图像处理与识别

——Part 8 图像特征描述(II)

主讲：张磊

Main Content

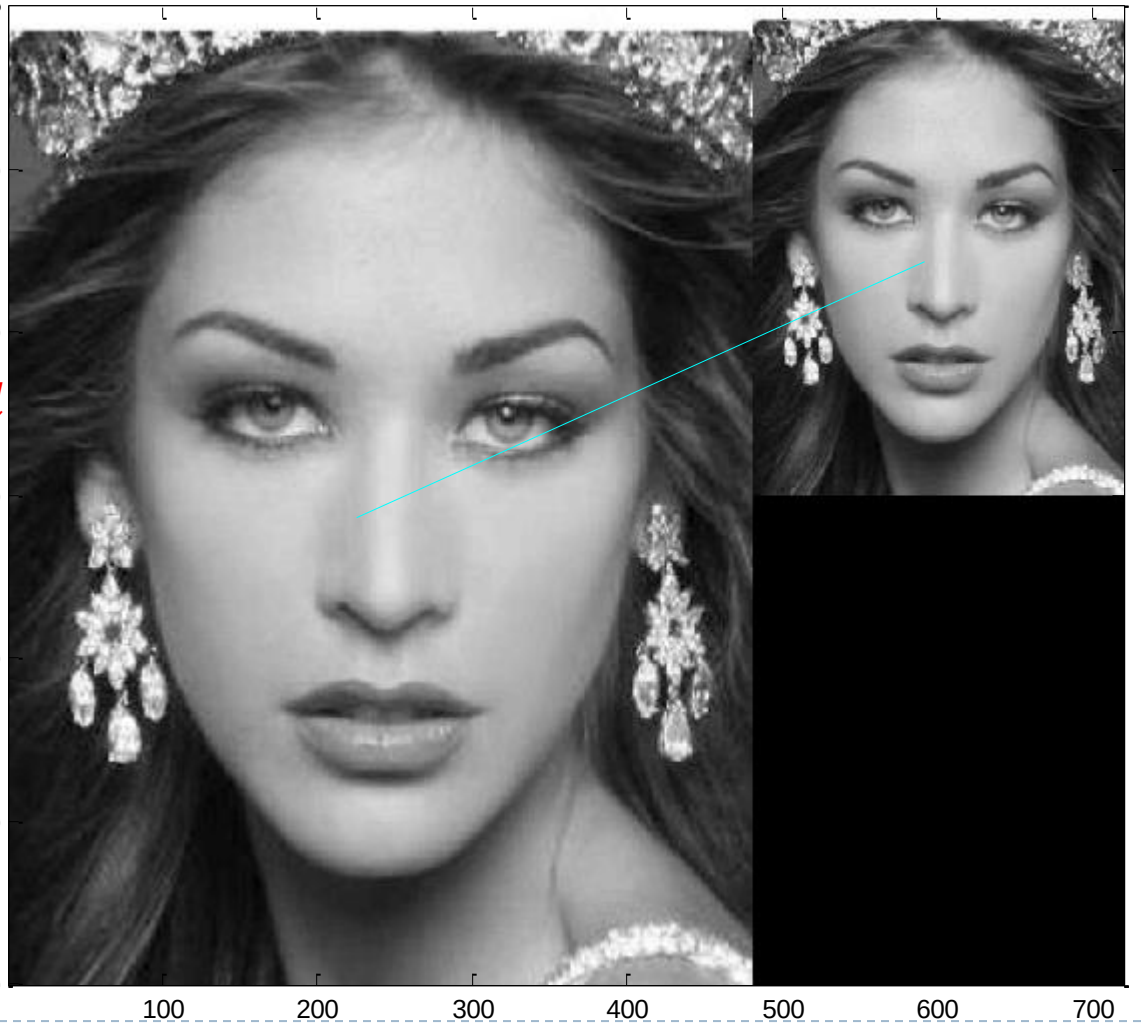
- Feature matching
- LBP descriptor
- HOG
- Color
- Shape

Feature Matching

- Two images

Feature Matching

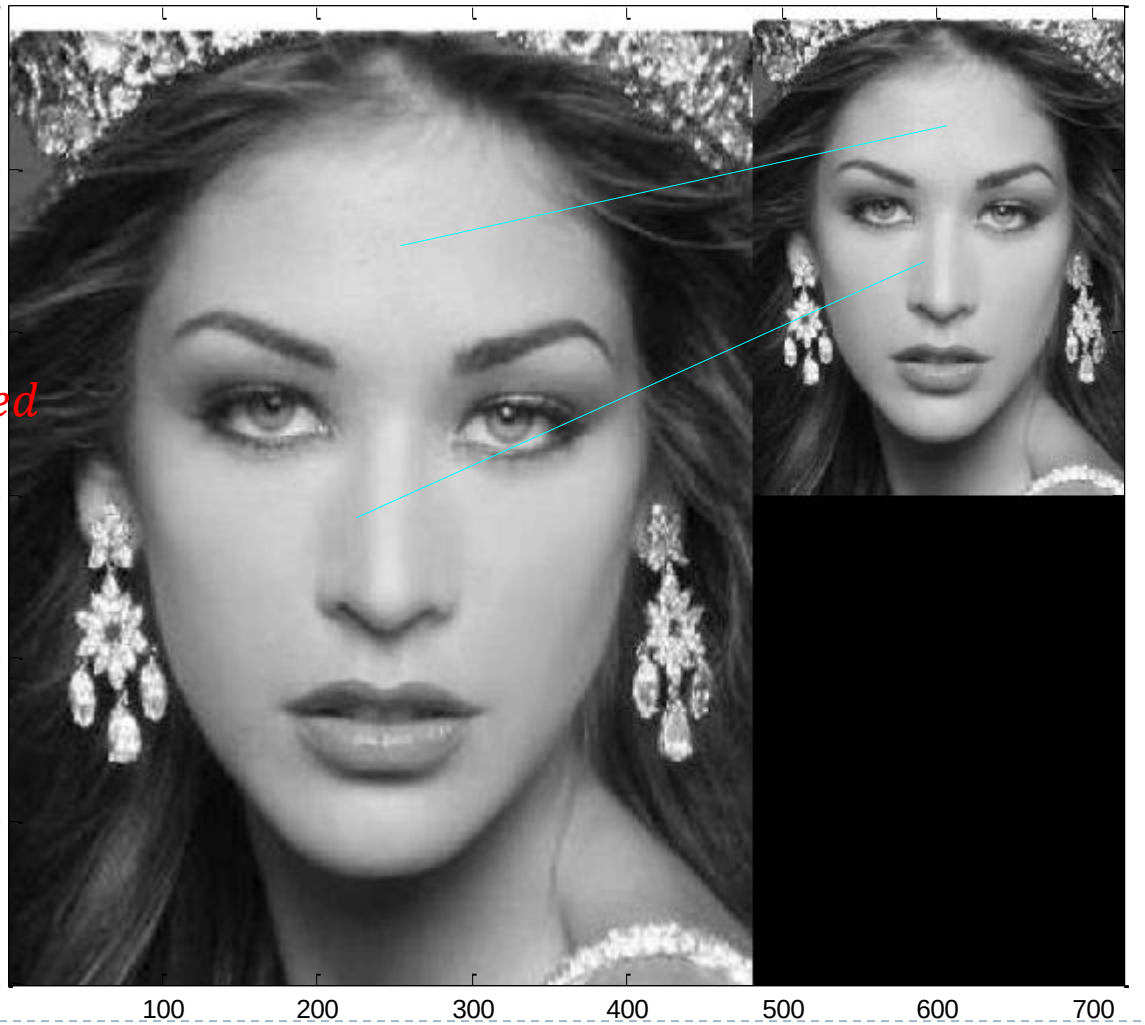
- Two images



one key point is matched

Feature Matching

- Two images



Two key points is matched

Feature Matching

- Two images

all matched key points

Feature Matching

- Key points of SIFT

1350 key points in the original image are detected based on SIFT descriptor.

423 key points in the scaled image are detected.

How to implement the accurate match between the two images?

Each key point is 128-dimensional (SIFT)

(16x16 patch around the key point)

Feature Matching

- K-NN

K-nearest neighbors is the most intuitive data mining method, without training.

When K=1, it becomes the nearest neighbor algorithm.

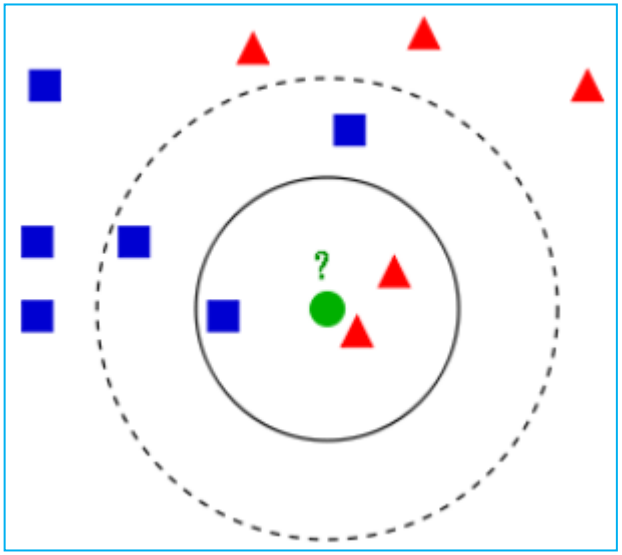
In other words, for a given **x**, the index is easily given by finding the nearest “distance”.

Generally, the identity of **x** depends on its k-nearest neighbors based on “majority” principle.

Feature Matching

- K-NN

Two classes with blue and red points are shown.



What is the green one?

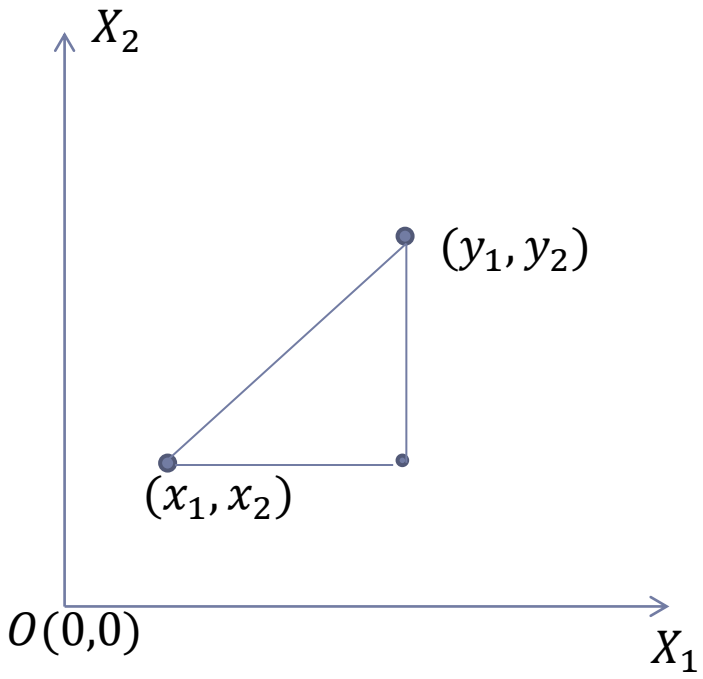
Feature Matching

- K-NN

To find the neighbors, it depends on the "distance" in high-dimensional data space. So, how to calculate the distance?

For a 2-dimensional Space (left)

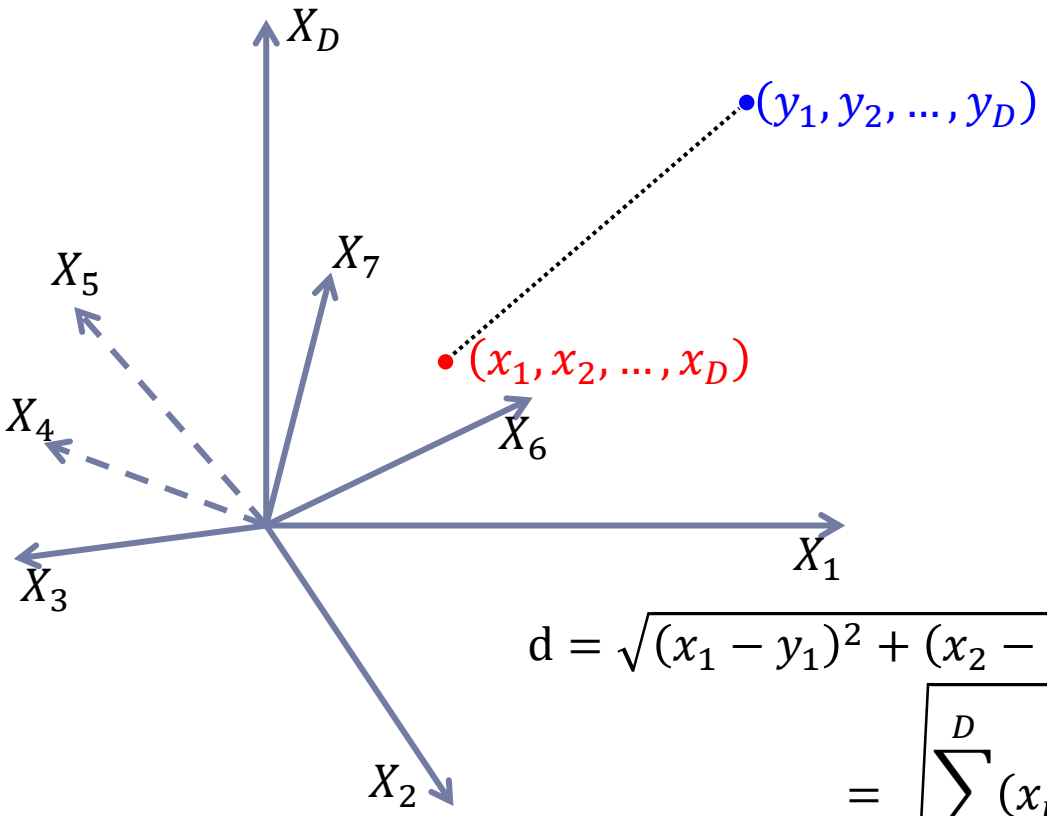
$$d=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$$



Feature Matching

- K-NN

In D-dimensional space, the distance between two points



Euclidean distance

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_D - y_D)^2}$$

$$= \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2}$$

Feature Matching

- K-NN

Distance function:

(1) Euclidean distance (欧式距离)

(2) Mahalanobis distance (马氏距离)

(3) Manhattan distance (曼哈顿距离)

(4) Hamming distance (汉明距离)

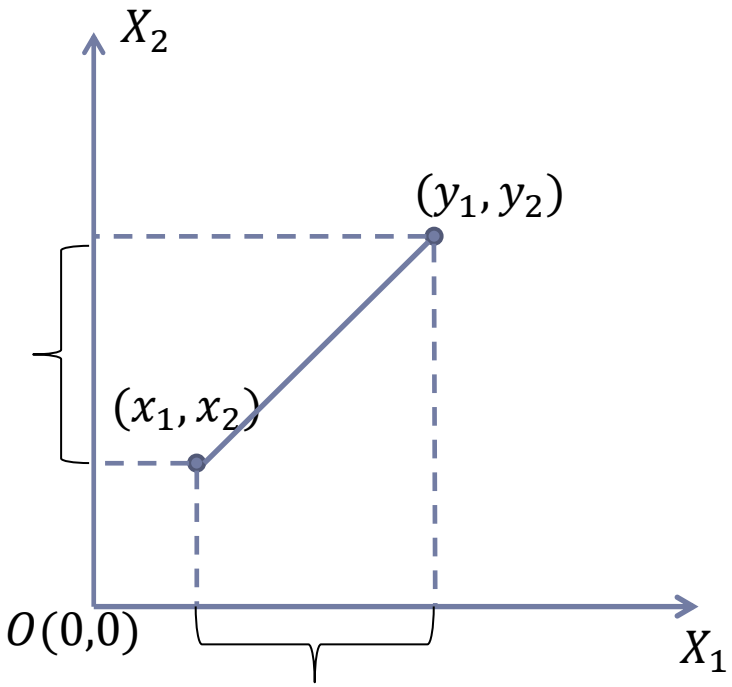
Feature Matching

- K-NN

Manhattan distance

Suppose a two-dimensional data space, there is

$$d=|x_1-y_1|+|x_2-y_2|$$



Feature Matching

- Kd-tree (k-dimensional tree) for KNN algorithm

K-nn is simple, intuitive, easily understood, but not efficient in large-scale retrieval.

Please imagine if there are 1 million images, with 10000 key points per image, how about the complexity?

To find out the nearest neighbors of a query quickly and accurately, spatial indexing structure and approximating algorithms are proposed.

Feature Matching

- Kd-tree for KNN algorithm

Idea: the data often shows clustering states, it is possible to construct data index, and then match quickly.

Index tree is a kind of tree structure index method, and divide the search space into multiple parts.

Kd-tree is clipping based, and R-tree is overlapping based.

Kd-tree is searching space partition based tree, the first step is to partition the space, then do search in each sub-space.

Feature Matching

- Kd-tree for KNN algorithm

Suppose there are six 2-dimensional data points,

$$\{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)\}$$

These points lie in a 2-dimensional space.

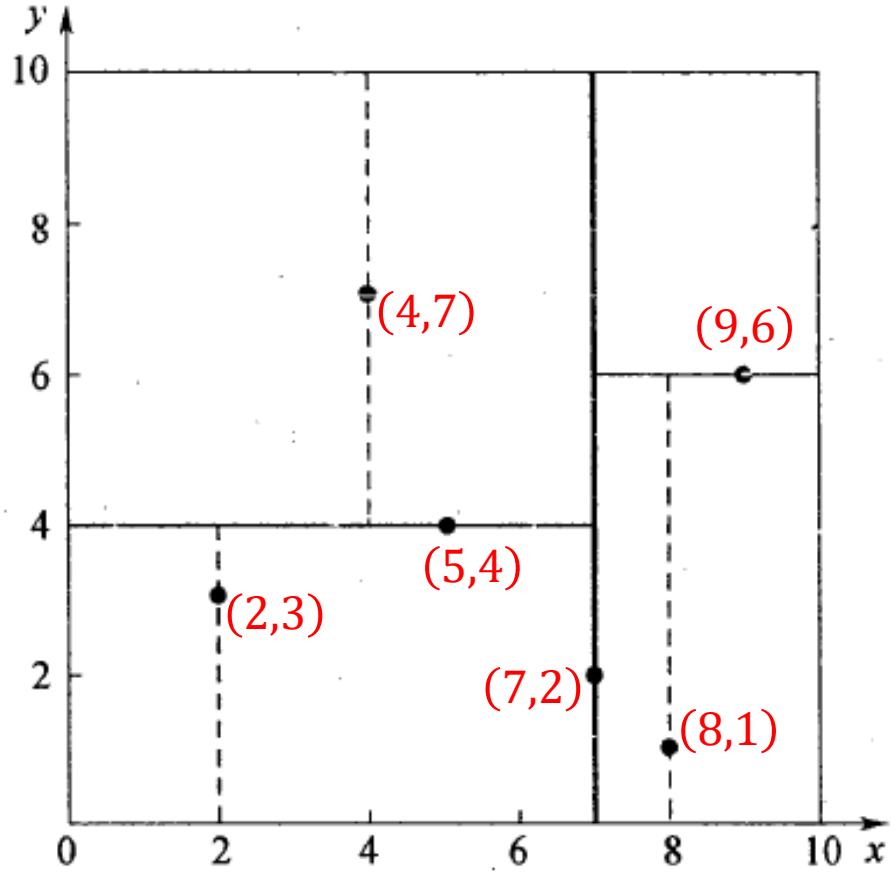
For a given query point (x,y), how to find its nearest neighbor?

Divide the space into several parts.

Feature Matching

- Kd-tree for KNN algorithm

{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)}

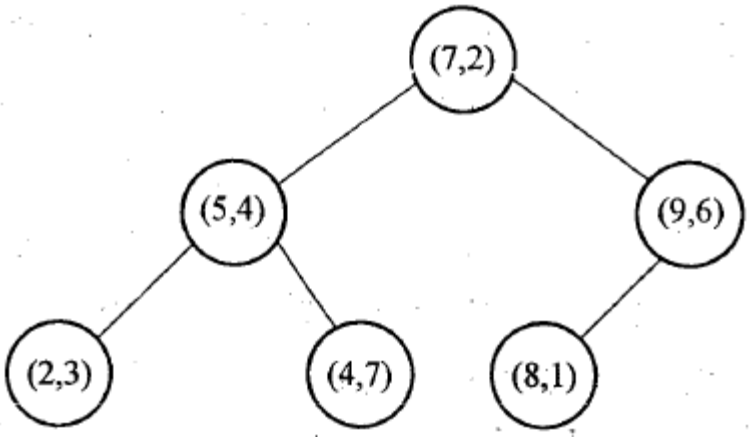


$Kd-tree\ space$

Feature Matching

- Kd-tree for KNN algorithm

{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)}



$Kd-tree$

Two variables:

split domain: *x or y axix*?

Node data: *which point?*

The segmentation hyperplane depends on the split and node data

Feature Matching

- Kd-tree for KNN algorithm

$$\{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)\}$$

The kd-tree construction process:

(1) Determine the split domain. Compute the variance of each dimension on all data points (X dim and Y dim). The dim w.r.t the maximum variance is the split domain.

(2) Determine the Node-data. Sort the data in split dim, the median value is the node data.

(3) Determine the segmentation hyperplane. Orthogonal to split domain, with x or y=node

(4) Repeat, until only one data point is included in a space.

The segmentation hyperplane depends on the split and node data

Feature Matching

- Kd-tree for KNN algorithm

$$\{(2,3);(5,4);(9,6);(4,7);(8,1);(7,2)\}$$

The kd-tree construction process:

(1) Determine the split domain. The variance X-dim and Y-dim is 6.97 and 5.37. Therefore, the split domain is X.

(2) Determine the Node-data. Sort the data in X-dim, (2, 4, 5, 7, 8, 9). the median value is 7, i.e. Node=7.

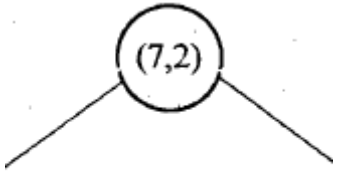
(3) Determine the segmentation hyperplane. Orthogonal to X, with x=7.

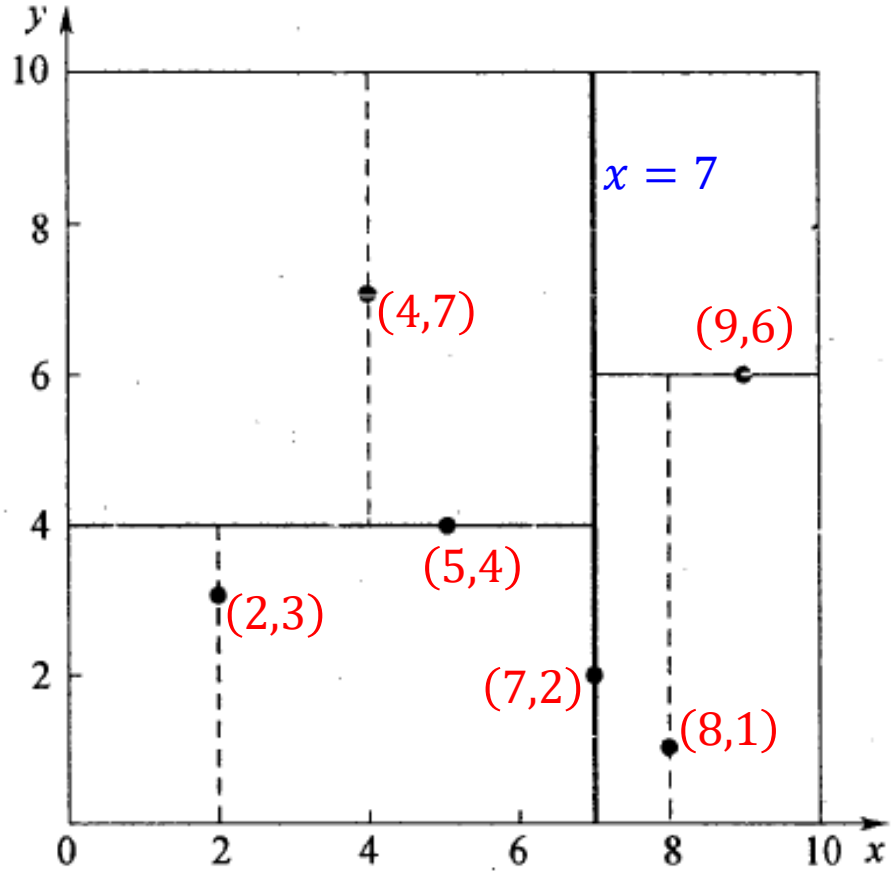
(4) Repeat, until only one data point is included in a space.

Feature Matching

- Kd-tree for KNN algorithm

{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)}



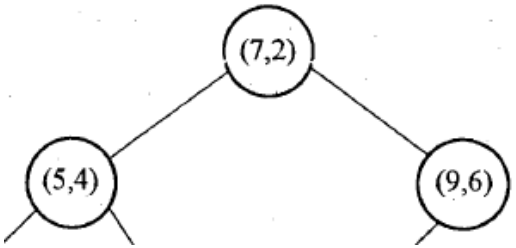


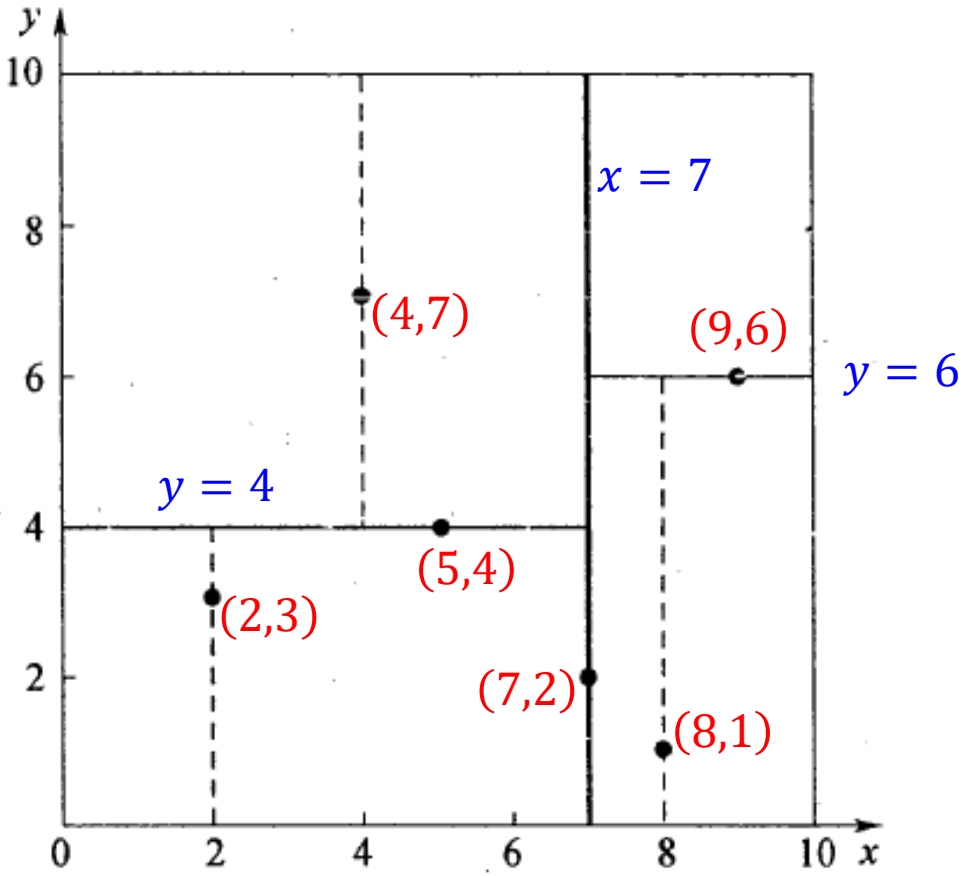
Kd − tree space

Feature Matching

- Kd-tree for KNN algorithm

{(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)}



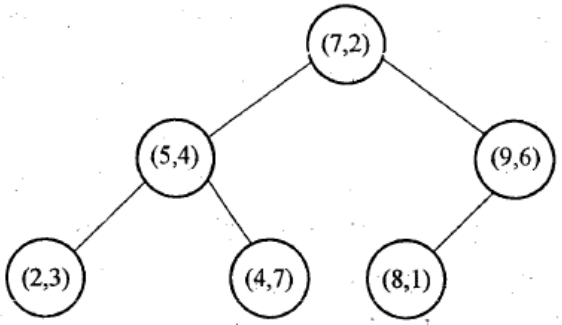


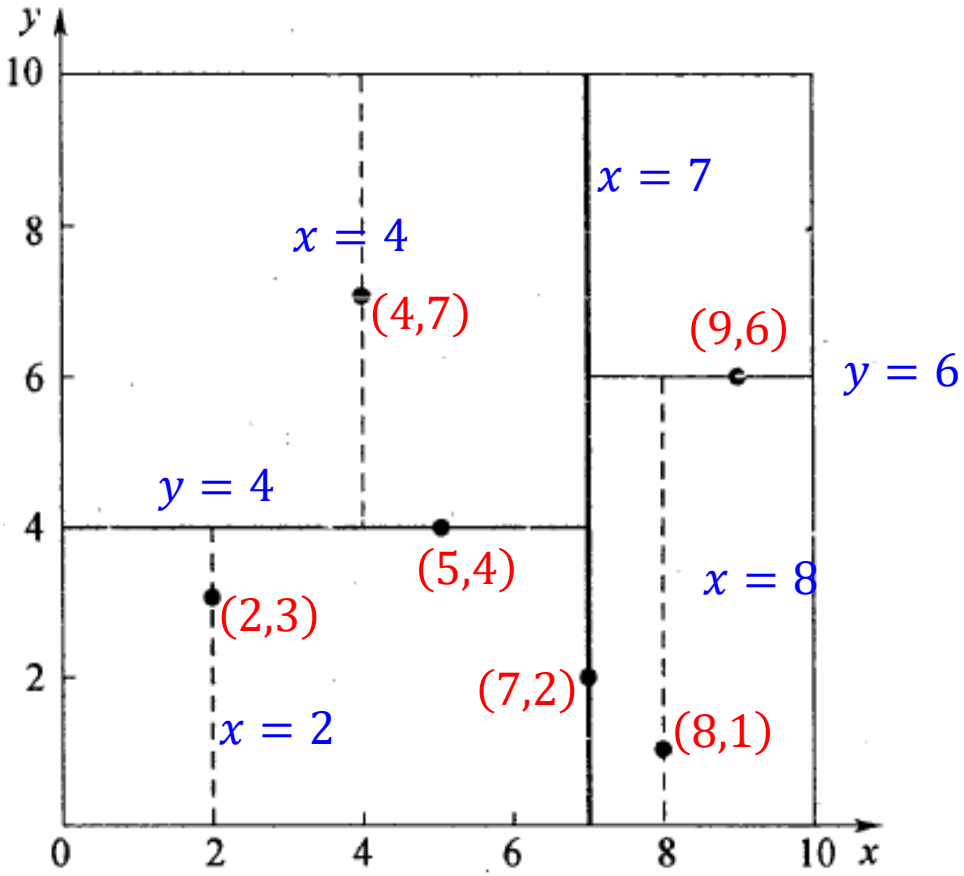
Kd − tree space

Feature Matching

- Kd-tree for KNN algorithm

  {(2,3); (5,4); (9,6); (4,7); (8,1); (7,2)}





$Kd-tree\ space$

Feature Matching

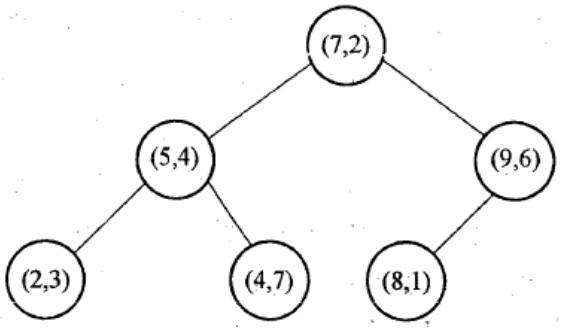
- Kd-tree for KNN algorithm

Given a query point (2.1,3.1), how to find its nearest point?

Step 1: Binary tree search. The start point is (7,2), then (5,4) and finally (2,3). The searching path is {(7,2),(5,4), (2,3)}

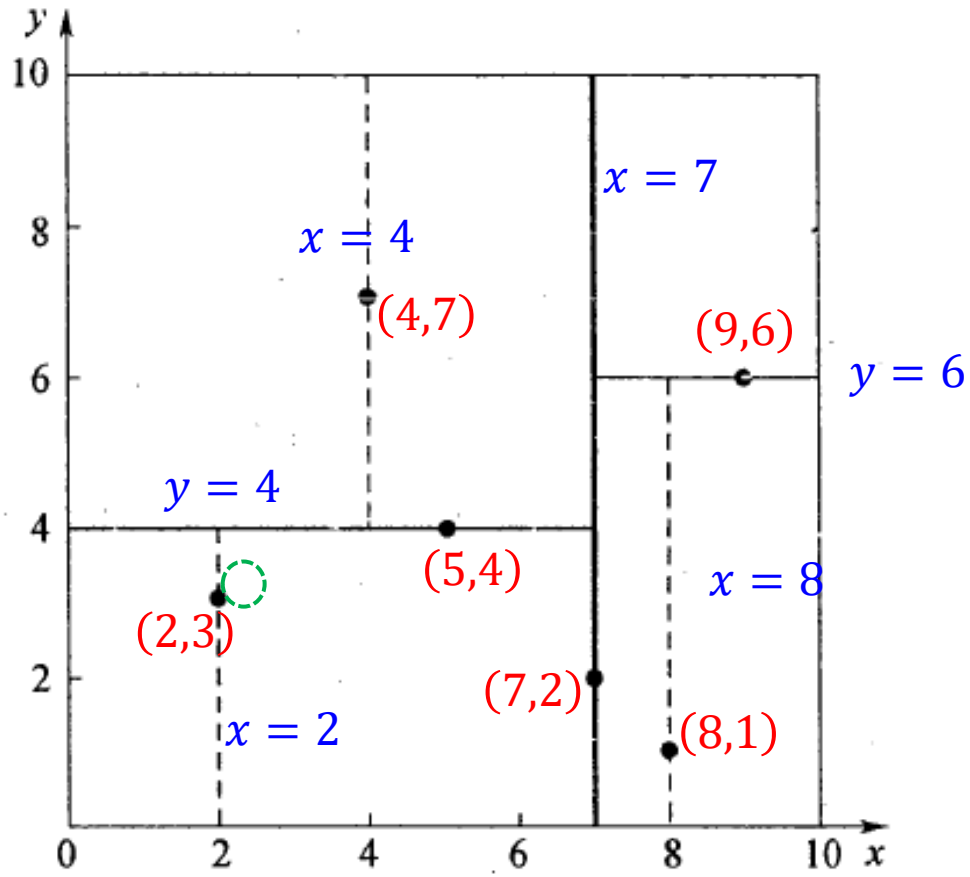
Step 2: Backtracking search. From (2,3), and go back to its farther point (5,4), search other space. Take (2.1,3.1) as center, make a circle with radius as 0.14.

Step 3: Backtracking search. Go back to (7,2). No overlapping.



Feature Matching

- Kd-tree for KNN algorithm



$Kd-tree\ space$

Feature Matching

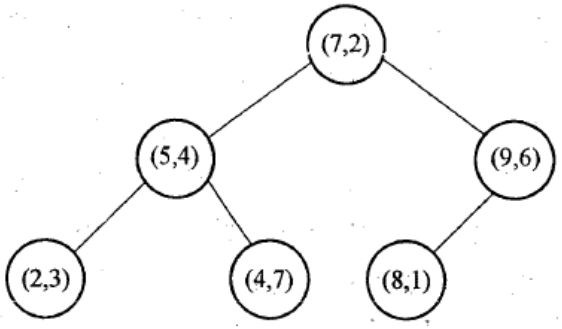
- Kd-tree for KNN algorithm

Given a query point (2,4.5), how to find its nearest point?

Step 1: Binary tree search. The start point is (7,2), then (5,4) and finally (4,7). The searching path is {(7,2),(5,4), (4,7)}

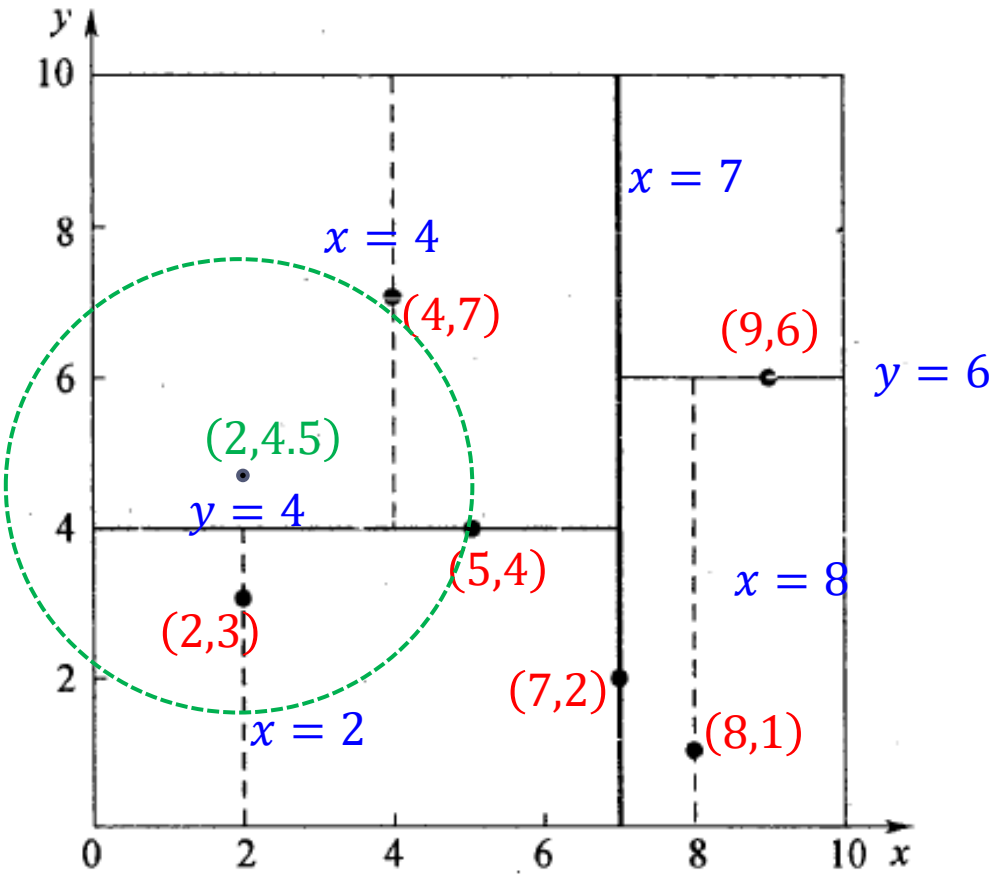
Step 2: Backtracking search. From (4,7), and go back to its farther point (5,4), search other space. Take (2,4.5) as center, make a circle with radius as 3.04 (<3.2). There is overlapping. Then, search (2,3) and the distance is 1.5<3.04

Step 3: Backtracking search. Go back to (7,2). Make circle with radius as 1.5, no Overlapping.



Feature Matching

- Kd-tree for KNN algorithm



$Kd-tree\ space$

Feature Matching

- Kd-tree for KNN algorithm

Not all nodes experience the distance computation with the query point.

Therefore, the complexity is largely reduced in nearest neighbor search.

David Lowe also proposed an improved Kd-tree search algorithm, i.e. BBF (Best-Bin-First)

Feature Matching

- David Lowe’ matching method in SIFT

To avoid the unstable key points, such as complex background, shelter etc. Lowe proposed to compare the nearest neighbor and the secondary nearest neighbor.

近邻与次近邻的距离比 ratio

For the two key points, if the ratio between the nearest distance and the secondary nearest distance is smaller than a threshold T, match.

$$if \quad \frac{d_1}{d_2} < T, match$$

Feature Matching

- David Lowe’ matching method in SIFT

Generally, a larger ratio corresponds to more inaccurate matches.

If the threshold T is reduced, the number of successfully matched key points of SIFT will also decrease, but more stable.

Lowe suggests a ratio threshold 0.8. With more experience of authors, based on a number of scale, rotation, illumination of two images, the best ration threshold is 0.4~0.6.

Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)

SPM, as an improvement of BOVW model, can fully use the local image information. BOW的缺点是忽略了空间位置信息，且无序

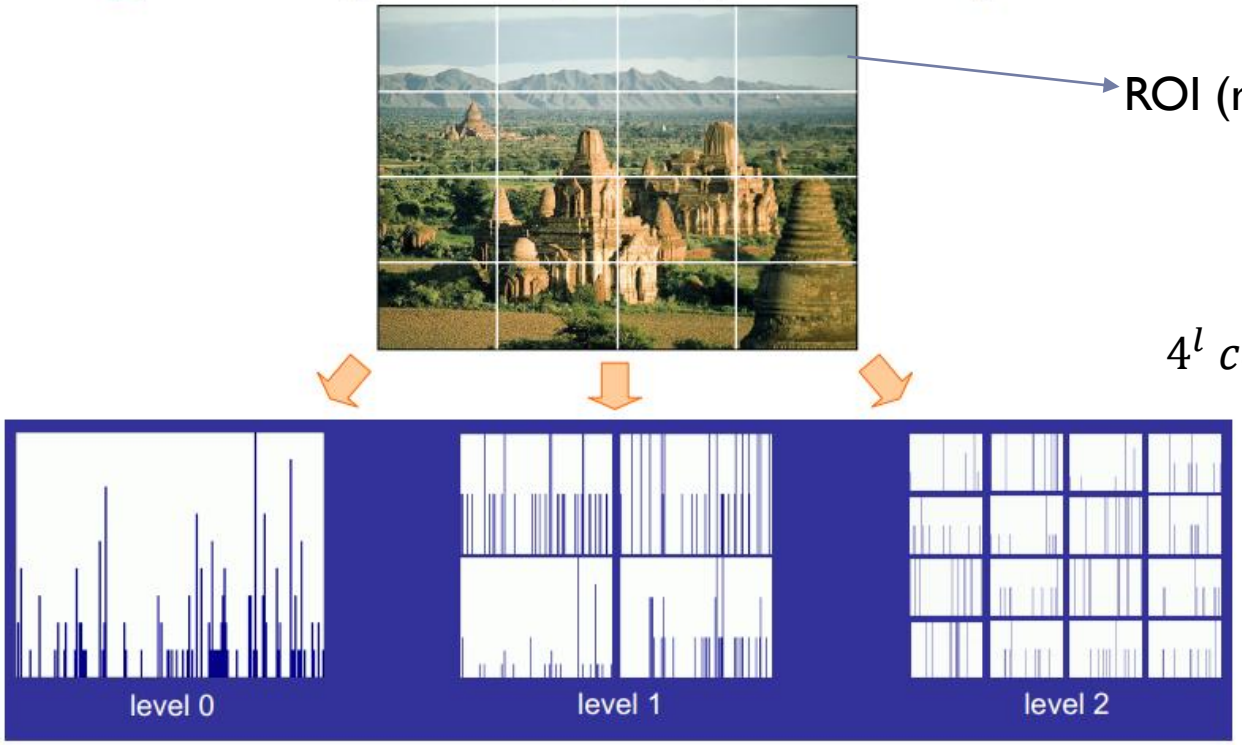
SPM computes the feature key points under different resolutions, such that the local information is captured.

UIUC伊利诺伊大学
一香槟分校

S. Lazebink, C. Schmid, J. Ponce, Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories, CVPR'06.

Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)
  - Extension of a bag of features
  - Locally orderless representation at several levels of resolution
  - Based on *pyramid match kernels* Grauman & Darrell (2005)
    - Grauman & Darrell: build pyramid in feature space, discard spatial information
    - Our approach: build pyramid in image space, quantize feature space



Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)

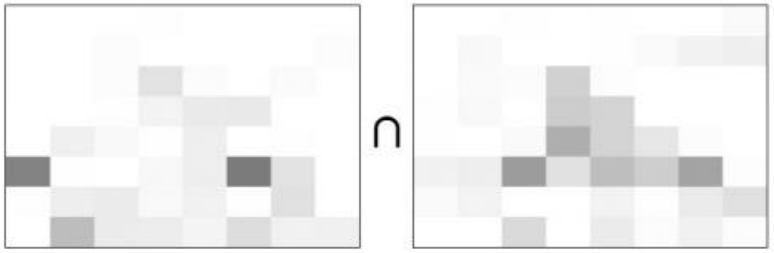
Find maximum-weight matching (weight is inversely proportional to distance)

Original images

如何计算两幅图的匹配程度？

Feature histograms:

Level 3



$= \mathcal{I}_3$

Histogram intersection

$\mathcal{I}^\ell$

$$\mathcal{I}(H_X^\ell, H_Y^\ell) = \sum_{i=1}^{D} \min\left(H_X^\ell(i), H_Y^\ell(i)\right)$$

Level 2

$= \mathcal{I}_2$

Level 1

$= \mathcal{I}_1$

Level 0

$= \mathcal{I}_0$

Word m

Total weight (value of *pyramid match kernel*): $\mathcal{I}_3 + \frac{1}{2}(\mathcal{I}_2 - \mathcal{I}_3) + \frac{1}{4}(\mathcal{I}_1 - \mathcal{I}_2) + \frac{1}{8}(\mathcal{I}_0 - \mathcal{I}_1)$

尺度权重：$\frac{1}{2^{L-\ell}}$

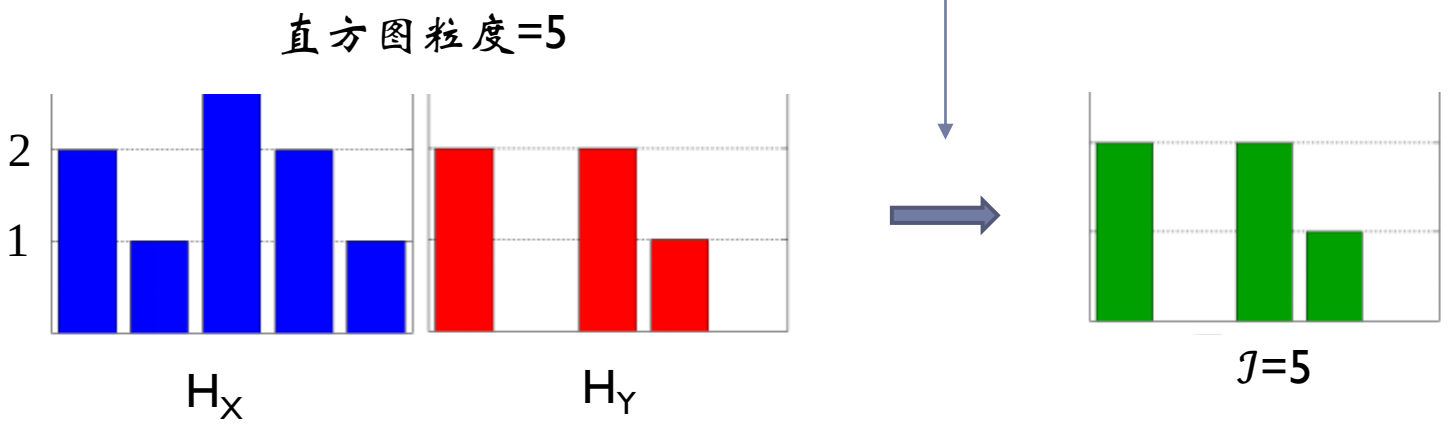
D: bins/cells数目； $H_X(i)$ 图像X的点落入第i个cell的数量

Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)

Histogram intersection（直方图相交/直方图交叉算子）

$$\mathcal{I}(H_X^\ell, H_Y^\ell) = \sum_{i=1}^{D} \min\left(H_X^\ell(i), H_Y^\ell(i)\right)$$



Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)

Find maximum-weight matching (weight is inversely proportional to distance)

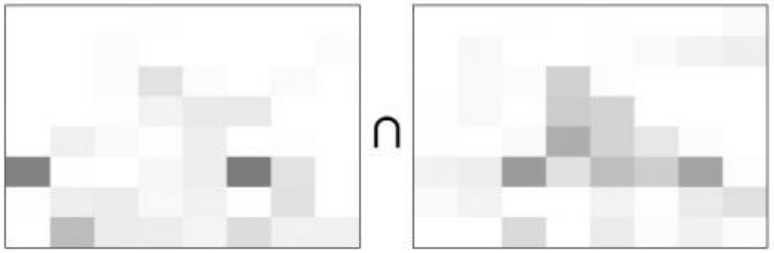
Original images

如何计算两幅图的匹配程度？

BOW直方图

Feature histograms:

Level 3



$= \mathcal{I}_3$

Histogram intersection

$\mathcal{I}^\ell$

$$\mathcal{I}(H_X^\ell, H_Y^\ell) = \sum_{i=1}^{D} \min\left(H_X^\ell(i), H_Y^\ell(i)\right)$$

Level 2

$= \mathcal{I}_2$

Level 1

$= \mathcal{I}_1$

Level 0

$= \mathcal{I}_0$

Pyramid Match Kernel:

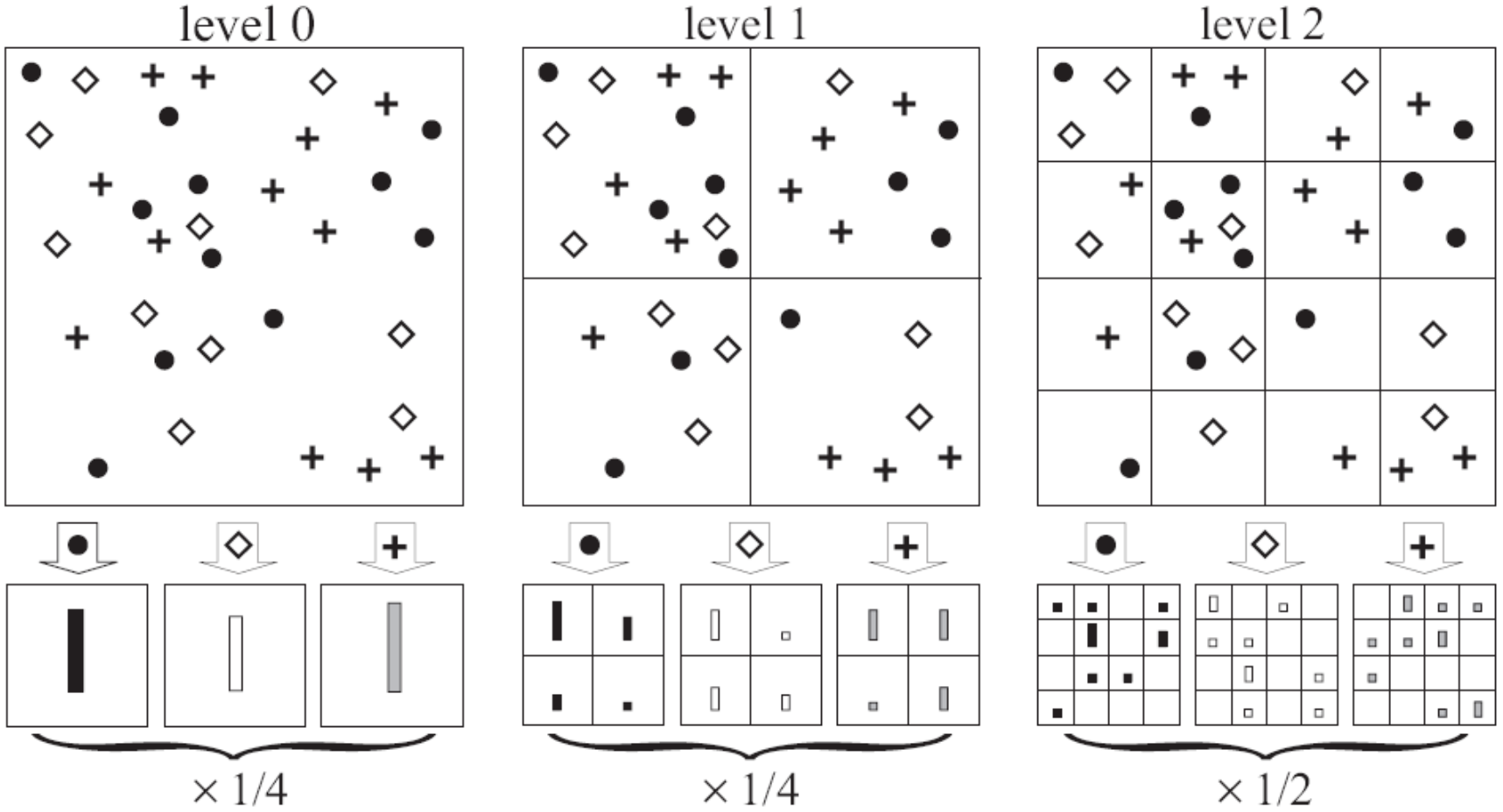
$$\kappa^L(X,Y) = \mathcal{I}^L + \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}}\left(\mathcal{I}^\ell - \mathcal{I}^{\ell+1}\right)$$

$$= \frac{1}{2^L}\mathcal{I}^0 + \sum_{\ell=1}^{L} \frac{1}{2^{L-\ell+1}}\mathcal{I}^\ell .$$

Word m

Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)



举例：假设有3个视觉单词(聚类中心)

Feature Matching

- Spatial Pyramid Matching (SPM, 空间金字塔匹配)

For *L* levels, the final kernel is the sum of each channel kernel, (M is the number of words by K-means)

$$K^L(X,Y) = \sum_{m=1}^{M} \kappa^L(X_m, Y_m)$$

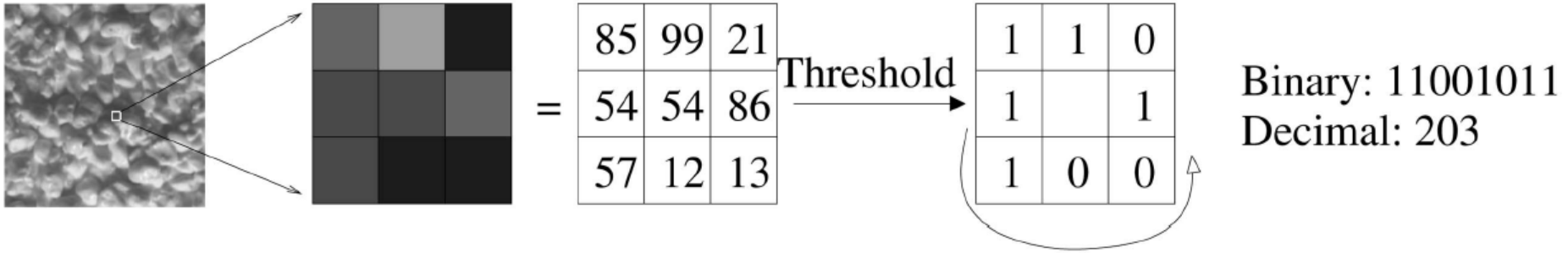
The final feature dimension by concatenating the appropriately weighted histograms of all channels at all resolutions is

$$M \sum_{\ell=0}^{L} 4^{\ell} = M\frac{1}{3}(4^{L+1} - 1)$$

Local Binary Pattern-Texture descriptor

- LBP


LBP operator was originally designed by Ojala 1994 for texture description. By thresholding a 3x3 neighborhood of each pixel with the center pixel, a binary code is obtained.



The basic LBP operator

Local Binary Pattern-Texture descriptor

- LBP

original image

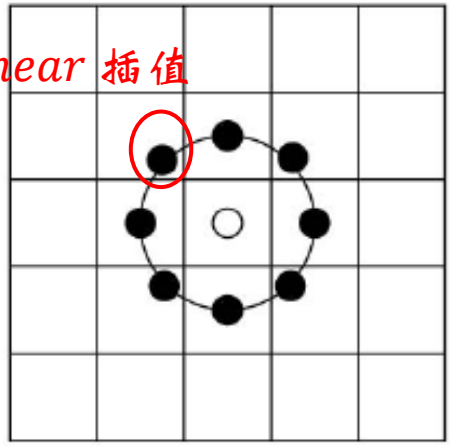
LBP image

Local Binary Pattern-Texture descriptor

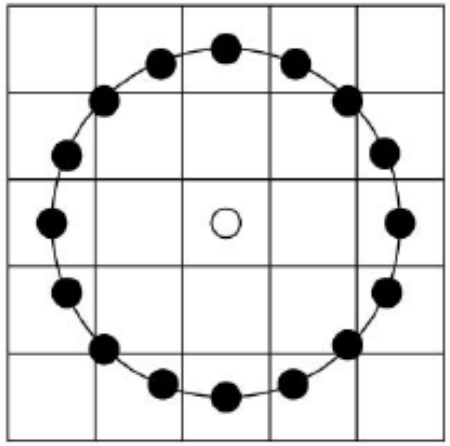
- LBP

To be able to deal with textures at different scales, LBP operator was extended to use neighborhoods of different sizes, evenly spaced on a circle centered at the pixel. (e.g. radius R and the number P of sampling points).

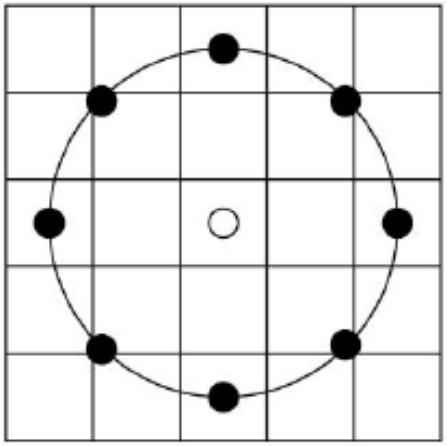
binlinear 插值



R=1, P=8



R=2, P=16



R=2, P=8

Local Binary Pattern-Texture descriptor

- Uniform LBP (等价模式)

With the increasing number P of sampling points, the number of LBP values (decimal), i.e. 2^P is too large. (降维)

Therefore, uniform LBP is proposed, i.e. the binary pattern contains at most two bitwise transitions from 0 to 1, or 1 to 0.

For example,

00000000 → 0 transition (0次跳变)
00001111 → 1 transition (1次跳变)
00110000 → 2 transition (2次跳变)

Uniform binary pattern

00110001 → 3 transition (3次跳变)
00110100 → 4 transition (4次跳变)

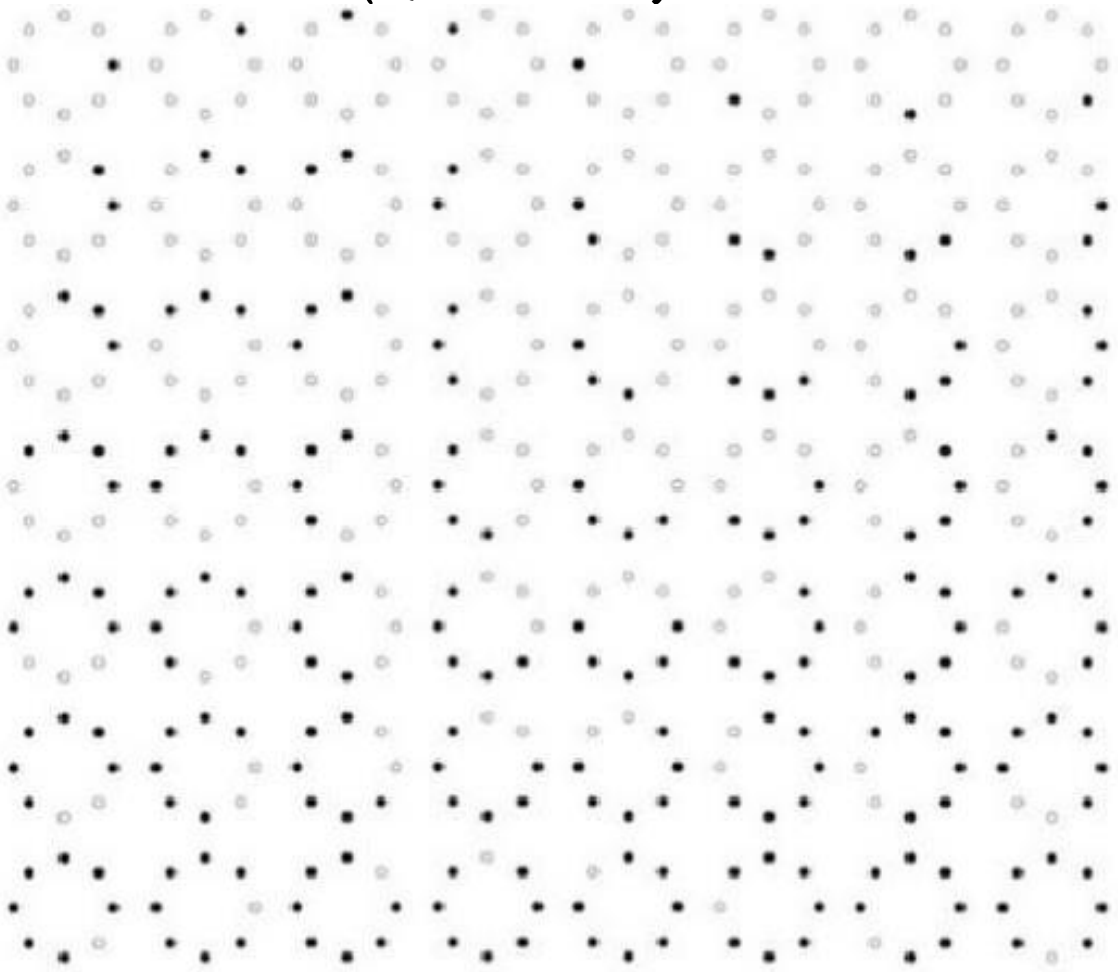
Non − uniform binary pattern

For 3x3, P=8; For 5x5, P=24

Local Binary Pattern-Texture descriptor

- Uniform LBP (等价模式)

56种



$Non-uniform\ pattern$
$is\ viewed\ as\ one\ bin$

$Pattern\ number$: $2^P \rightarrow P\ (P-1)+2+1;\ 256 \rightarrow 59$

Local Binary Pattern-Texture descriptor

- Rotation invariant LBP (旋转不变LBP)

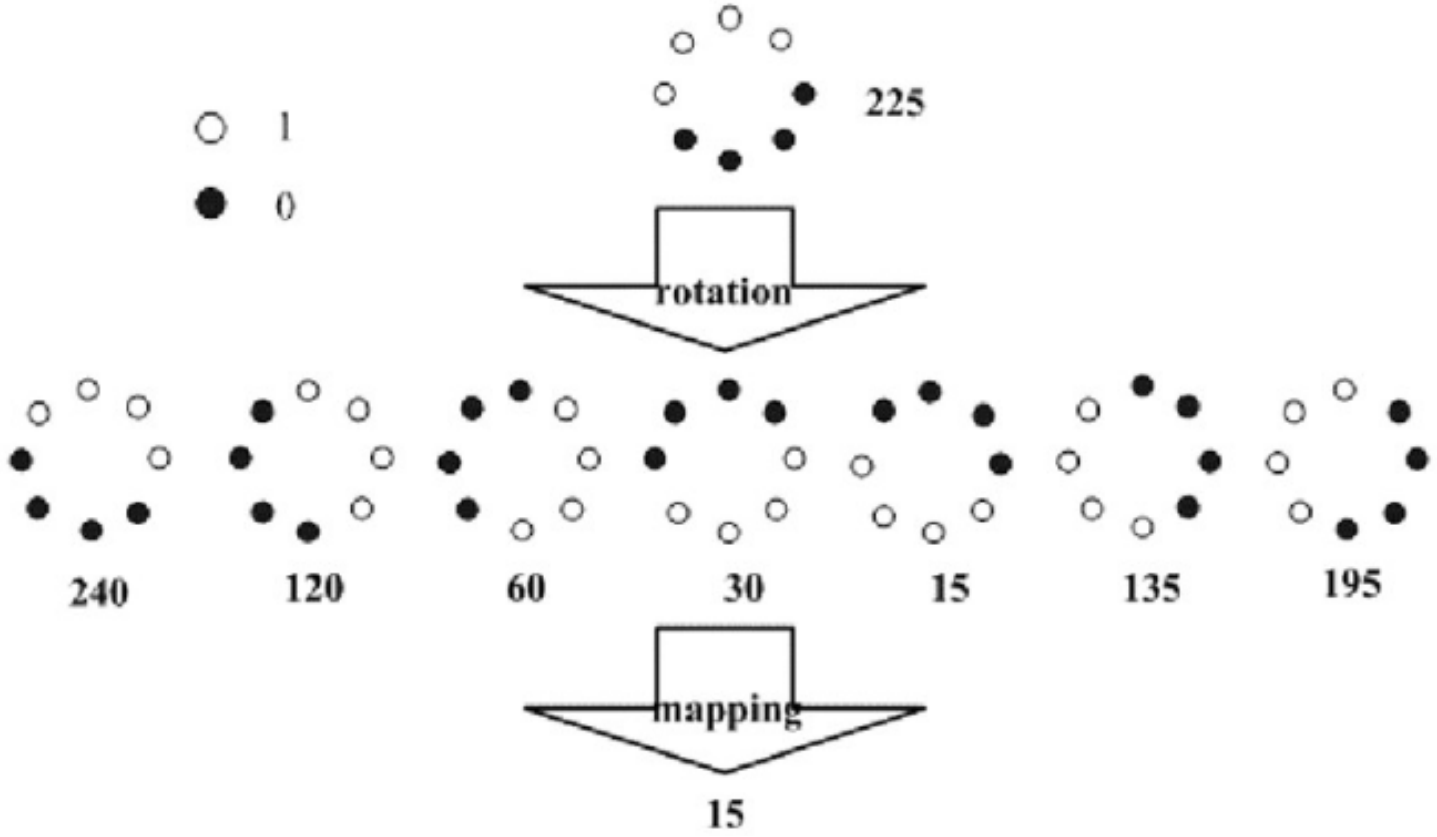
Obviously, LBP is gray scale invariant，但非旋转不变.

How to achieve rotation invariant?

$$LBP^{ri}_{P,R} = min\{ROR(LBP_{P,R}, i) \quad | \quad i = 0, 1, \ldots, \mathrm{P} - 1\}$$

Local Binary Pattern-Texture descriptor

- Rotation invariant LBP (旋转不变LBP)



$$LBP^{ri}_{P,R} = min\{ROR(LBP_{P,R}, i) \quad | \quad i = 0, 1, \ldots, \mathrm{P} - 1\}$$

Local Binary Pattern-Texture descriptor

- LBP feature representation

original image

LBP image

Uniform LBP image

Ojala find that the uniform patterns occupy 90% of the image

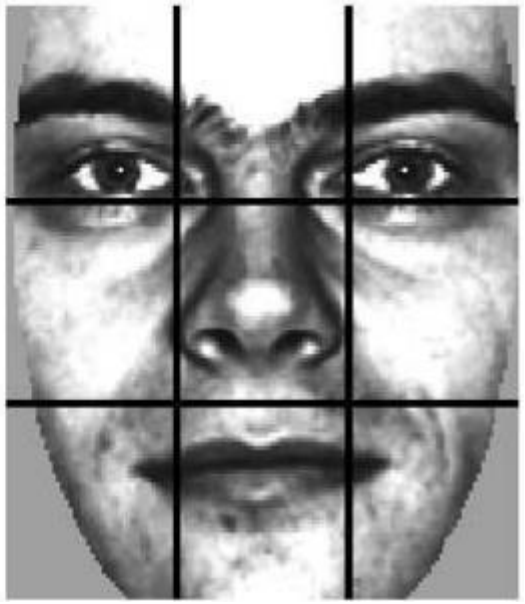
Local Binary Pattern-Texture descriptor

- LBP feature representation

LBP feature representation of an image:

Step 1: Subdivide the image into multiple patches



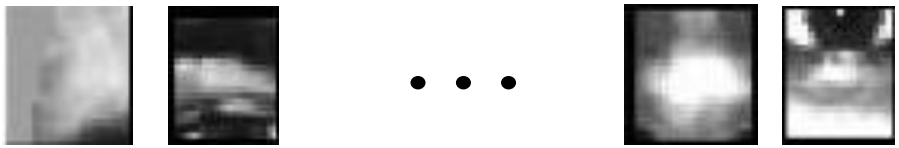


Local Binary Pattern-Texture descriptor

- LBP feature representation

LBP feature representation of an image:

Step 2: Take 3x3 neighborhood on each patch, and extract LBP binary code for each pixel, and obtain the LBP-value (decimal).



Step 3: Obtain the histogram of each patch with LBP-value.

Step 4: Concatenate the histogram vector of all patches together, and a long feature vector is formed (i.e. LBP vector of an image)

Histogram of Oriented Gradient

- HOG

HOG was first proposed by Dalal and Triggs, in CVPR'2005 for human detection.

The basic idea is that local object appearance and shape can often be characterized very well by the distribution of local intensity gradients or edge directions.

N. Dalal, Histograms of Oriented Gradients for Human Detection, IEEE CVPR'05.

Histogram of Oriented Gradient

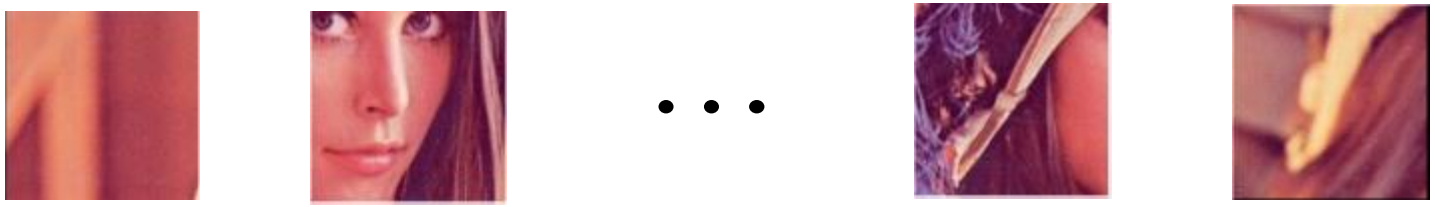
- HOG

(1) **Cell** division of the image (small spatial regions). The size of each cell (non-overlap) is 8*8 pixel.

Histogram of Oriented Gradient

- HOG

(2) Compute the gradient image and gradient directions of each cell based on some gradient operator (正交梯度算子或方向梯度算子).



$$M(x,y) = \sqrt{I_x^2 + I_y^2}$$

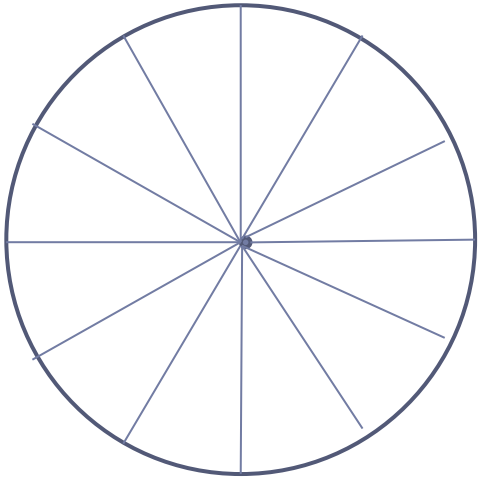
$$\theta(x,y) = tan^{-1}\left(\frac{I_y}{I_x}\right)$$

Histogram of Oriented Gradient

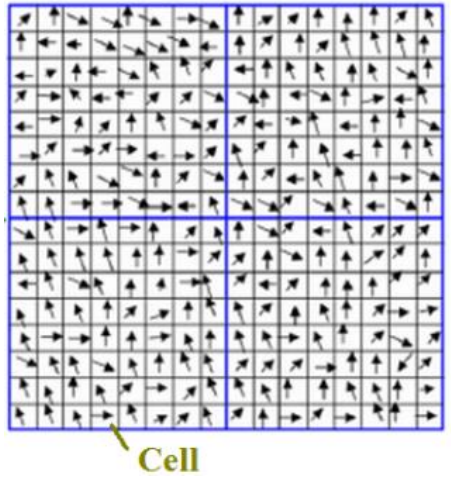
- HOG

(3) Cell Orientation Histogram

Suppose there are 12 bins, with each of 30°. For each cell,



统计每个像素点的梯度方向所属的bin

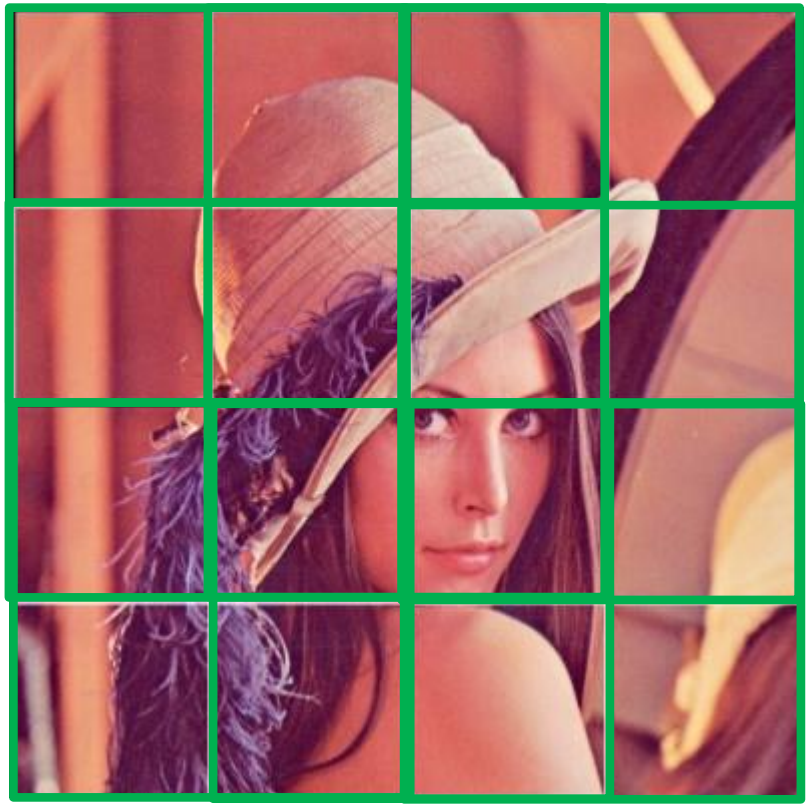


Compute the statistics in each bin by using weighted $\theta(x, y)$. The weight is $M(x, y)$. 12-dimensional vector is achieved.

Histogram of Oriented Gradient

- HOG

(3) Block formulation by 2*2 cells by overlapping.



Block normalization *to reduce the effect of contrast changes, illumnation,* where **b** is the block feature vector by concatenating each cell

$$\mathbf{b} \leftarrow \frac{\mathbf{b}}{\sqrt{\|\mathbf{b}\|^2 + \epsilon}}$$

Histogram of Oriented Gradient

- HOG

(4) HOG Feature formulation.

Each cell is 12-dimensional vector.

Each block is 2*2 cell, forms a 48-dimensional vector.

In the Lena image, 9 overlapping blocks are achieved, and therefore 9*48=432 dimensional HOG feature vector **h** is obtained.

$$\mathbf{h} \leftarrow \frac{\mathbf{h}}{\sqrt{\|\mathbf{h}\|^2 + \epsilon}}$$

$$h_n \leftarrow \min(h_n, \tau)$$

$$\mathbf{h} \leftarrow \frac{\mathbf{h}}{\sqrt{\|\mathbf{h}\|^2 + \epsilon}}$$

Histogram of Oriented Gradient

- PHOG (Pyramid HOG)

HOG进一步延伸，利用金字塔原理，考虑不同尺度的HOG，能够检测不同尺度的特征。计算复杂度更高。

自行学习！