Intelligent Control Systems

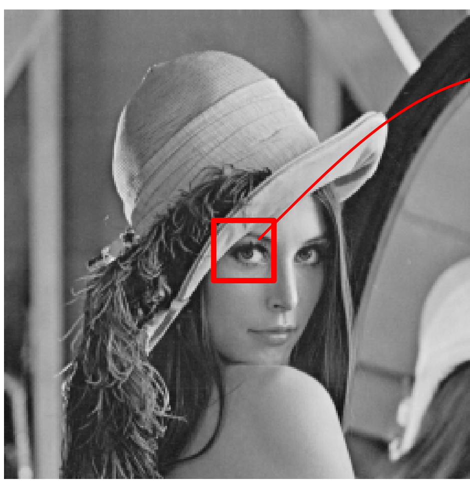
Visual Tracking (2)
— Feature-based Methods —

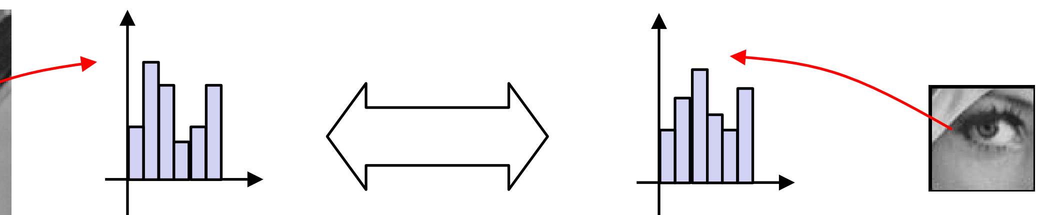
Shingo Kagami
Graduate School of Information Sciences,
Tohoku University
swk(at)ic.is.tohoku.ac.jp

http://www.ic.is.tohoku.ac.jp/ja/swk/

Feature-based Methods vs Direct Methods

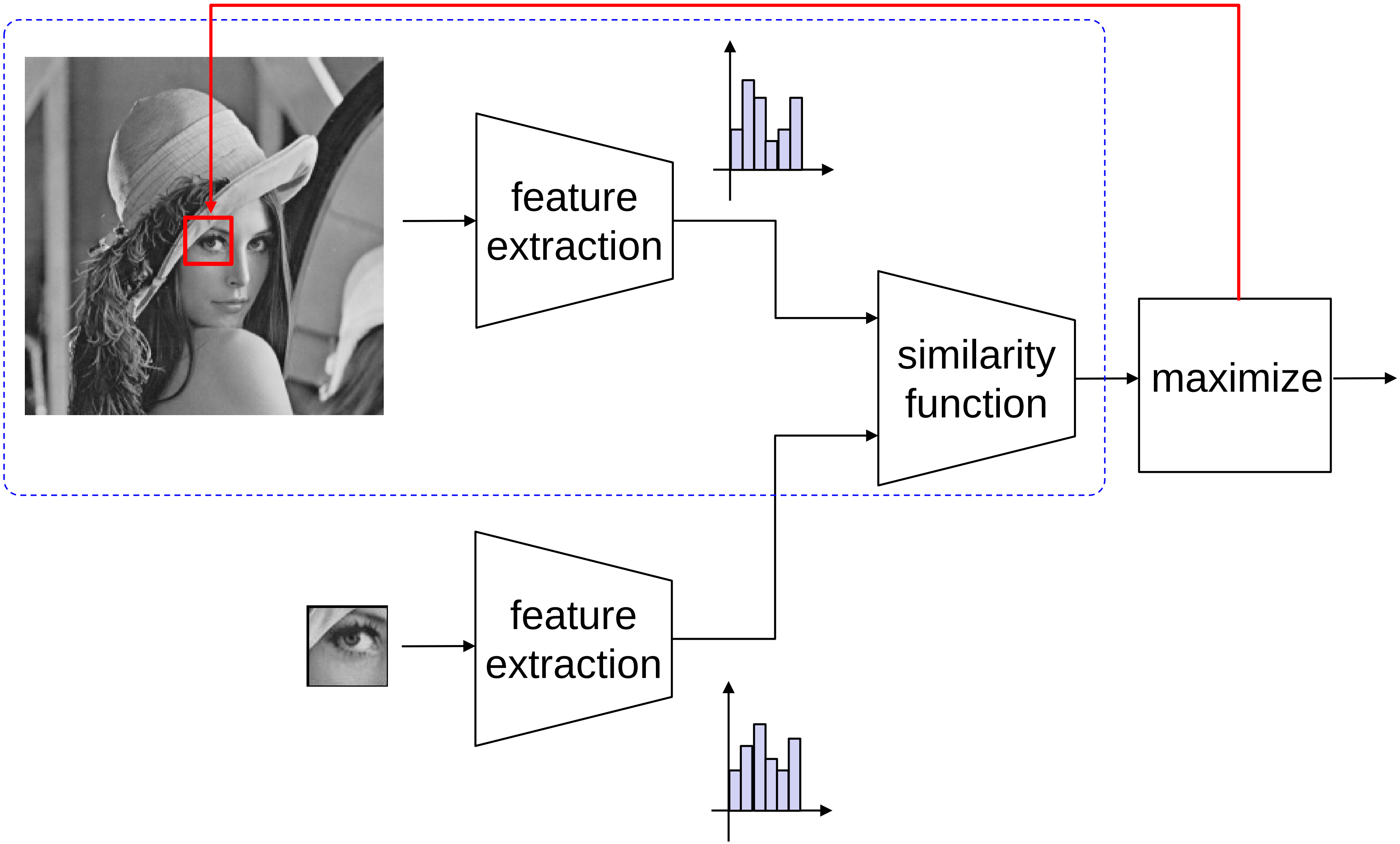
direct comparison of
pixel values



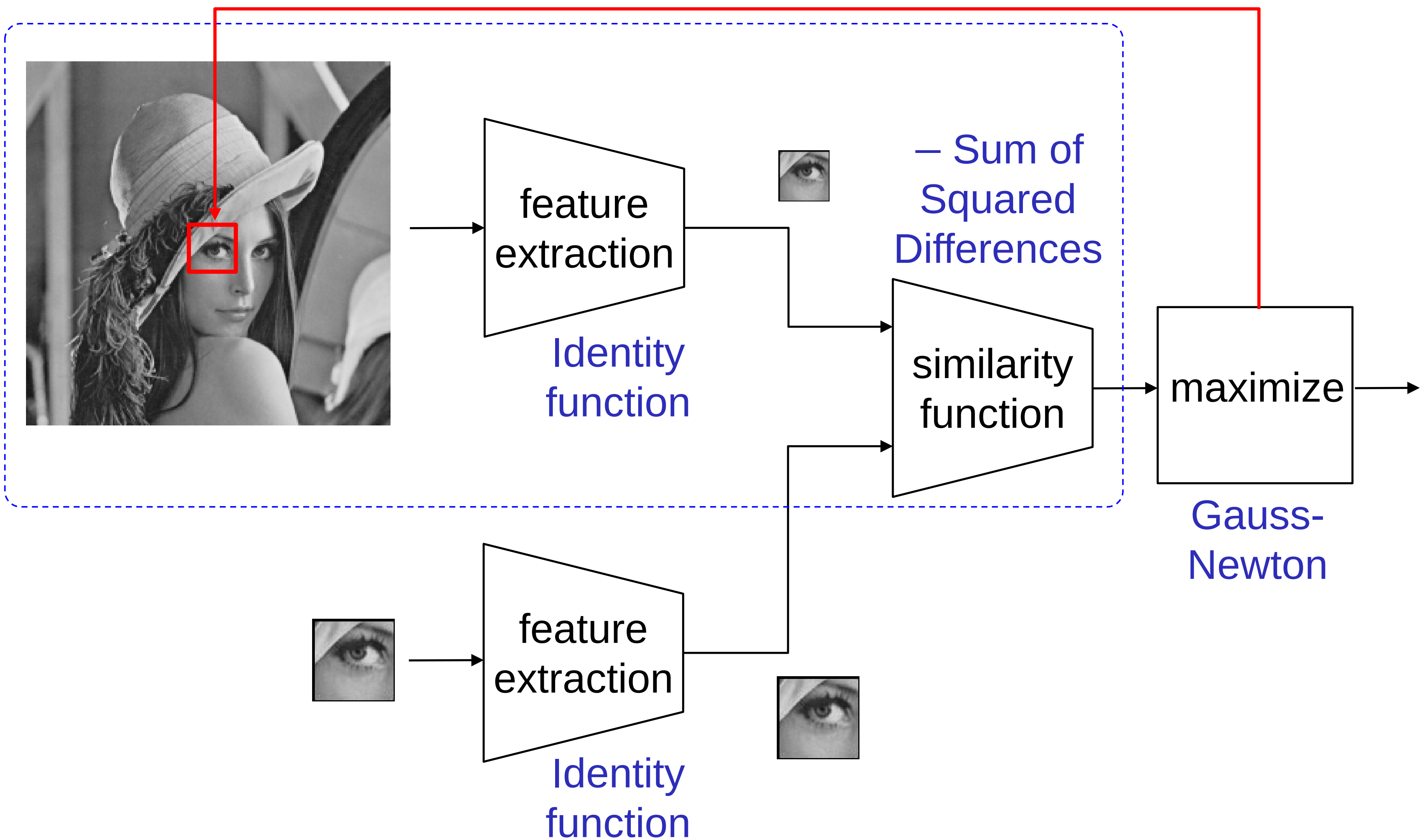


comparison of feature values
computed from images (e.g.
histograms, edge positions, …)

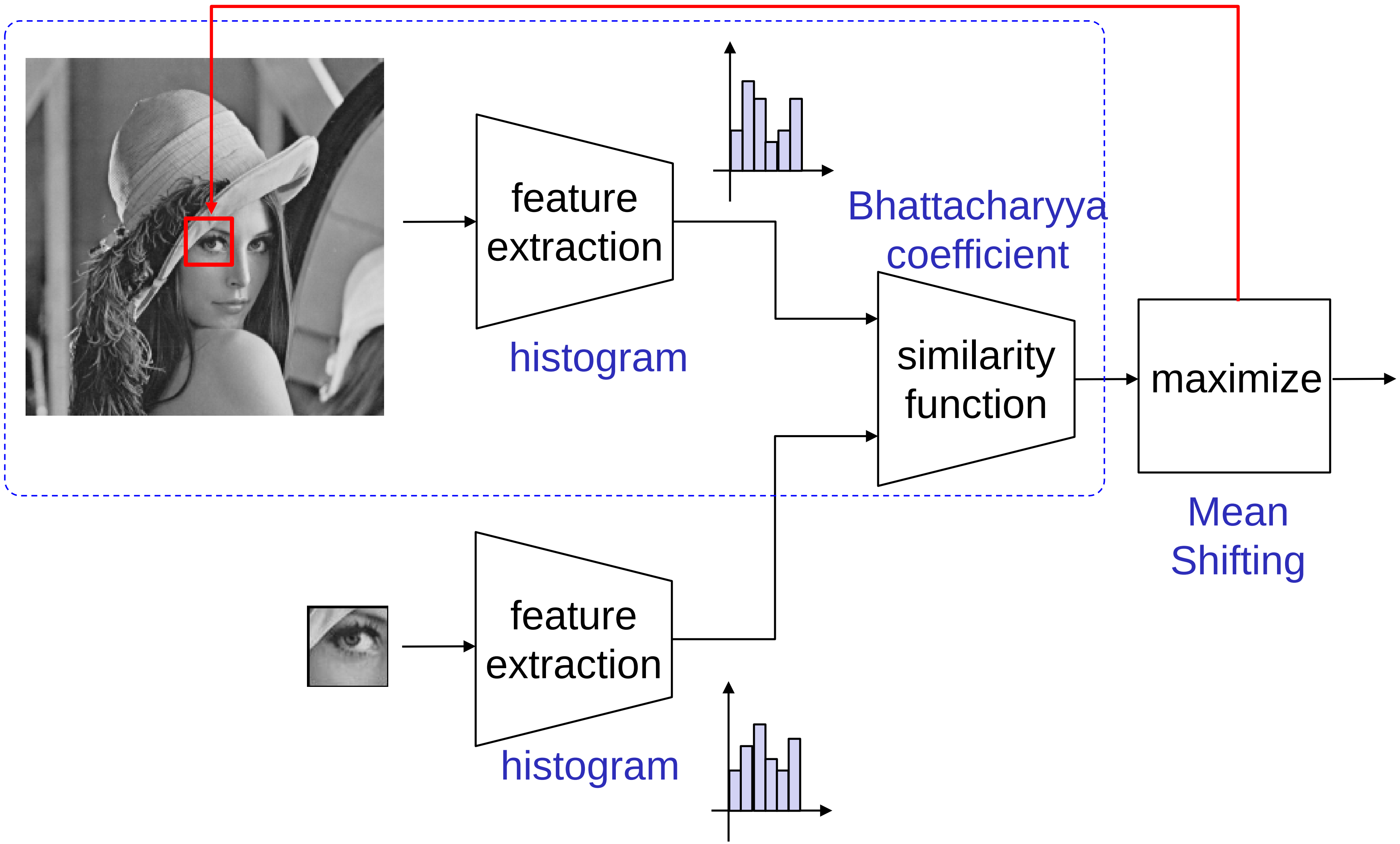
General Framework



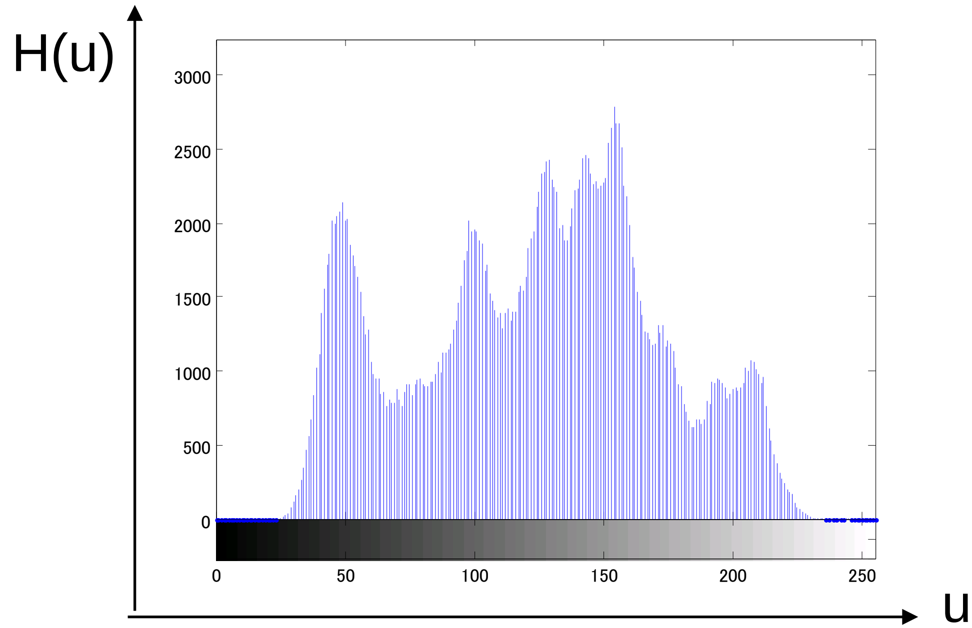
Lucas-Kanade Tracking (last week)



Histogram-based Mean Shift Tracking



(Grayscale) Histogram



$$\boldsymbol{H} = \{H_u\}_{u=1,2,\cdots,m}, \ H_u = \sum_{\boldsymbol{x} \in S(u)} 1$$

where S(u) is a set of pixels having values belonging to the bin u

$$\boldsymbol{p} = \{p_u\}, \ p_u \propto H_u, \ \sum_{u=1}^{m} p_u = 1$$

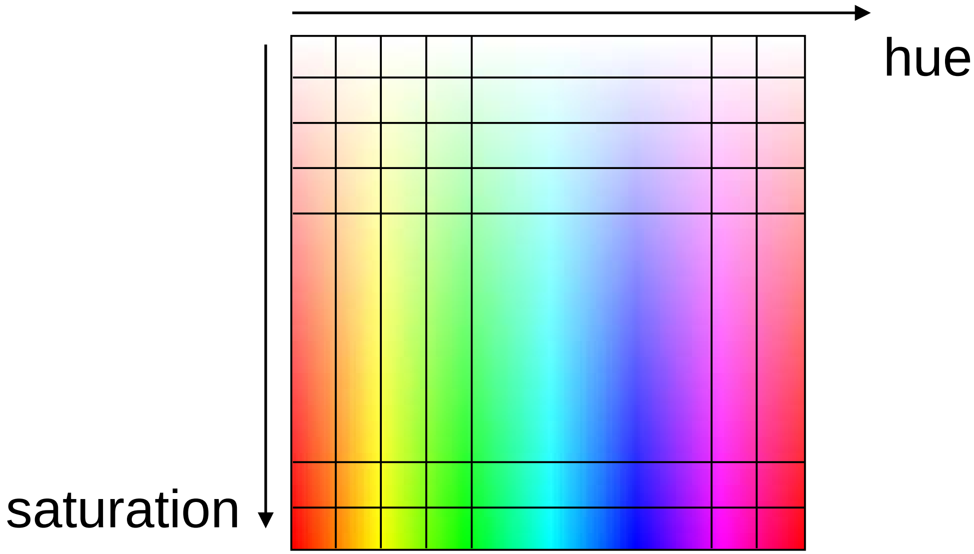
(normalized histogram)

Color Histograms

e.g.1) By splitting each of RGB components into 16 bins, we have histogram over 16 x 16 x 16 bins

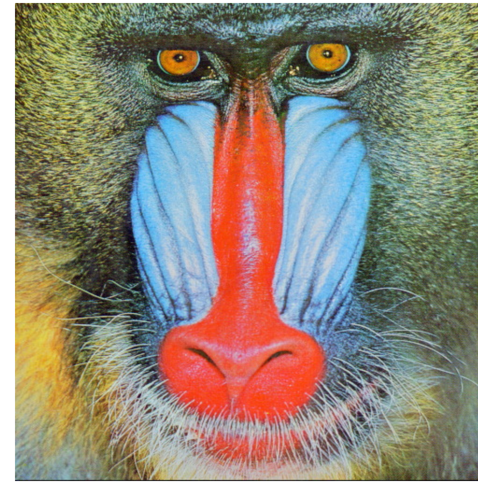
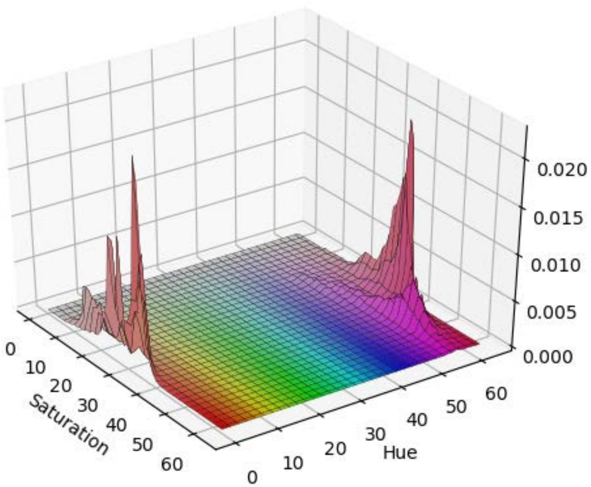
e.g.2) By splitting each of Hue and Saturation components into 64 bins (and ignoring Value component), we have histogram over 64 x 64 bins

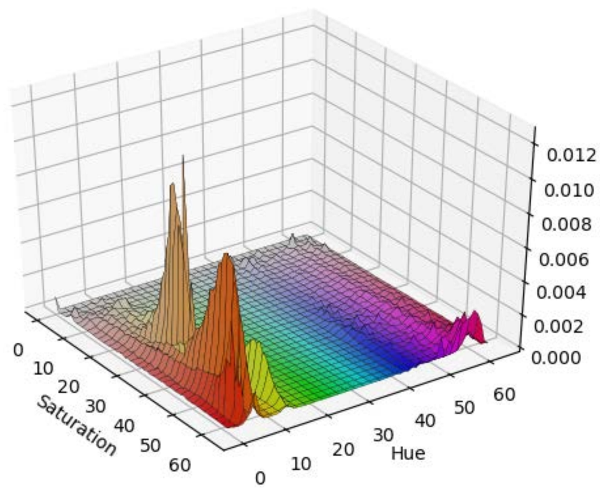
- Less affected by illumination change

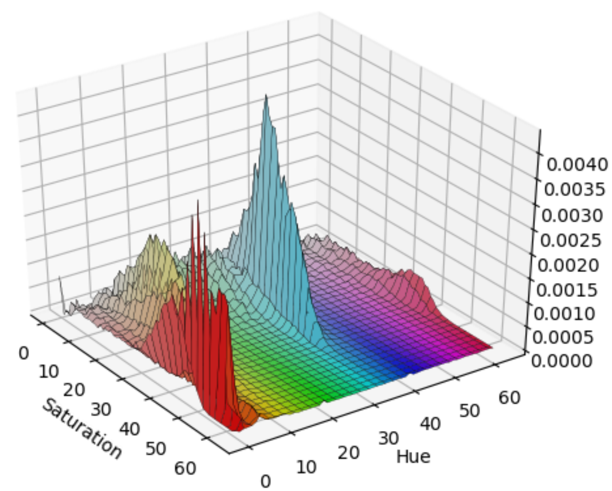


Hue-Saturation Histograms







04_color_histogram.py
(Also download color_histogram_utils.py and put it in the same folder)

Weighted Histogram

- The pixels near boundaries should have small influence
- Discontinuity in the similarity map is not favored

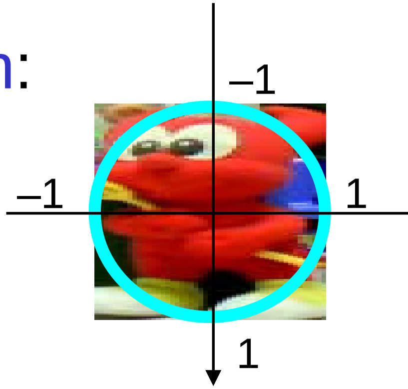
  ! **weight the voting depending of pixel locations**

Object Model:

$$q_u \propto \sum_{\boldsymbol{x} \in S_0(u)} k(\|\boldsymbol{x}\|^2)$$

Histogram of candidate region:

$$p_u(\boldsymbol{y}) \propto \sum_{\boldsymbol{x} \in S(u)} k(\|\boldsymbol{y} - \boldsymbol{x}\|^2)$$



S_0(u): set of pixels whose pixel values belong to bin u in the model image

S(u): the same above in the current image

k(): weight function or kernel function

Image coordinates **x** = (x, y) are normalized such that it fits a unit circle

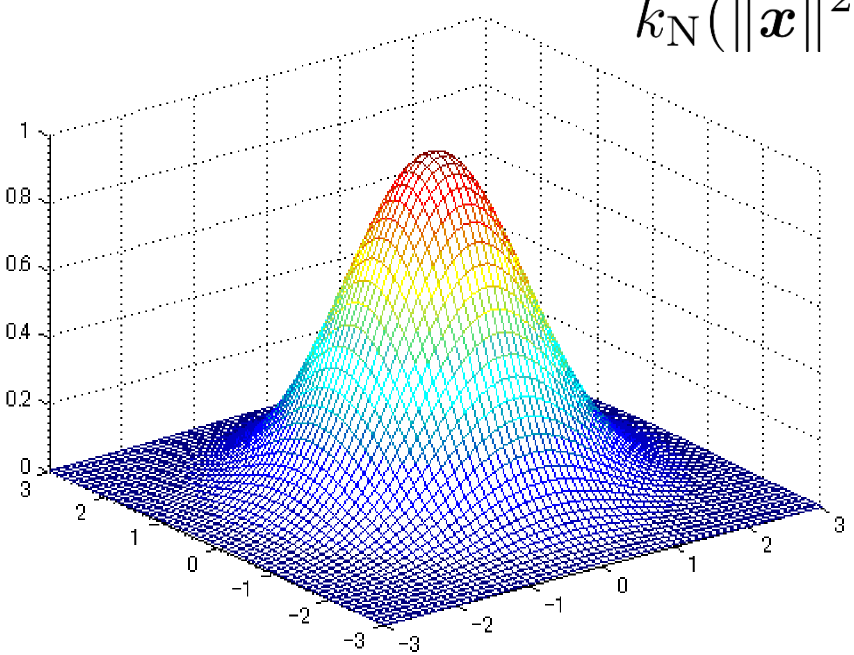
unit circle centered at **y**

Kernel Function Examples

e.g.1) Gauss kernel

$$k_{\mathrm{N}}(t) \propto \exp\left(-\frac{t}{2}\right)$$

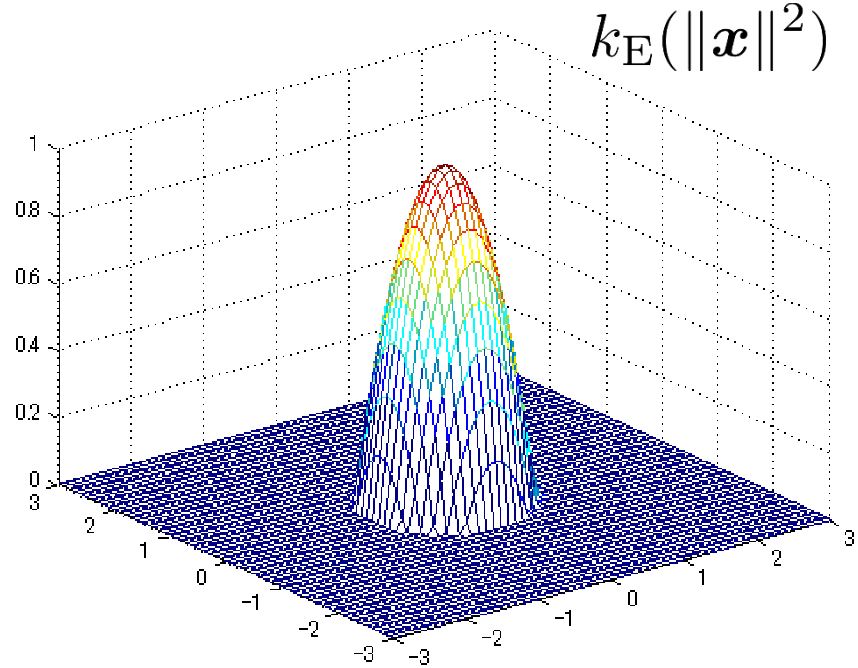
$k_{\mathrm{N}}(\|\boldsymbol{x}\|^2)$



e.g.2) Epanechnikov kernel

$$k_{\mathrm{E}}(t) \propto \begin{cases} 1-t, & t \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

$k_{\mathrm{E}}(\|\boldsymbol{x}\|^2)$



Similarity of Histograms

- Our objective is to find a region with histogram similar to that of a given model
- How do we measure the similarity?

Bhattacharyya Coefficient

- is a metric for similarity of two probabilistic distributions (and thus, of two normalized histograms) **p** and **q**

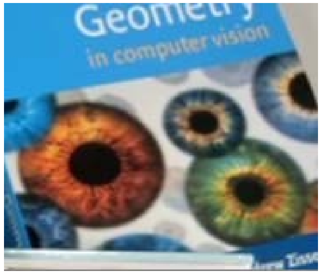
$$\rho(\boldsymbol{p}, \boldsymbol{q}) = \sum_{u=1}^{m} \sqrt{p_u q_u}$$

- Geometric interpretation: inner product of $(\sqrt{q_1}, \sqrt{q_2}, \cdots, \sqrt{q_m})^{\mathrm{T}}$ and $(\sqrt{p_1}, \sqrt{p_2}, \cdots, \sqrt{p_m})^{\mathrm{T}}$, which lie on the unit sphere surface

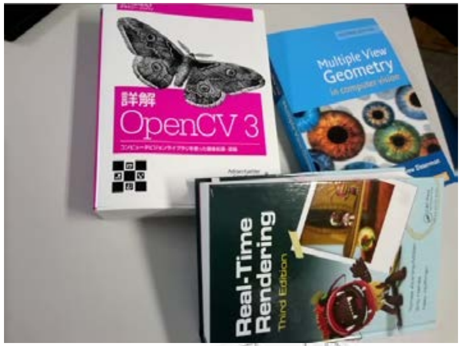
Why not other similarity measure?: Simply because this is convenient for the mean shift method

Similarity Map with Weighted Histogram

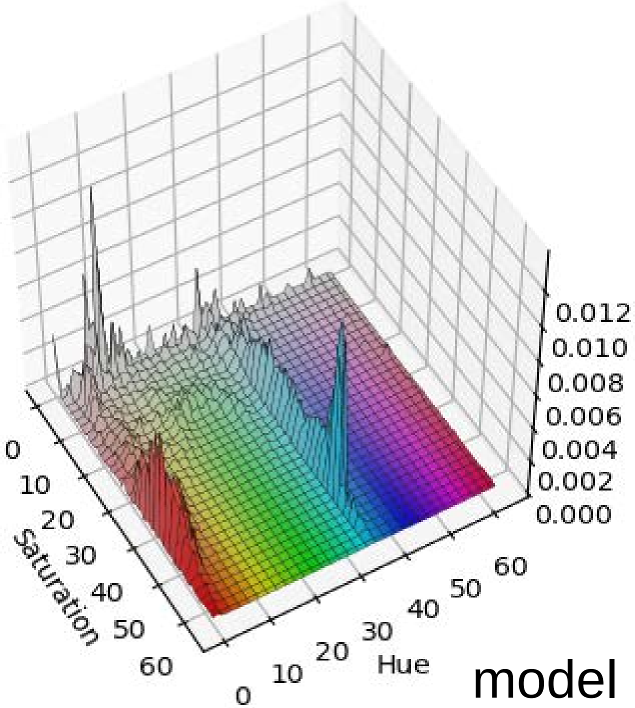
04_color_histogram_similarity.py (and color_histogram_utils.py)



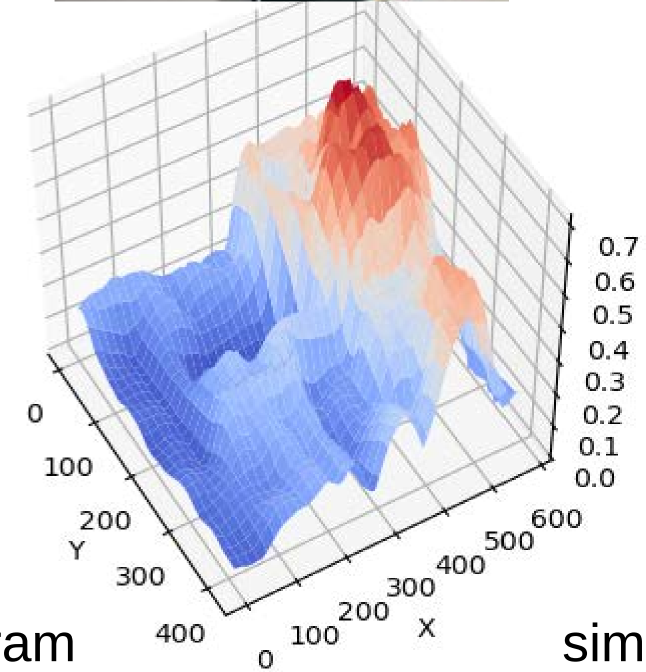
object model



current image



model histogram



similarity map

Dictionary in Python

```
dic = { "a": 123, "b": (10, 20), "c": "Foo" }
dic["a"]
  -> 123
dic["b"]
  -> (10, 20)
dic["c"] = "Bar"
dic["c"]
  -> 'Bar'
```

cf.

```
lis = [123, (10, 20), "Foo"]
lis[0]
  -> 123
lis[1]
  -> (10, 20)
lis[2]
  -> 'Foo'
```

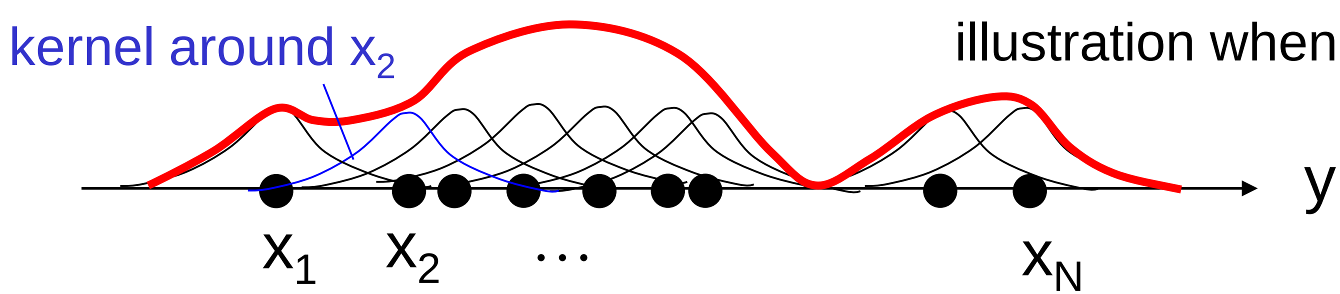
Kernel Density Estimation (KDE)

- Since brute-force search for maximum similarity is too time consuming, let us think of using a gradient-based method
- We introduce Mean Shift Method that finds a local maximum of probability density distribution estimated through data samples [Fukunaga 1975]

Given data samples $\boldsymbol{x}_i$ with weights $w(\boldsymbol{x}_i)$ drawn from an unknown probability distribution $p(\boldsymbol{y})$, KDE (or Parzen estimation) of $p(\boldsymbol{y})$ is given by

$$p(\boldsymbol{y}) \sim \sum_i w(\boldsymbol{x}_i) k(\|\boldsymbol{y} - \boldsymbol{x}_i\|^2)$$

k(): kernel function



Mean Shift Method

- An efficient method to find a local maximum of a probability distribution estimated by KDE

$$f_k(\boldsymbol{y}) = \sum_{\boldsymbol{x} \in \text{samples}} w(\boldsymbol{x}) k(\|\boldsymbol{y} - \boldsymbol{x}\|^2)$$

Gradient at **y**: $$\nabla f_k(\boldsymbol{y}) = \frac{\partial}{\partial \boldsymbol{y}} f_k(\boldsymbol{y}) = \sum_{\boldsymbol{x}} k'(\|\boldsymbol{y} - \boldsymbol{x}\|^2) \cdot 2(\boldsymbol{y} - \boldsymbol{x}) w(\boldsymbol{x})$$

Defining $g(x) = -k'(x)$, we have

$$\nabla f_k(\boldsymbol{y}) = 2 \sum_{\boldsymbol{x}} g(\|\boldsymbol{y} - \boldsymbol{x}\|^2)(\boldsymbol{x} - \boldsymbol{y}) w(\boldsymbol{x})$$

$f_g(\boldsymbol{y})$: KDE with kernel g()

$$= 2 \left[\sum_{\boldsymbol{x}} \{\boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2)\} - \boldsymbol{y} \sum_{\boldsymbol{x}} \{w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2)\} \right]$$

$$= 2 f_g(\boldsymbol{y}) \left[\frac{\sum_{\boldsymbol{x}} \{\boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2)\}}{f_g(\boldsymbol{y})} - \boldsymbol{y} \right]$$

$\boldsymbol{m}_g(\boldsymbol{y})$: mean shift vector

Interpretation of Mean Shift Vector

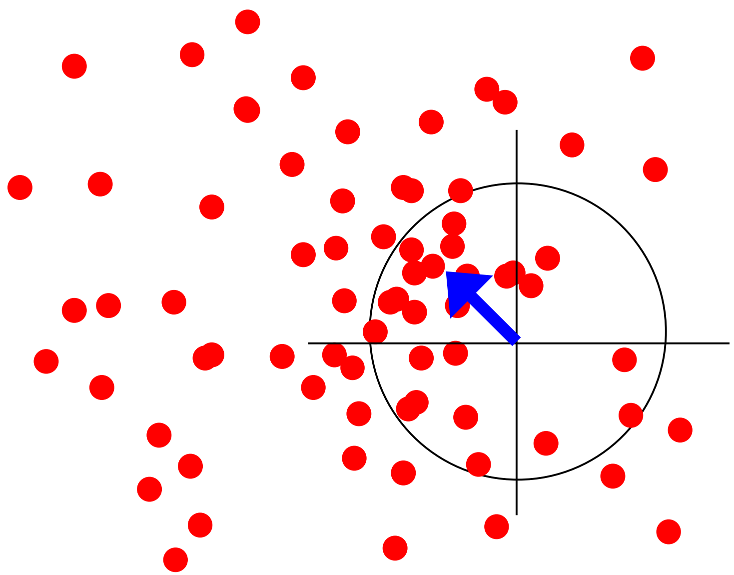
$$\boldsymbol{m}_g(\boldsymbol{y}) = \frac{\sum_{\boldsymbol{x}} \{\boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y}-\boldsymbol{x}\|^2)\}}{f_g(\boldsymbol{y})} - \boldsymbol{y}$$
$$= \frac{\sum_{\boldsymbol{x}} \{\boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y}-\boldsymbol{x}\|^2)\}}{\sum_{\boldsymbol{x}} \{w(\boldsymbol{x}) g(\|\boldsymbol{y}-\boldsymbol{x}\|^2)\}} - \boldsymbol{y}$$

When Epanechnikov kernel is used as k, $g = -k'$ becomes 1 within the unit circle around **y**, and 0 otherwise.

$$k(t) \propto \begin{cases} 1-t, & t \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\boldsymbol{m}_g(\boldsymbol{y}) = \frac{\sum_{\boldsymbol{x} \in \text{unit circle}} \boldsymbol{x} w(\boldsymbol{x})}{\sum_{\boldsymbol{x} \in \text{unit circle}} w(\boldsymbol{x})} - \boldsymbol{y}$$

center of gravity within unit circle



Mean Shift Method

Recalling that $\nabla f_k(\boldsymbol{y}) = 2 f_g(\boldsymbol{y}) \boldsymbol{m}_g(\boldsymbol{y})$ (i.e. $\boldsymbol{m}_g(\boldsymbol{y}) = \dfrac{\nabla f_k(\boldsymbol{y})}{2 f_g(\boldsymbol{y})}$), we see

- Mean shift vector is toward the direction $f_k(\mathbf{y})$ becomes larger
- Mean shift vector is large when $f_g(\mathbf{y})$ is small (i.e. goal may be further), and small when $f_g(\mathbf{y})$ is large (i.e. goal may be closer)

Procedure of Mean Shift Method with Epanechnicov kernel:

1. Compute center of gravity of samples around current position
2. Move to the center of gravity (mean shifting)
3. Return to 1. unless the mean shift vector becomes too small

Approximating the Histogram Similarity

Coming back to the tracking problem, consider approximating the Bhattacharyya coefficient $\rho(\mathbf{p}(\mathbf{y}), \mathbf{q})$ such that it fits the Mean Shift framework

- Let $\mathbf{y}_0$ denote the initial candidate position
- Consider 1st order Taylor expansion to $\rho(\mathbf{p}(\mathbf{y}), \mathbf{q})$ with respect to $\mathbf{p}(\mathbf{y})$ around $\mathbf{p}(\mathbf{y}_0)$

$$
\begin{aligned}
\rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}) &= \sum_u \sqrt{p_u(\boldsymbol{y})}\sqrt{q_u} \\
&\approx \sum_u \sqrt{q_u} \left\{ \sqrt{p_u(\boldsymbol{y}_0)} + \frac{1}{2} p_u(\boldsymbol{y}_0)^{-1/2} \left(p_u(\boldsymbol{y}) - p_u(\boldsymbol{y}_0) \right) \right\} \\
&= \sum_u \sqrt{q_u} \left(\sqrt{p_u(\boldsymbol{y}_0)} + \frac{1}{2} p_u(\boldsymbol{y}) \frac{1}{\sqrt{p_u(\boldsymbol{y}_0)}} - \frac{1}{2}\sqrt{p_u(\boldsymbol{y}_0)} \right) \\
&= \frac{1}{2} \sum_u \sqrt{q_u}\sqrt{p_u(\boldsymbol{y}_0)} + \frac{1}{2} \sum_u p_u(\boldsymbol{y}) \frac{\sqrt{q_u}}{\sqrt{p_u(\boldsymbol{y}_0)}}
\end{aligned}
$$

Since the 1st term does not depend on **y**, what we should maximize is the 2nd term:

$$\sum_u p_u(\boldsymbol{y}) \frac{\sqrt{q_u}}{\sqrt{p_u(\boldsymbol{y}_0)}}$$

Recalling that $p_u(\boldsymbol{y}) \propto \sum_{\boldsymbol{x}\in S(u)} k(\|\boldsymbol{y}-\boldsymbol{x}\|^2)$, this comes down to maximization of

$$\sum_{u\in\text{all bins}} \; \sum_{\boldsymbol{x}\in\text{all pixels belonging to } u} k(\|\boldsymbol{y}-\boldsymbol{x}\|^2) \frac{\sqrt{q_u}}{\sqrt{p_u(\boldsymbol{y}_0)}}$$

$$= \sum_{\boldsymbol{x}\in\text{all pixels}} \sqrt{\frac{q_{b(\boldsymbol{x})}}{p_{b(\boldsymbol{x})}(\boldsymbol{y}_0)}} k(\|\boldsymbol{y}-\boldsymbol{x}\|^2)$$

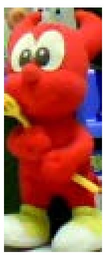
where b(**x**) is the bin to which **x** belongs.

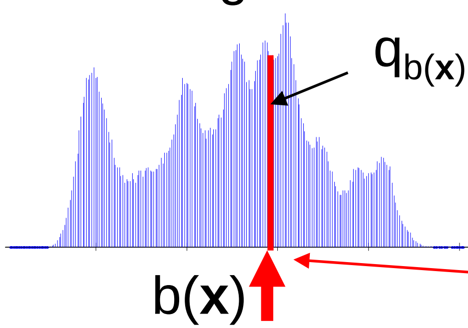
So, what we should maximize is:

$$\sum_{\boldsymbol{x} \in \text{all pixels}} \sqrt{\frac{q_{b(\boldsymbol{x})}}{p_{b(\boldsymbol{x})}(\boldsymbol{y}_0)}} k(\|\boldsymbol{y}-\boldsymbol{x}\|^2) = \sum_{\boldsymbol{x} \in \text{all pixels}} w(\boldsymbol{x}) k(\|\boldsymbol{y}-\boldsymbol{x}\|^2)$$

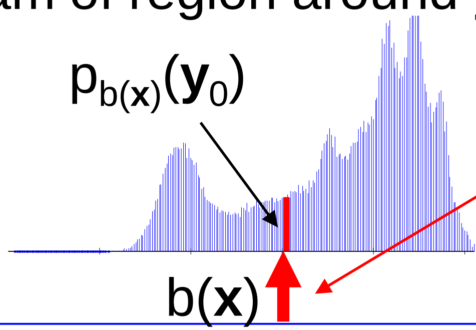
KDE with kernel k() where the pixels are treated as samples

model histogram





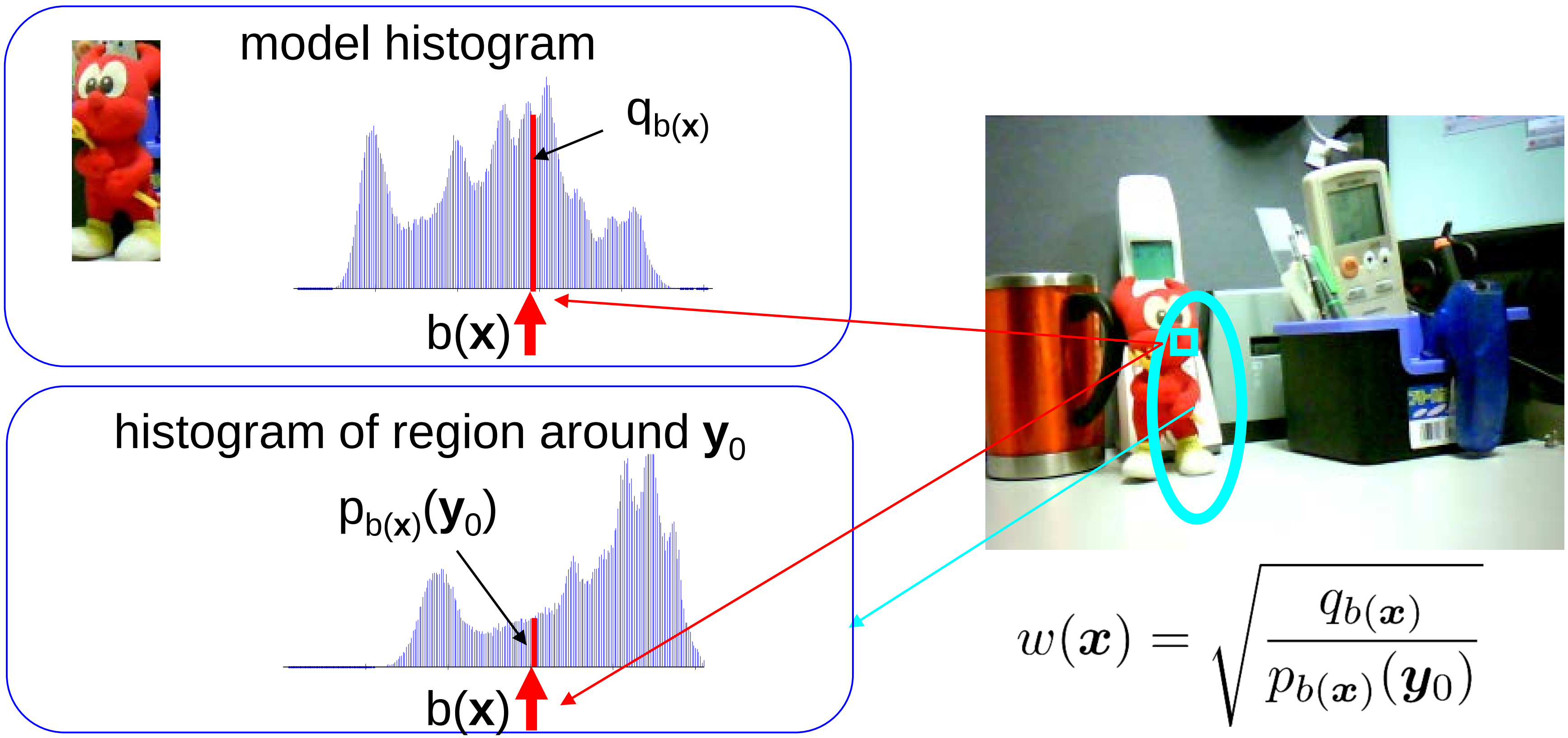
histogram of region around $\mathbf{y}_0$



For each pixel **x** (in the kernel range), find b(**x**) to look up q and p, and compute w(x).

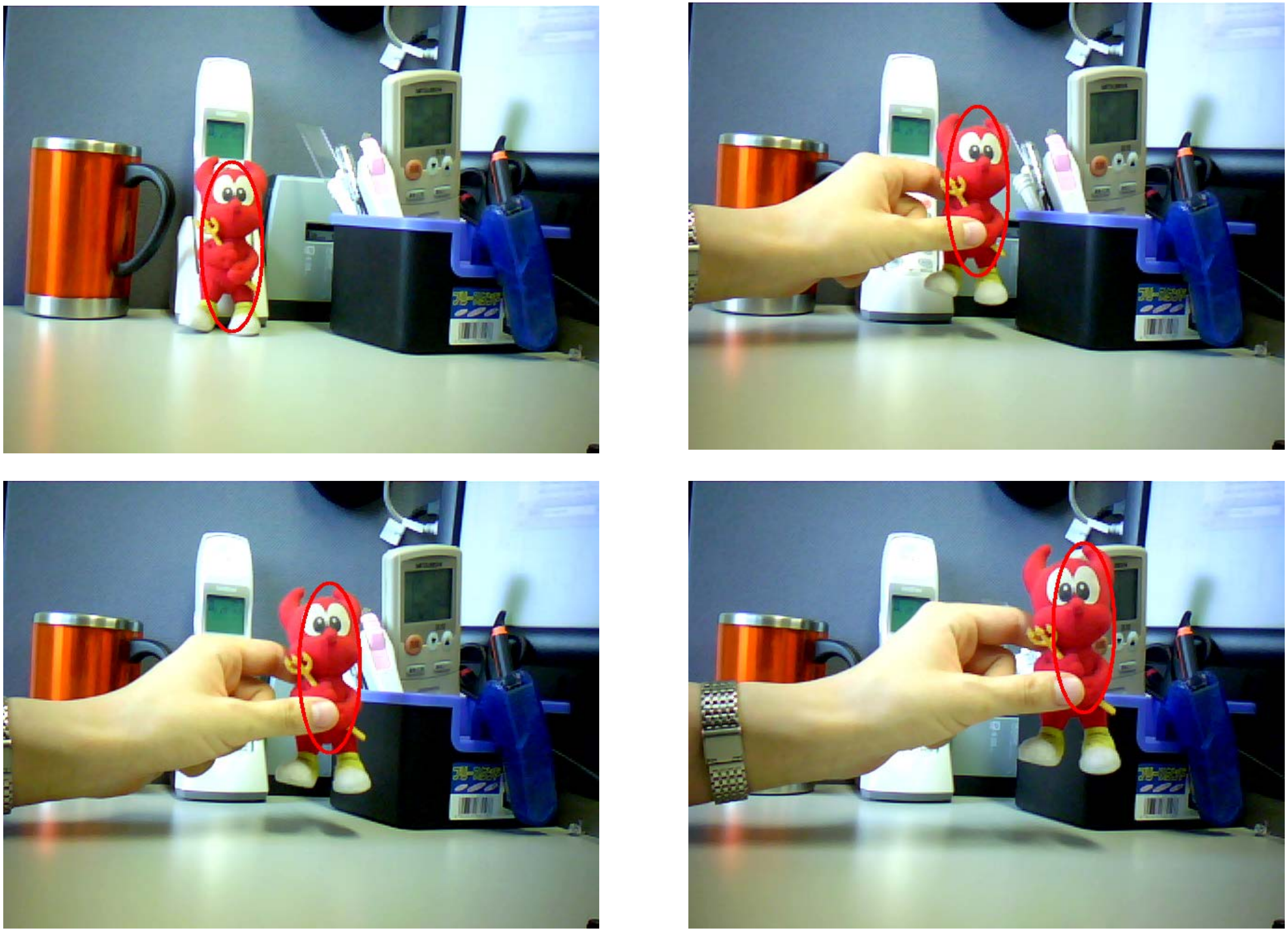
Mean Shift Tracking [Comaniciu 2003]

1. Compute the weighted histogram p($\mathbf{y}_0$) around $\mathbf{y}_0$
2. Move $\mathbf{y}_0$ to the center of gravity of w($\mathbf{x}$), by finding b($\mathbf{x}$) and looking up $\mathbf{q}$ an $\mathbf{p}$ for each pixel $\mathbf{x}$ around $\mathbf{y}_0$
3. Return to 1. unless the move becomes too small



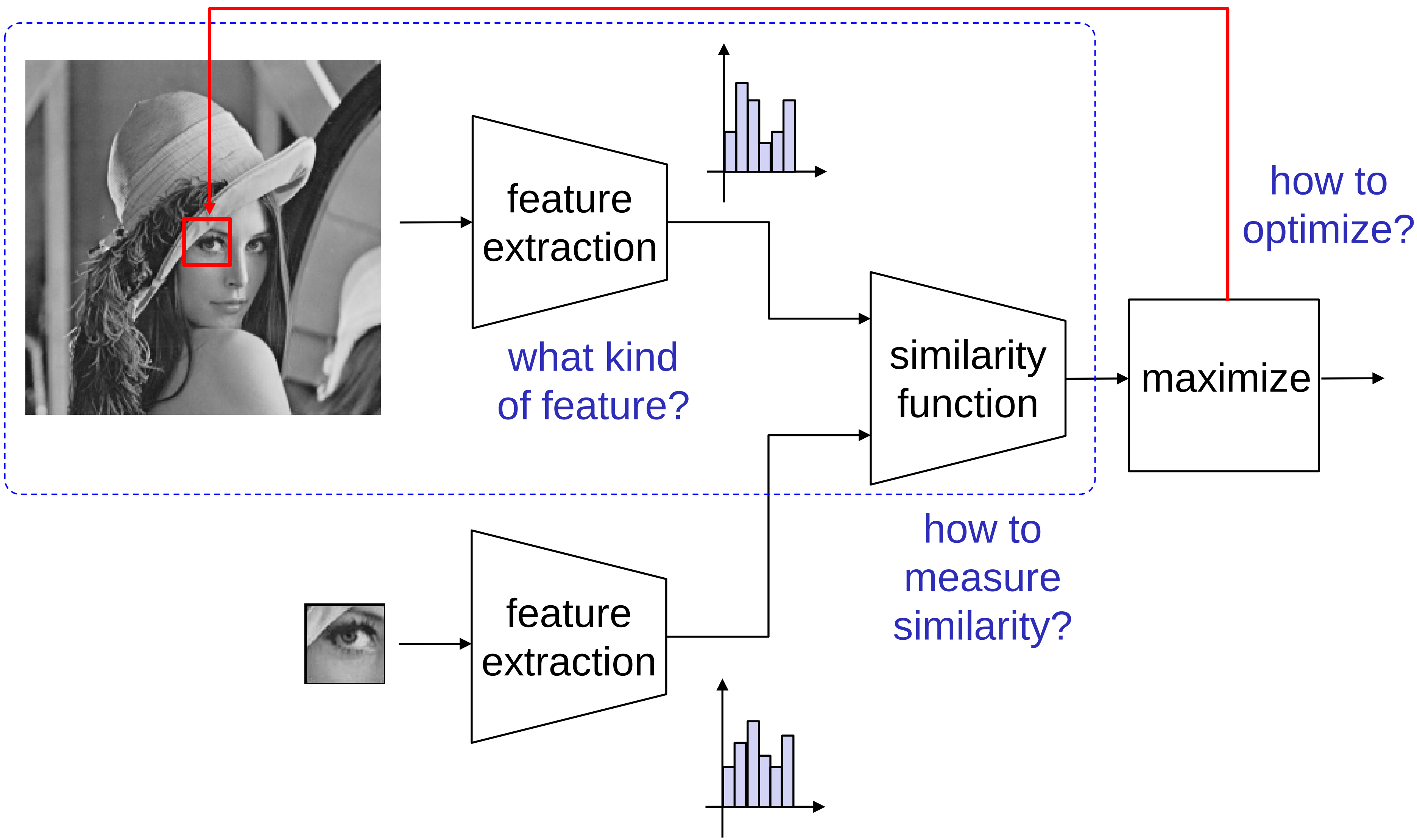
$$w(\boldsymbol{x}) = \sqrt{\frac{q_{b(\boldsymbol{x})}}{p_{b(\boldsymbol{x})}(\boldsymbol{y}_0)}}$$

04_mean_shift_color_histogram.py (and color_histogram_utils.py)

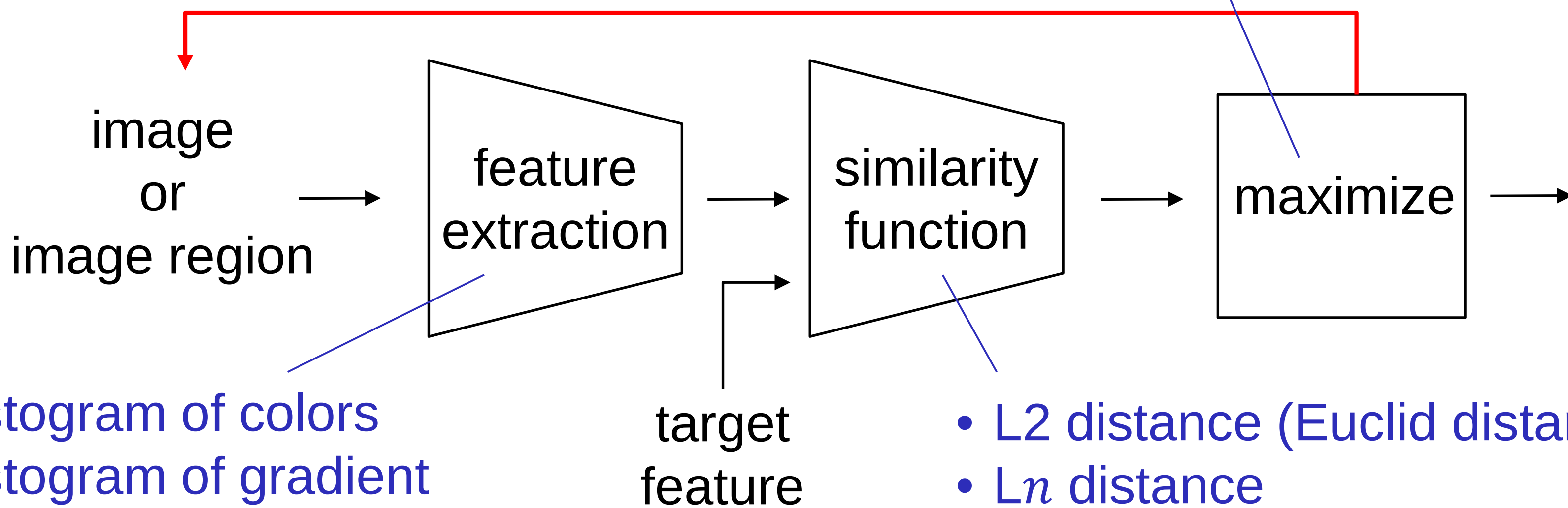


Further Examples

General Framework



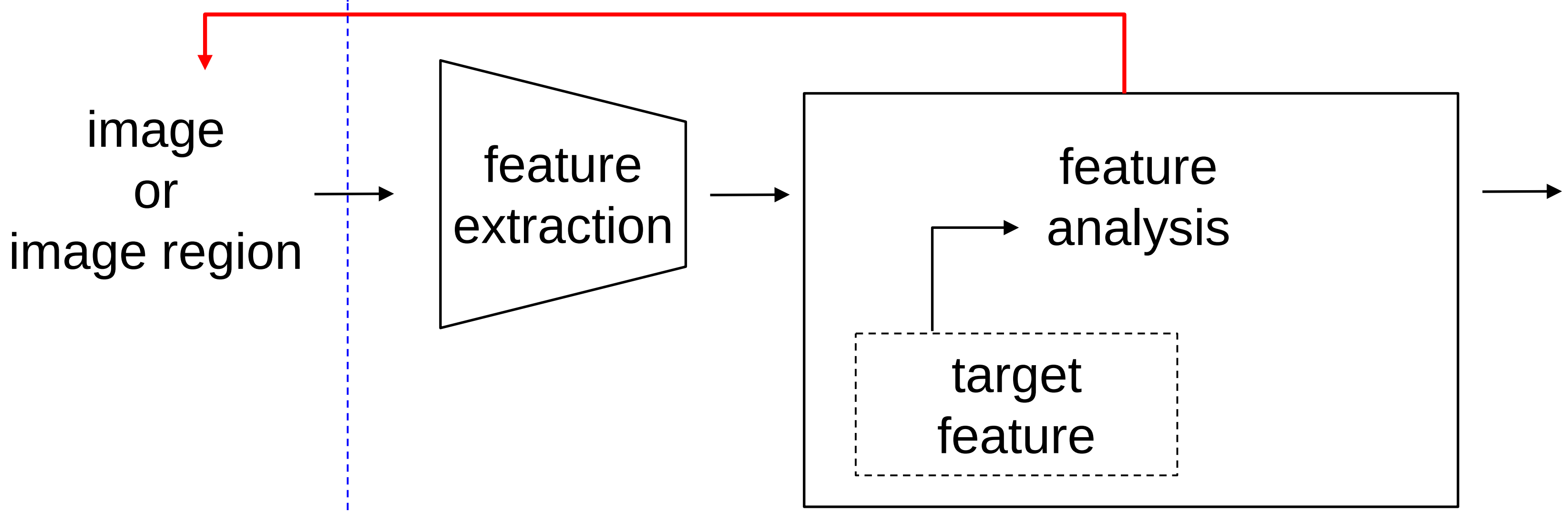
- brute-force search over entire image
- brute-force search in a local search area
- brute-force search over feature points
- gradient-based search
- …



- histogram of colors
- histogram of gradient orientations
- responses from linear filters
- comparisons of pixel pairs
- …

- L2 distance (Euclid distance)
- Ln distance
- Correlation
- Bhattacharyya coefficient
- Kullback-Leibler divergence
- …

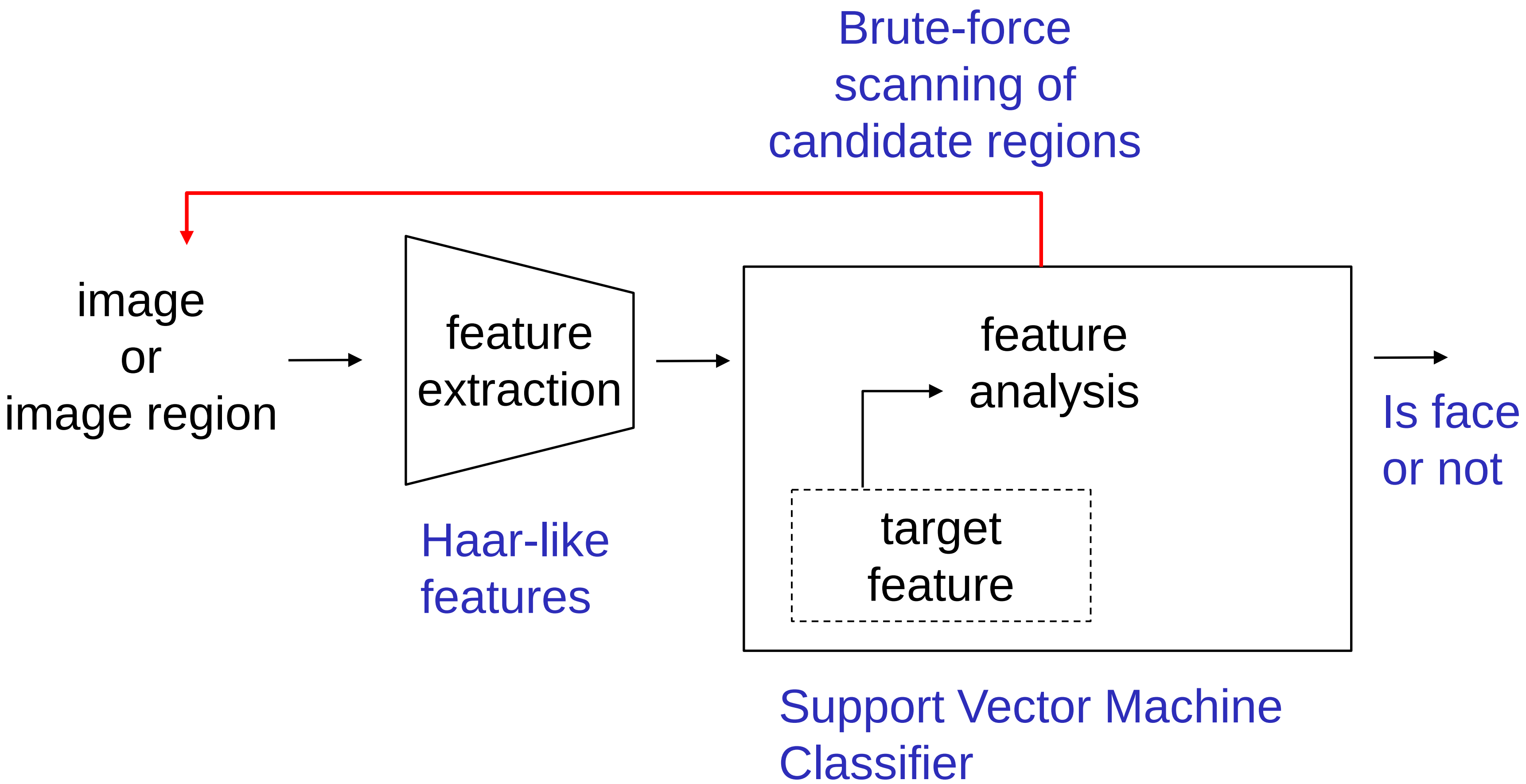
Further Generalization



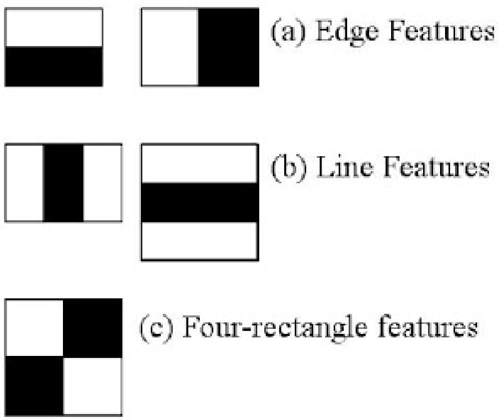
- object category (recognition task)
- is target object or not (detection task)
- incremental position update (tracking task)
- candidate region proposal
- …

"Extraction + analysis" can possibly be learnt in an end-to-end manner, particularly when deep learning methods are used

Face Detection Example



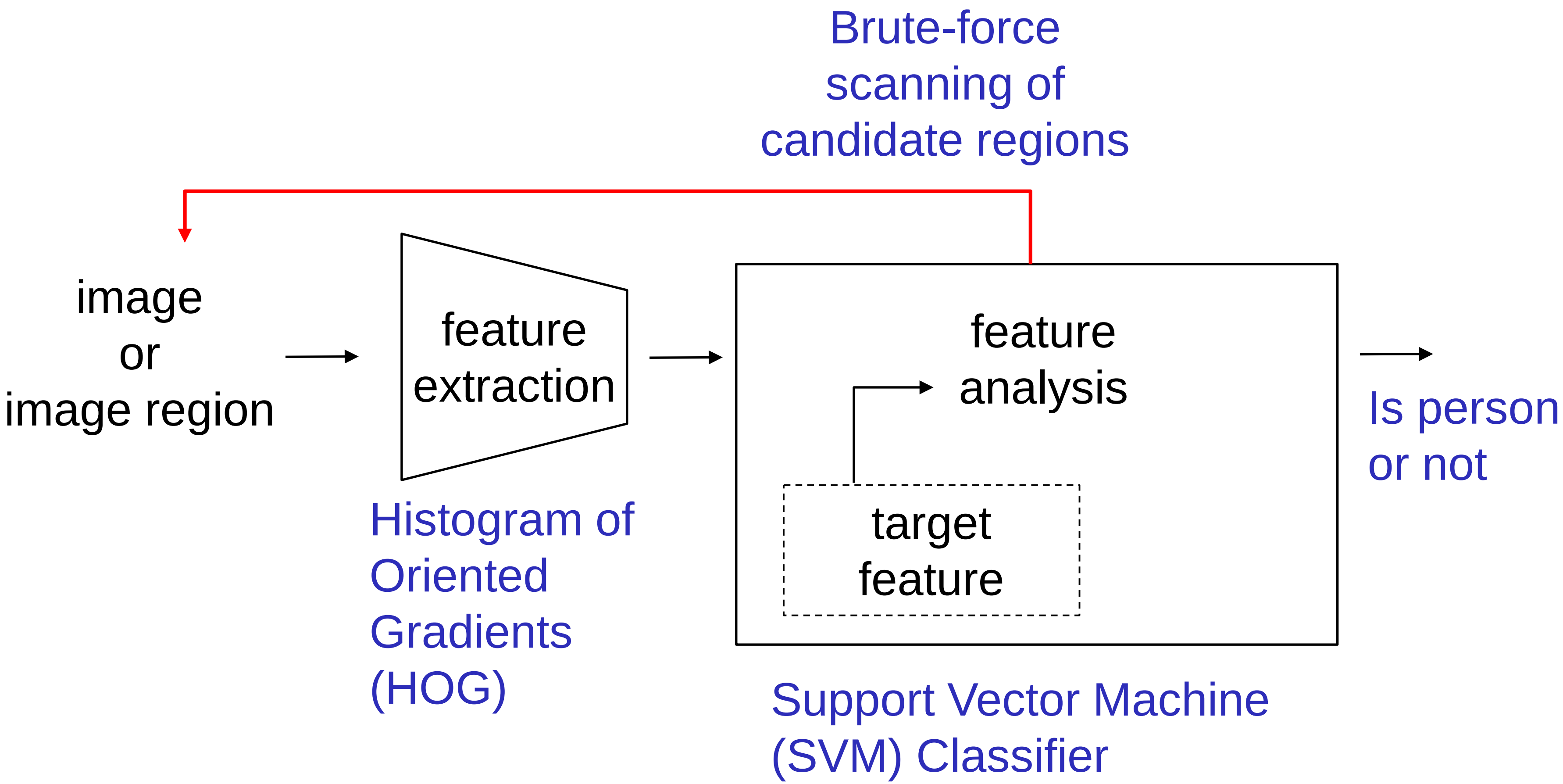
Haar-like features



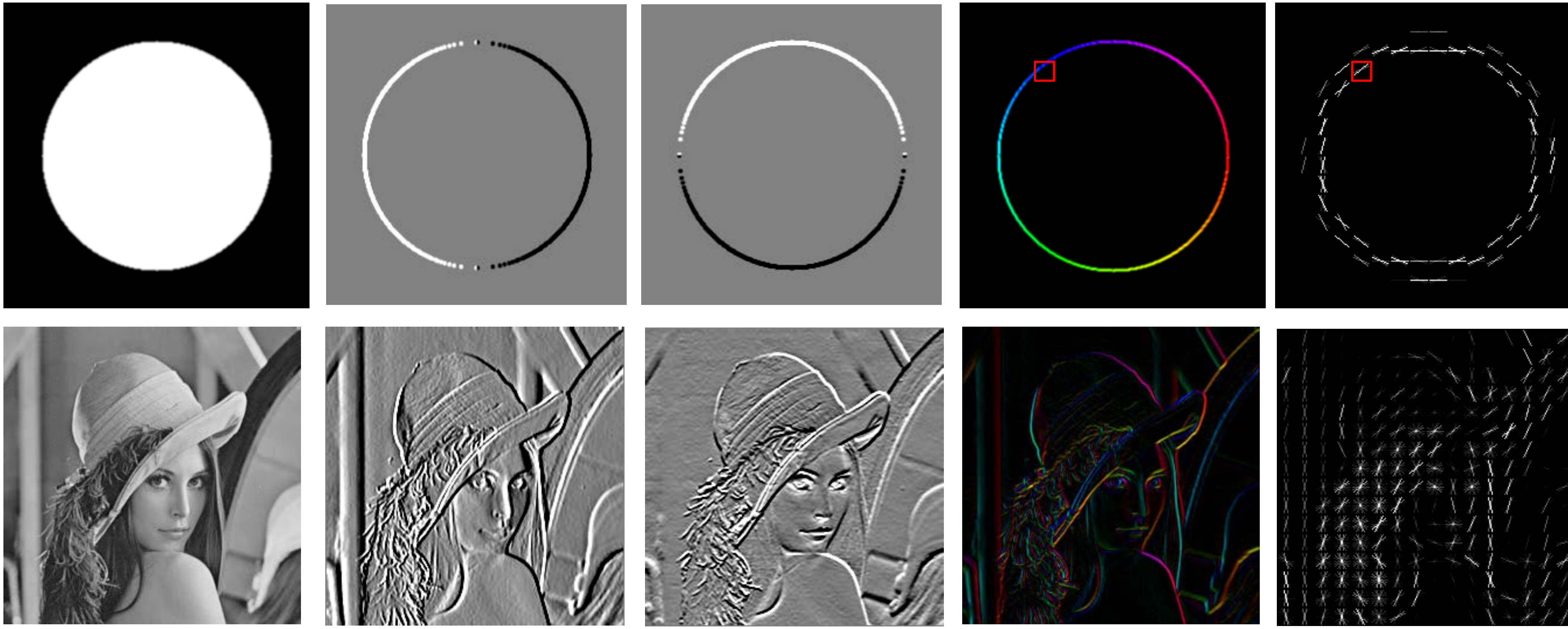
https://docs.opencv.org/master/d2/d64/tutorial_table_of_content_objdetect.html

- Convolving an image with such box-shaped kernels at many image positions can be accelerated through a technique called the “integral image”
- Feature value obtained from a single kernel has little information, but aggregating many of them works well for face detection [Viola 2001]

Person Detection Example



Histogram of Spatial Gradient Orientations



I | I_x | I_y

Hue $\propto \operatorname{atan}\frac{I_x}{I_y} + \pi$

Saturation = 1

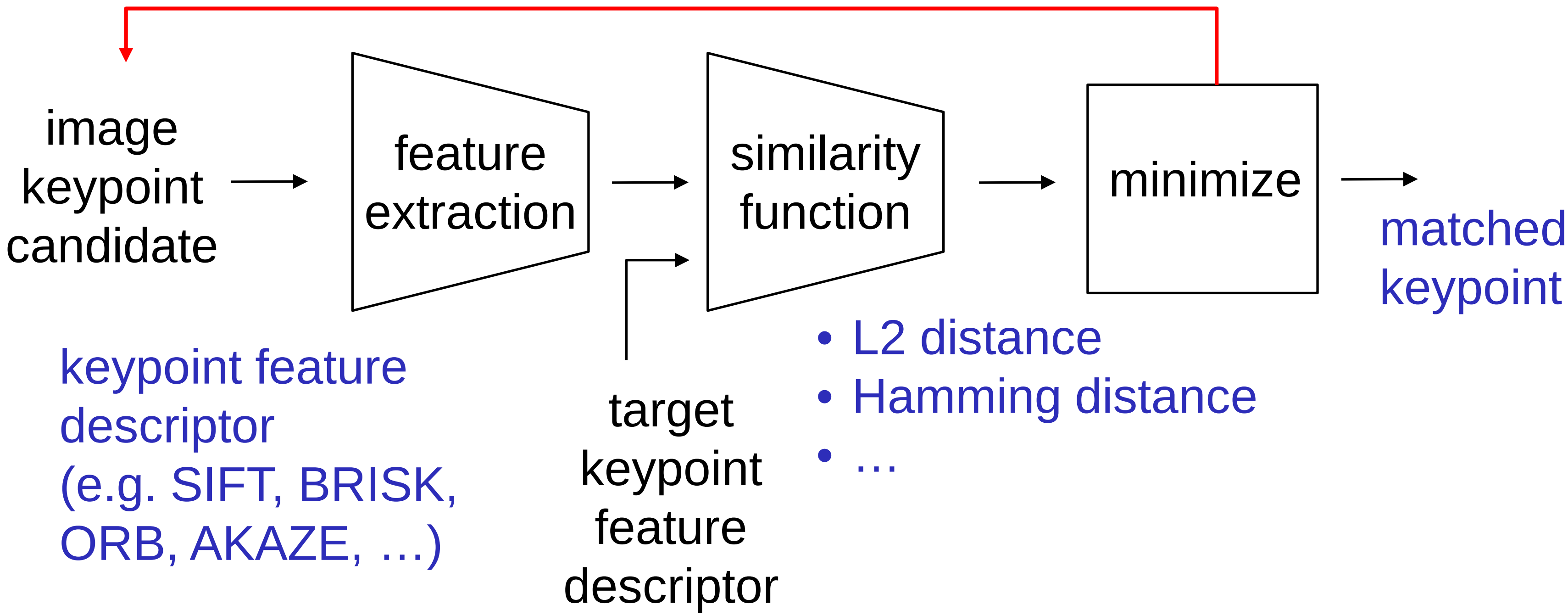
Value $\propto \sqrt{I_x^2 + I_y^2}$

gradient orientation histograms visualized

Often referred to as HOG (Histogram of Oriented Gradients) when combined with local block-wise normalization [Datal 2005]

Local Image Features around Keypoints

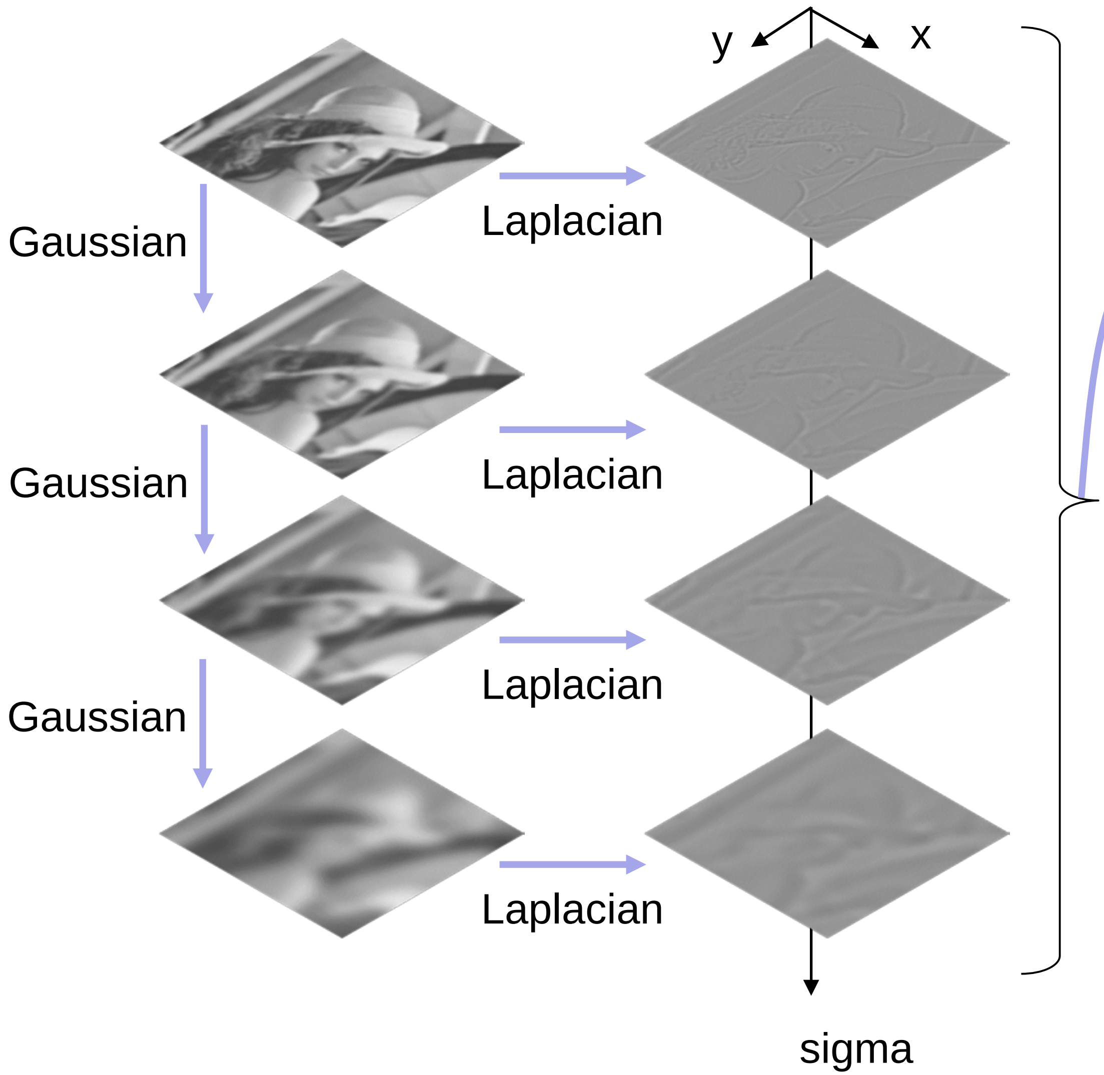
- Brute-force scanning of keypoints
- Nearest-neighbor search in feature space
- …



keypoint feature descriptor (e.g. SIFT, BRISK, ORB, AKAZE, …)

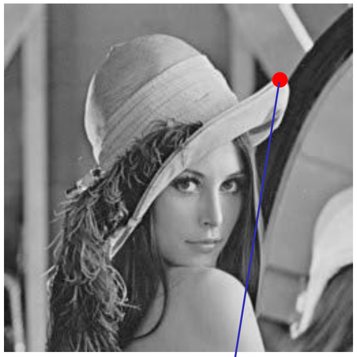
- L2 distance
- Hamming distance
- …

SIFT Keypoint [Lowe 2004]



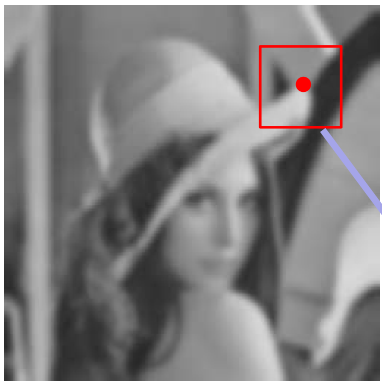
Keypoint Detection

- Locate peaks in (x, y, sigma) space
- Eliminate edge responses (by analyzing 2x2 Hessian matrix)

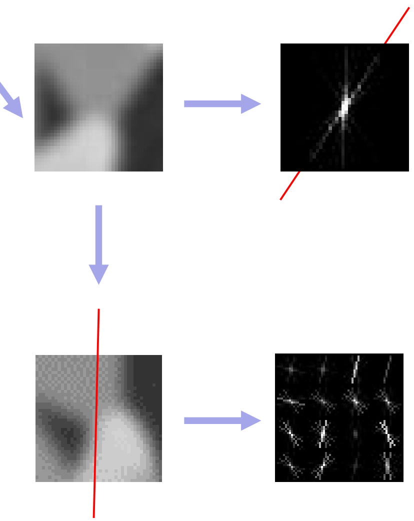


keypoint at (e.g.)
(x = 203, y = 53, sigma = 10)

SIFT Local Image Descriptor [Lowe 2004]



Given a keypoint at (*x, y, sigma*):



Orientation Assignment:

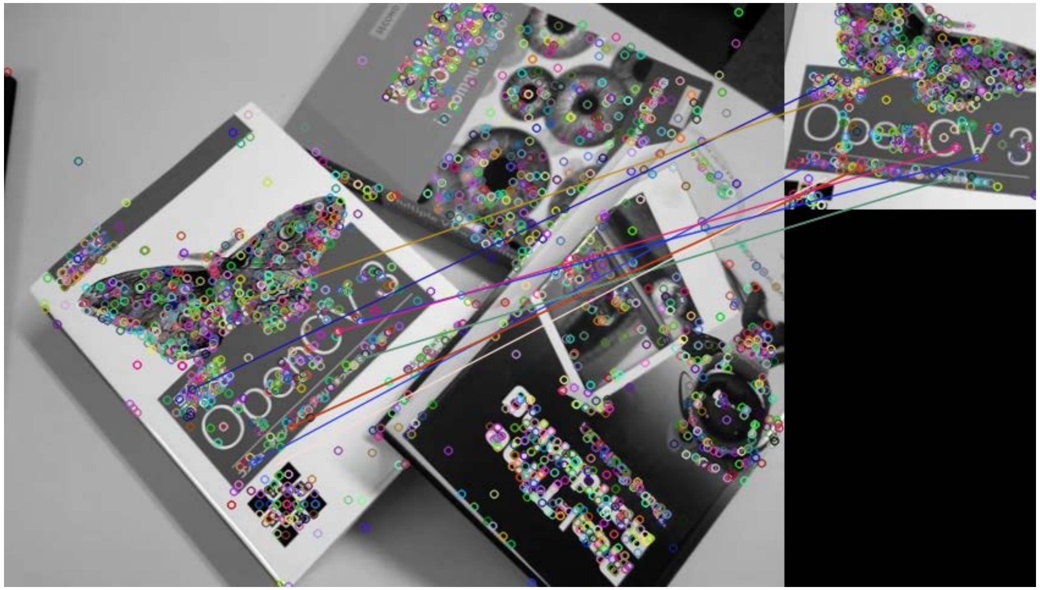
- Look at the patch around the point with the size determined by *sigma*
- Find the dominant orientation by finding peak in gradient histogram

Feature Descriptor Computation:

- The patch is aligned to the dominant orientation
- Compute the gradient orientation histograms with 8 orientation bins in 4x4 cells, resulting in 128-D feature vector that is invariant to scale and rotation

- This procedure (and the resulting feature vector itself) is called Scale Invariant Feature Transformation (SIFT)
- Useful in point-to-point matching of images

Plane Tracking Example by Keypoint Matching



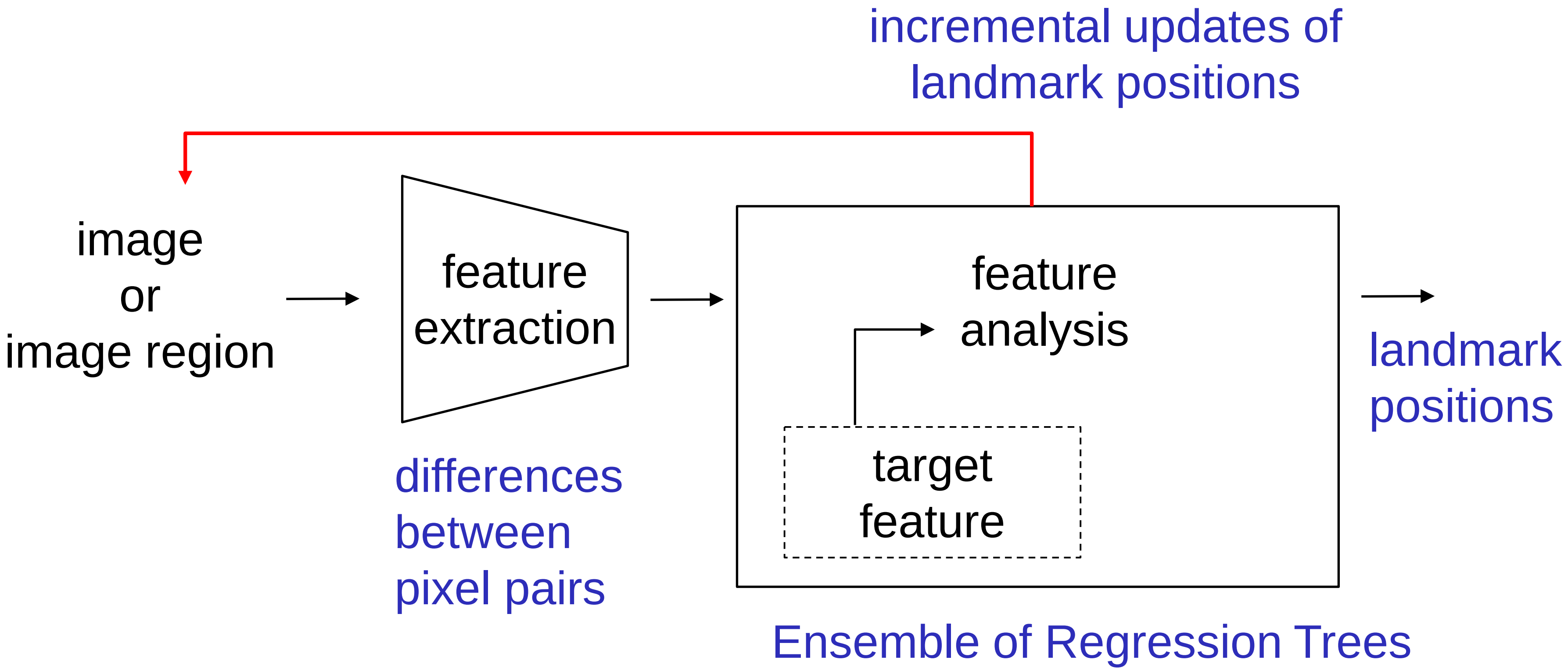
SIFT keypoint matching

Estimation of homography transformation using the matched keypoint pairs

- More recent approaches (e.g. BRISK, ORB, AKAZE, …) generate binary valued feature vector through comparisons of pairs of pixel values

04_keypoint_match.py

Face Landmark Alignment Example [Kazemi 2014]



Preparation for Face Landmark Alignment

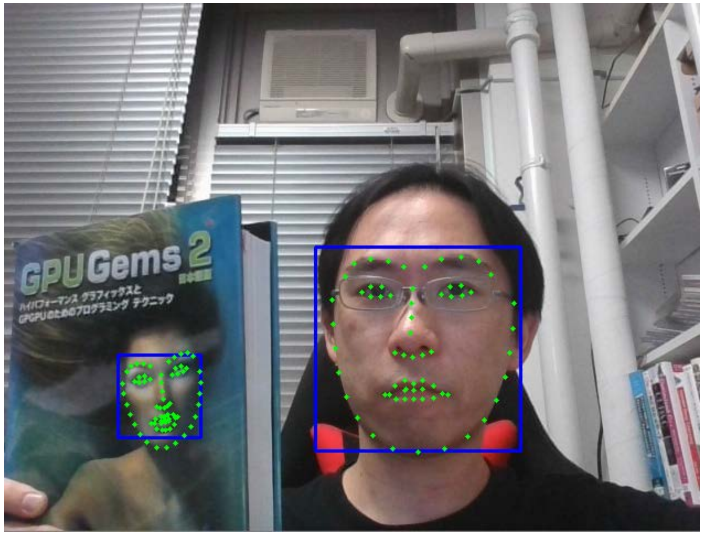
- Before trying the sample code for face alignment, run the following command in the command prompt opened by `C:¥ic2020¥ic_python_env.bat` (This may take 10 minutes or so)

```
pip install dlib
```

- Then download `shape_predictor_68_face_landmarks.dat.bz2` from http://dlib.net/files/, uncompress it, and put it in the `sample` directory (This consumes 100 MB or so)

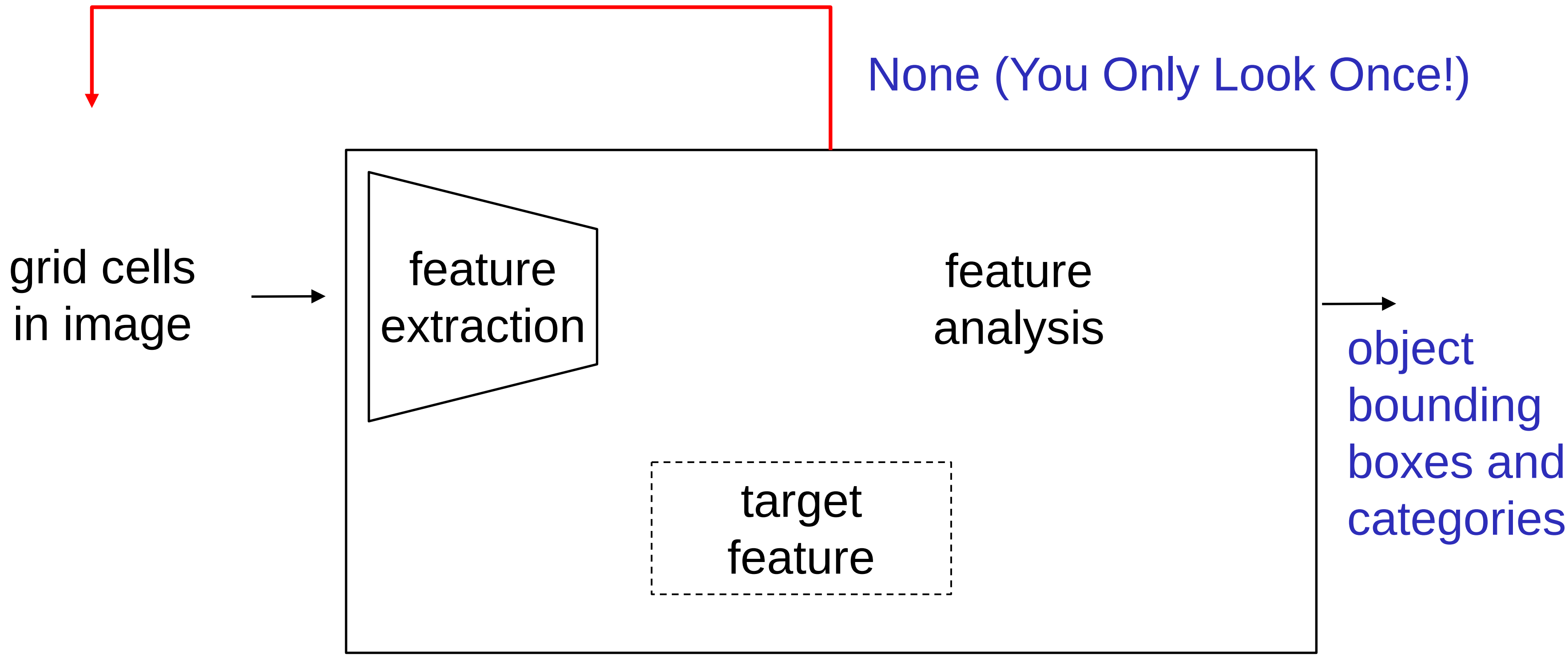
- Now you are ready to run:

  04_face_landmarks.py



Object Detection/Recognition Example

YOLO [Redmon 2016]



- Bounding box (with confidence) generation network
- Object class probability estimation network

Preparation for YOLO v3

- Run the following command in the command prompt opened by `C:¥ic2020¥ic_python_env.bat` (which requires 350 MB disk space)

```
pip install pillow
pip install tensorflow==1.13.1
pip install keras==2.2.4
```

- Download https://github.com/qqwweee/keras-yolo3 in zip file and extract somewhere (say, `C:¥ic2020¥keras-yolo3`). Or, you can use git

```
git clone https://github.com/qqwweee/keras-yolo3
```

- Download https://pjreddie.com/media/files/yolov3.weights into `C:¥ic2020¥keras-yolo3`. File size is 250 MB and downloading may take long time.

- Run the following command in `C:¥ic2020¥keras-yolo3` (which generate another 250-MB file)

```
python convert.py yolov3.cfg yolov3.weights model_data¥yolo.h5
```

Running YOLO v3

```
import sys
from yolo import YOLO
from yolo import detect_video

if __name__ == '__main__':
    cap_src = 'vtest.avi'
    if len(sys.argv) == 2:
        if sys.argv[1].isdecimal():
            cap_src = int(sys.argv[1])
        else:
            cap_src = sys.argv[1]

    detect_video(YOLO(), cap_src)
```

- Put the above program into a file `yolo_cam.py` in `C:¥ic2020¥keras-yolo3`, and run the following

```
python yolo_cam.py path_to_a_video_file_or_camera_number
```

References

- D. Comaniciu, V. Ramesh and P. Meer: Kernel-Based Object Tracking, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.25, no.5, 2003.
- D. Comaniciu and P. Meer: Mean Shift: A Robust Approach Toward Feature Space Analysis, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.25, no.5, 2003.
- K. Fukunaga and L. D. Hostetler: The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition, IEEE Trans. on Information Theory, vol.IT-21, no.1, 1975.
- N. Dalal and B. Triggs: Histograms of Oriented Gradients for Human Detection, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2005), 2005.
- P. Viola and M. J. Jones: Rapid Object Detection Using a Boosted Cascade of Simple Features. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.
- D. G. Lowe: Distinctive Image Features from Scale-Invariant Keypoints, International J. of Computer Vision, vol.60, no.2, 2004.
- V. Kazemi and J. Sullivan: One Millisecond Face Alignment with an Ensemble of Regression Trees, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2014), 2014.
- J. Redmon, S. Divvala, R. Girshick and A. Farhadi: You Only Look Once: Unified, Real-Time Object Detection, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2016), 2016.