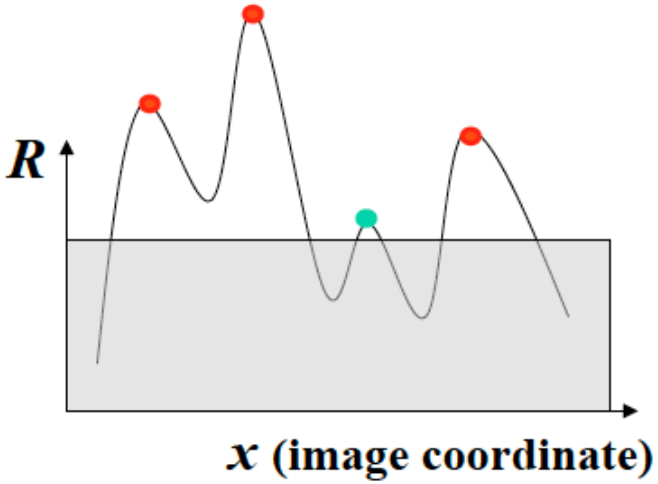
$I \rightarrow a\,I + b$

- **Only derivatives are used =>**

invariance to intensity shift $I \rightarrow I + b$

- **Intensity scaling: $I \rightarrow a\,I$**

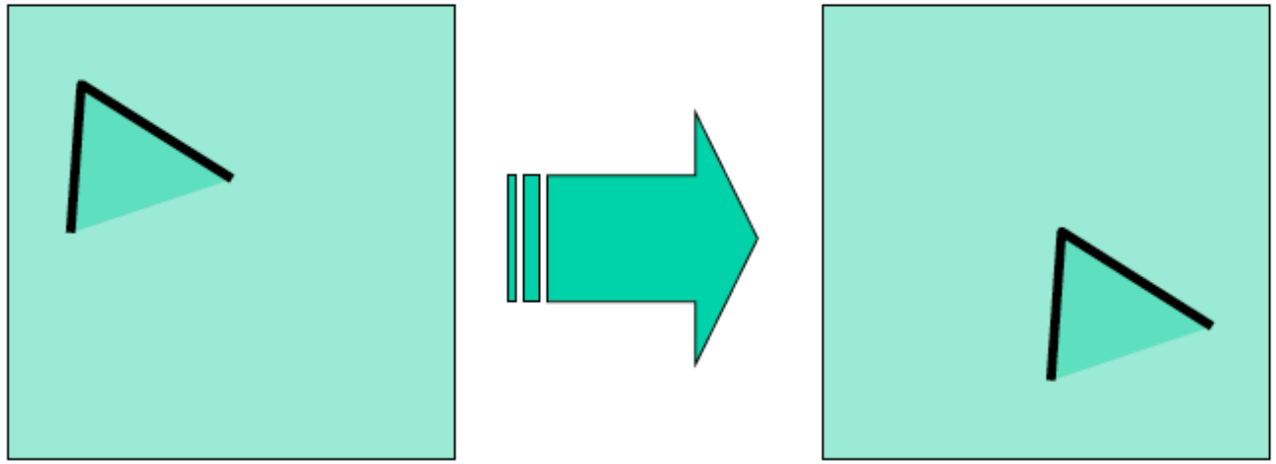




***Partially invariant* to affine intensity change**

Translation/Rotation Covariance

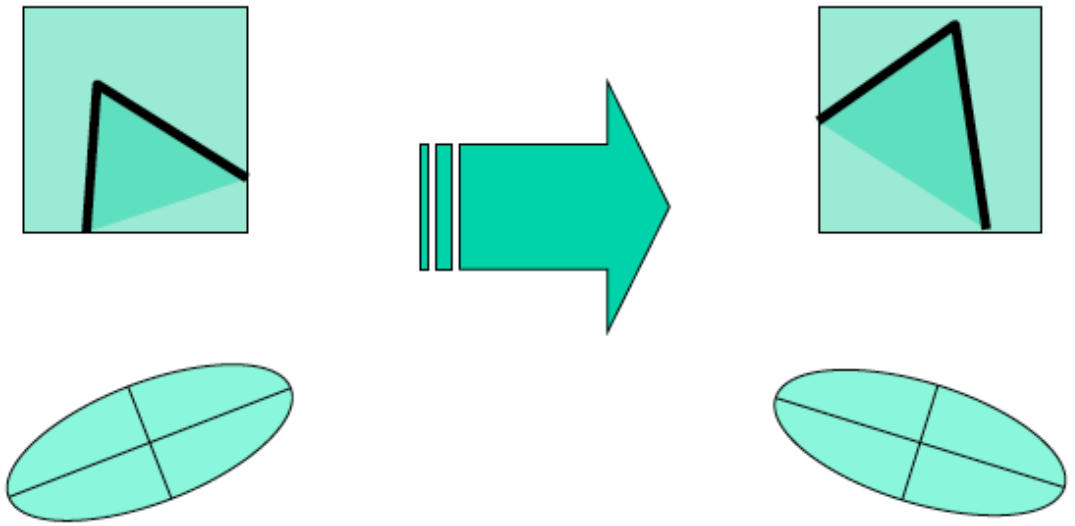
Image translation



- **Derivatives and window function are shift-invariant**

Corner location is covariant w.r.t. translation

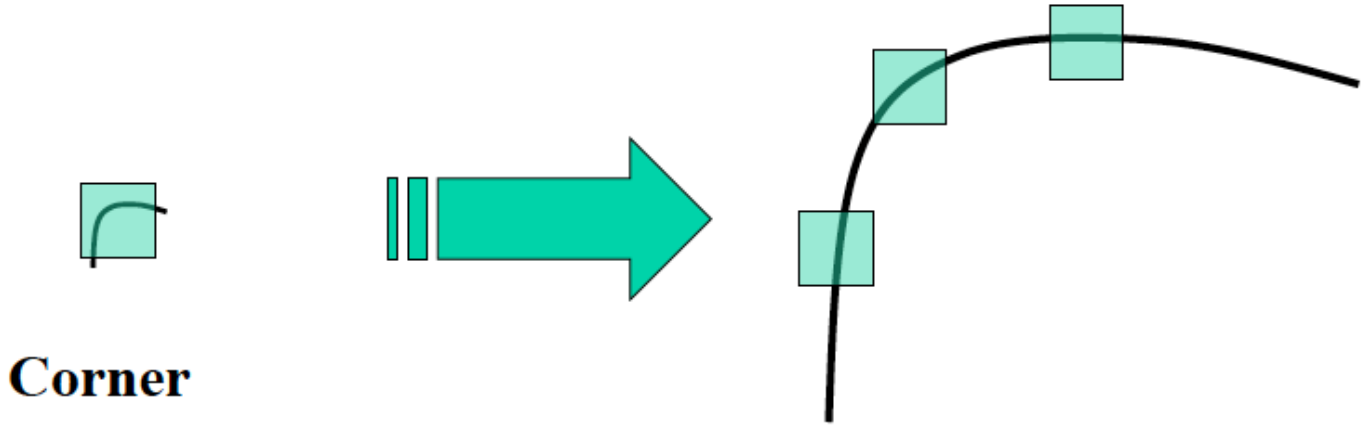
Image rotation



Second moment ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner location is covariant w.r.t. rotation

Scaling



Corner

All points will be classified as edges

Corner location is not covariant to scaling!

Harris Corner Detector

Rotation invariant?

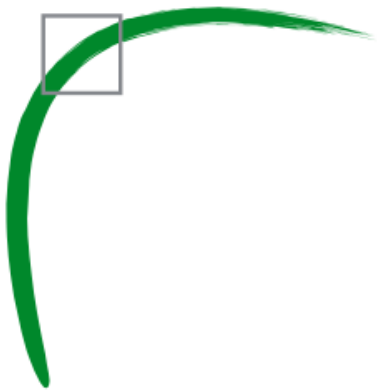
Scale invariant?

Harris Corner Detector

Rotation invariant?

Scale invariant?

edge!



corner!

How can we handle scale?

After feature detection, how can we match features in multiple images (feature description and matching)

How can we handle scale?

After feature detection, how can we match features in multiple images (feature description and matching)

Two Questions

1. How can we make a feature detector ***scale invariant ?***

2. How can we ***automatically select the scale ?***

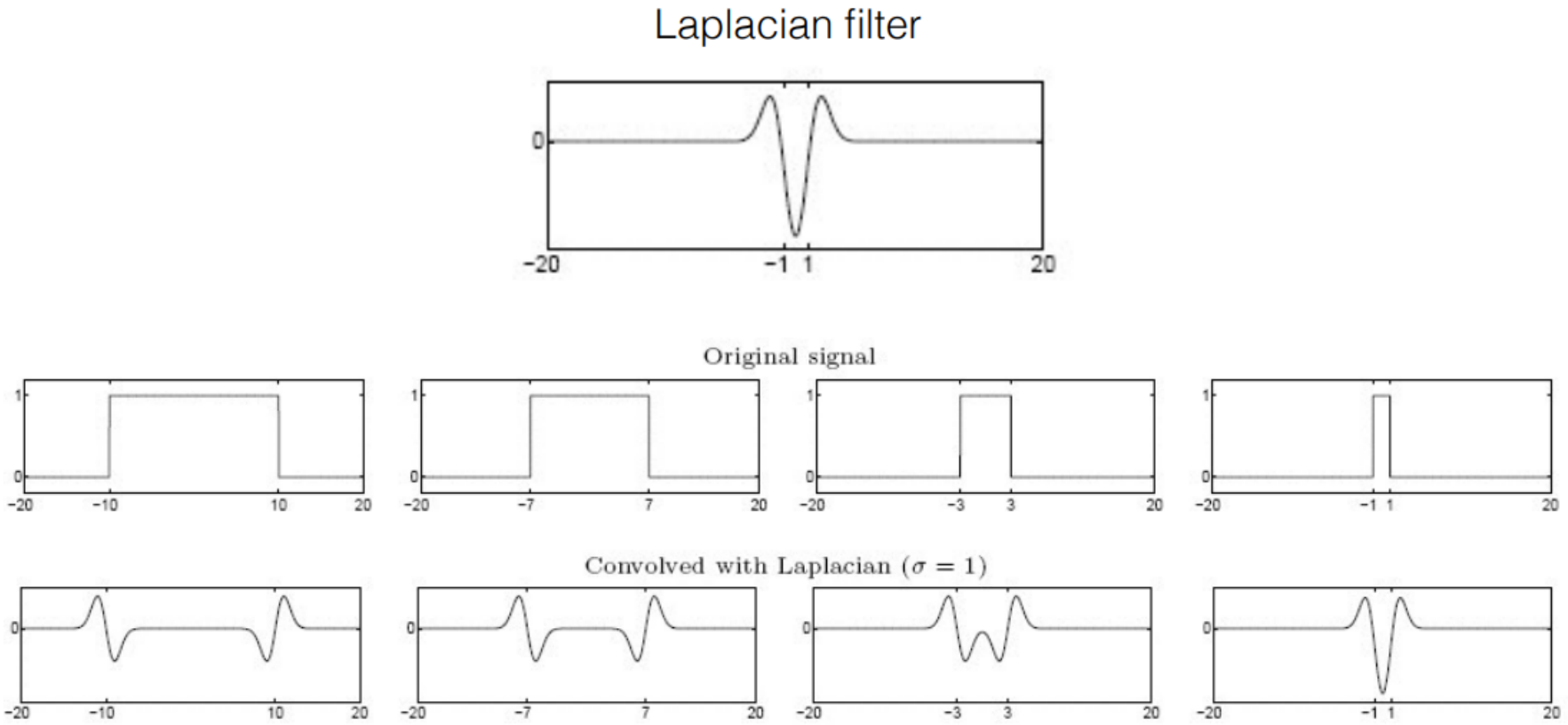
Multi-Scale Methods

1. Multi-Scale Detection

2. Scale-Space Normalization

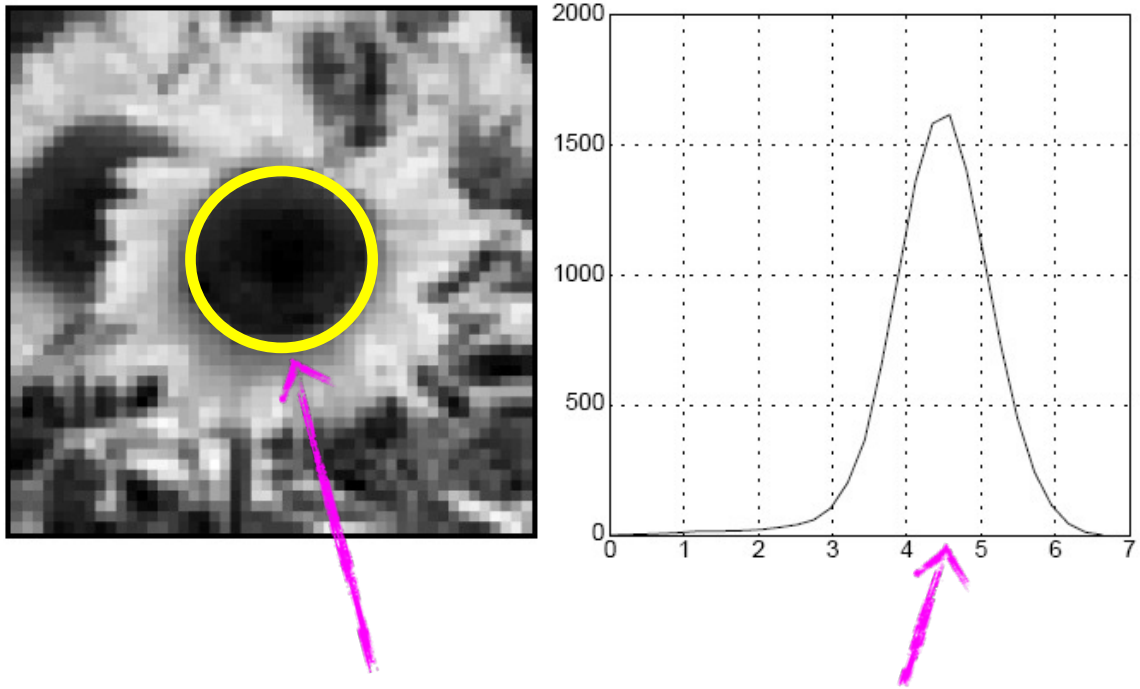
Multi-Scale 2D Blob Detector

Laplacian Filter !!!



Highest response when the signal has the same **characteristic scale** as the filter

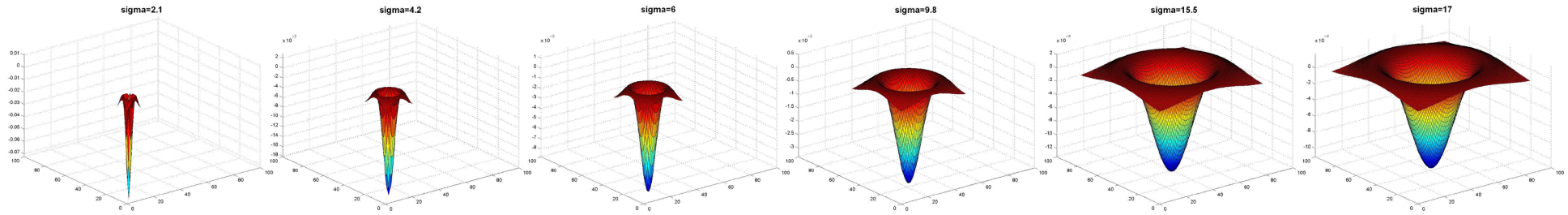
characteristic scale - the scale that produces peak filter response



characteristic scale

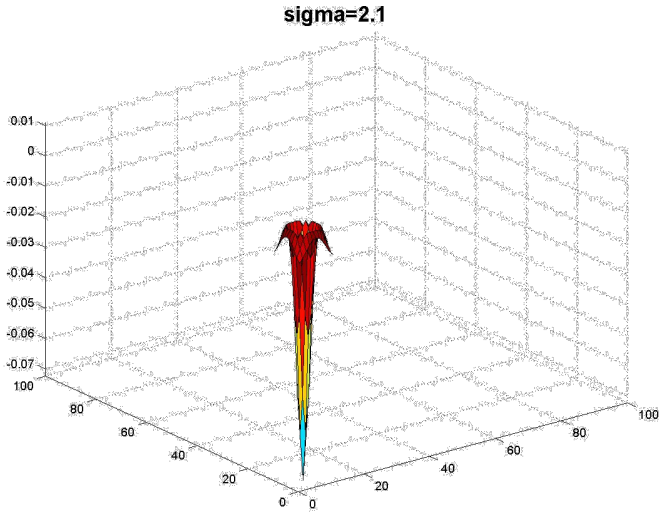
we need to search over characteristic scales

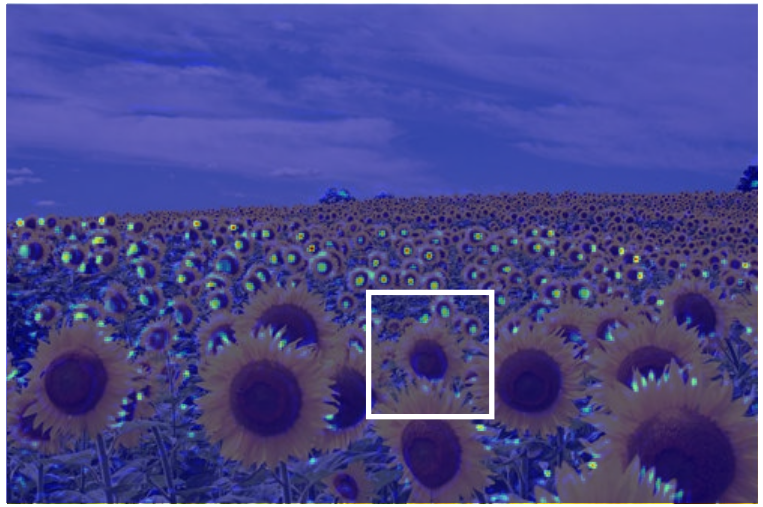
What happens if you apply different Laplacian filters?

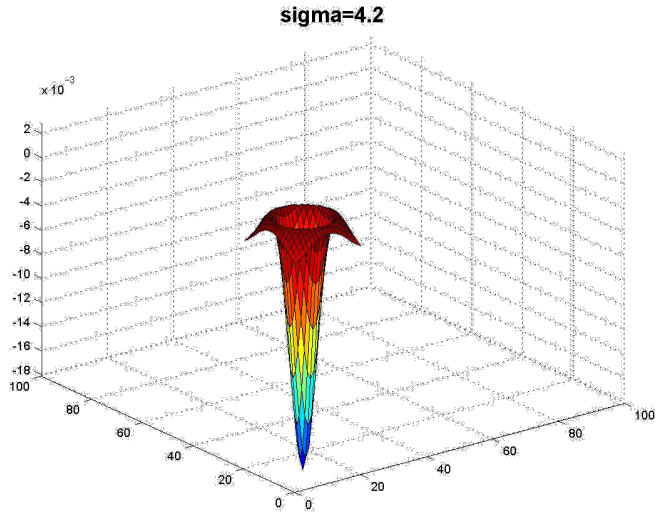


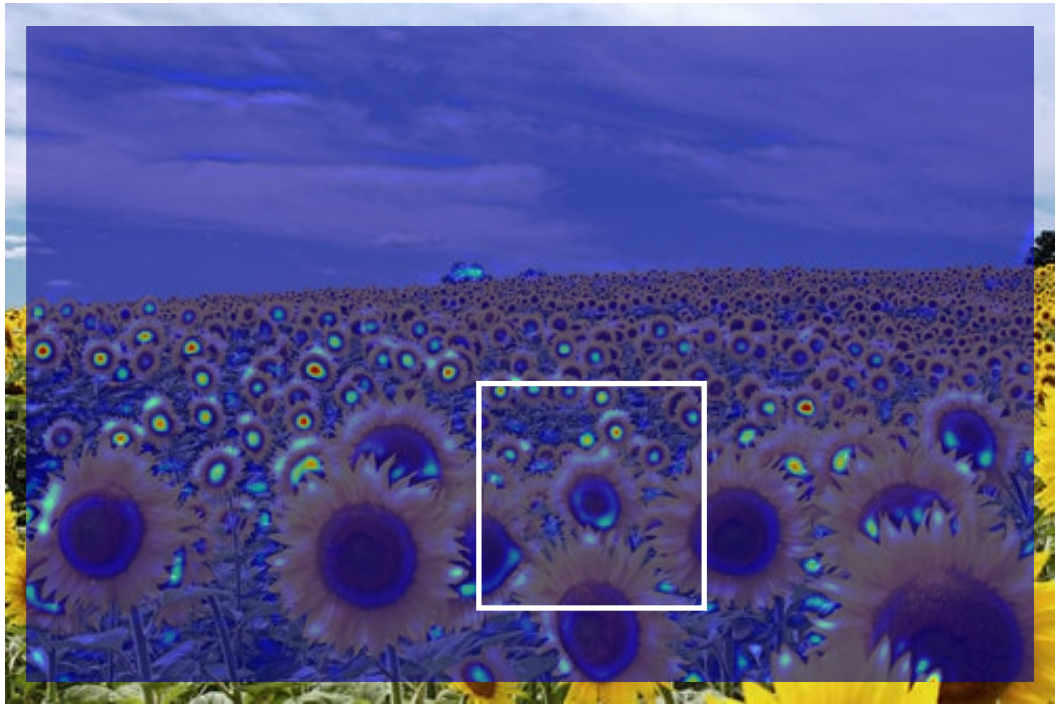
Full size

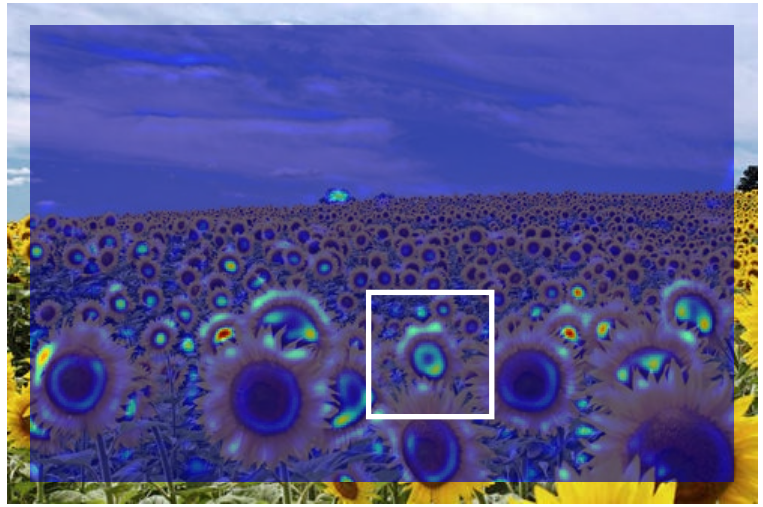
3/4 size

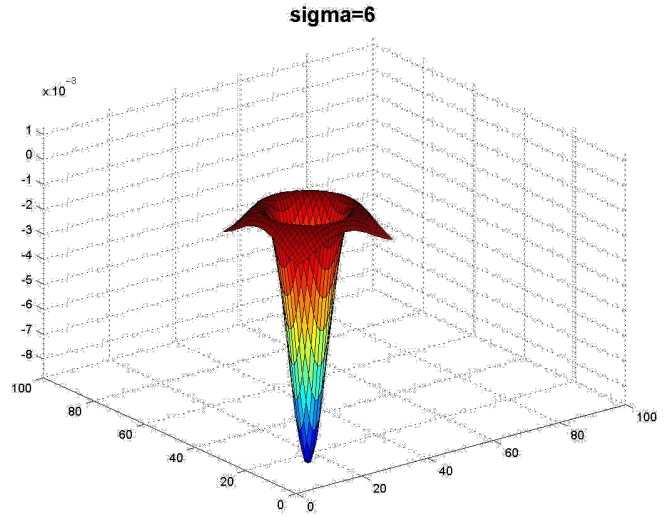

sigma=2.1
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
100
80
60
40
20
0
0
20
40
60
80
100

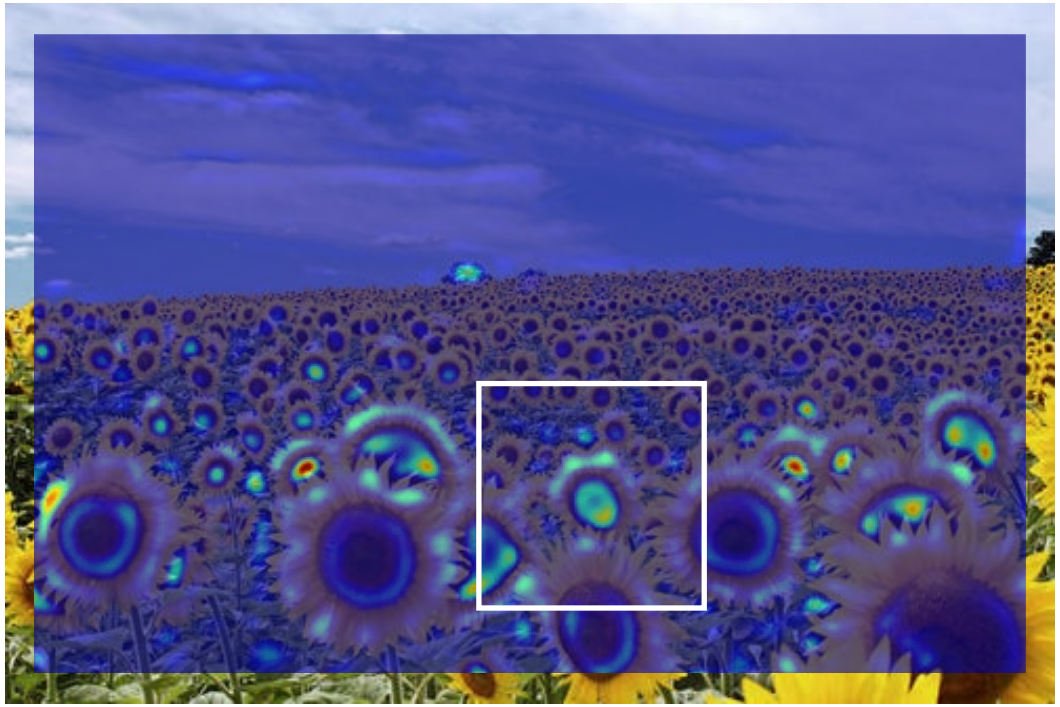



sigma=4.2
x 10^-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100

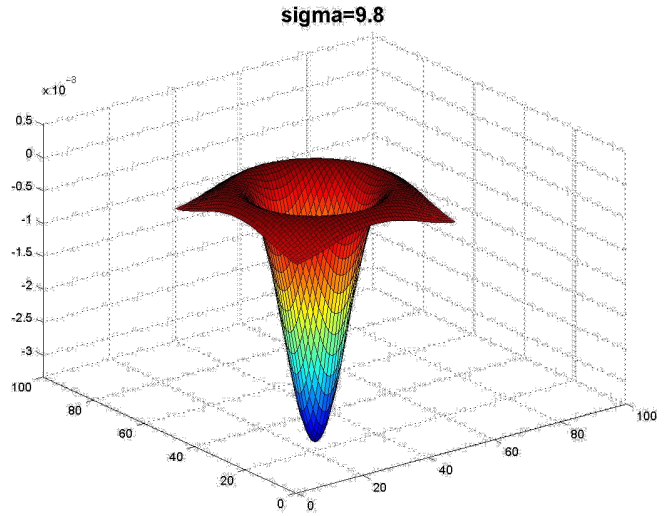




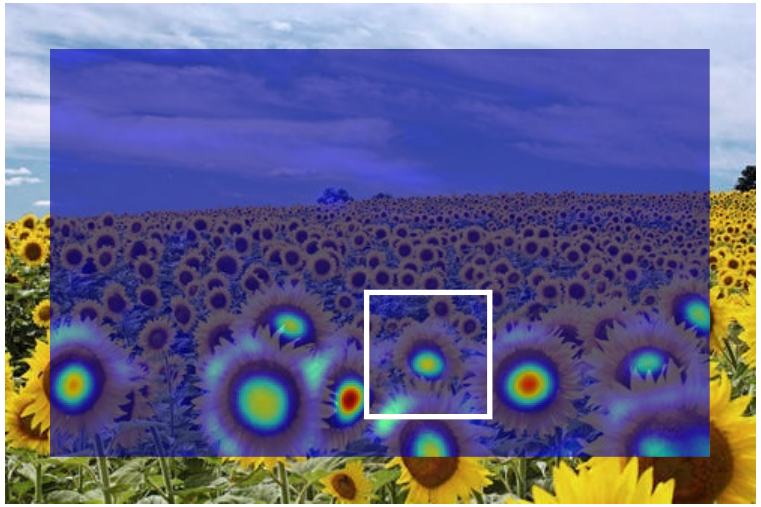

sigma=6
x 10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
-8
100
80
60
40
20
0
0
20
40
60
80
100

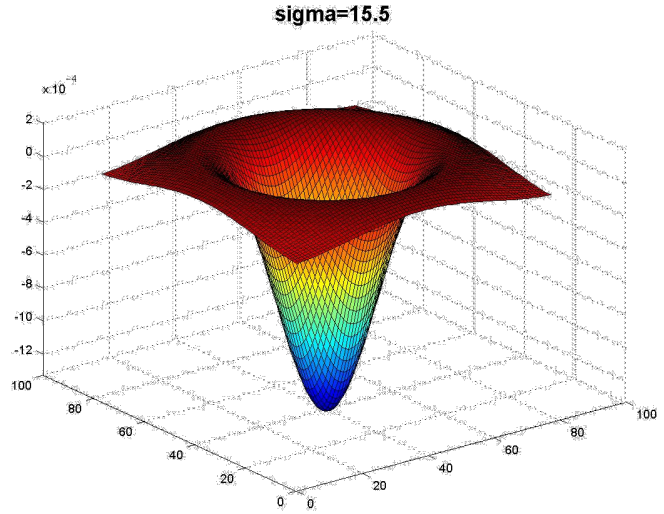


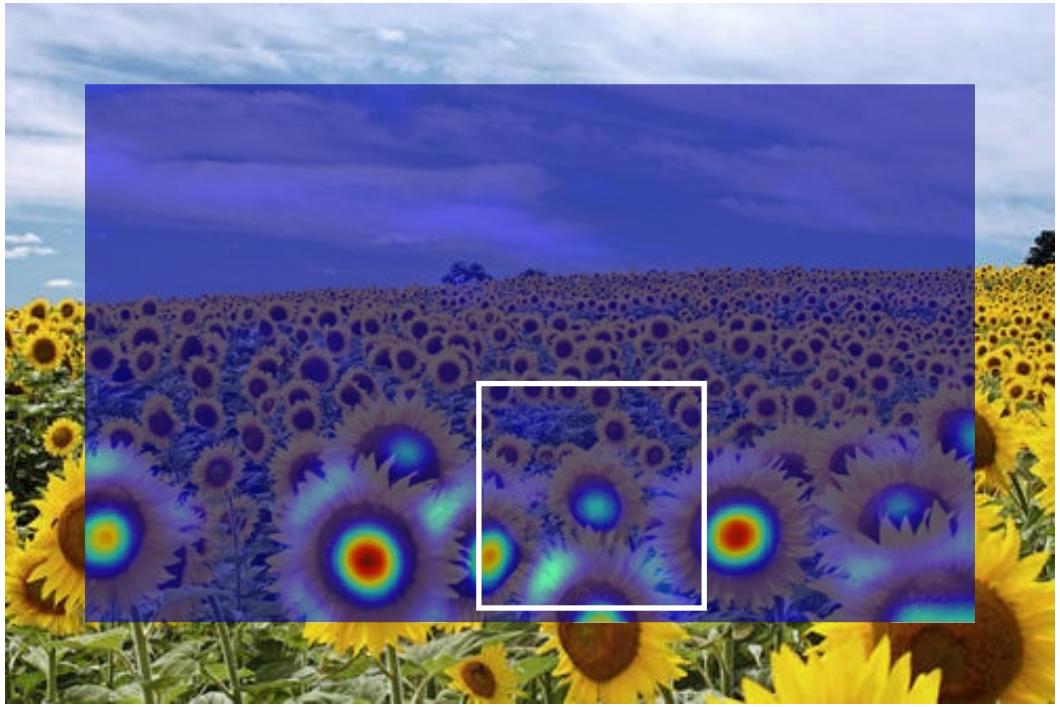

peak!

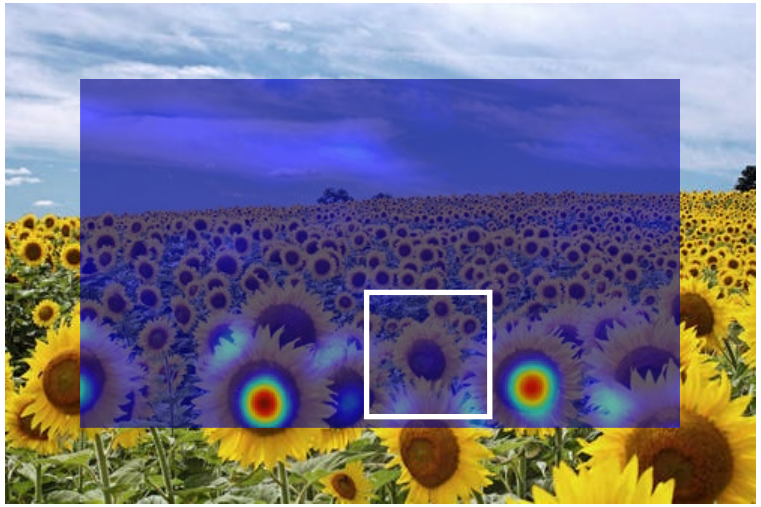

sigma=9.8
x 10^-3
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
100
80
60
40
20
0
0
20
40
60
80
100

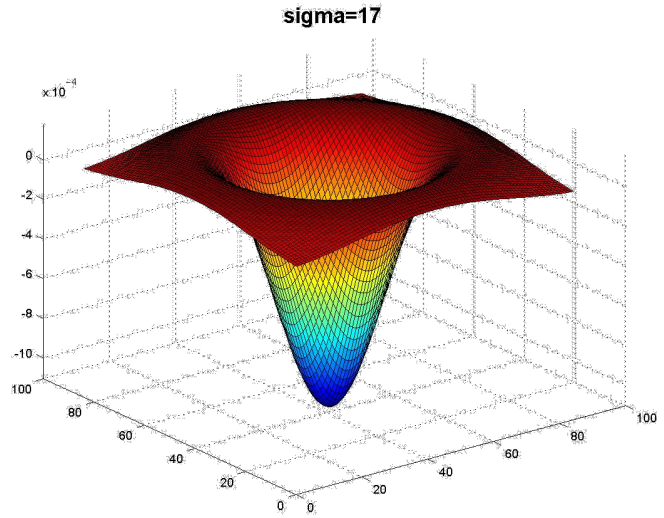

peak!

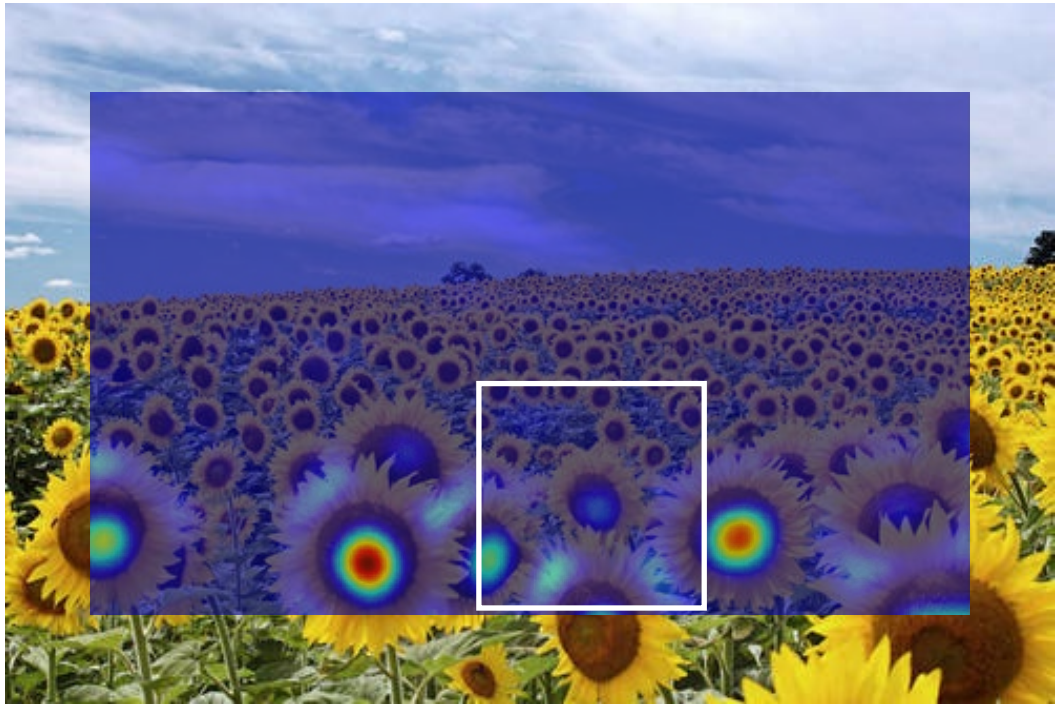


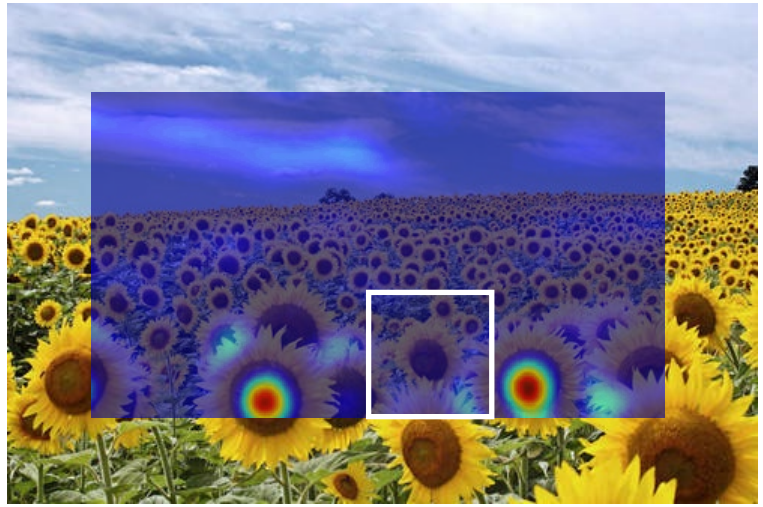

sigma=15.5
x 10^-4
2
0
-2
-4
-6
-8
-10
-12
100
80
60
40
20
0
0
20
40
60
80
100

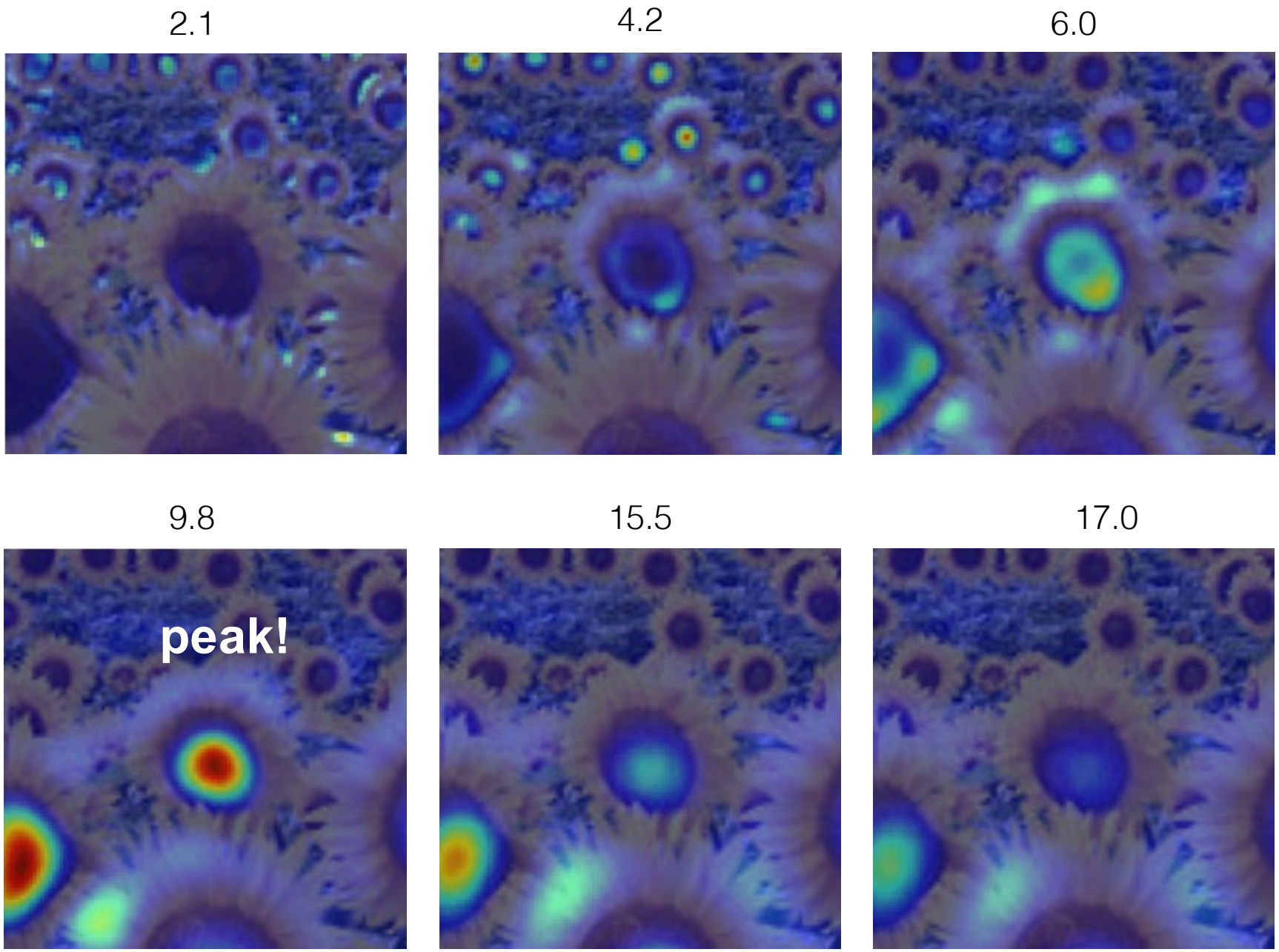


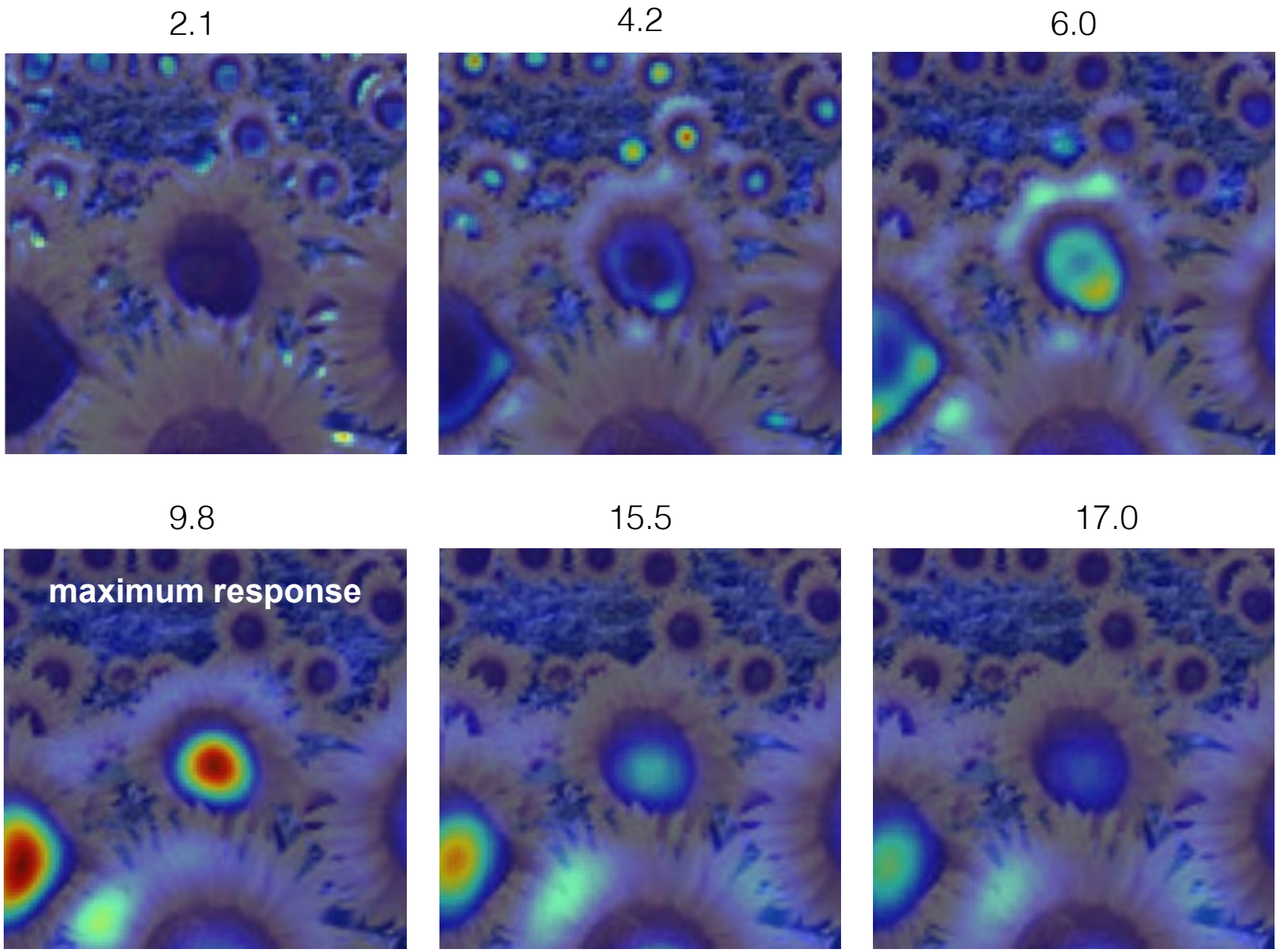



sigma=17
x 10^-4
0
-2
-4
-6
-8
-10
100
80
60
40
20
0
0
20
40
60
80
100

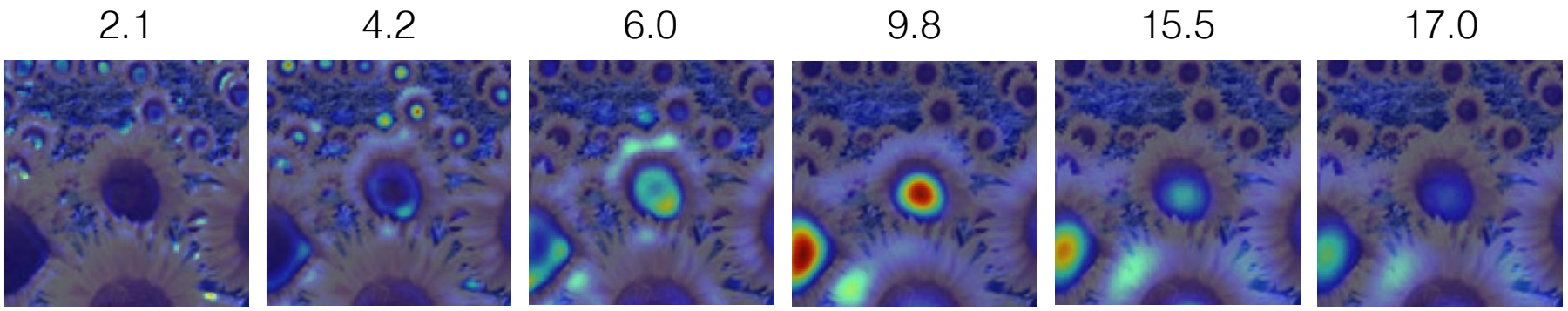




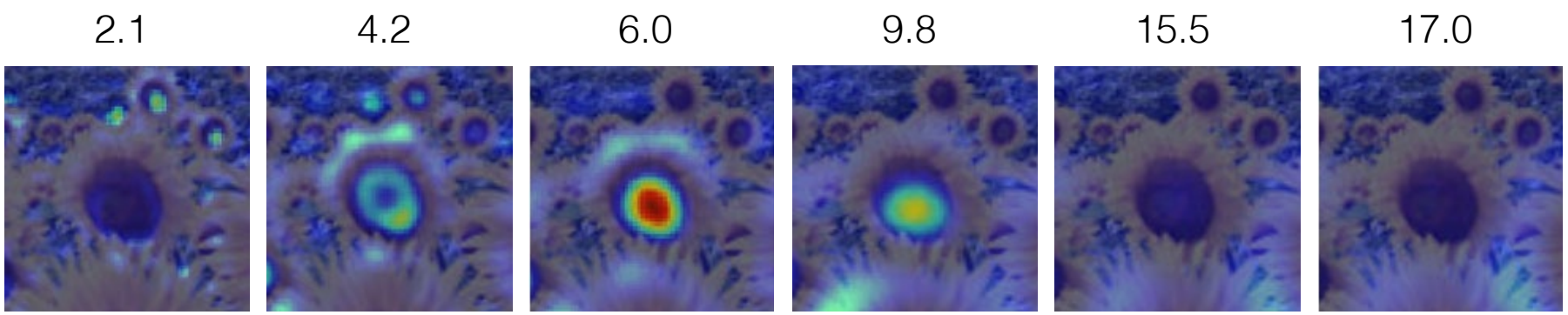

2.1
4.2
6.0
9.8
peak!
15.5
17.0


2.1
4.2
6.0
9.8
maximum response
15.5
17.0

optimal scale

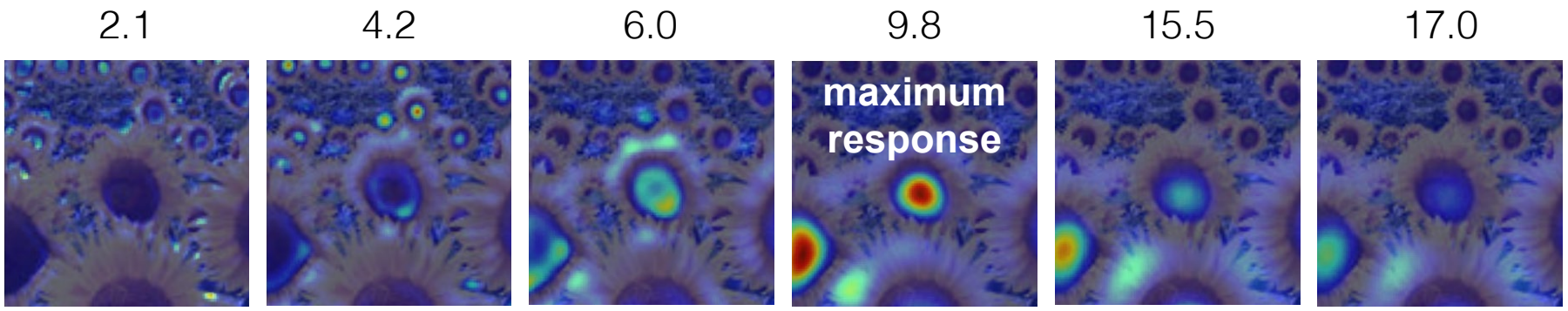


Full size image

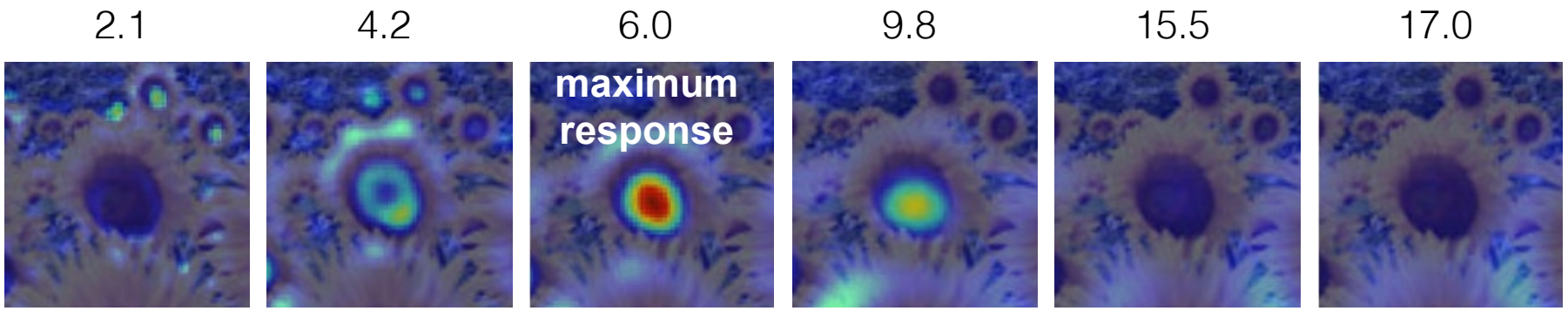


3/4 size image

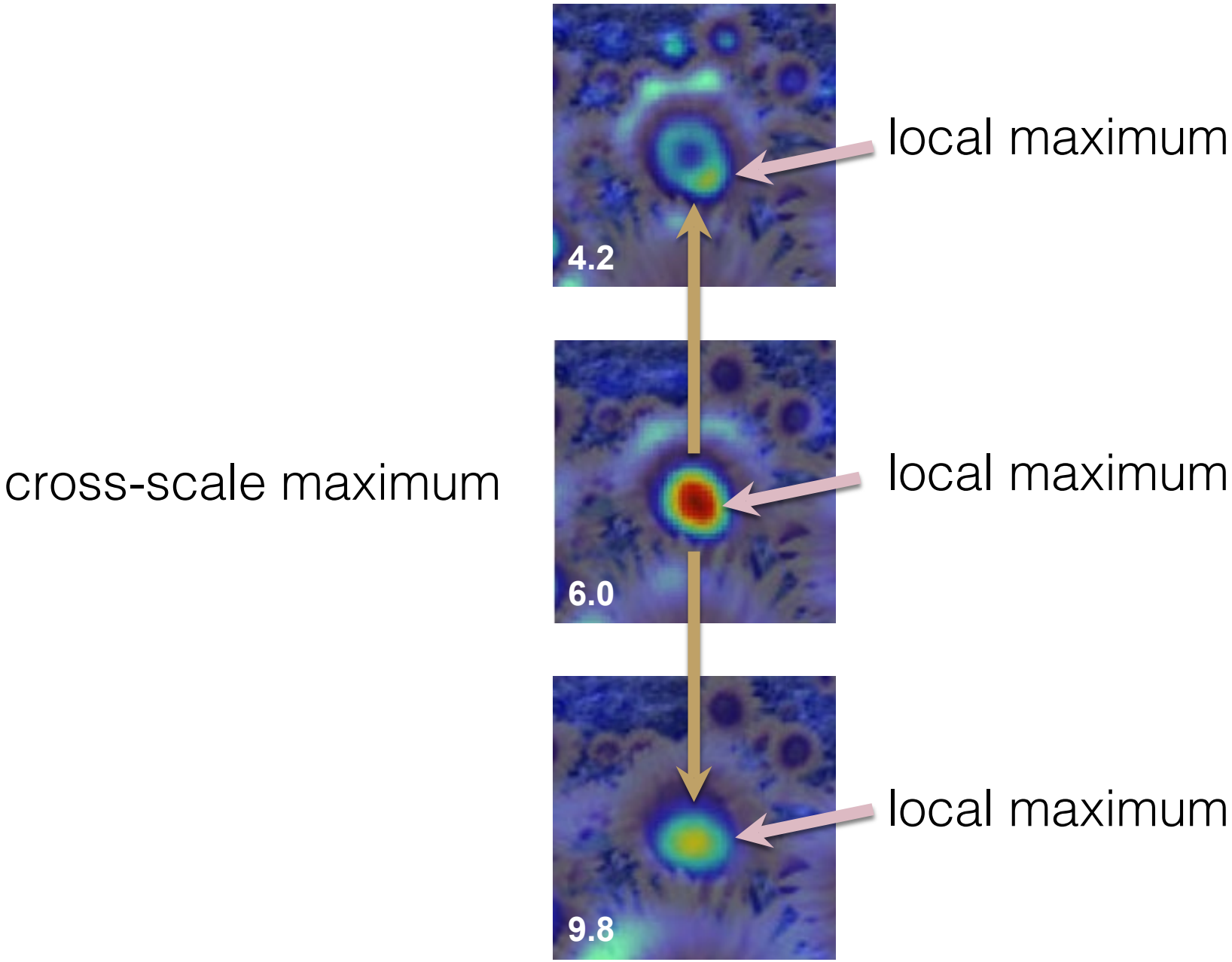
optimal scale



Full size image



3/4 size image


local maximum
4.2
cross-scale maximum
local maximum
6.0
local maximum
9.8

Multi-Scale 2D Blob Detector

Implementation

For each level of the Gaussian Pyramid:

- Compute feature response
- If *local maximum* AND *cross-scale*
  - Save location and scale of feature (x, y, s)

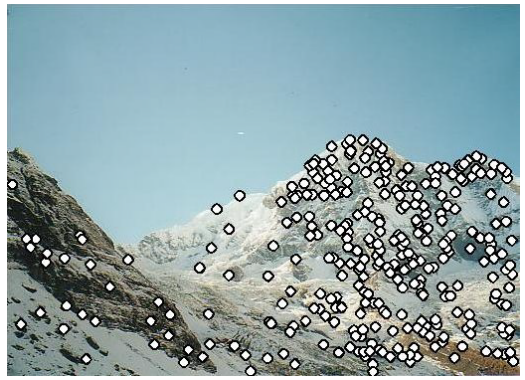
We have detected corners and blobs. But what is it useful for?

So that we can match them with related points

But how do we know that one point is similar to another point?
DESCRIPTORS

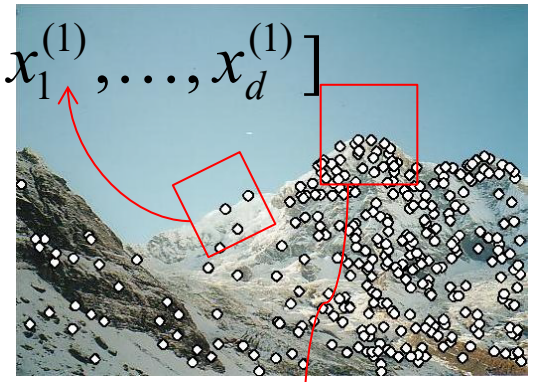
Features: Main Components

1. **DETECTION**
   Identify "interest points"



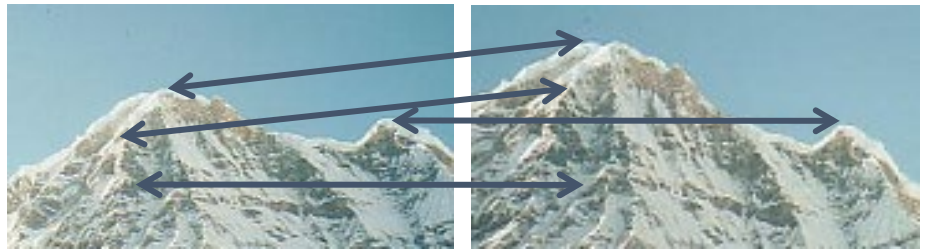
2. **DESCRIPTION**
   Extract "feature descriptor" vectors surrounding each interest point

$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



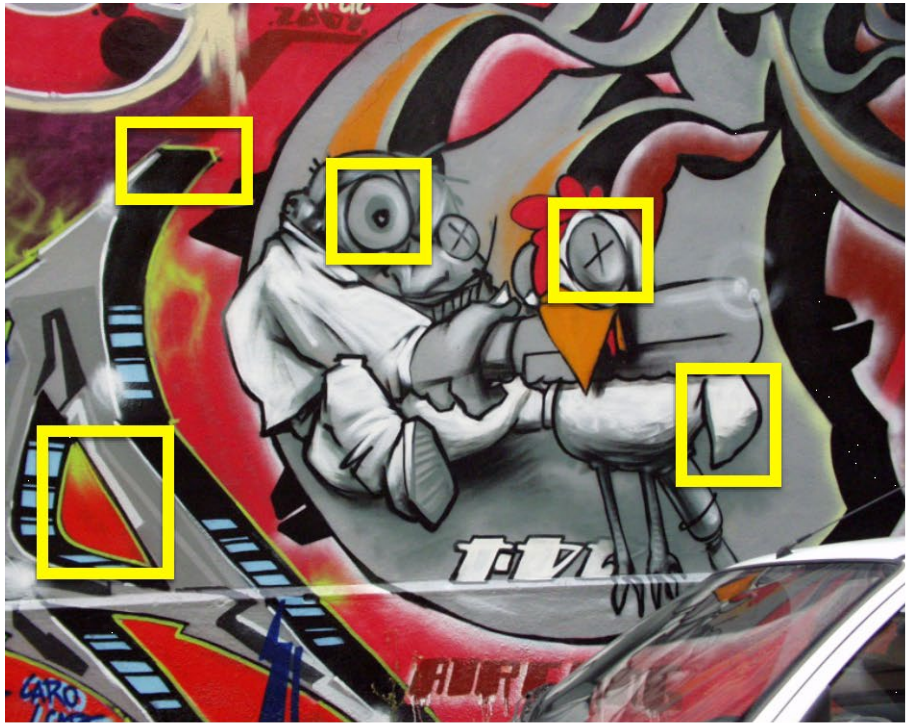
$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

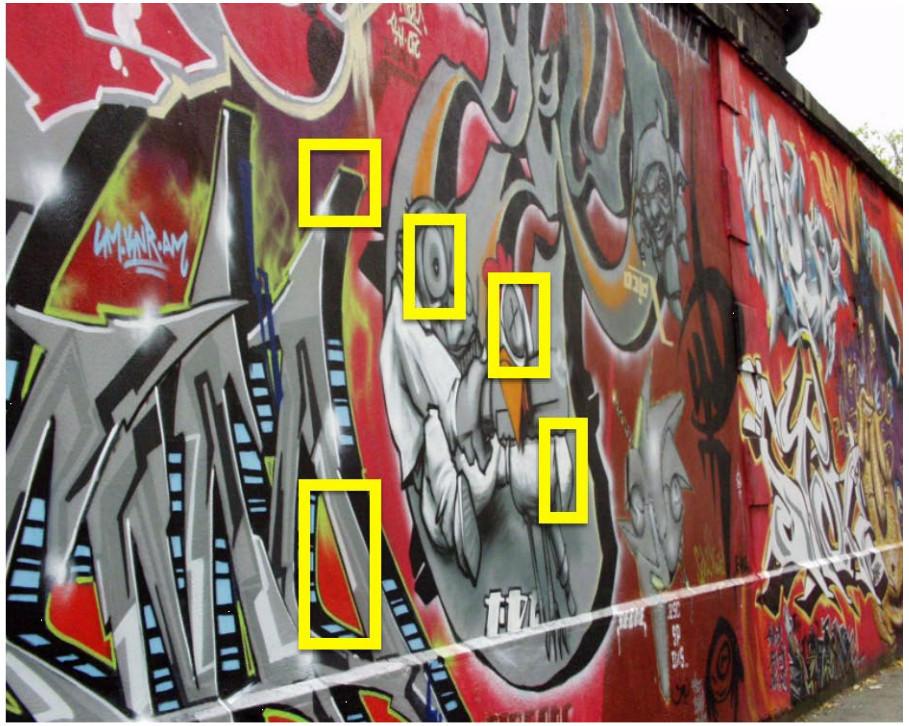
3. **MATCHING**
   Determine correspondence between descriptors in two views



Slide Credit: Kristen Grauman

Feature Description





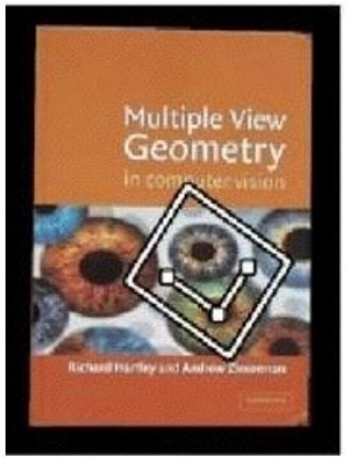
If we know where the good features are,
how do we match them?


How do we describe an image patch?
Patches with similar content should have similar descriptors.

Challenges with Designing A Feature Descriptor

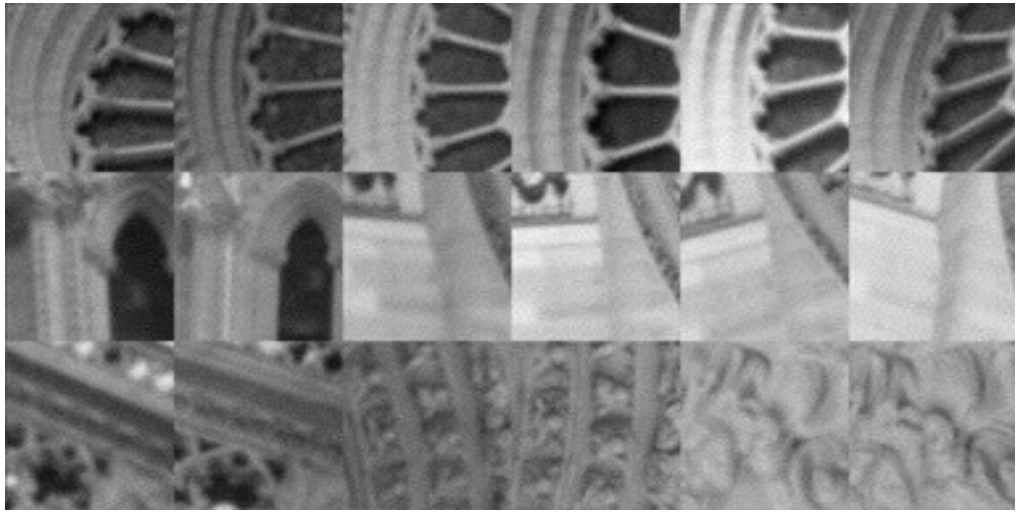
Photometric transformations

Geometric transformations



objects will appear at different scales,
translation and rotation

What is the best descriptor for an image feature?



Idea 1: Image Patches

Image patch

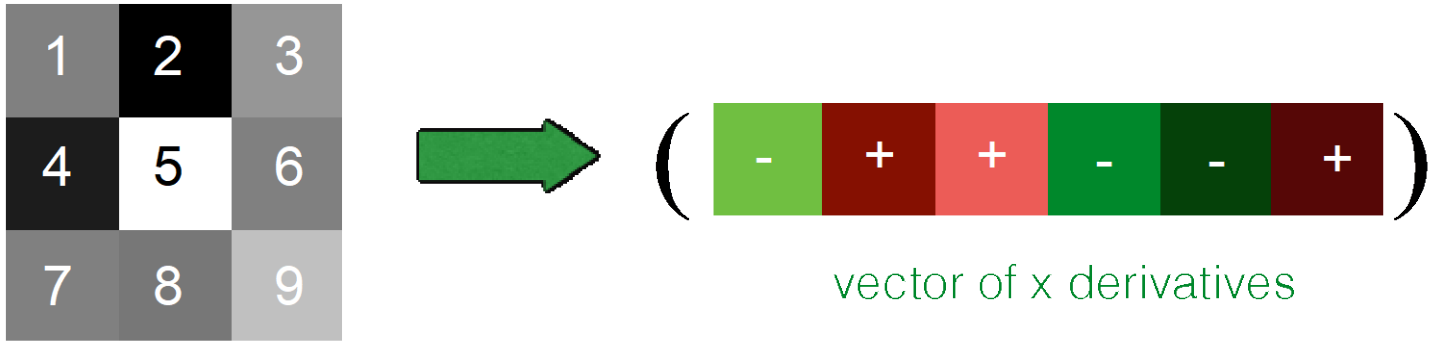
Just use the pixel values of the patch

Perfectly fine if geometry and appearance is unchanged

(a.k.a. template matching)

Image gradients

Use pixel differences



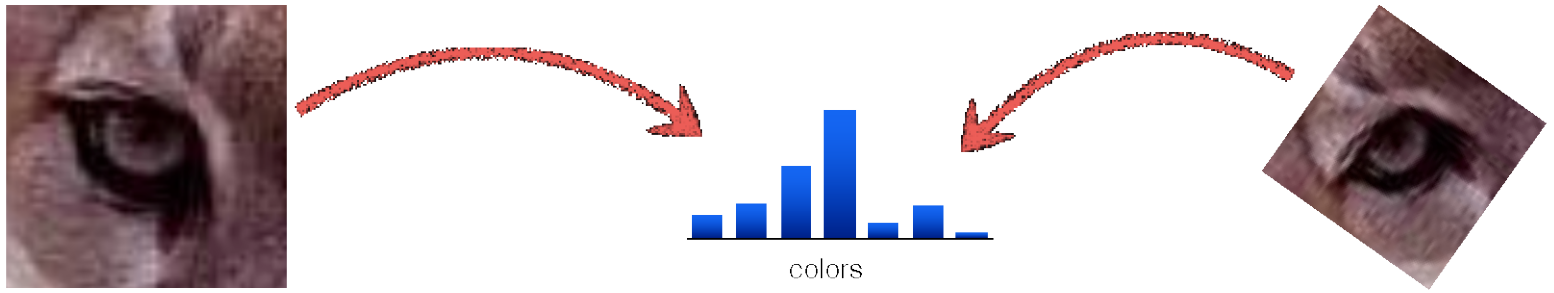
vector of x derivatives

'binary descriptor'

Feature is invariant to absolute intensity values

Color histogram

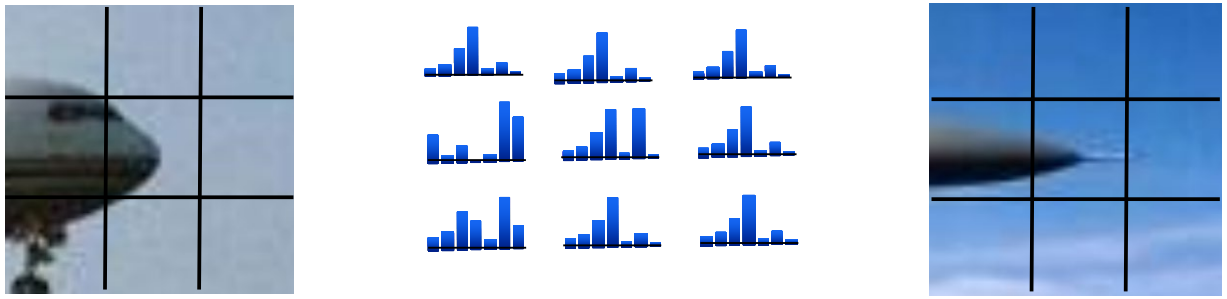
Count the colors in the image using a histogram



Invariant to changes in scale and rotation

Spatial histograms

Compute histograms over spatial ‘cells’

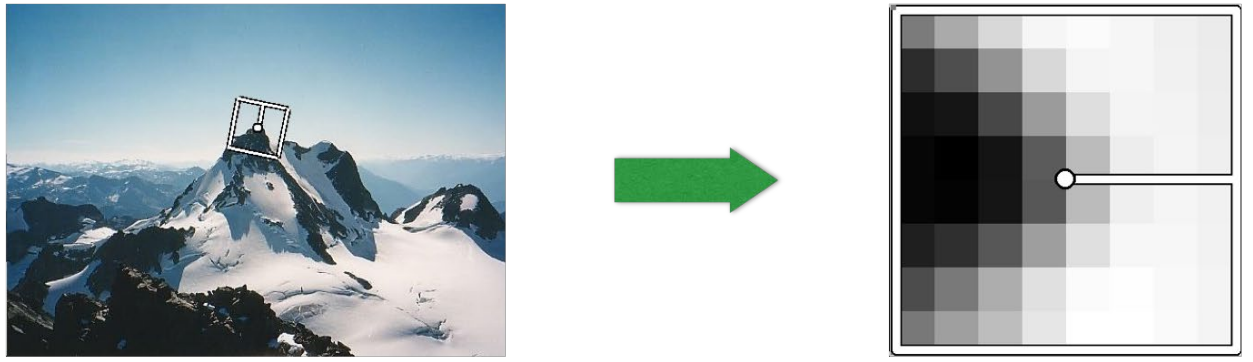


Retains rough spatial layout

Some invariance to deformations

Orientation normalization

Use the dominant image gradient direction to normalize the orientation of the patch



save the orientation angle θ along with (x, y, s)

Many more feature detectors

- MOPS (Multi-Scale Oriented Patches)
- Haar-Wavelet filterbank
- GIST features (uses Gabor filterbank)
- Textons
- HOG (Histogram of Oriented Gradients)
- SURF (Speeded-Up Robust Features)
- BRIEF (Binary Robust Independent Elementary Features)
- …
- …

Invariance and Discriminability

- Invariance:
  - Descriptor shouldn’t change even if image is transformed
- Discriminability:
  - Descriptor should be highly unique for each point

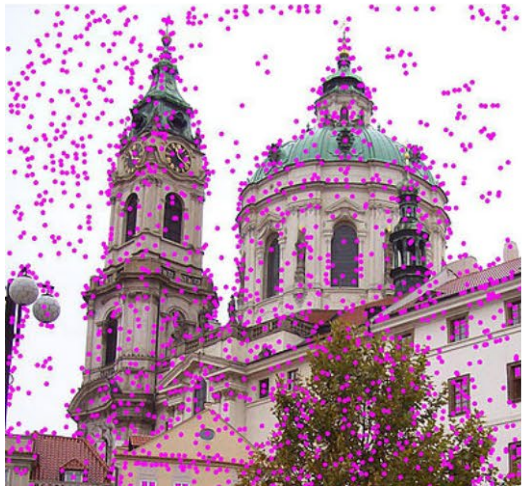
Invariant descriptors

- We looked at invariant / equivariant **detectors**

- Most feature descriptors are also designed to be invariant to:
  - Translation, 2D rotation, scale

- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transforms (some are fully affine invariant)
  - Limited illumination/contrast changes

Main (classical) feature used: SIFT



SIFT

(Scale Invariant Feature Transform)

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

Some history of SIFT

- The SIFT paper by David Lowe was rejected multiple times

David Lowe relates the story:

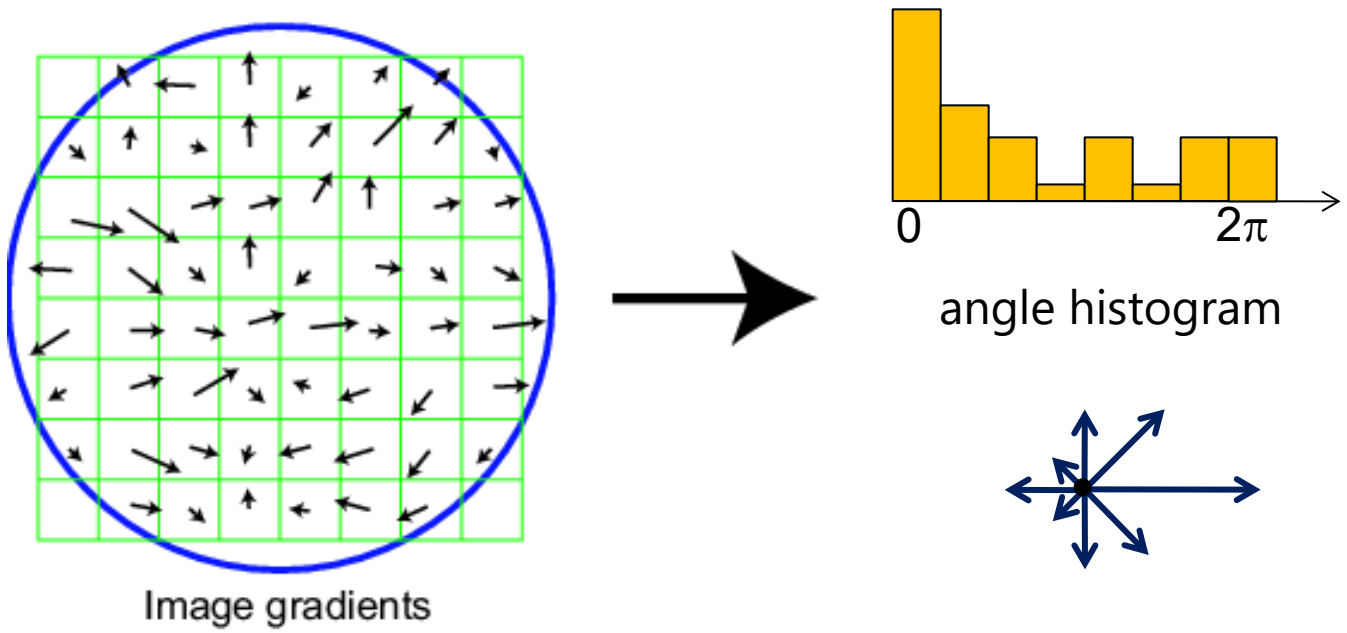
> *I did submit papers on earlier versions of SIFT to both ICCV and CVPR (around 1997/98) and both were rejected. I then added more of a systems flavor and the paper was published at ICCV 1999, but just as a poster. By then I had decided the computer vision community was not interested, so I applied for a patent and intended to promote it just for industrial applications.*
>
> *Another recent example is Rob Fergus's tiny images paper, which never did appear in a conference, but already has had a strong impact. I'm sure there are hundreds of other examples.*

Source: http://yann.lecun.com/ex/pamphlets/publishing-models.html

- SIFT went on to become the most highly cited paper in all of engineering sciences in 2005.

Scale Invariant Feature Transform

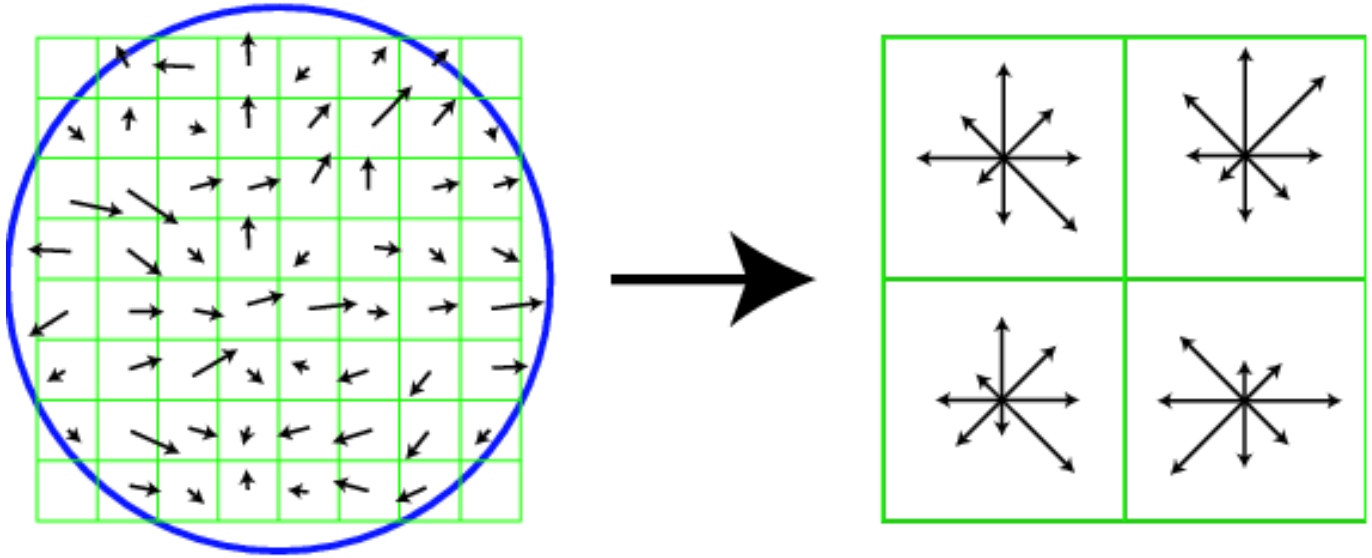
- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations
- Shift the bins so that the biggest one is first



SIFT descriptor

Full version

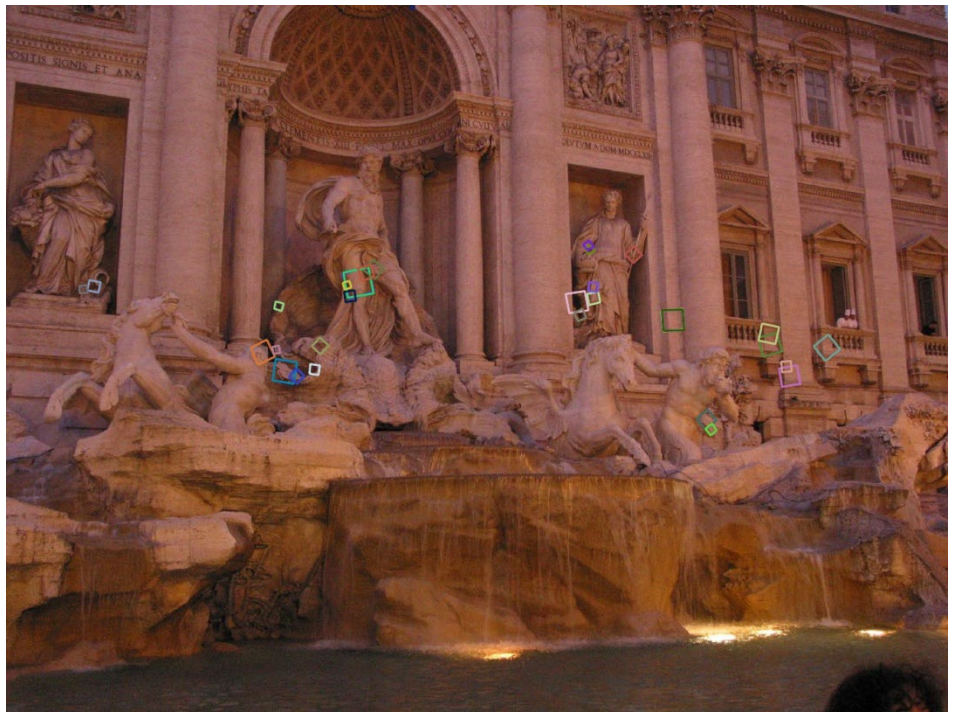
- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



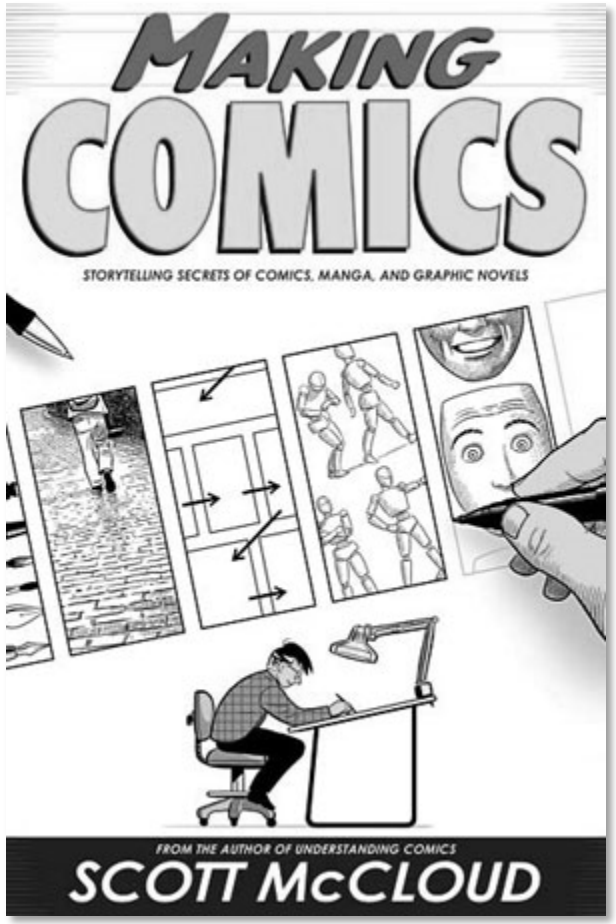
Properties of SIFT

Extraordinarily robust matching technique

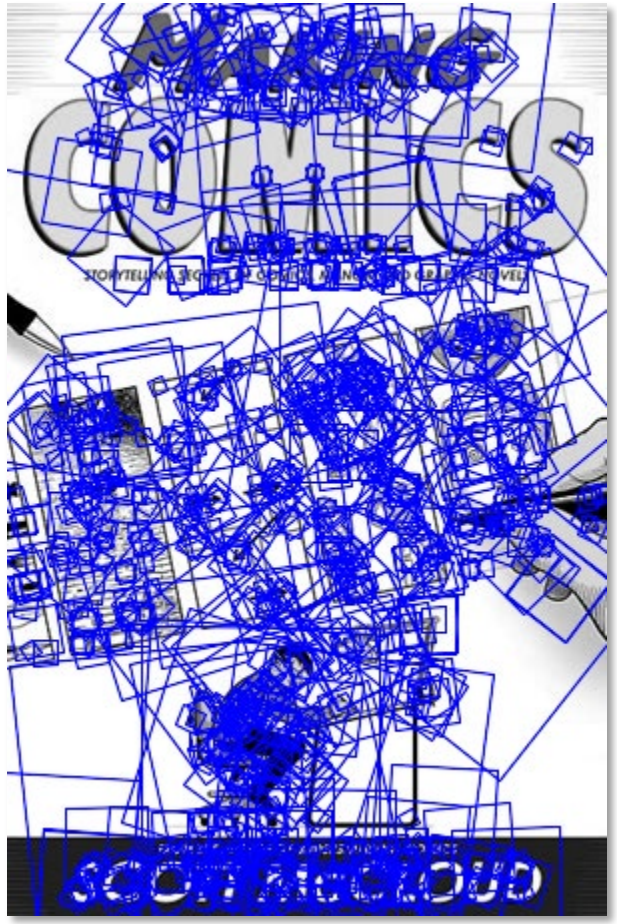
- Can handle changes in viewpoint (up to about 60 degree out of plane rotation)
- Can handle significant changes in illumination (sometimes even day vs. night (below))
- Pretty fast—hard to make real-time, but can run in <1s for moderate image sizes
- Lots of code available



SIFT Example



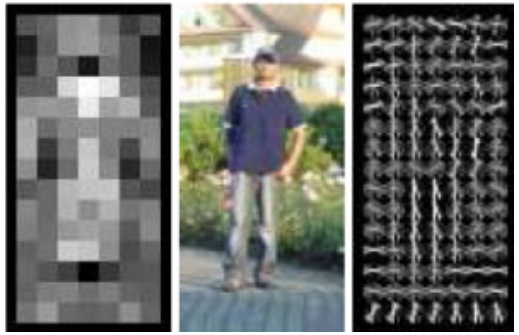
sift



868 SIFT features

Other descriptors

- HOG: Histogram of Gradients (HOG)
  - Dalal/Triggs
  - Sliding window, pedestrian detection



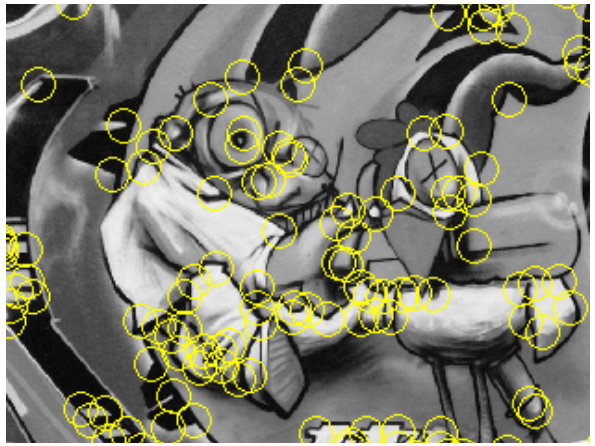
- FREAK: Fast Retina Keypoint
  - Perceptually motivated
  - Can run in real-time; used in Visual SLAM on-device

- LIFT: Learned Invariant Feature Transform
  - Learned via deep learning – along with many other recent features

https://arxiv.org/abs/1603.09114

Summary

- Keypoint detection: repeatable and distinctive
  - Corners, blobs
  - Harris, DoG



- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one