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/

Sample codes for this week

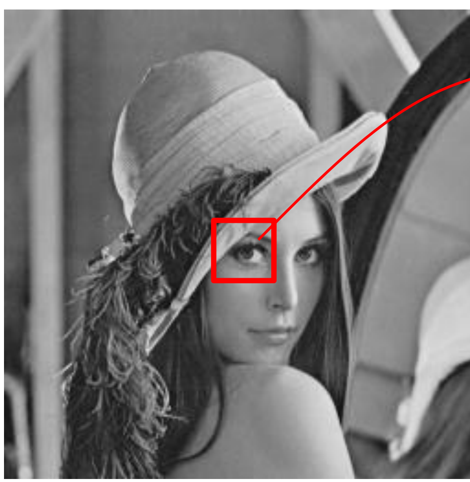
- Open https://github.com/shingo-kagami/ic.git
- Click the green button “Code” and click “Download Zip”
- Copy the files whose names start from ic04*** to C:¥ic2022¥sample

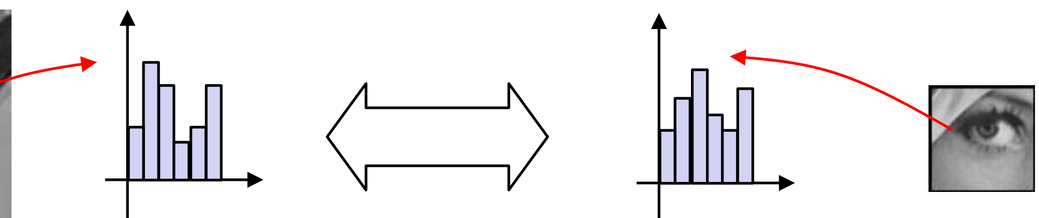
If you are a Git user, you may simply run:

```
cd C:¥ic2022¥sample
git pull
```

Feature-based Methods vs Direct Methods

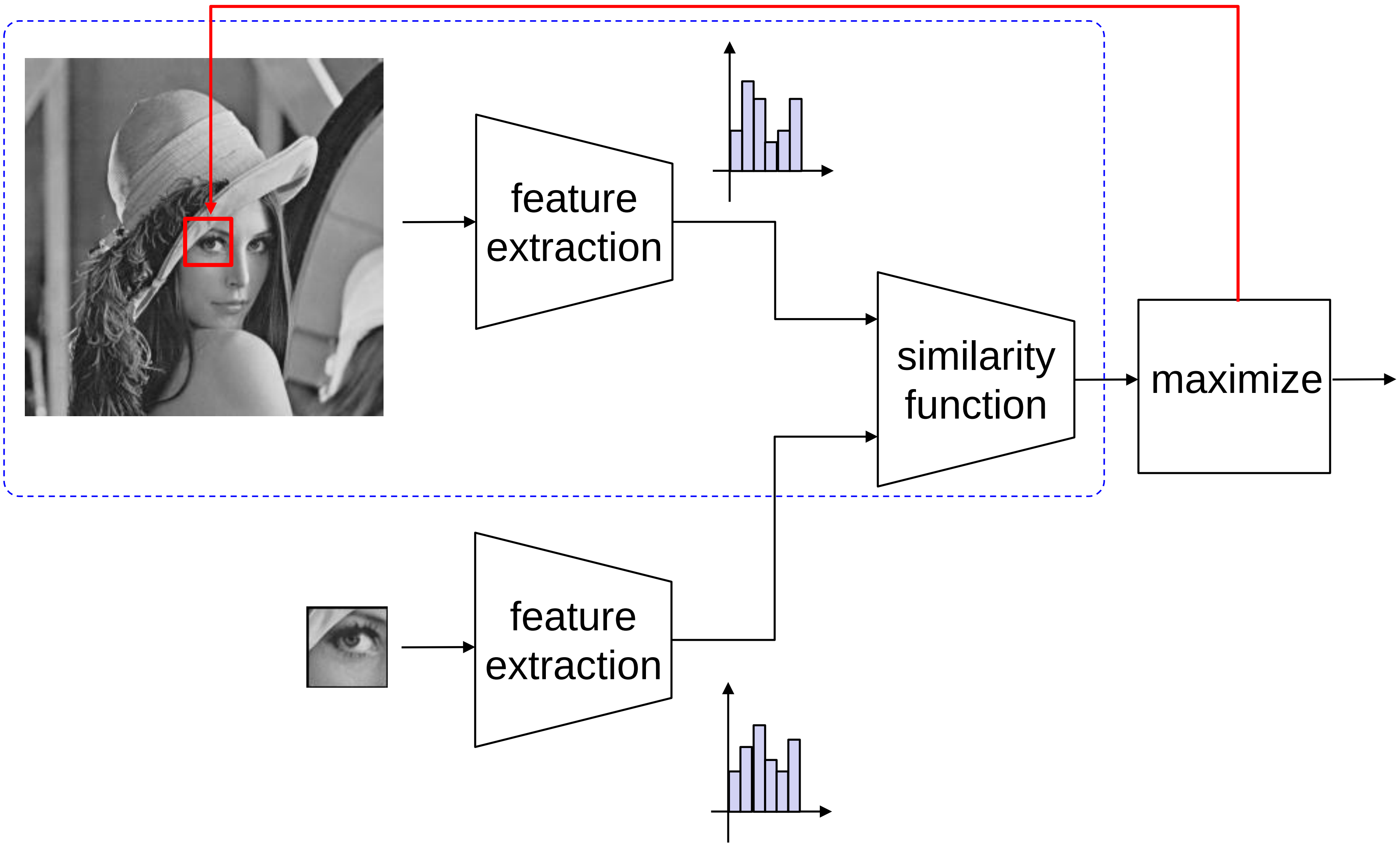
direct comparison of
pixel values



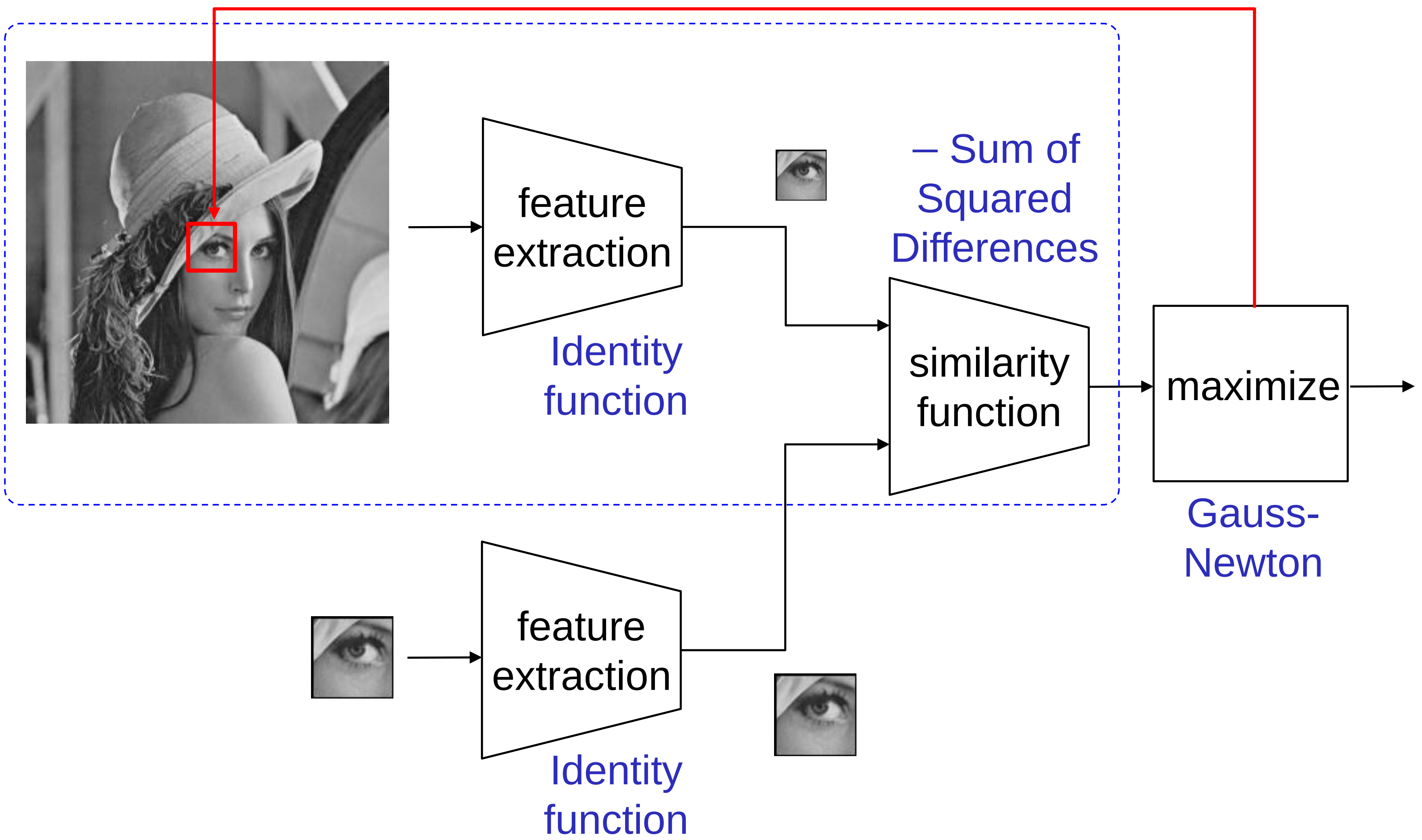


comparison of feature values
computed from images (e.g.
histograms, edge positions, …)

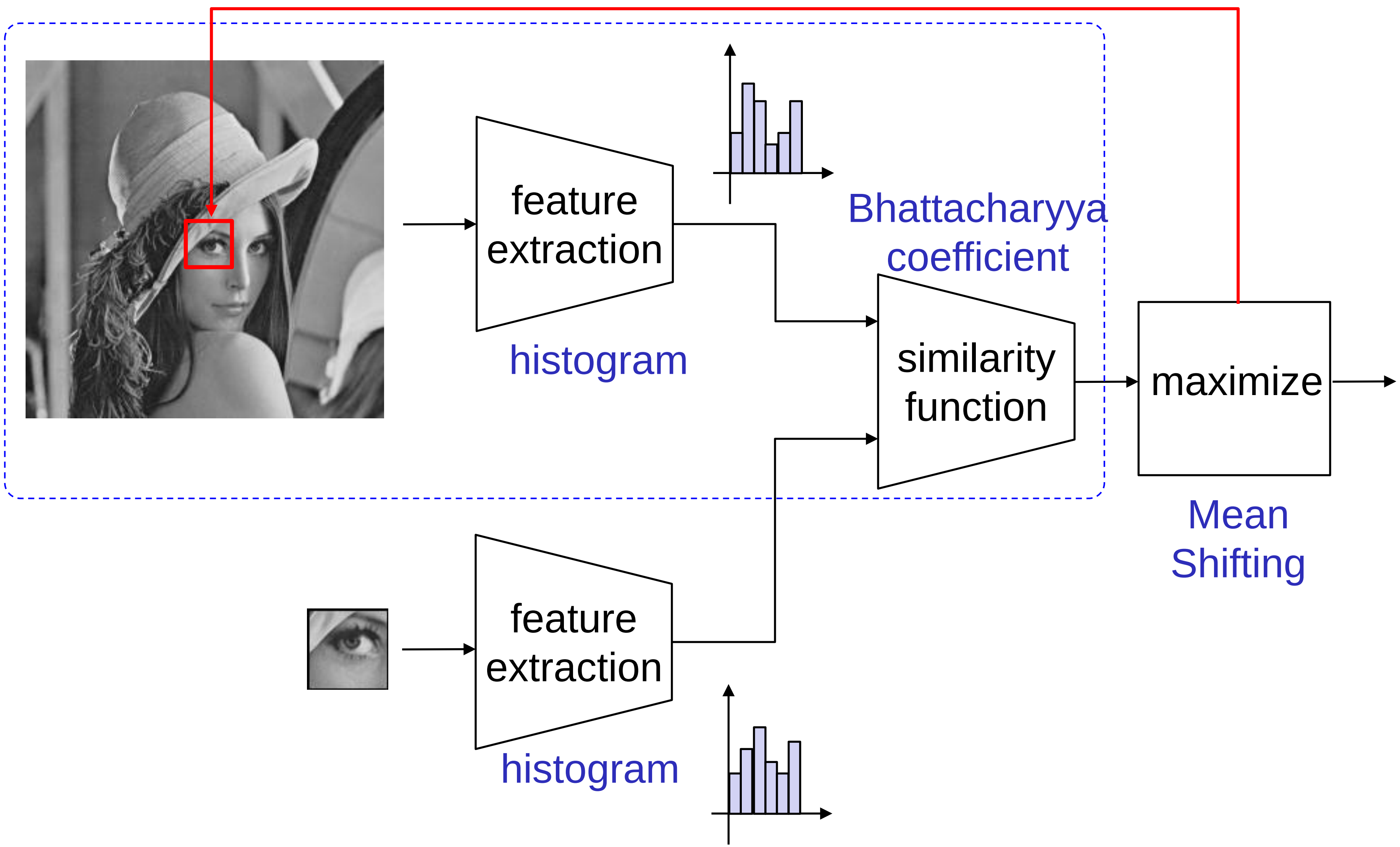
General Framework



Lucas-Kanade Tracking (last week)



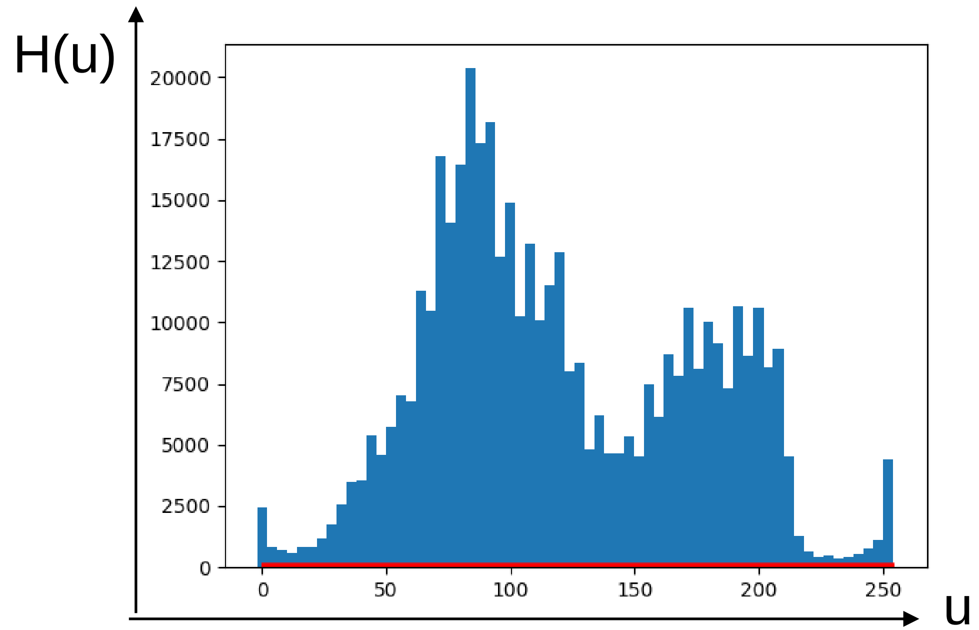
Histogram-based Mean Shift Tracking



Agenda

- Color Histogram
- Similarity Measure of Histogram
- Mean Shift Tracking based on Color Histogram
- Further Examples of Tracking and Detection

Histogram of Pixel Values (Gray Level)



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

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

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

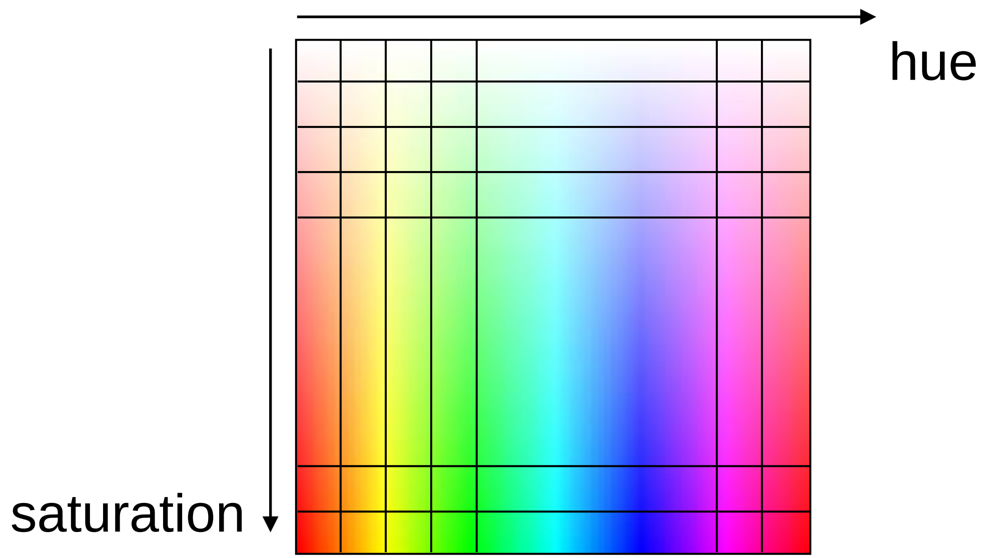
(normalized histogram)

Color Histograms

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

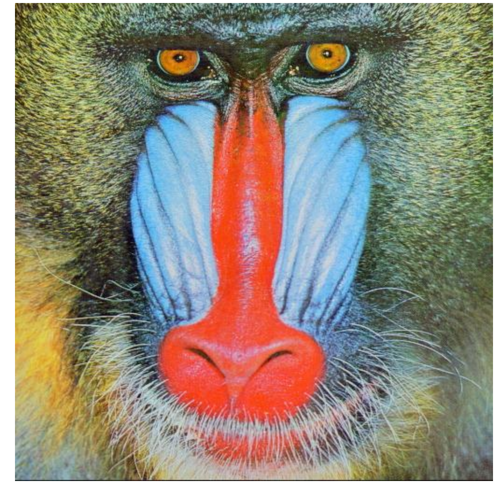
e.g.2) By splitting each of Hue and Saturation components into 64 bins (and ignoring Value component), we have histogram over 64 x 64 bins

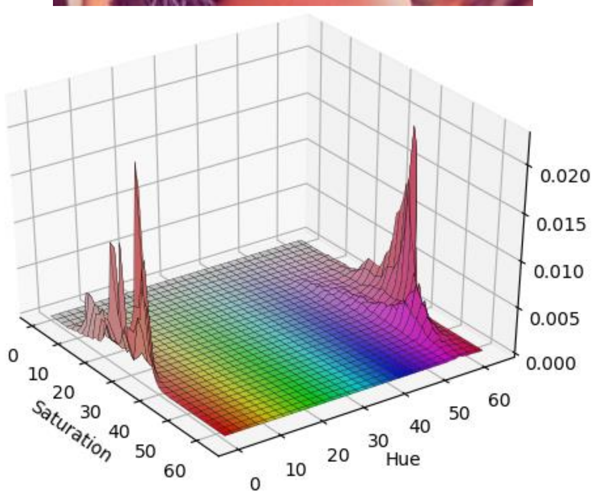
- Less affected by illumination change

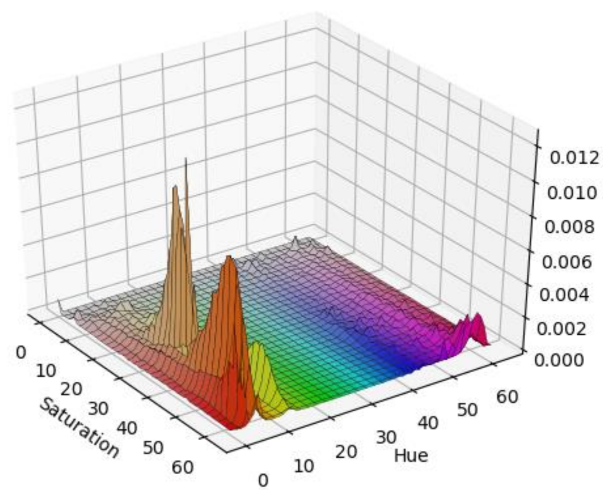


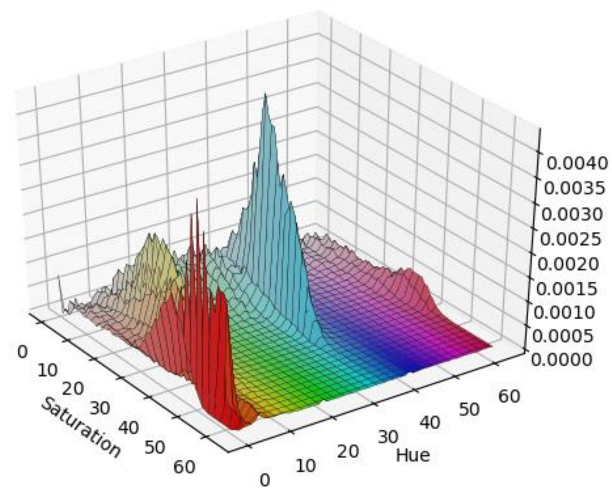
Hue-Saturation Histograms











Implementation of Hue-Saturation Histogram

ic04_color_histogram.py

```
hist = np.zeros((sbin, hbin))

for j in range(height):
    for i in range(width):
        weight = 1.0
        if is_weighted:
            x_norm = (i - cx) * w_normalizer
            y_norm = (j - cy) * h_normalizer
            rr = x_norm ** 2 + y_norm ** 2
            if rr <= 1:
                weight = 1 - rr
            else:
                weight = 0.0

        hue = image[j, i, 0]
        sat = image[j, i, 1]
        hist[int(sat * sbin / 256.0), int(hue * hbin / 180.0)] += weight

hist_sum = np.sum(hist)
hist /= hist_sum
```

center

2.0 / width

2.0 / height

Weighted Histogram

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

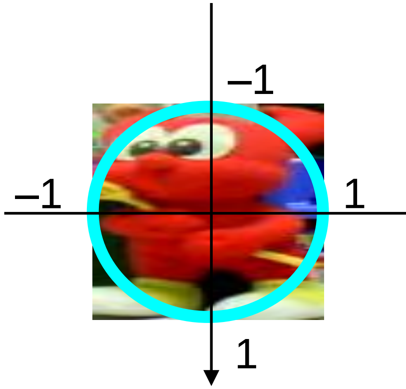
  → **weight the voting depending on pixel locations**

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 $\boldsymbol{x} = (x, y)$ are normalized such that it fits a unit circle

unit circle centered at $\boldsymbol{y}$

Kernel Function Examples

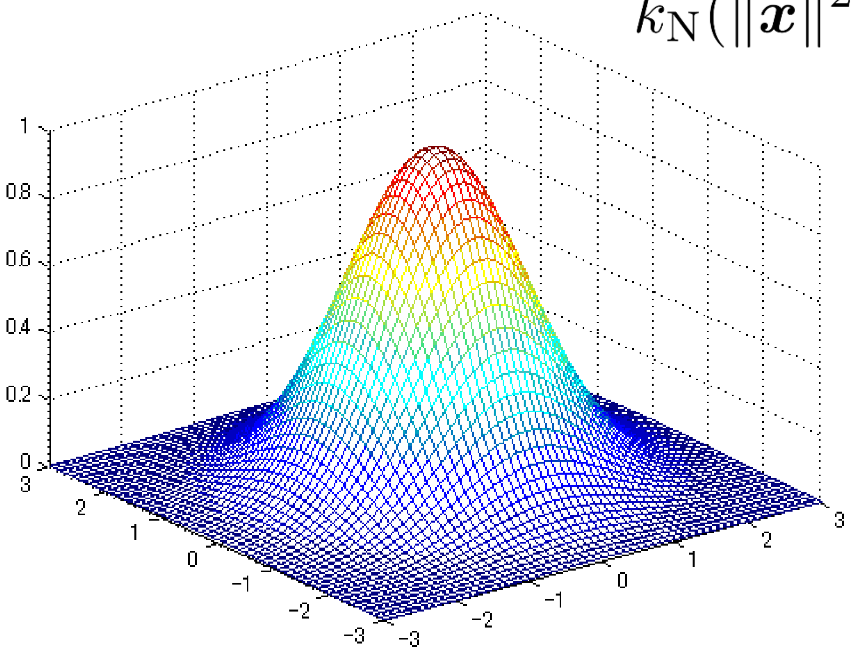
e.g.1) Gauss kernel

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

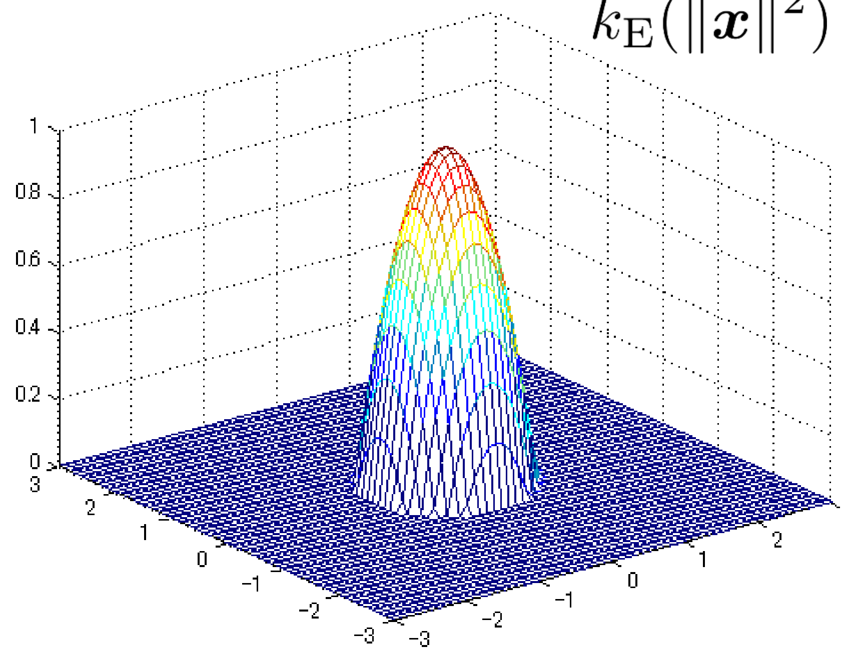
e.g.2) Epanechnikov kernel

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

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



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



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) $\boldsymbol{p}$ and $\boldsymbol{q}$

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

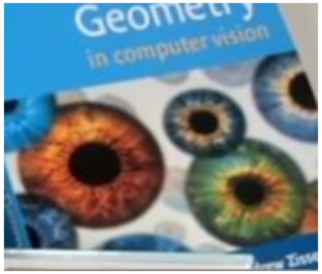
- Geometric interpretation:

  inner product of $\left(\sqrt{q_1}, \sqrt{q_2}, \cdots, \sqrt{q_m}\right)^T$ and $\left(\sqrt{p_1}, \sqrt{p_2}, \cdots, \sqrt{p_m}\right)^T$
- lies on the unit sphere surface

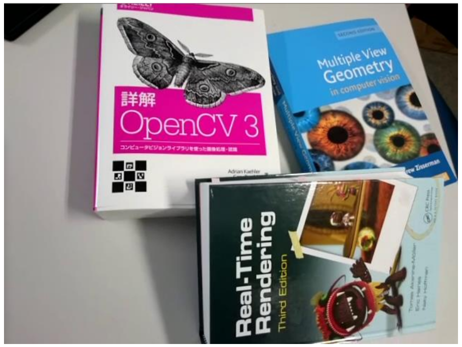
Why not other similarity measure?:

Simply because this is convenient for the mean shift method

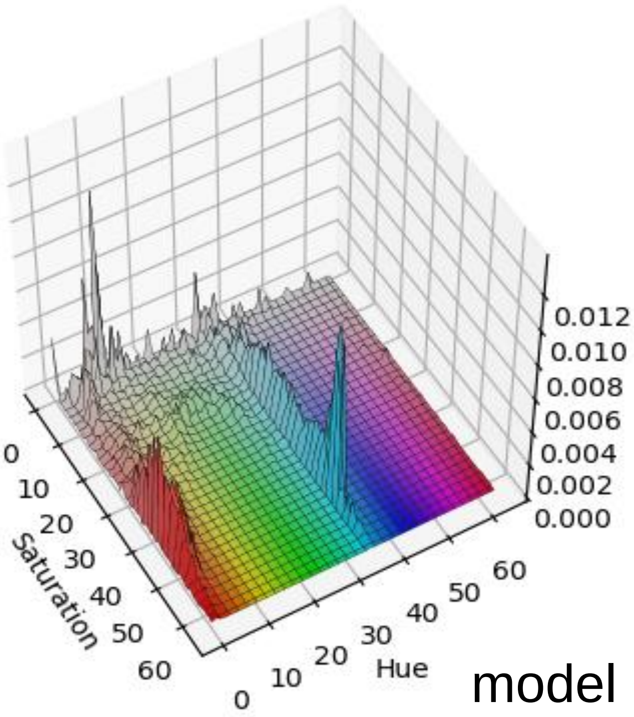
Similarity Map with Weighted Histogram



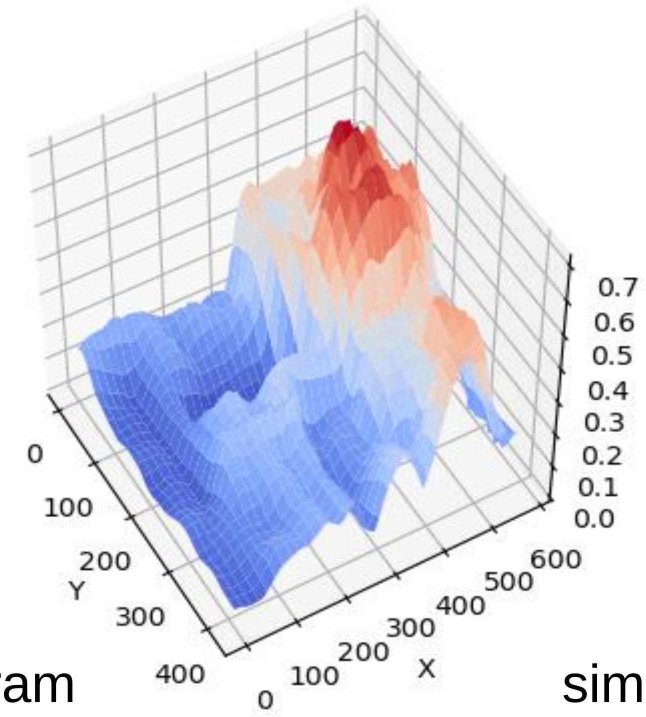
object model



current image



model histogram



similarity map

Implementation of Histogram Similarity Map

ic04_color_histogram_similarity.py

```
for j in range(0, sheight, step):
    for i in range(0, swidth, step):
        region = input_hsv[j:(j + theight), i:(i + twidth)]
        hist = generate_hue_sat_histogram(region, hist_size, is_weighted)
        bhattacharyya_coeff = np.sum(np.sqrt(hist_model * hist))
        sim_map[j:(j + step), i:(i + step)] = bhattacharyya_coeff

return sim_map
```

Similarity is evaluated at every `step` pixels
to reduce computational time

Approximating the Histogram Similarity (1/2)

We approximate the Bhattacharyya coefficient $\rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q})$ such that it fits the Mean Shift framework

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

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

Approximating the Histogram Similarity (2/2)

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

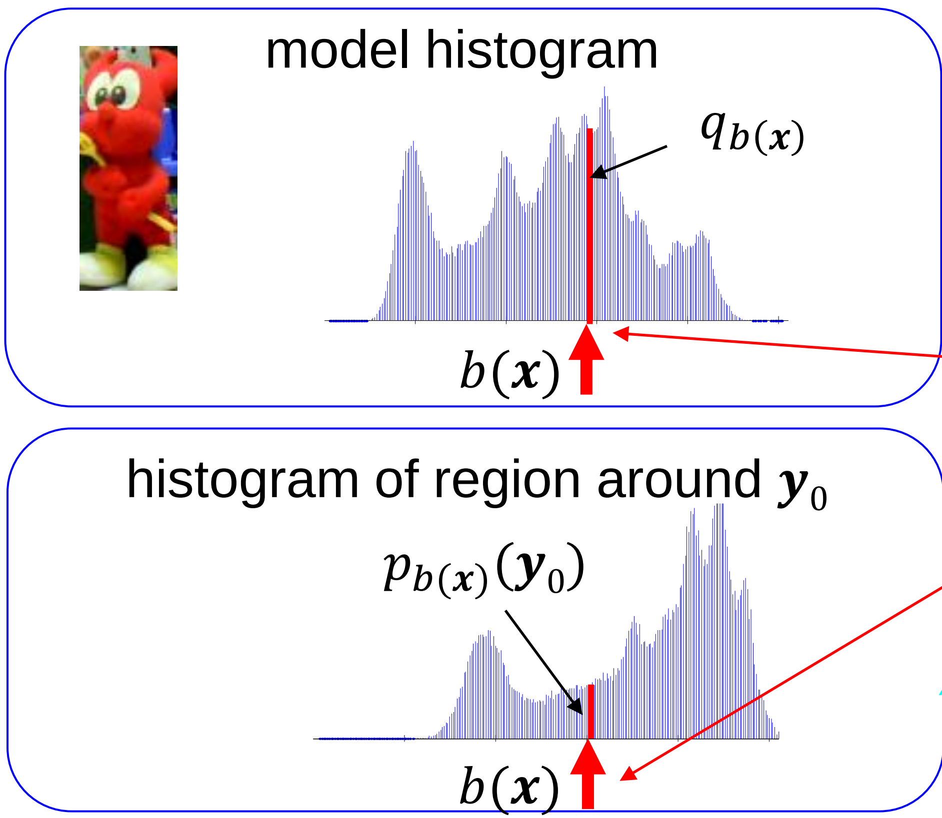
$$\sum_{u \in \text{all bins}} \sum_{\boldsymbol{x} \in \text{all pixels belonging to } u} k(\|\boldsymbol{y} - \boldsymbol{x}\|^2) \frac{\sqrt{q_u}}{\sqrt{p_u(\boldsymbol{y}_0)}}$$

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

where $b(\boldsymbol{x})$ is the bin to which $\boldsymbol{x}$ belongs

So, what we should maximize is:

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



For each pixel $\boldsymbol{x}$ (in the kernel range), find $b(\boldsymbol{x})$ to look up q and p, and compute $w(x)$

Mean Shift Method

Mean Shift is an efficient method to find a local maximum of a probability distribution represented by:

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

Its gradient at $\boldsymbol{y}$ is

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

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

$$\begin{aligned}
\nabla f_k(\boldsymbol{y}) &= 2 \sum_{\boldsymbol{x}} g(\|\boldsymbol{y} - \boldsymbol{x}\|^2)(\boldsymbol{x} - \boldsymbol{y}) w(\boldsymbol{x}) \\
&= 2 \left[\sum_{\boldsymbol{x}} \left\{ \boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2) \right\} - \boldsymbol{y} \overbrace{\boxed{\sum_{\boldsymbol{x}} \left\{ w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2) \right\}}}^{f_g(\boldsymbol{y})} \right] \\
&= 2 f_g(\boldsymbol{y}) \boxed{\left[\frac{\sum_{\boldsymbol{x}} \left\{ \boldsymbol{x} w(\boldsymbol{x}) g(\|\boldsymbol{y} - \boldsymbol{x}\|^2) \right\}}{f_g(\boldsymbol{y})} - \boldsymbol{y} \right]}
\end{aligned}$$

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

Interpretation of Mean Shift Vector

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

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

When k is Epanechnikov kernel, $g = -k'$ becomes 1 within the unit circle around $\boldsymbol{y}$, and 0 otherwise.

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

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

center of gravity within unit circle

Putting Them All Together

Mean Shift Tracking [Comaniciu et al. 2003]

1. Compute the weighted histogram $p(\boldsymbol{y}_0)$ around $\boldsymbol{y}_0$
2. Move $\boldsymbol{y}_0$ to the center of gravity of $w(\boldsymbol{x})$, by finding $b(\boldsymbol{x})$ and looking up the histograms $\boldsymbol{q}$ and $\boldsymbol{p}$ for each pixel $\boldsymbol{x}$ around $\boldsymbol{y}_0$

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

3. Return to 1. unless the move becomes too small

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 $f_k(\boldsymbol{y})$ becomes larger
- The length of mean shift vector is large when $f_g(\boldsymbol{y})$ is small (i.e. goal may be further), and small when $f_g(\boldsymbol{y})$ is large (i.e. goal may be closer)

Implementation of Mean Shift Tracking (1/2)

ic04_mean_shift_color_histogram.py

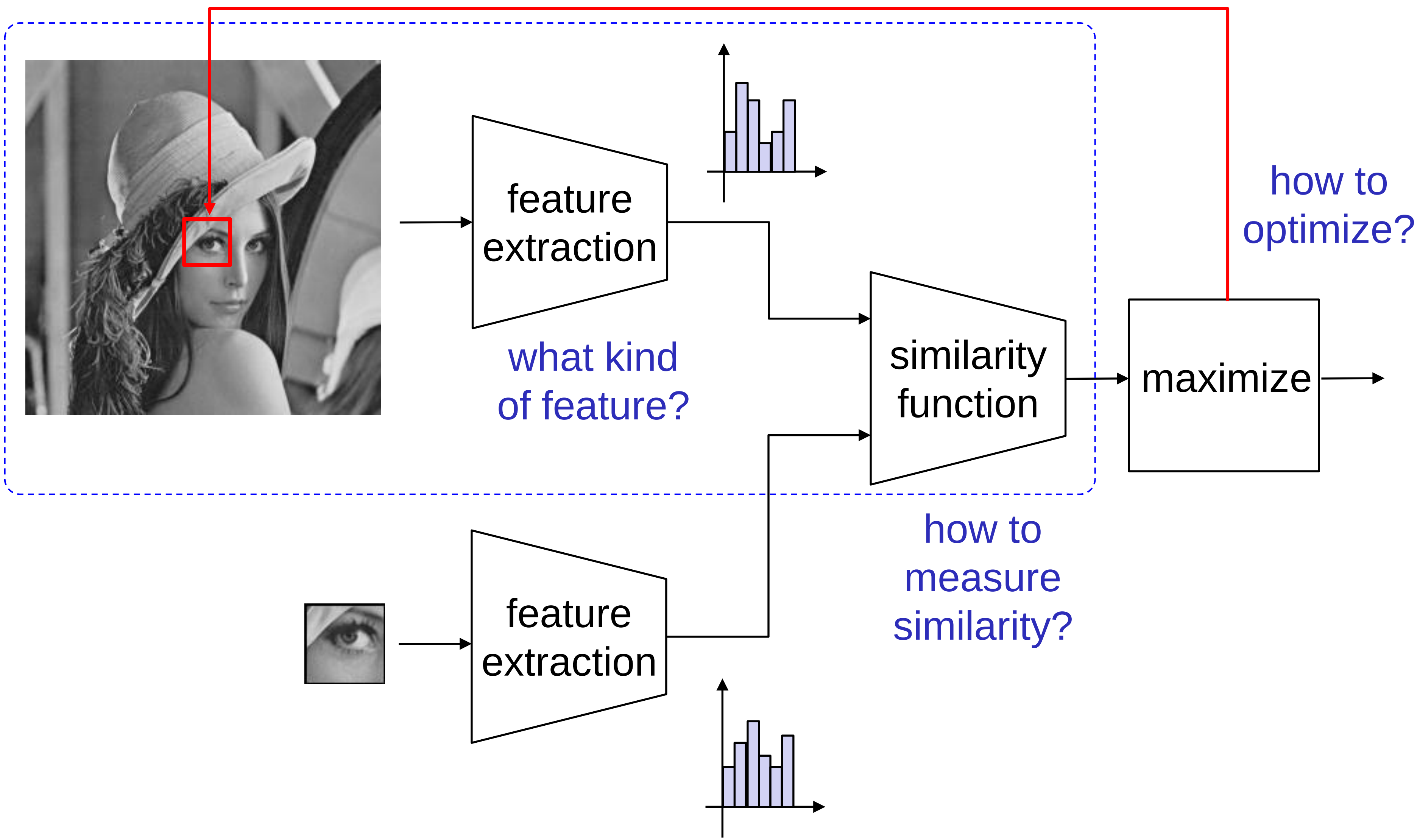
```
def mean_shift_vector(image_area, hist_model, hist_cand):
    # ...
    m0 = mx = my = 0.0
    for j in range(height):
        for i in range(width):
            hue = image_area[j, i, 0]
            sat = image_area[j, i, 1]
            at_sbin = int(sat * sbin / 256.0)
            at_hbin = int(hue * hbin / 180.0)
            q = hist_model[at_sbin, at_hbin]
            p = hist_cand[at_sbin, at_hbin]
            x_norm = (i - cx) * w_normalizer
            y_norm = (j - cy) * h_normalizer
            rr = x_norm ** 2 + y_norm ** 2
            if rr < 1 and p != 0.0:
                w = math.sqrt(q / p)
                m0 += w
                mx += w * i
                my += w * j
    if m0 == 0:
        return 0.0, 0.0
    else:
        return mx/m0 - cx, my/m0 - cy
```

Implementation of Mean Shift Tracking (2/2)

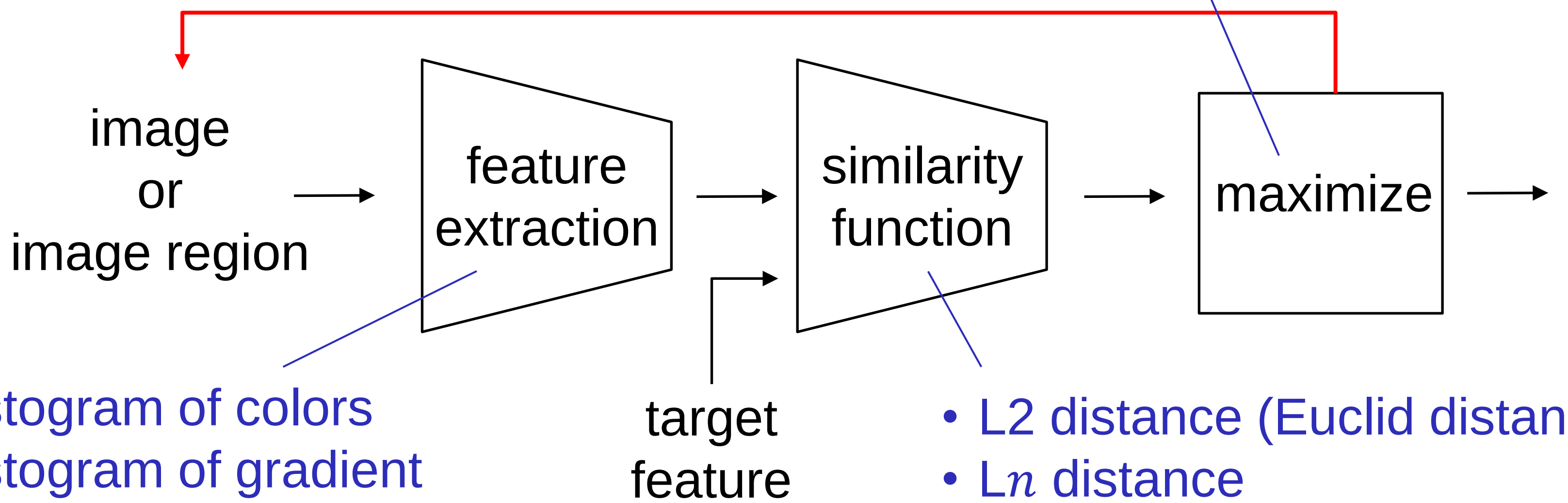
```
def track_by_mean_shift(image, tcenter, tsize, hist_model, max_iter=20):
    twidth = tsize[0]
    theight = tsize[1]
    hist_size = (hist_model.shape[1], hist_model.shape[0])

    for iter in range(max_iter):
        candidate = cv2.getRectSubPix(image, (twidth, theight), tcenter)
        hist_cand = generate_hue_sat_histogram(candidate, hist_size, True)
        msv_x, msv_y = mean_shift_vector(candidate, hist_model, hist_cand)
        tcenter = (tcenter[0] + msv_x, tcenter[1] + msv_y)
        dd = msv_x ** 2 + msv_y ** 2
        if dd < 1.0:
            break

    bhattacharyya_coeff = np.sum(np.sqrt(hist_model * hist_cand))
    return np.int16(tcenter), bhattacharyya_coeff
```

Further Examples

General Framework



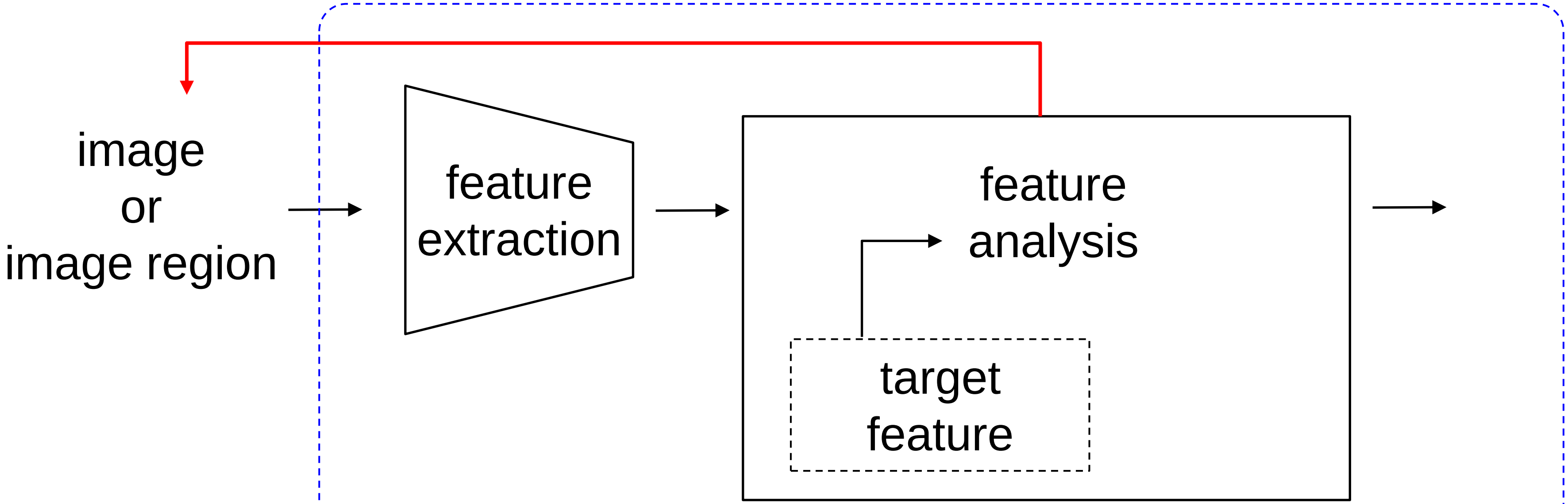
- brute-force search over entire image
- brute-force search in a local search area
- brute-force search over feature points
- gradient-based search
- …



- 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
- …

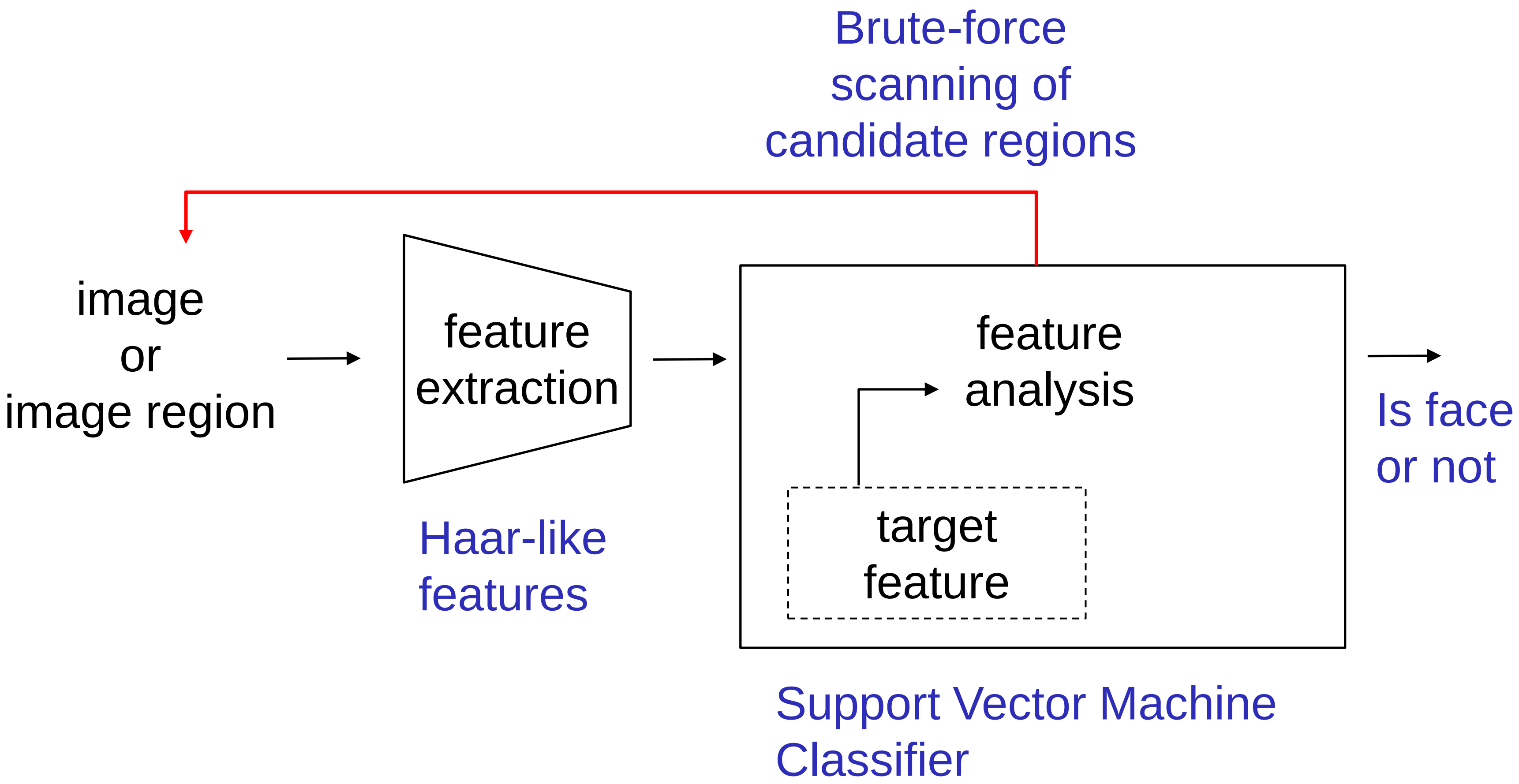
Further Generalization



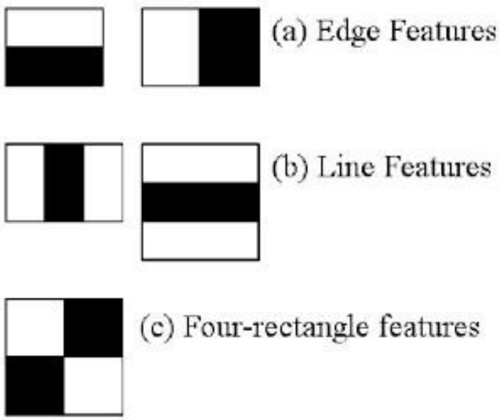
- object category (recognition task)
- is target object or not (detection task)
- incremental position update (tracking task)
- candidate region proposal
- …

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

Face Detection Example



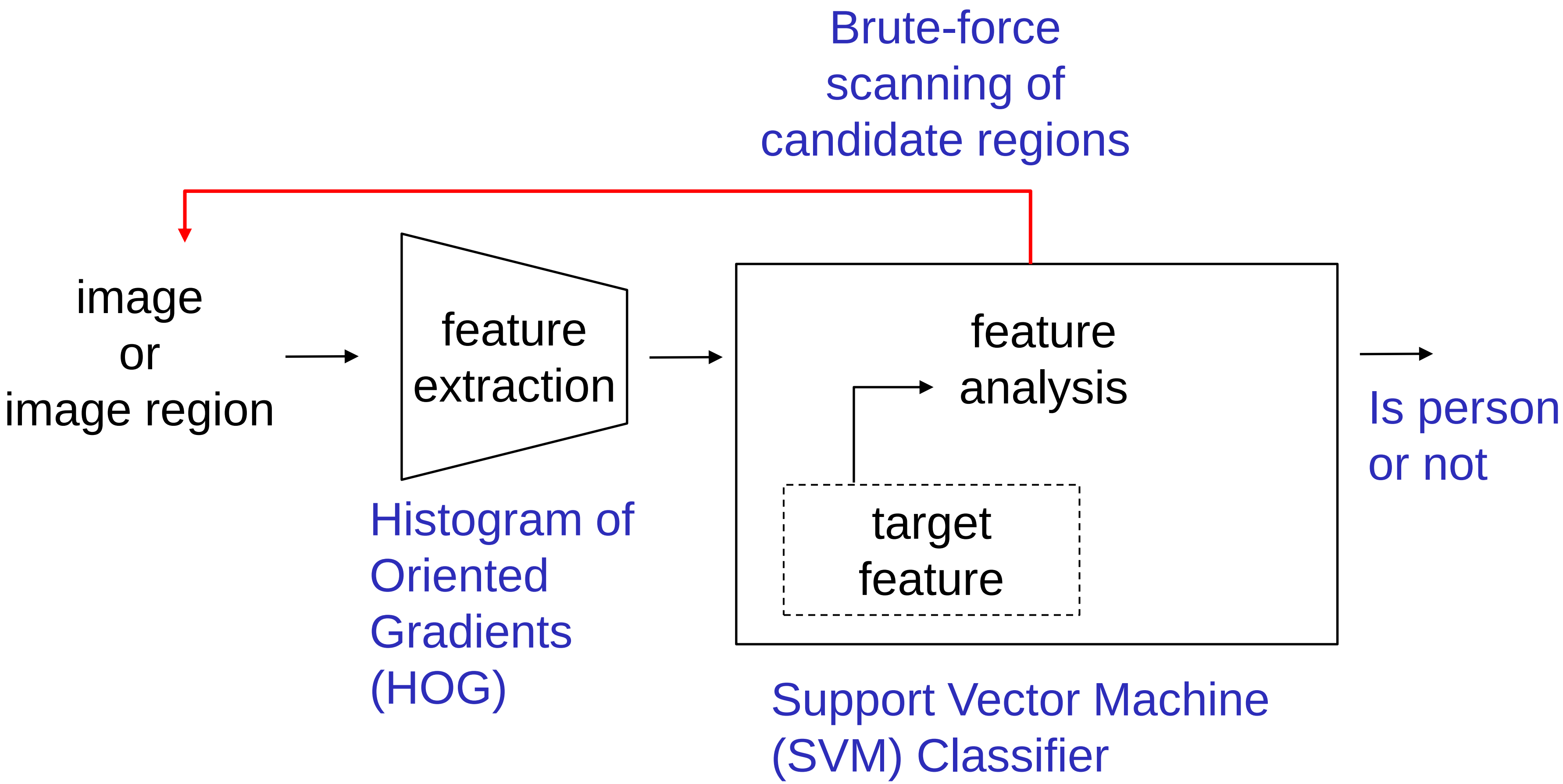
Haar-like features



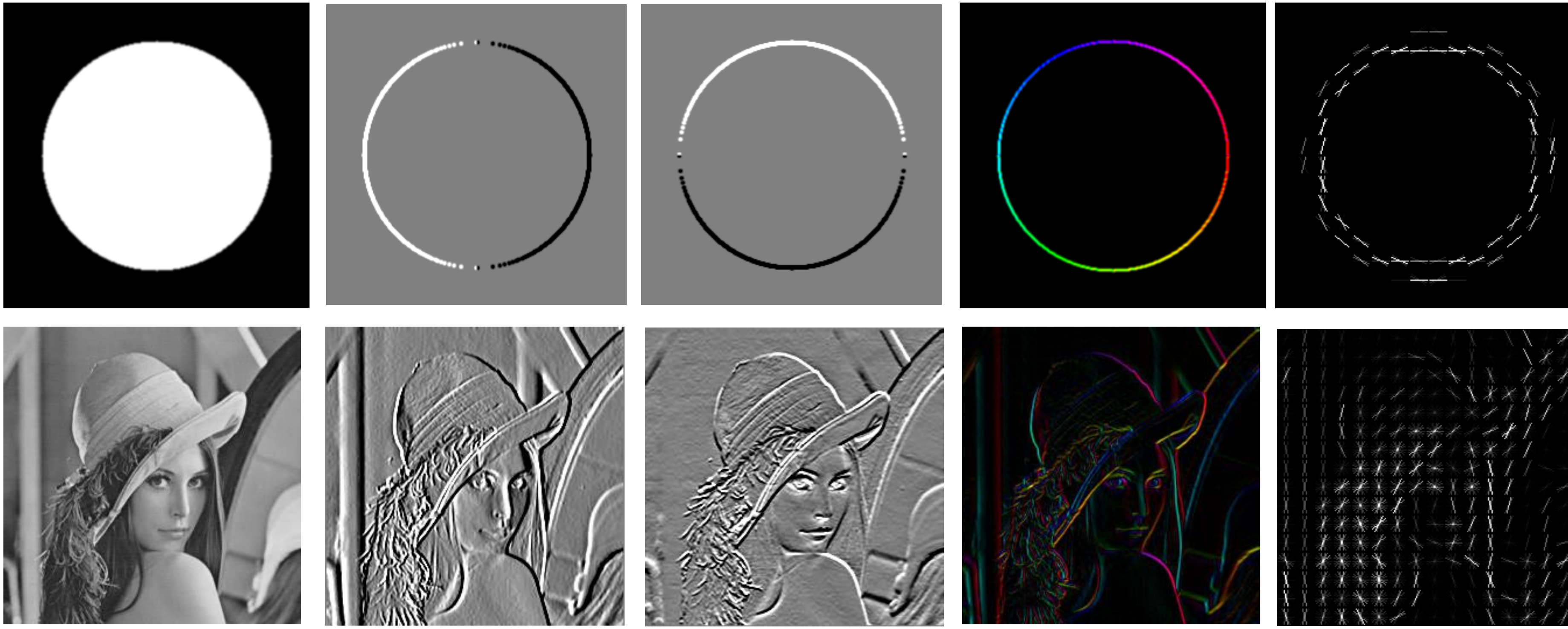
https://docs.opencv.org/4.x/dc/d88/tutorial_traincascade.html

- 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 and Jones 2001]

Person Detection Example



Histogram of Spatial Gradient Orientations



I

I_x

I_y

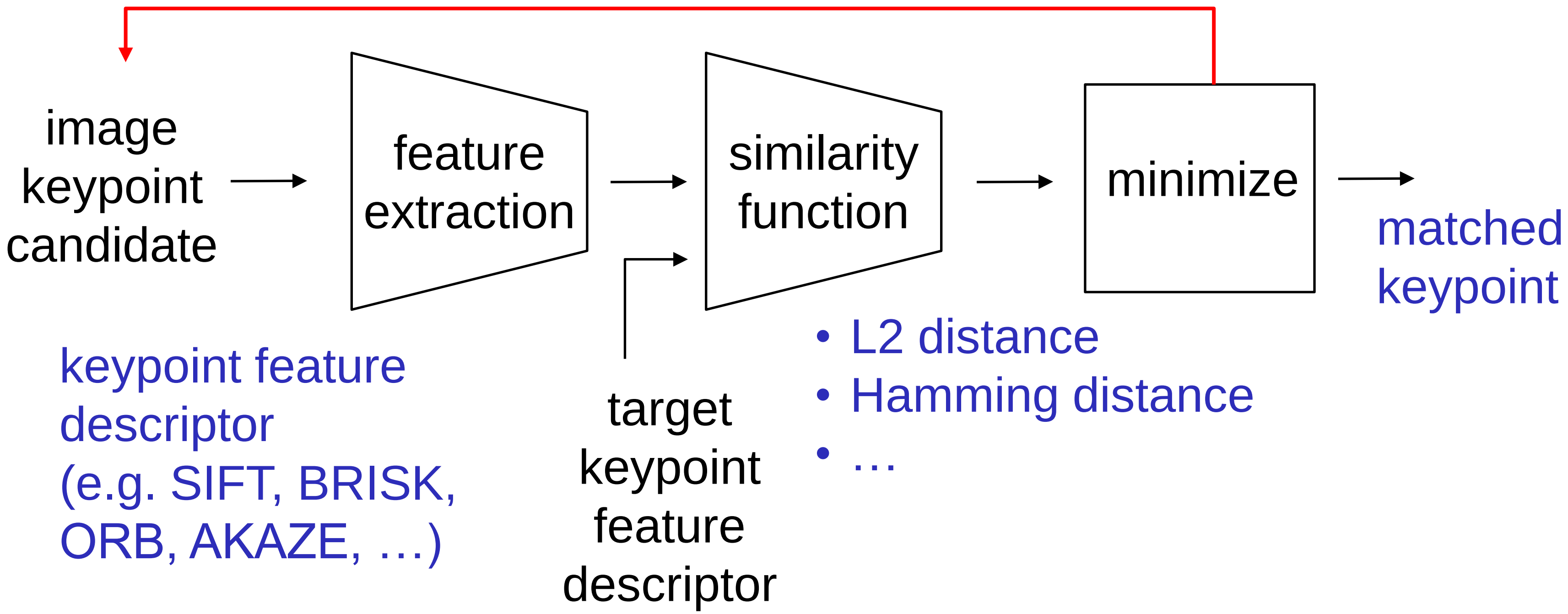
Hue $\propto \mathrm{atan2}(\mathrm{I_y}, \mathrm{I_x}) + \pi$
Sat = 1
Value $\propto \sqrt{I_x^2 + I_y^2}$

gradient orientation histograms visualized

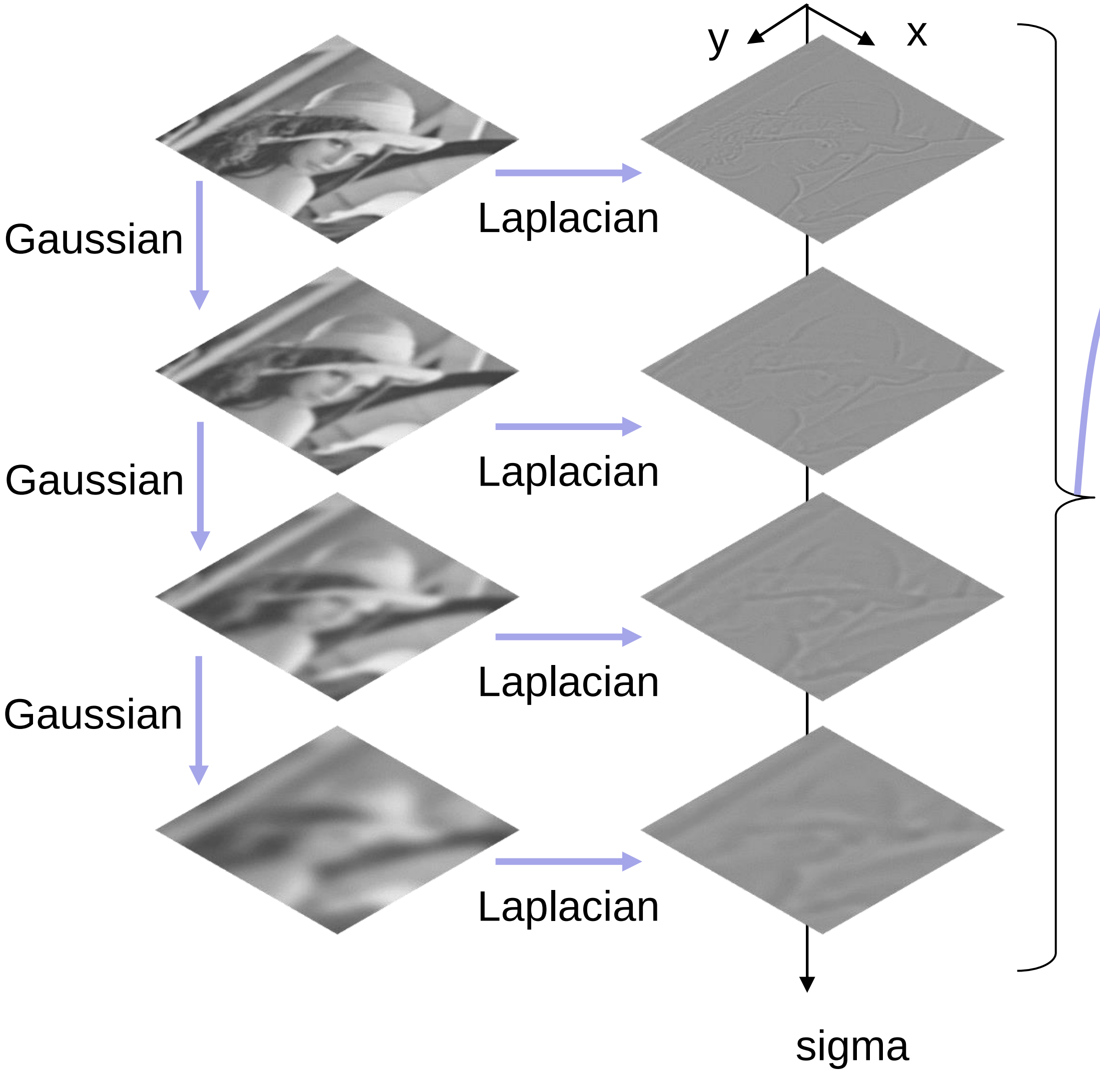
Often referred to as HOG (Histogram of Oriented Gradients) when combined with local block-wise normalization [Datal and Triggs 2005]

Local Image Features around Keypoints

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

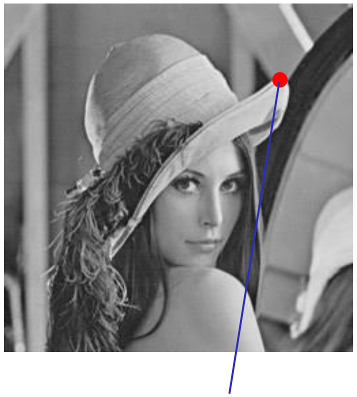


SIFT Keypoint [Lowe 2004]



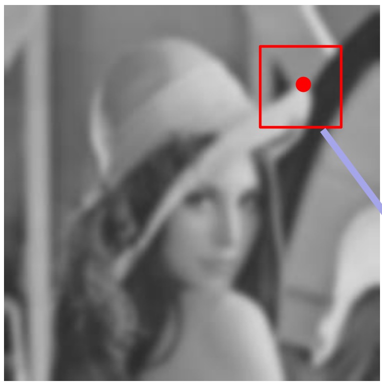
Keypoint Detection

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

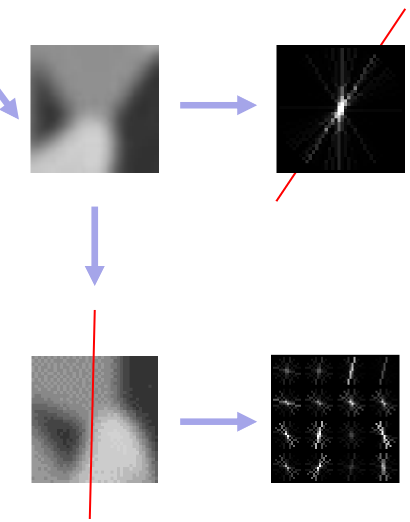


keypoint at (e.g.)
(x = 203, y = 53, sigma = 10)

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

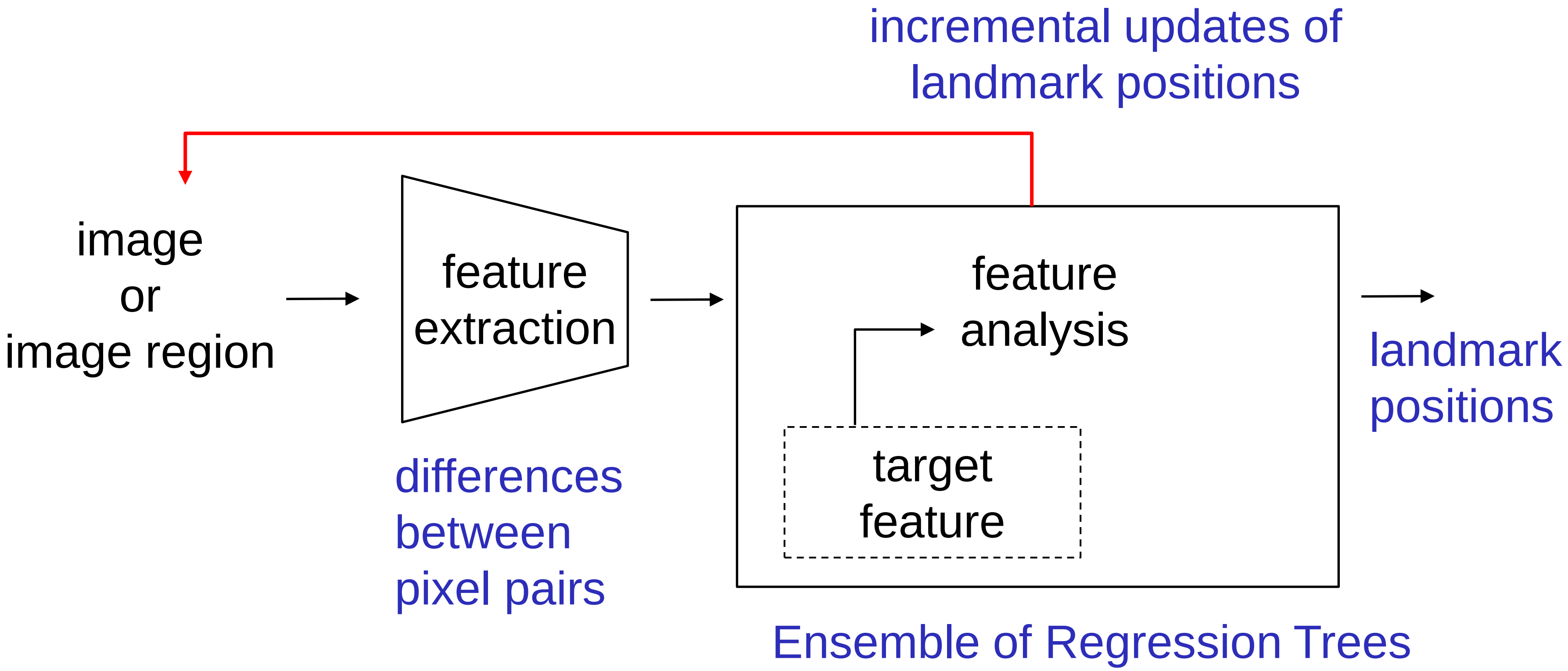
SIFT keypoint matching

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

ic04_keypoint_match.py

Face Landmark Alignment Example

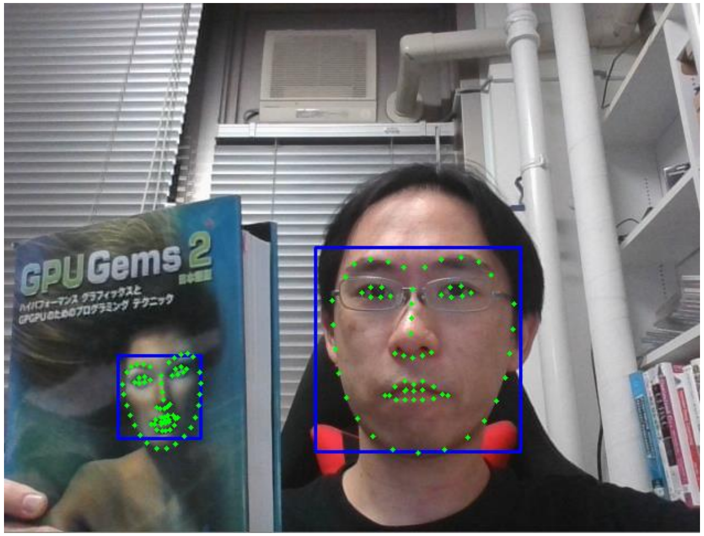


Face Landmark Tracking

ic04_face_dlib.py

[Kazemi and Sullivan 2014]

This sample code requires the Optional Step (3) in kagami_ic2022_install.pdf (building and installing dlib). Note also that you need shape_predictor_5_face_landmarks.dat and/or shape_predictor_68_face_landmarks.dat



You can also try another face detection model implemented in OpenCV:

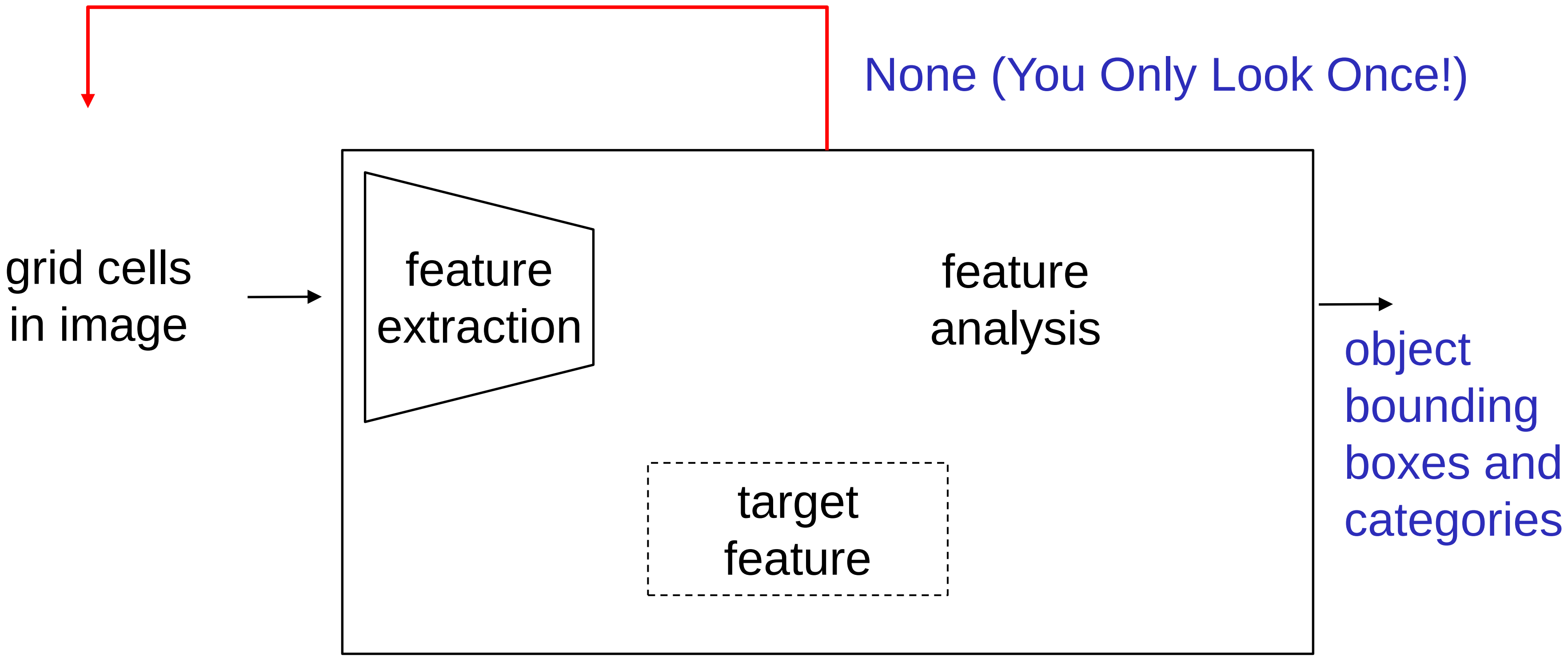
ic04_face_yunet.py

https://github.com/ShiqiYu/libfacedetection

This requires a file face_detection_yunet_2022mar.onnx from https://github.com/opencv/opencv_zoo/tree/master/models/face_detection_yunet

Object Detection/Recognition Example

YOLO Object Detector [Redmon et al. 2016]



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

Running YOLO v5

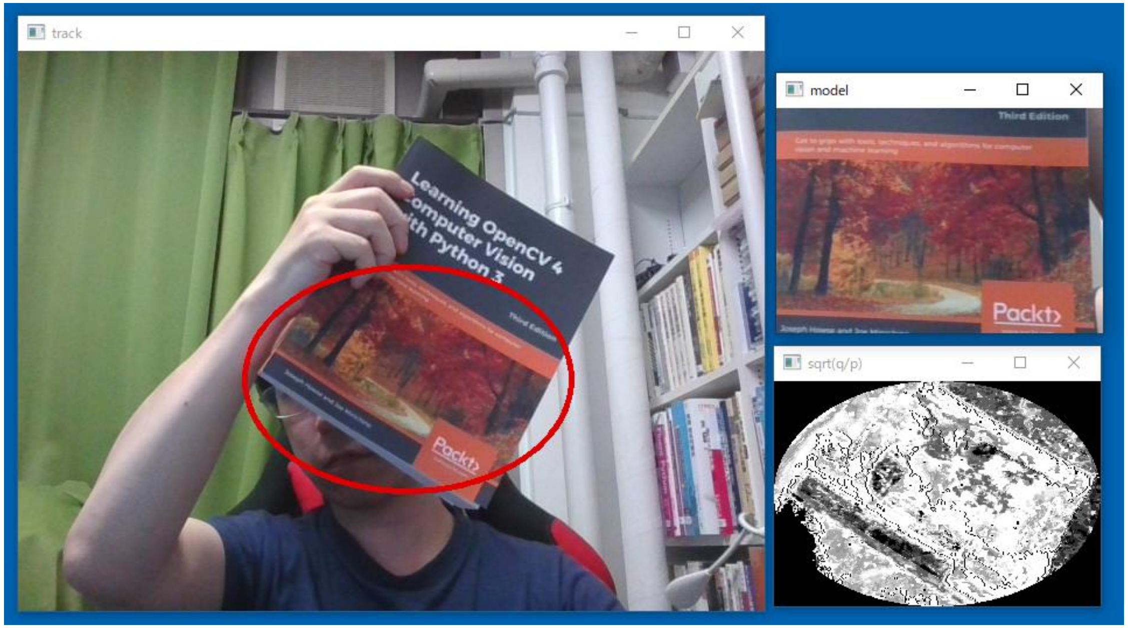
ic04_yolov5.py

https://github.com/ultralytics/yolov5
https://github.com/ultralytics/yolov5/issues/36

- This equires Optional Step (2) in kagami_ic2022_install.pdf (installing PyTorch etc.)
- When it is run for the first time, the parameter files are automatically downloaded into ./hub_cache folder

Exercises (Not Assignments)

Copy and modify ic04_mean_shift_color_histogram.py to visualize the values of $\sqrt{\frac{q}{p}}$ in the normalized unit circle around $\boldsymbol{y}_0$ (current_center) for every camera input frame.



Copy and modify ic04_mean_shift_color_histogram.py to implement adaptation to the object size in the image. A possible approach is to evaluate Bhattacharyya coefficients also for larger and smaller areas (e.g., 1.1 times and 0.9 times, respectively) and update the size (e.g., 1.02 times or 0.98 times, respectively) if the larger or smaller one gives a better result.

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.