Intro to Computer Vision with OpenCV-Python (Day 2)

2017-05-26

Juyong Kim(juyong.kim@vision.snu.ac.kr)

SEOUL NATIONAL UNIV.
VISION & LEARNING

Contents

[Day 1]

- OpenCV-Python
  - Introduction / Image manipulation / Draw objects
- Edge Detection
  - Image gradient / Canny edge detection

[Day 2]

- Object Classification
  - Object classification / Image feature / Classifier
- Object Tracking
  - Histogram-based Object Tracking / LK optical flow / OpenCV tracking API

Object Classification

Object classification

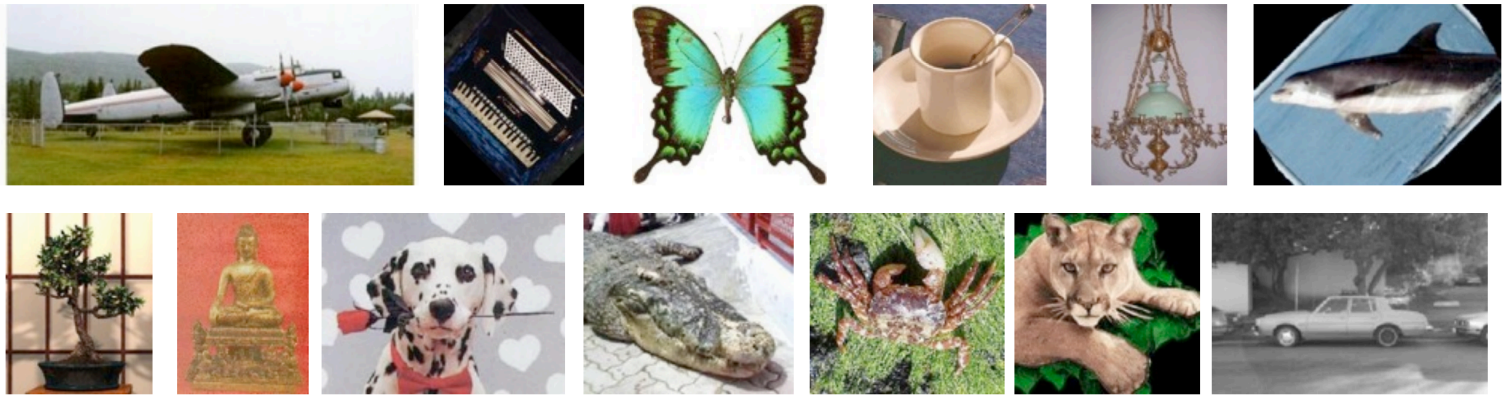
Image feature

- Dense SIFT(PHOW)
- VBoW
- Spatial histogram

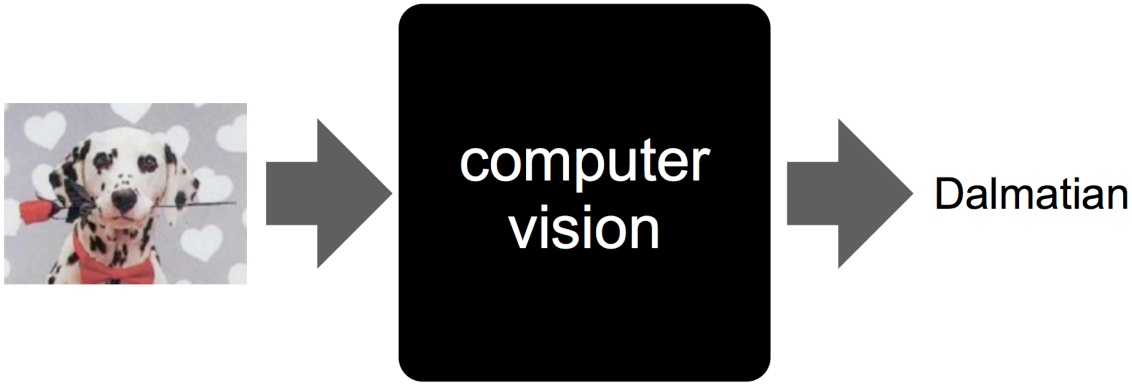
Classifier

Object Classification

- The Dataset – Caltech 101



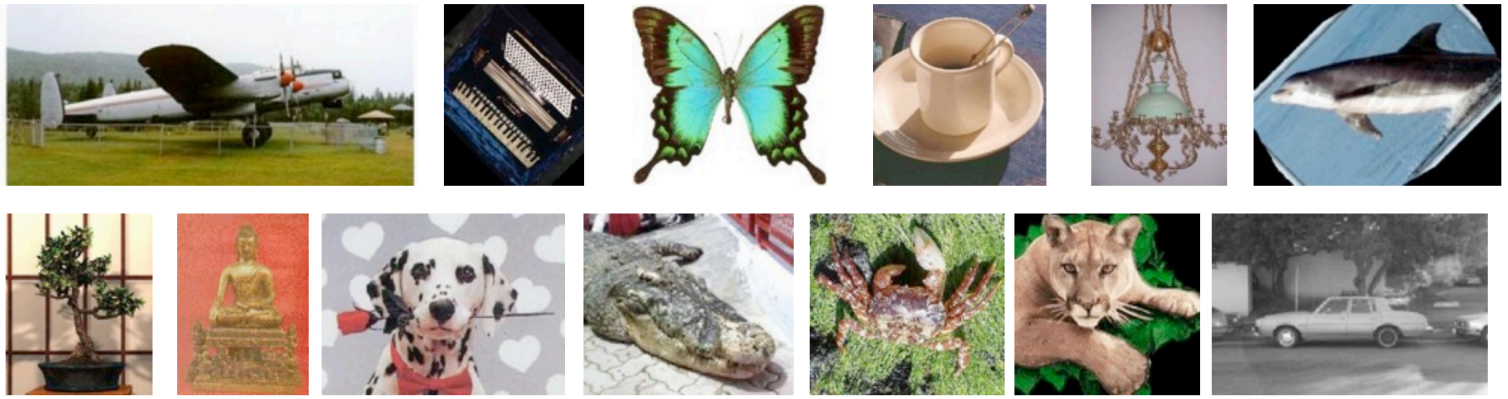
[Fei-Fei et al. 2003]



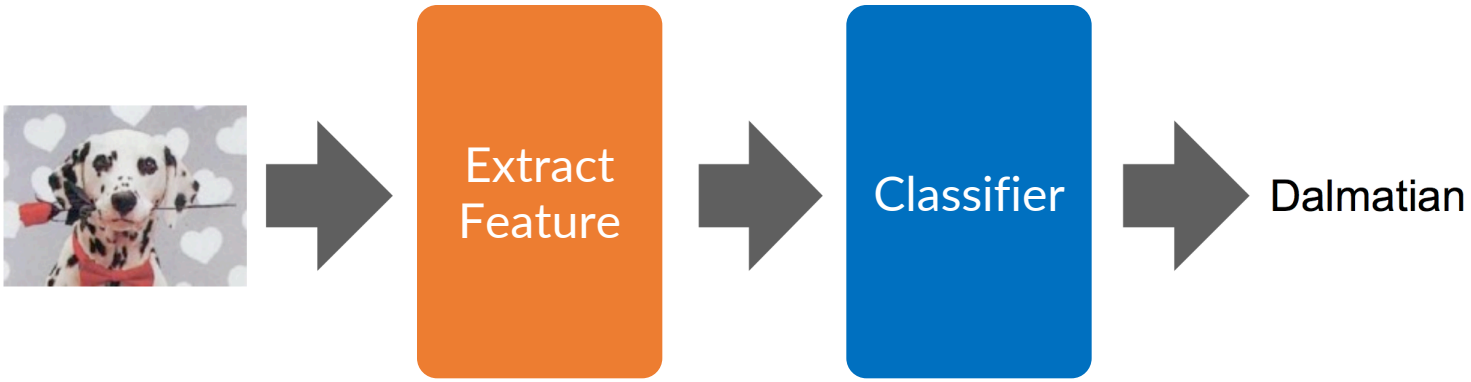
Andrea Vedaldi, Brian Fulkerson, *VLFeat Tutorial* [http://vision.ucla.edu/~vedaldi/assets/pubs/vedaldi10vlfeat-tutorial.pdf]

Object Classification

- The Dataset – Caltech 101



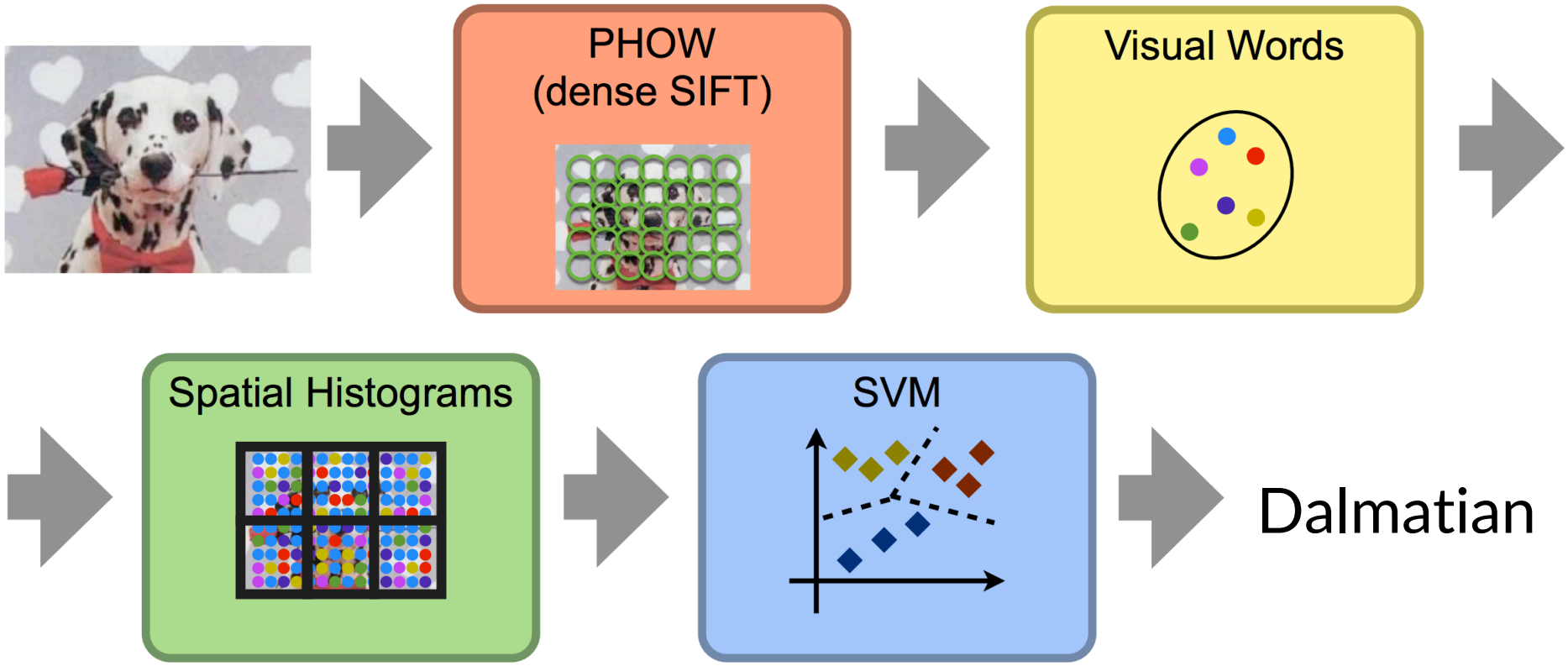
[Fei-Fei et al. 2003]



Andrea Vedaldi, Brian Fulkerson, *VLFeat Tutorial* [http://vision.ucla.edu/~vedaldi/assets/pubs/vedaldi10vlfeat-tutorial.pdf]

Object Classification

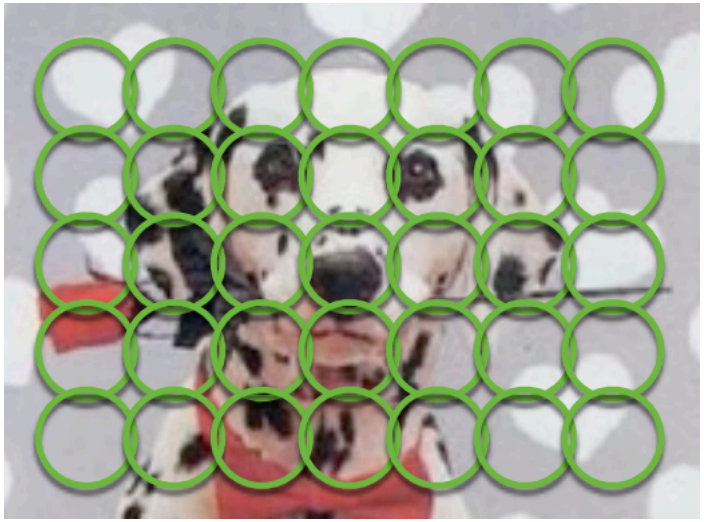
- At a Glance - A Pipeline

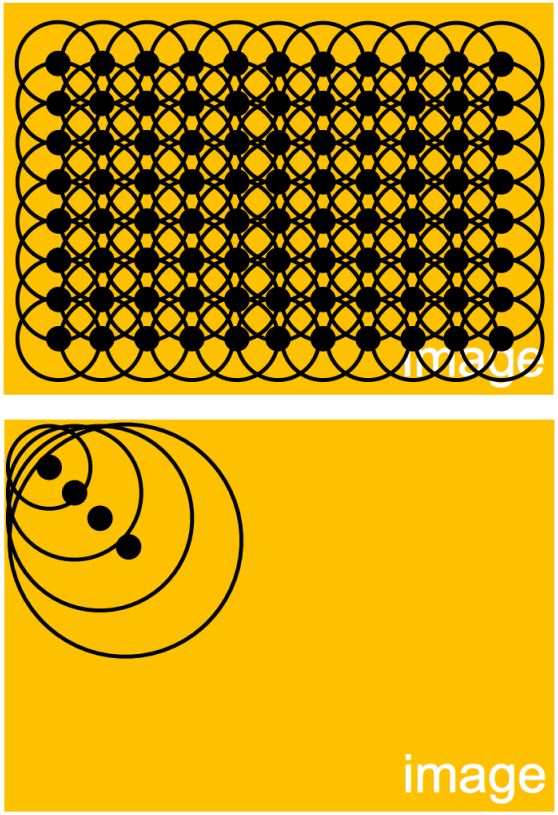


Andrea Vedaldi, Brian Fulkerson, *VLFeat Tutorial* [http://vision.ucla.edu/~vedaldi/assets/pubs/vedaldi10vlfeat-tutorial.pdf]

Feature Extraction

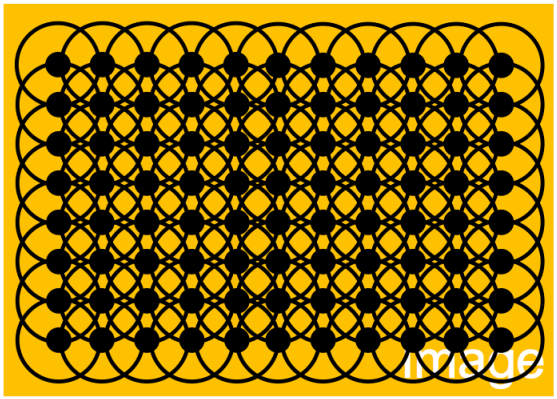
- Dense SIFT
  - Uniform keypoints – No detection
  - Dense multiscale SIFT
  - Descriptors from the uniform keypoints





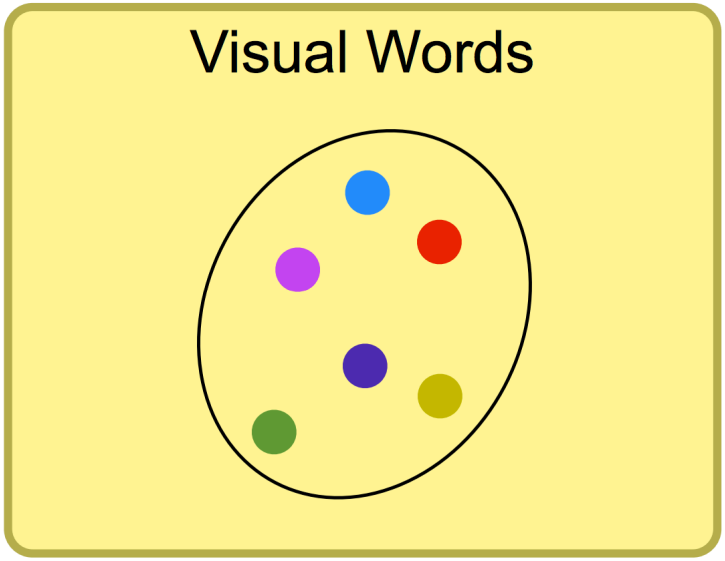
Feature Extraction

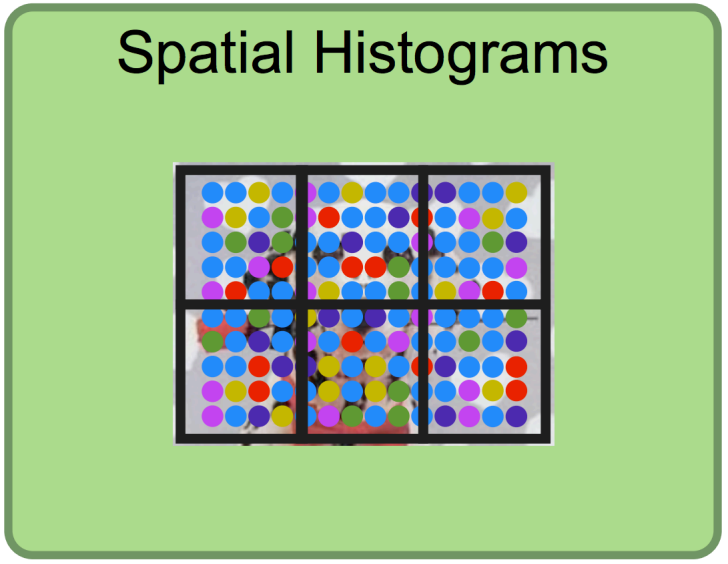
- Dense SIFT
  - Uniform keypoints – No detection
  - Dense multiscale SIFT
  - Descriptors from the uniform keypoints




```
sift = cv2.xfeatures2d.SIFT_create()
# Dense SIFT(Extract SIFT descriptor in grid points over an image)
def denseSIFT(img, step = 5, size = 7):
    rows, cols = img.shape[:2]
    kp = []
    for x in xrange(step,cols,step):
        for y in xrange(step,rows,step):
            kp.append(cv2.KeyPoint(x, y, size))
    kp, des = sift.compute(img, kp)
    return des
```

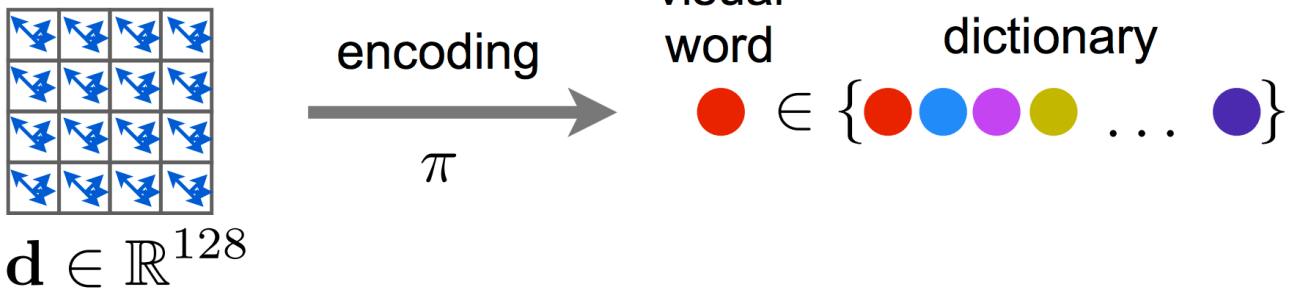
Feature Extraction



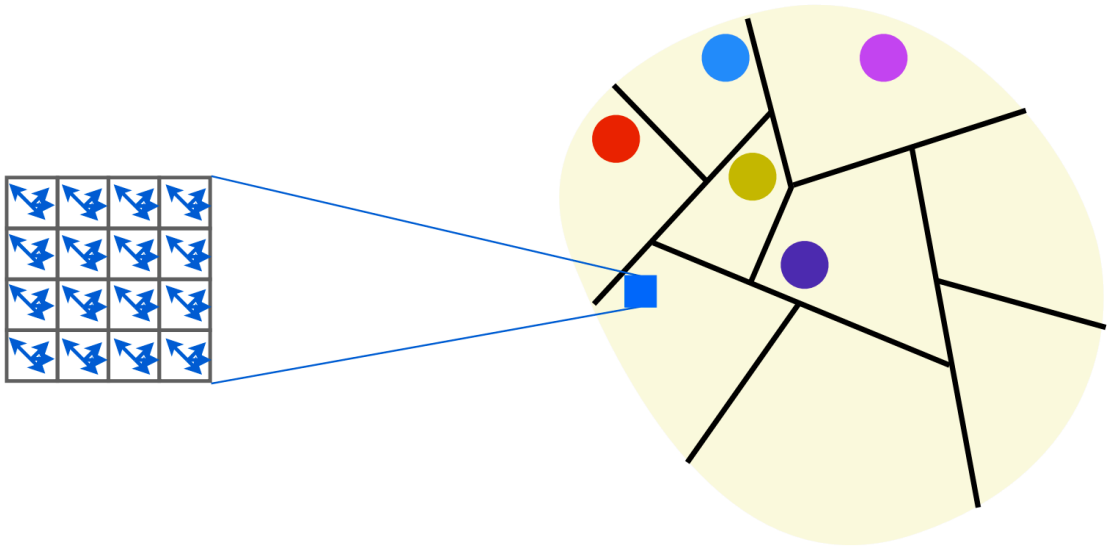


Feature Extraction

- Visual words



- Encoding = clustering [Sivic and Zisserman 2003]
  - vector quantization (k-means) [Lloyd 1982]
  - agglomerative clustering [Leibe et al. 2006]
  - affinity propagation [Frey and Dueck 2007]
  - ...

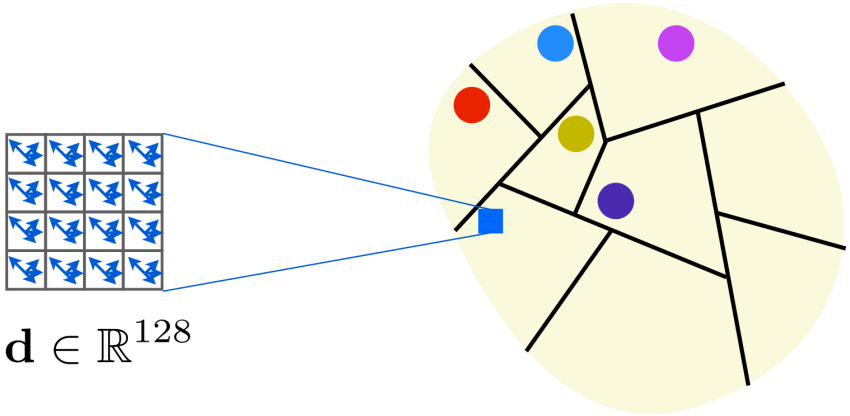


Feature Extraction

- K-means clustering in OpenCV

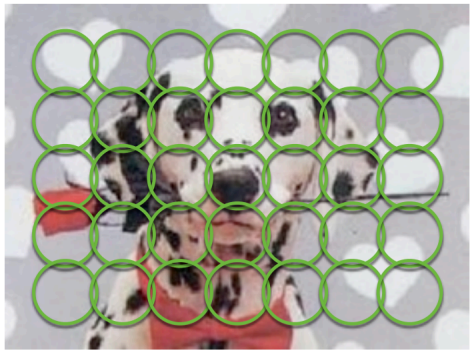
```
# Quantize the descriptors to get the visual words
print "Running K-means clustering (%d -> %d)..." % (PHOW_descrs.shape[0],
                                                    numWords)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 500, 1.0)
attempts = 10
flags = cv2.KMEANS_RANDOM_CENTERS
retval, bestLabels, vocab = cv2.kmeans(PHOW_descrs, numWords,
                                       None, criteria, attempts, flags)
```

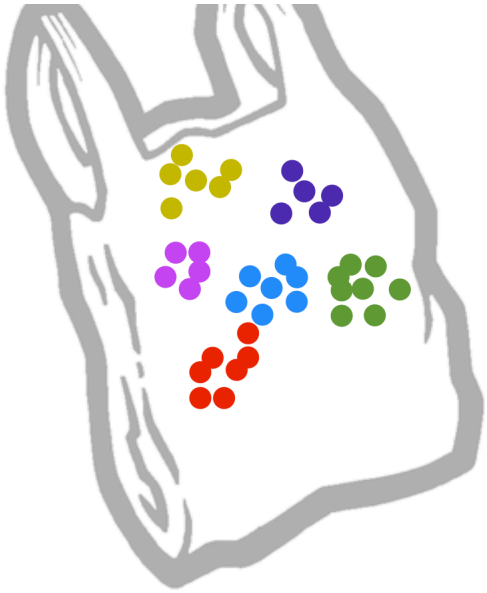
```
# match SIFT features to the words from K-means
bf = cv2.BFMatcher()
matches = bf.knnMatch(des, vocab, k=1)
words = [m[0].trainIdx for m in matches]
```

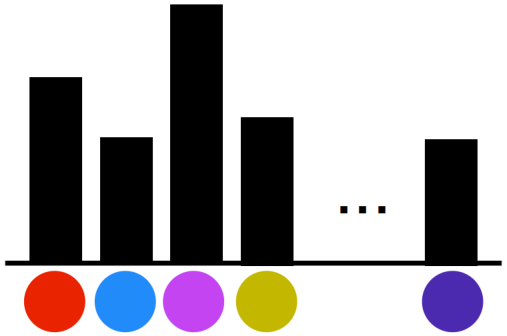


Feature Extraction

- Visual Bag of Words





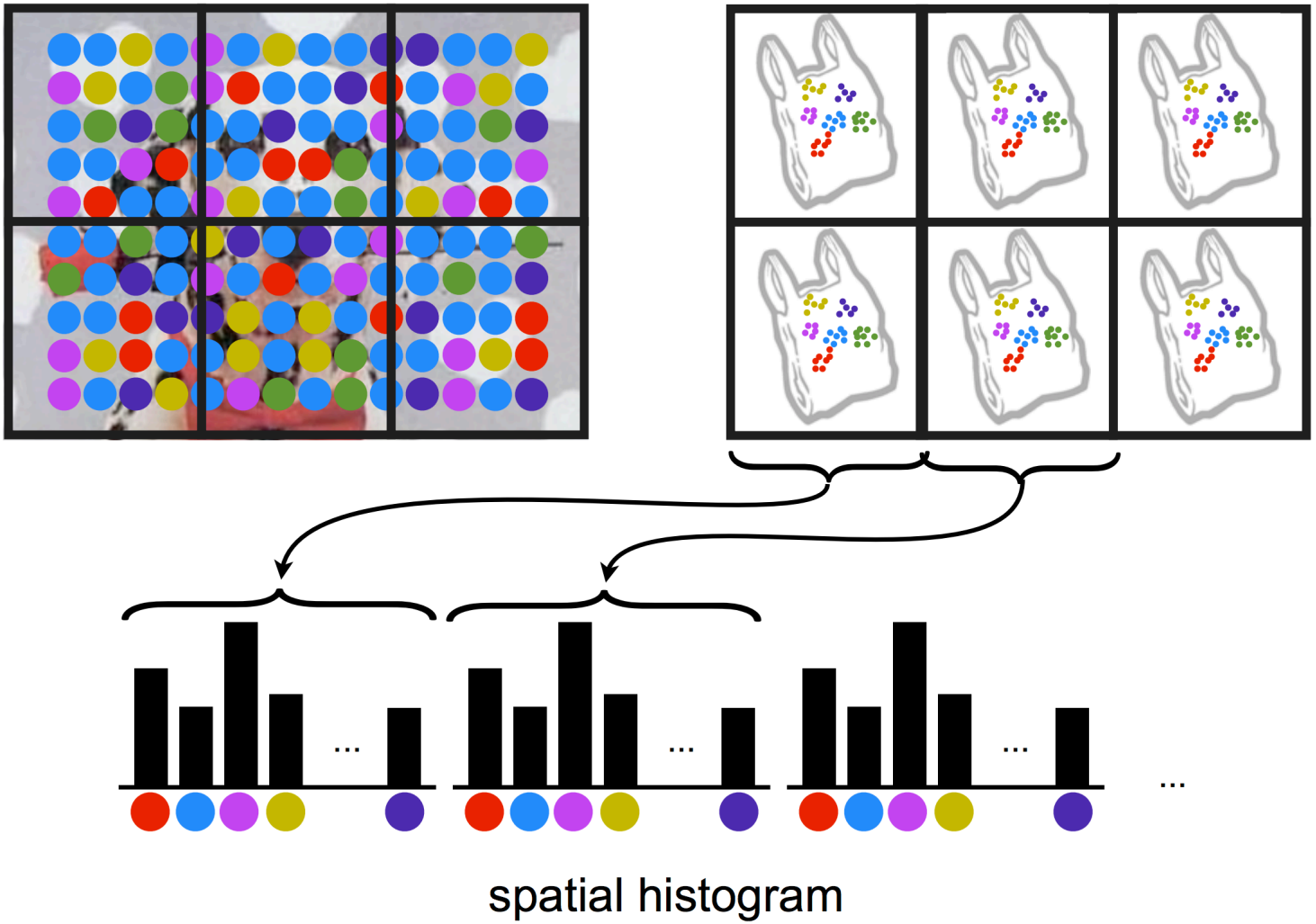


[Csurka et al. 2004]

histogram (bag) of visual words

Feature Extraction

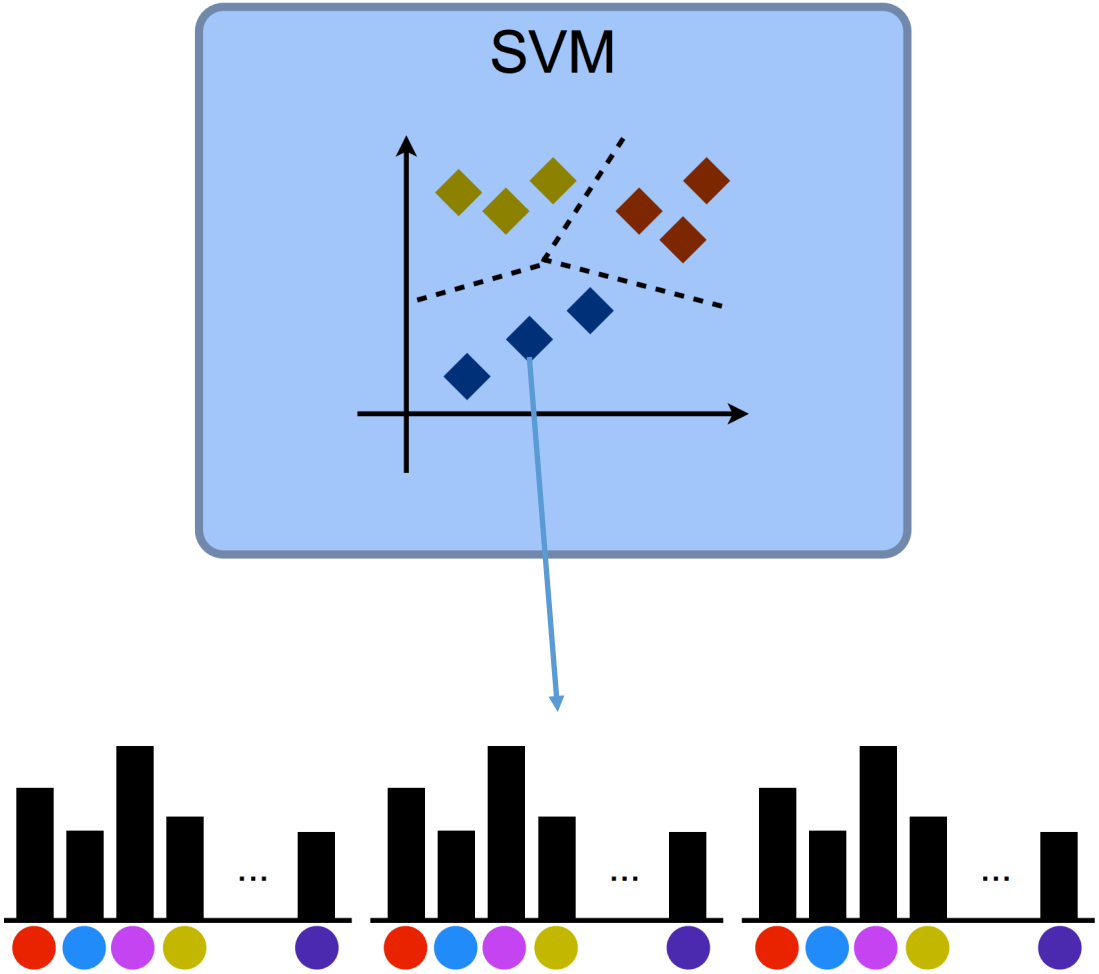
- Spatial Histogram



[Lazebnik et al. 2004]

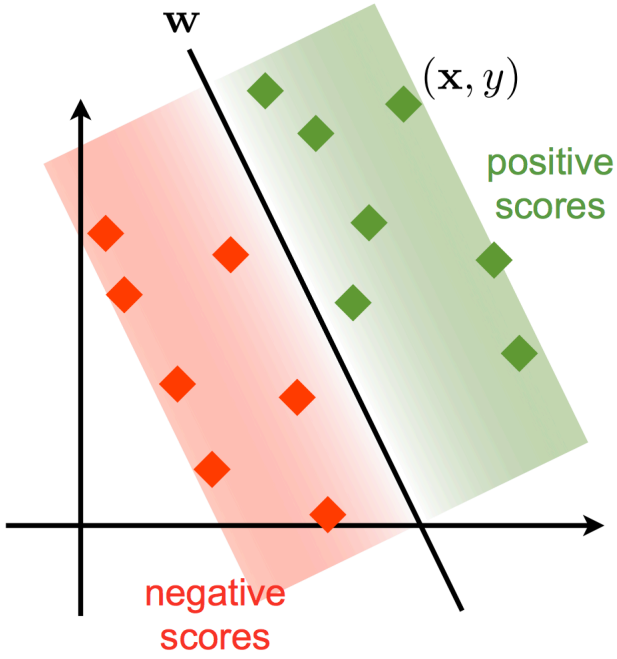
Classifier

- SVM



Classifier

- SVM



Discriminant score

$$f(\mathbf{x}; \mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{w}; (\mathbf{x}_i, y_i))$$

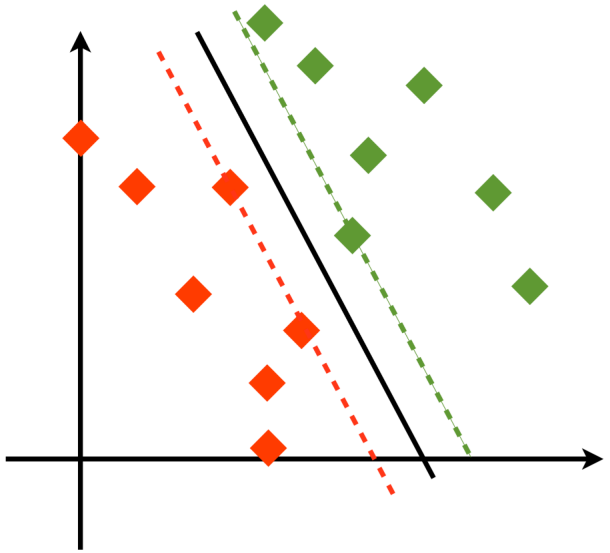
Classifier

- Linear SVM

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; (\mathbf{x}_i, y_i))$$

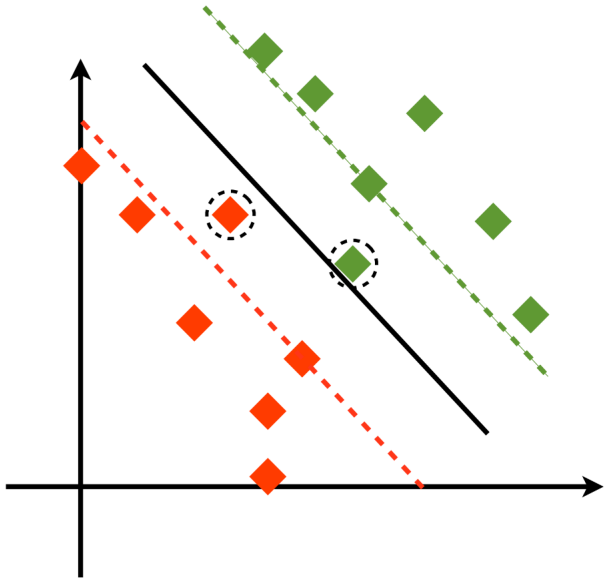
hard loss

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0, & y\langle \mathbf{w}, \mathbf{x} \rangle > 1, \\ +\infty, & \text{otherwise.} \end{cases}$$



hinge loss

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0, & y\langle \mathbf{w}, \mathbf{x} \rangle > 1, \\ 1 - y\langle \mathbf{w}, \mathbf{x} \rangle, & \text{otherwise.} \end{cases}$$



Classifier

- SVM in OpenCV
  - Training SVM

```
# Train SVM
print 'Training SVM...'
svm = cv2.ml.SVM_create()
svm.setType(cv2.ml.SVM_C_SVC)  # classification(n > 2)
svm.setKernel(cv2.ml.SVM_LINEAR)  # linear kernel
svm.setC(0.01)
svm.setTermCriteria((cv2.TERM_CRITERIA_COUNT, 10, 1.0)) # term. criteria

svm.train(train_bow, cv2.ml.ROW_SAMPLE, train_labels)
```

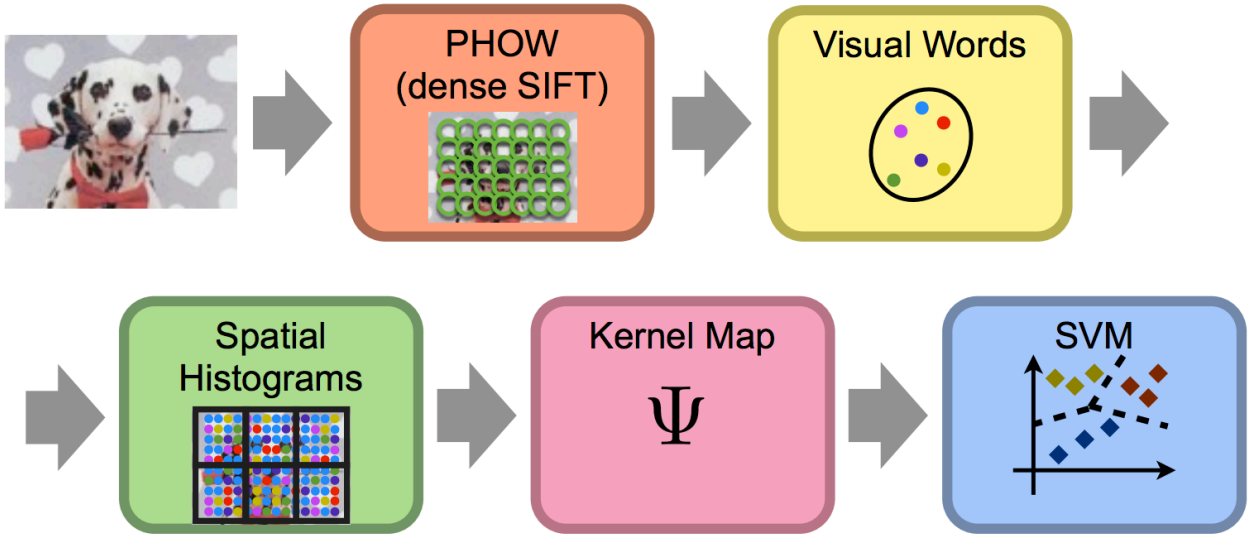
  - Predict with SVM

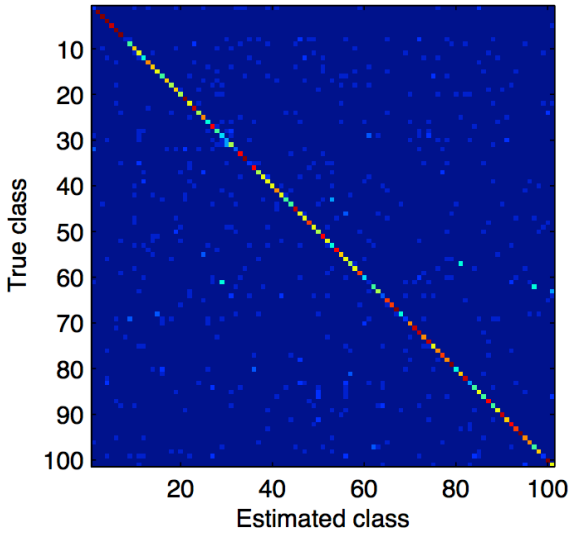
```
train_preds = svm.predict(train_bow)[1]
print('Training Accuracy: %.6f' % np.average(train_preds == train_labels))
```

```
Training Accuracy: 0.990850
```

Object Classification

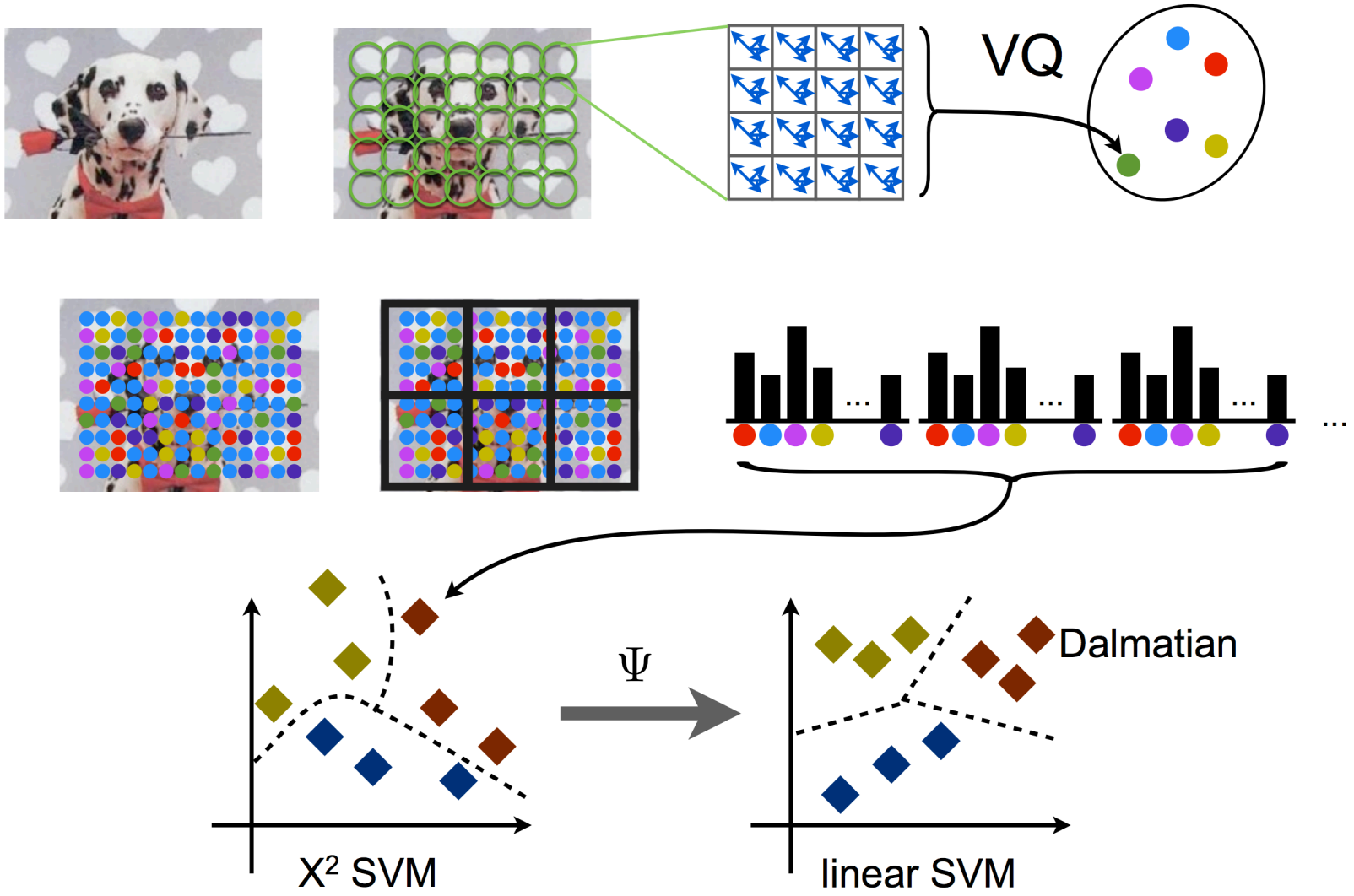
- Results
  - ~60% Accuracy using Dense SIFT / VBoW / Spatial Histogram





Object Classification

- Object Classification Summary



Let’s Check the Code

3_classification.ipynb

Object Tracking

Histogram-based object tracking

- MeanShift and CamShift
- Histogram-based tracking in OpenCV

Lucas-Kanade optical flow

- Optical flow
- LK optical flow in OpenCV

OpenCV tracking API

Video Input(cv2.VideoCapture)

- Capturing video from video files or devices

```
v_in = cv2.VideoCapture("video.mp4") # Open a video
v_in = cv2.VideoCapture(0) # Open the default camera
```

- Grab a frame from the file/device

```
ret, frame = v_in.read()
```

- Get properties of the video

```
prop = v_in.get(cv2.CAP_PROP_FRAME_WIDTH) # frame width
```

- Close the file/device

```
v_in.release()
```

Video Output(cv2.VideoWriter)

- Choose a video codec and initialize video writer

```
fourcc = cv2.VideoWriter_fourcc(*'X264') # Define FOURCC
v_out = cv2.VideoWriter('out.mp4', fourcc,
                        FPS, (width, height)) # Write video file
```

  - X264 is FourCC of H.264/MPEG-4 AVC
  - OpenCV look for a proper library of H.264 and initialize(openh264 is located in the tutorial root folder: openh264-1.6.0-win64msvc.dll)

- Write a frame(image) to the file

```
v_out.write(frame)  # image size should match
```

- Close the file

```
v_out.release()
```

Video I/O

Let’s Check the Code

video_tutorial.py

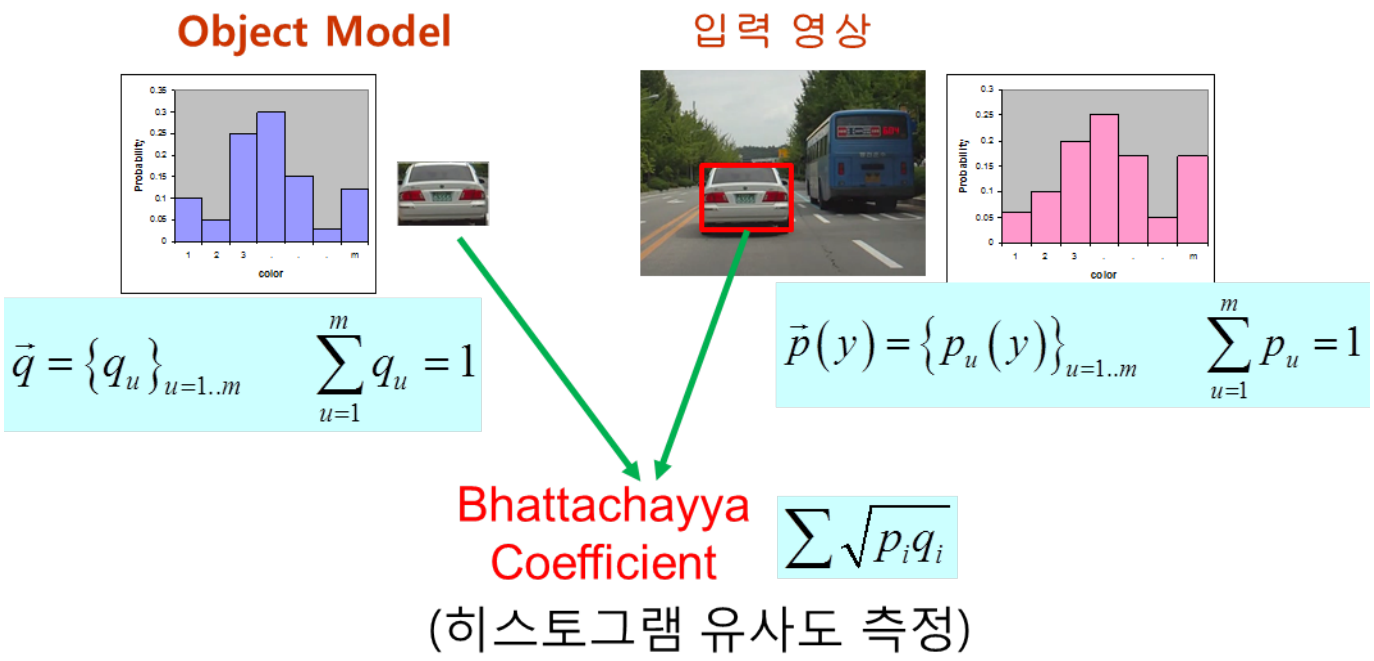
- To run the script:

```
> python video_tutorial.py ↵
```

Histogram-based Object Tracking

- Object tracking algorithms for locating a local maxima of a density function
  - MeanShift / CamShift

- What density function? A function of histogram matching
  - Histogram of (initial) object window & histogram of arbitrary window



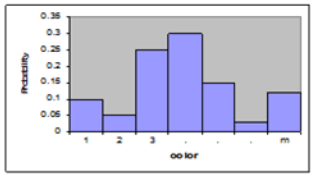
(Image credit: blog on MeanShift(Korean))

Histogram-based Object Tracking

- MeanShift/CamShift procedure

initialization

initial detection



object model (histogram)

$$H_{model} = \{q_u\}_{u=1..n} \qquad \sum_{u=1}^{n} q_u = 1$$

histogram backprojection

input image, I(x)

histogram backprojection

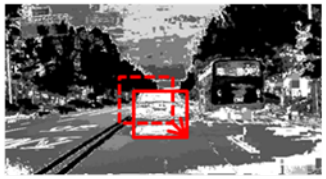
backprojection image

$$w(x) = \sqrt{\frac{H_{model}(I(x))}{H_{bkg}(I(x))}}$$

mean-shift localization

$$\Delta x = \frac{\sum w(x)(x - \bar{x}_{old})K(x - \bar{x}_{old})}{\sum w(x)K(x - \bar{x}_{old})}$$

$$\bar{x}_{new} = \bar{x}_{old} + \Delta x$$



mean-shift localization (density mean)

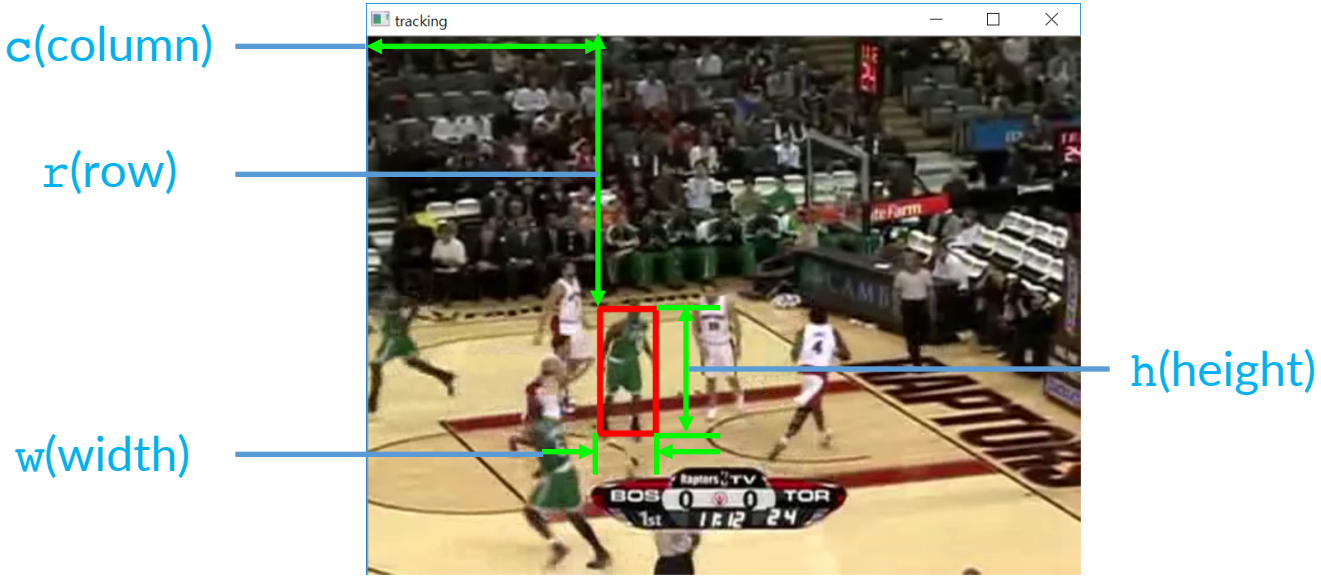
- CamShift has additional ellipse fitting of window after convergence

(Image credit: blog on MeanShift(Korean))

Histogram-based Object Tracking

- Calculate the histogram of the object

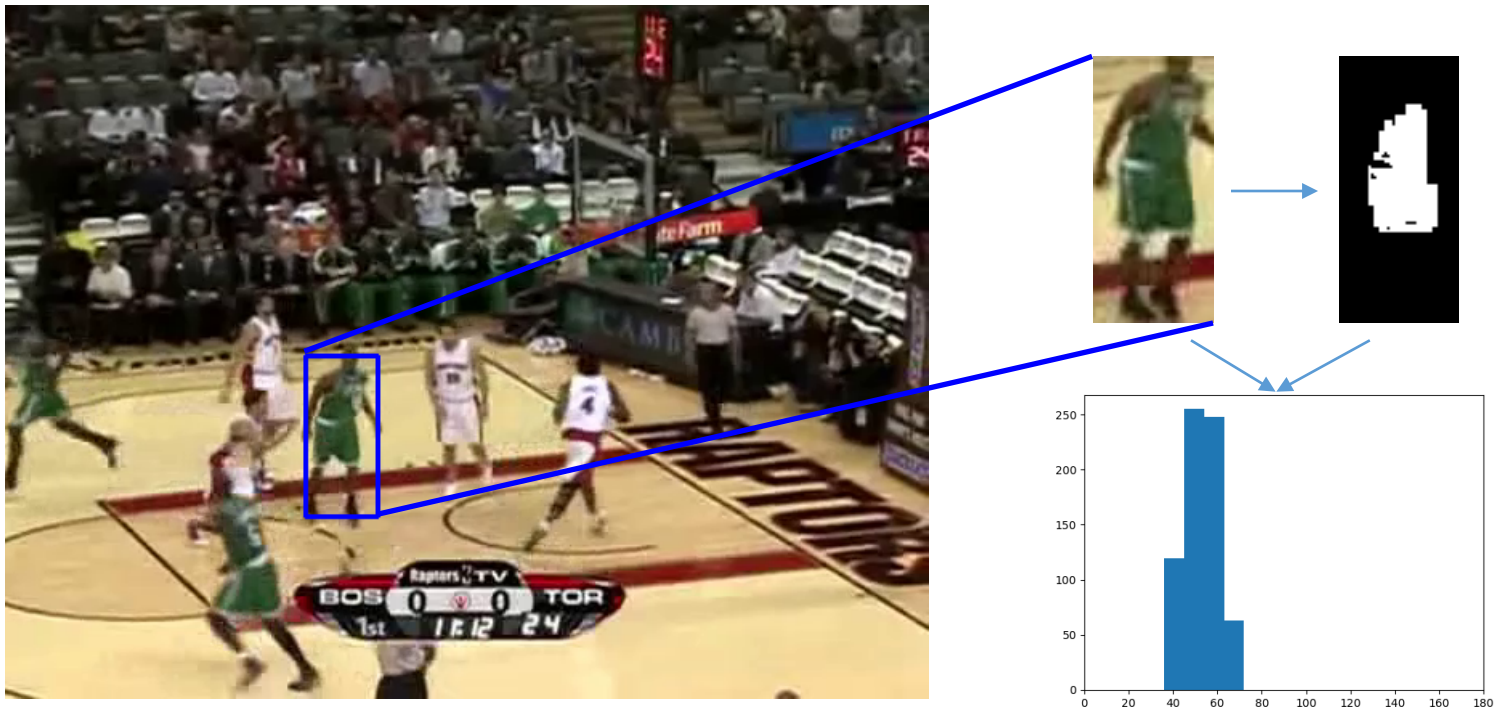
```
track_window = (187, 218, 45, 100) # ROI (c, r, w, h)
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((40.,50.,50.)), np.array((80.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [20], [0,180])  # Hue histogram
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
```



Histogram-based Object Tracking

- Calculate the histogram of the object

```
track_window = (187, 218, 45, 100) # ROI (c, r, w, h)
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((40.,50.,50.)), np.array((80.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [20], [0,180])  # Hue histogram
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
```



Histogram-based Object Tracking

- Grab a new frame and calculates the back projection of the histogram

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
dst = cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1)
```

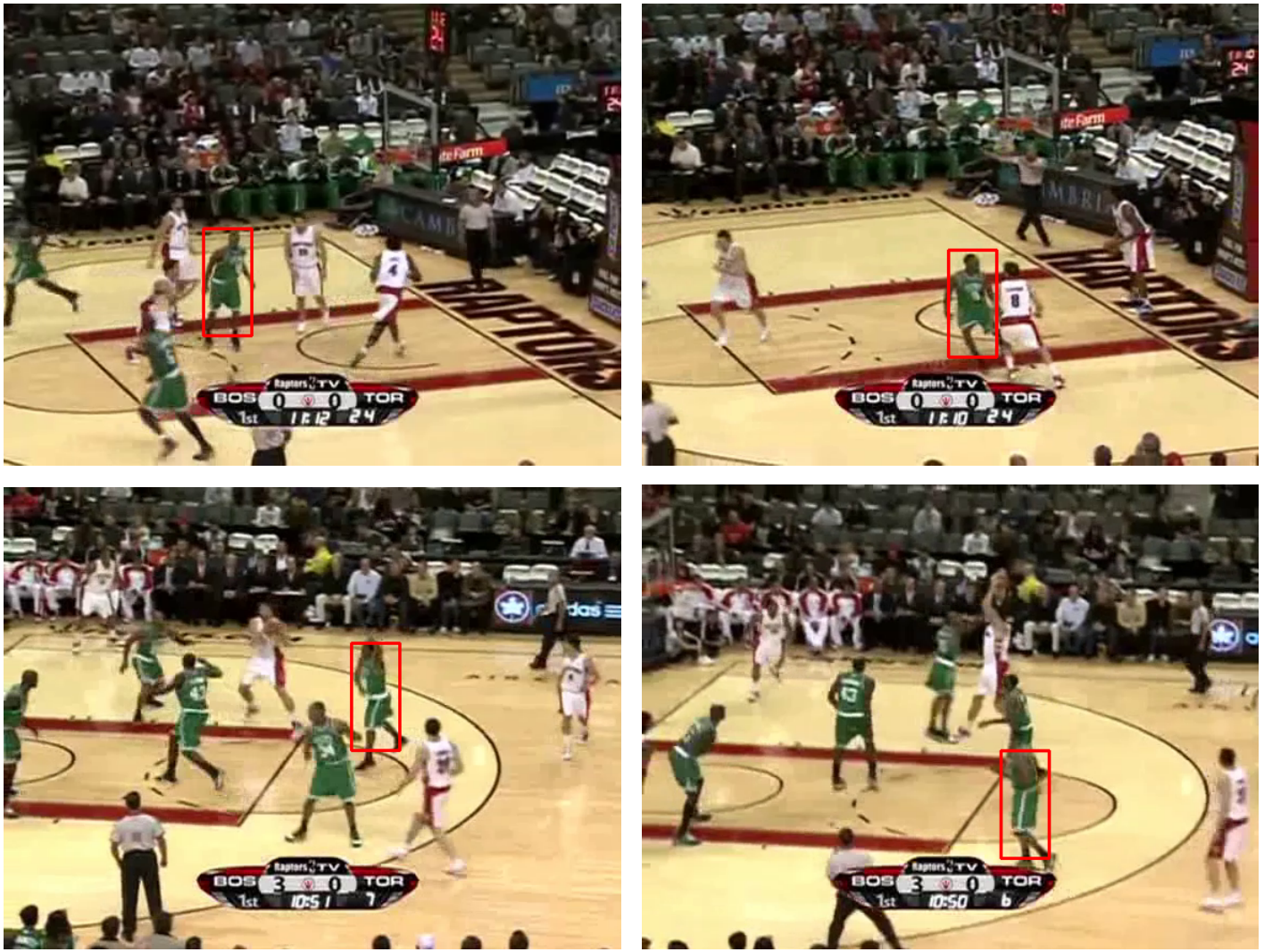
- Apply MeanShift/CamShift to update the tracking window

```
term_crit=(cv2.TERM_CRITERIA_EPS|cv2.TERM_CRITERIA_COUNT,10,1)
ret, track_window = cv2.meanShift(dst, track_window, term_crit)
#ret, track_window = cv2.CamShift(dst, track_window, term_crit)
```

  - `cv2.TERM_CRITERIA_EPS`: Termination criteria of change
  - `cv2.TERM_CRITERIA_COUNT`: Termination criteria of iteration

Histogram-based Object Tracking

- MeanShift/CamShift in OpenCV



Histogram-based Object Tracking

Let’s Check the Code

tracking_1_hist.py

- To run the script:

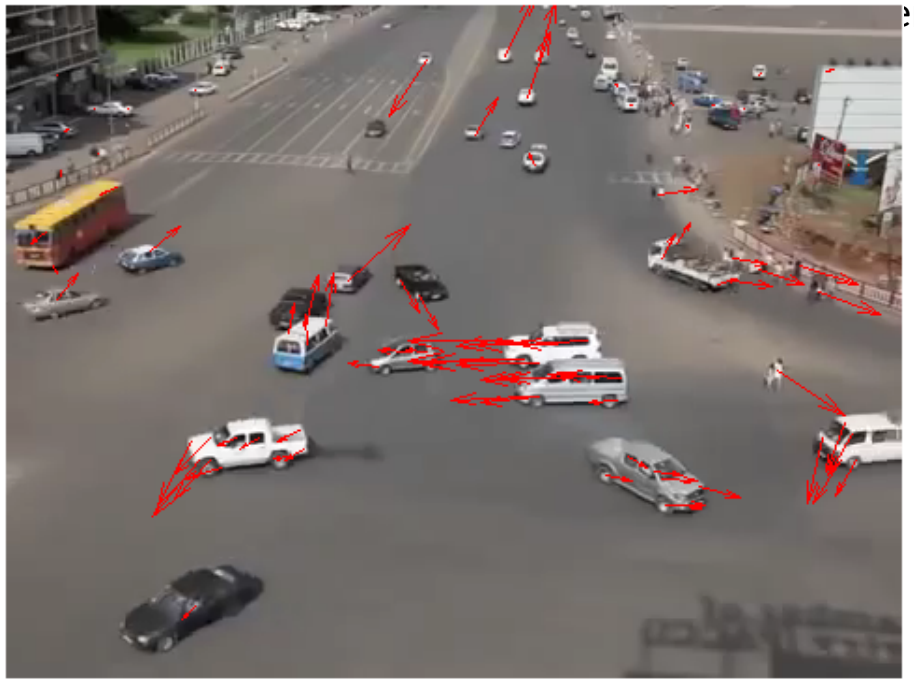
```
> python tracking_1_hist.py ↵
```

Lucas-Kanade Optical Flow

- Lucas-Kanade method
  - A widely used differential method for optical flow estimation

$$\begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_{x,y} I_x (T(x,y) - I(x,y)) \\ \sum_{x,y} I_y (T(x,y) - I(x,y)) \end{bmatrix}$$

  - $T(x, y)$: Templa ent



(Image credit: An LK example(MATLAB)

Lucas-Kanade Optical Flow

- LK optical flow in OpenCV-Python

```
nextPts, status, err = cv2.calcOpticalFlowPyrLK(prevImg,
                                    nextImg, prevPts, nextPts, …)
```

  - Compute optical flow of input points using the iterative LK method.
  - `prevImg`: First 8-bit(grayscale) input image.
  - `nextImg`: Second input image of the same size and type as `prevImg`
  - `prevPts`: Vector of 2D points for which the flow needs to be found
  - `nextPts`: Output vector of 2D points
  - `status`: Output status for each point of `prevPts` that are found in `nextImg`
  - `err`: Output error for each point

- Finding initial points – Shi-Satomi corner detector

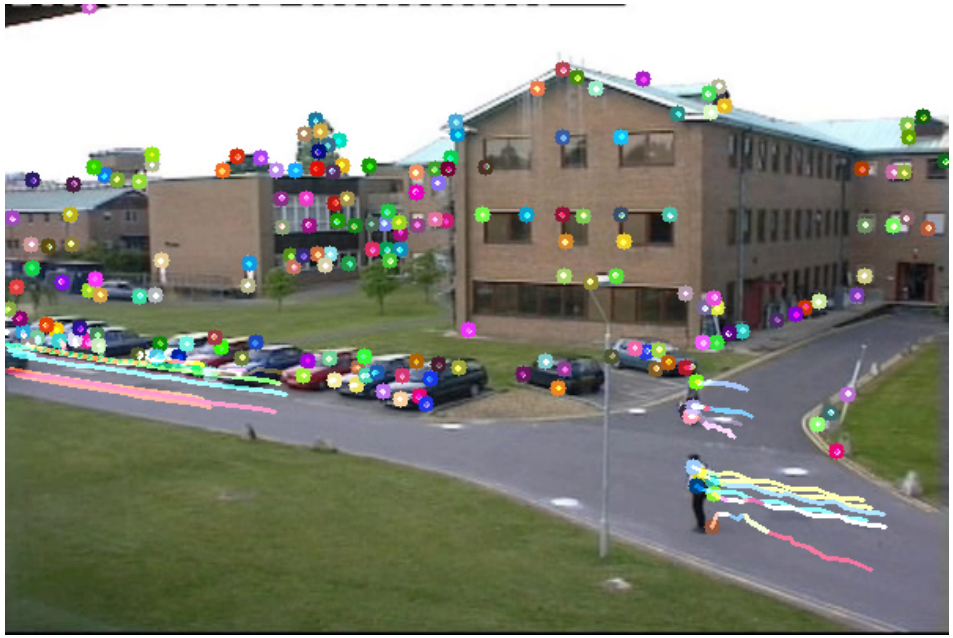
```
pts = cv2.goodFeaturesToTrack(gray_img, mask=None,
                                    **feature_params)
```

  - Finds corner points that are good to be tracked by trackers($\min(\lambda_1, \lambda_2) > k$)

Lucas-Kanade Optical Flow

- Point tracking with LK optical flow.

1. Find salient points from the first frame(`cv2.goodFeaturesToTrack()`)
2. For every frame
    1. Calculate optical flow and errors(`cv2.calcOpticalFlowPyrLK()`)
    2. Mark the tracks of moved points with low error



Lucas-Kanade Optical Flow

Let’s Check the Code

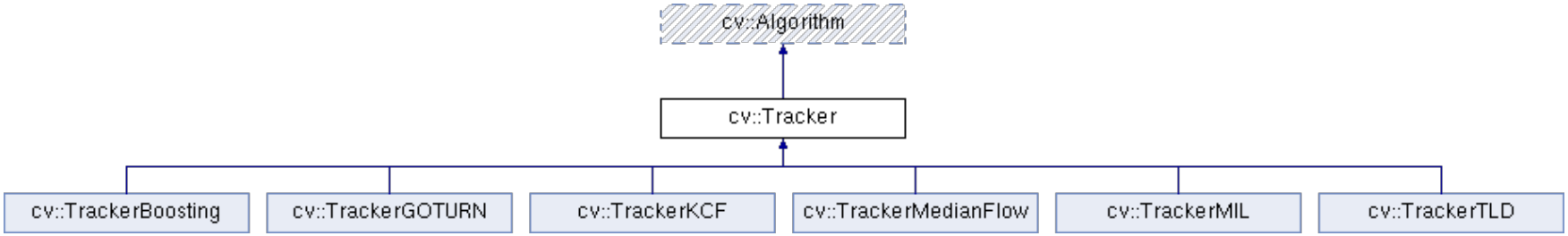
tracking_2_LK.py

- To run the script:

```
> python tracking_2_LK.py ↵
```

OpenCV Tracking API

- OpenCV provides common interfaces(C++) for object trackers



- Various types of recent object trackers are implemented
  - MIL: Visual Tracking with Online Multiple Instance Learning(CVPR 2009)
  - BOOSTING: Real-time tracking via on-line boosting(BMVC 2006)
  - MedianFlow: Forward-Backward Error: Automatic Detection of Tracking Failures(2010)
  - TLD: Tracking-Learning-Detection(PAMI 2010)
  - KCF: Exploiting the circulant structure of tracking-by-detection with kernels(ECCV 2012)
  - GOTURN: Learning to Track at 100 FPS with Deep Regression Networks(ECCV 2016)

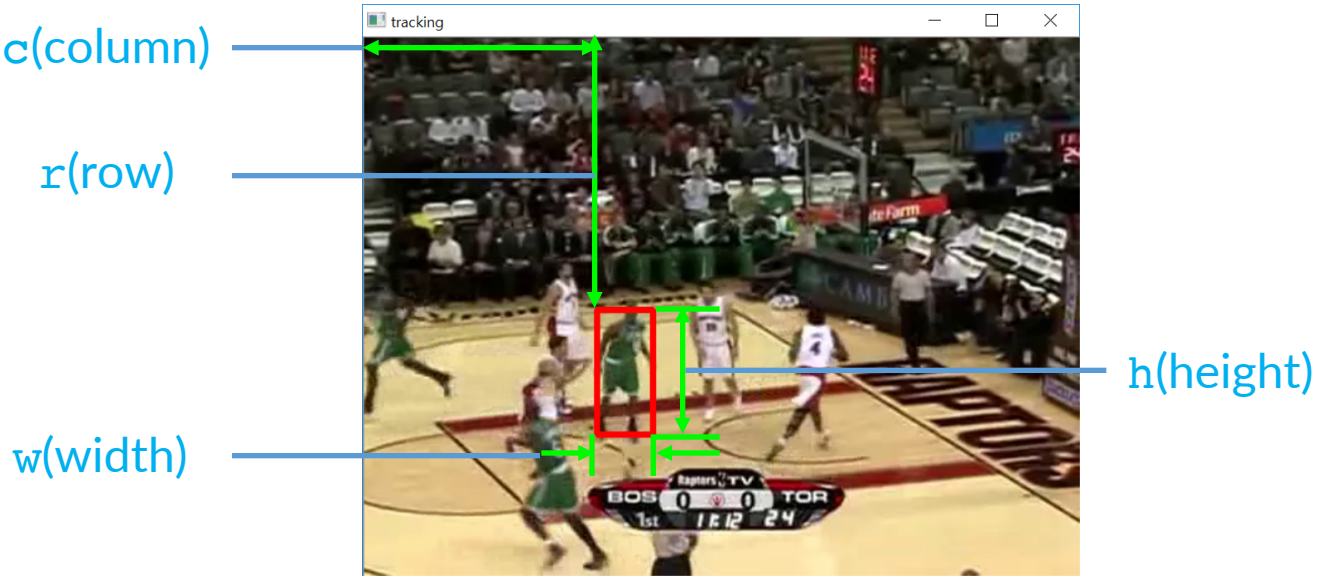
OpenCV Tracking API

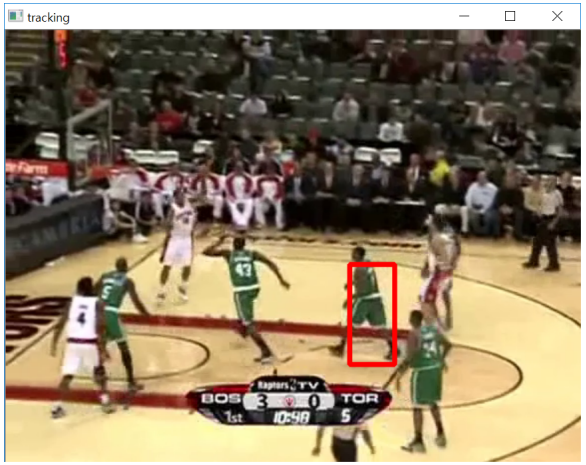
- Tracker API in OpenCV-Python – Simple!

```
# Set up tracker.
# MIL, BOOSTING, KCF, TLD, MEDIANFLOW or GOTURN
tracker = cv2.Tracker_create("KCF")

# Open a video and grab the first frame
v_in = cv2.VideoCapture("videos/basketball.mp4")
ret, frame = v_in.read()

# Initialize tracker with a bounding box
bbox = (187, 218, 45, 100)  # (r, c, w, h)
ret = tracker.init(frame, bbox)

while True:
    # ... (Grab next frame)
    ret, bbox = tracker.update(frame) # Update tracker window
    # ... (Process after tracker update)
```

OpenCV Tracking API

- Tracker API in OpenCV-Python
  - Bounding box





OpenCV Tracking API

- Tracker parameters
  - For each tracking algorithm, there are several parameters
  - Tracker parameters cannot be changed in OpenCV-python
  - You can change when using OpenCV(C++)
- Nevertheless, the performance is better than histogram-based trackers.
  - Performance of trackers vary from case to case

OpenCV Tracking API

Let's Check the Code

tracking_3_trackers.py

- To run the script:

```
> python tracking_3_trackers.py ↵
```

Practice Test – OpenCV Tracker API

- Given a video of figure skating, track the woman figure skater.

1. Create an object tracker using `cv2.Tracker_create()` and initialize.
2. Update the tracker for every incoming frame using `tracker.update()`
3. Draw a bounding box of the detected object using `cv.rectangle()`

- `bbox = (290, 70, 70, 200)`

- Hint
  - `tracking_3_trackers.py`

Q & A

Thank You!

Reference

- OpenCV-Python Tutorials
  http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
- The object classification lecture is largely inspired by VLFeat tutorial
  http://vision.ucla.edu/~vedaldi/assets/pubs/vedaldi10vlfeat-tutorial.pdf
  http://www.vlfeat.org/applications/apps.html
- You can download the Jupyter Notebooks and the codes of this lecture at
  https://github.com/dalgu90/1705_sk_opencv