Feature Extraction – COMP4423 Computer Vision

Xiaoyong Wei (魏驍勇)

x1wei@polyu.edu.hk

Department of Computing
電子計算學系

The Hong Kong Polytechnic University
香港理工大學

Opening Minds • Shaping the Future
啟迪思維 • 成就未來

A New Toy

A New Toy

Outline

- Features and Quantization
- Metrics (Distance and Similarity)
- Global and Local Features (Color Histograms, LBP, SIFT)

How do you group them?

It's easy!

	Yellow Hair, Blue Eyes, White Skin
	Black Hair, Brown Eyes, Yellow Skin
	Black Hair, Brown Eyes, Yellow
	Yellow Hair, Brown Eyes, White Skin
	Black Hair, Brown Eyes, Black Skin

It's easy!

	Hair Color	Skin Color
	Yellow	White
	Black	Yellow
	Black	Yellow
	Yellow	White
	Black	Black

This is how human see it.

Is there a way for computers to "see" it?

Quantization

	Hair Color	Skin Color
	Yellow (2)	White (1)
	Black (3)	Yellow (2)
	Black (3)	Yellow (2)
	Yellow (2)	White (1)
	Black (3)	Black (3)

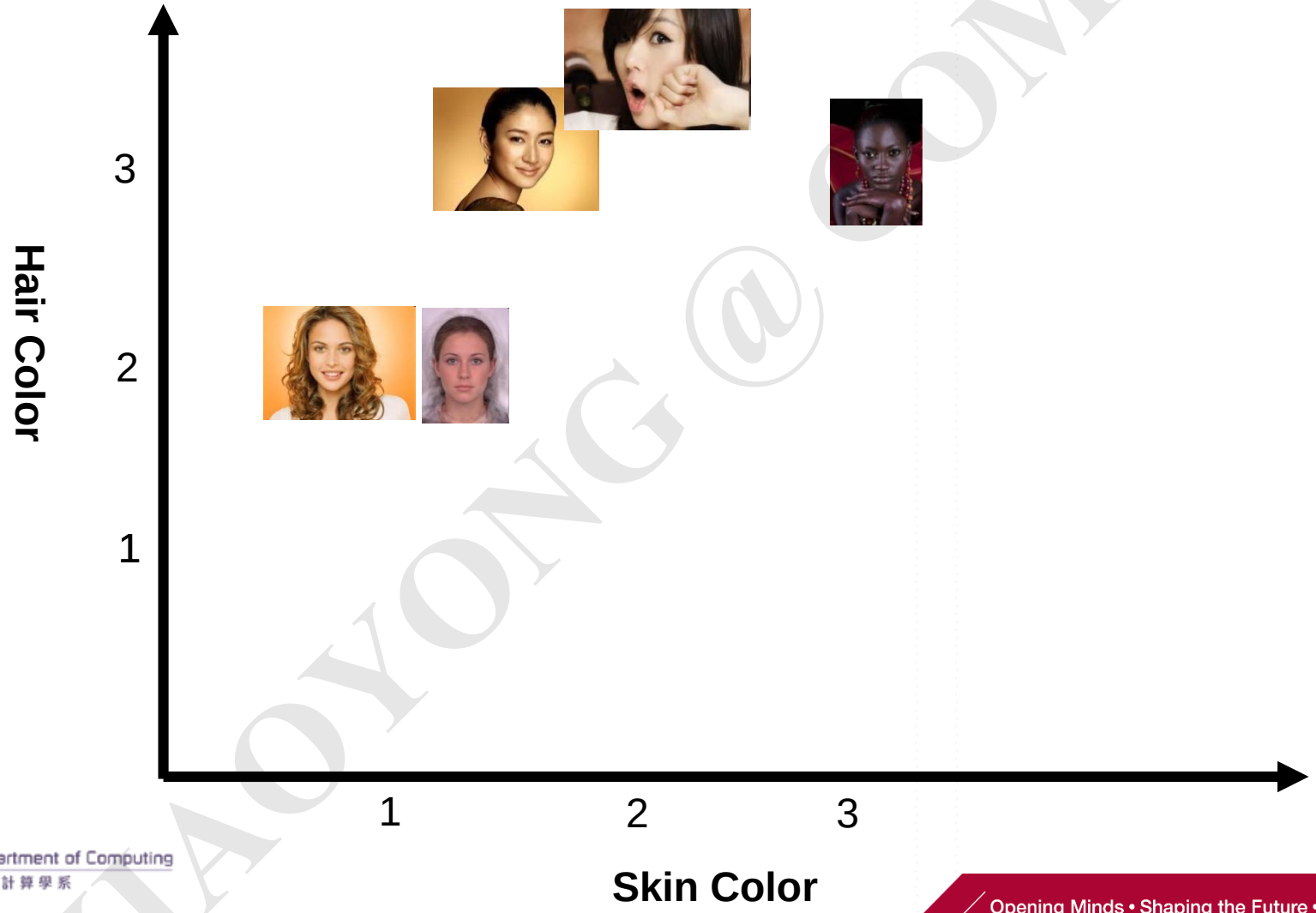
Great! Seems that now we can build grouping rules easily using if-else?

Right and wrong!

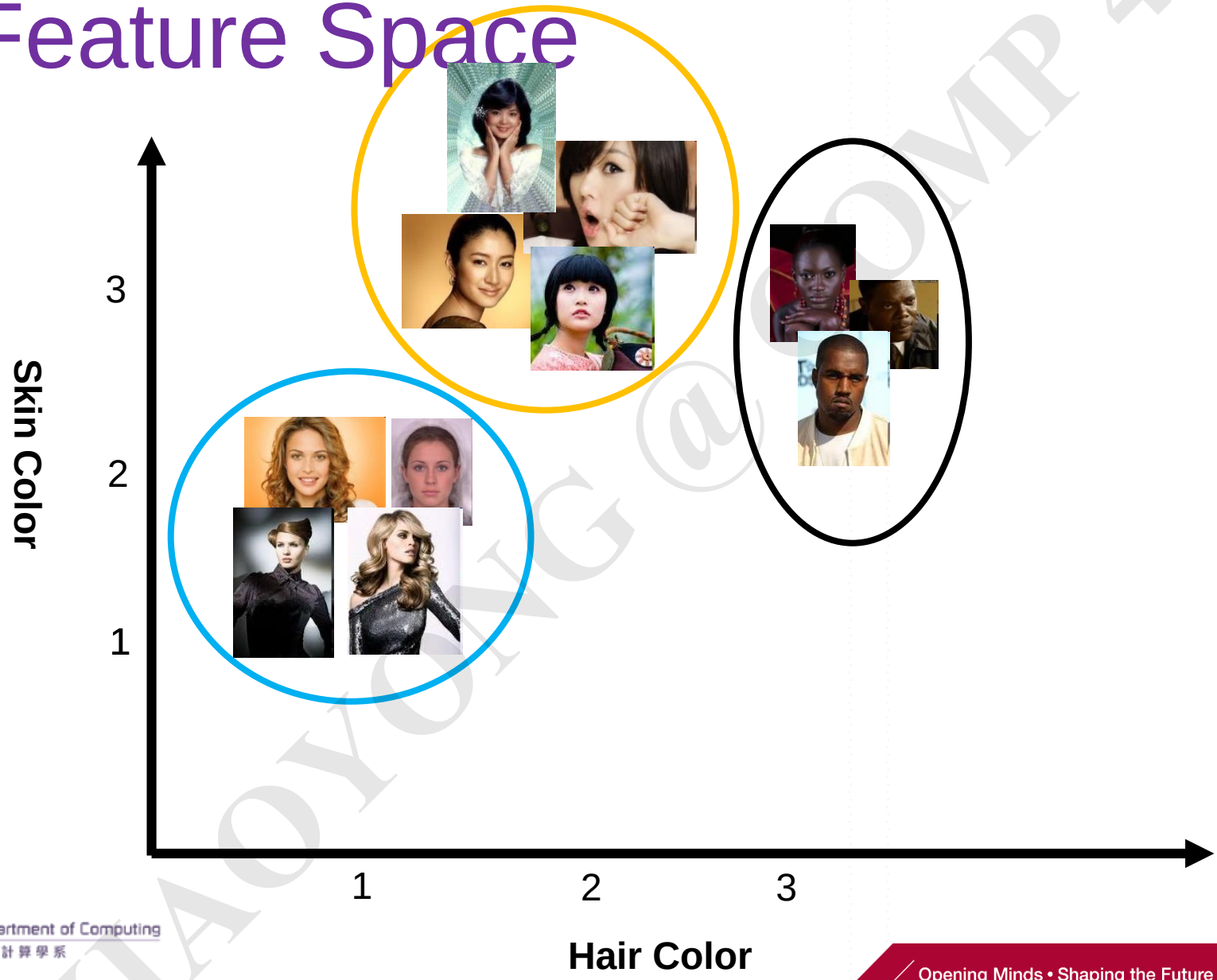
Feature Vectors - Quantization

	Hair Color	Skin Color
	Yellow **(2.135)**	White **(1.05)**
	Black **(3.324)**	Yellow **(2.008)**
	Black **(2.945)**	Yellow **(2.355)**
	Yellow **(2.035)**	White **(1.035)**
	Black **(2.994)**	Black **(3.208)**

Feature Space



Feature Space



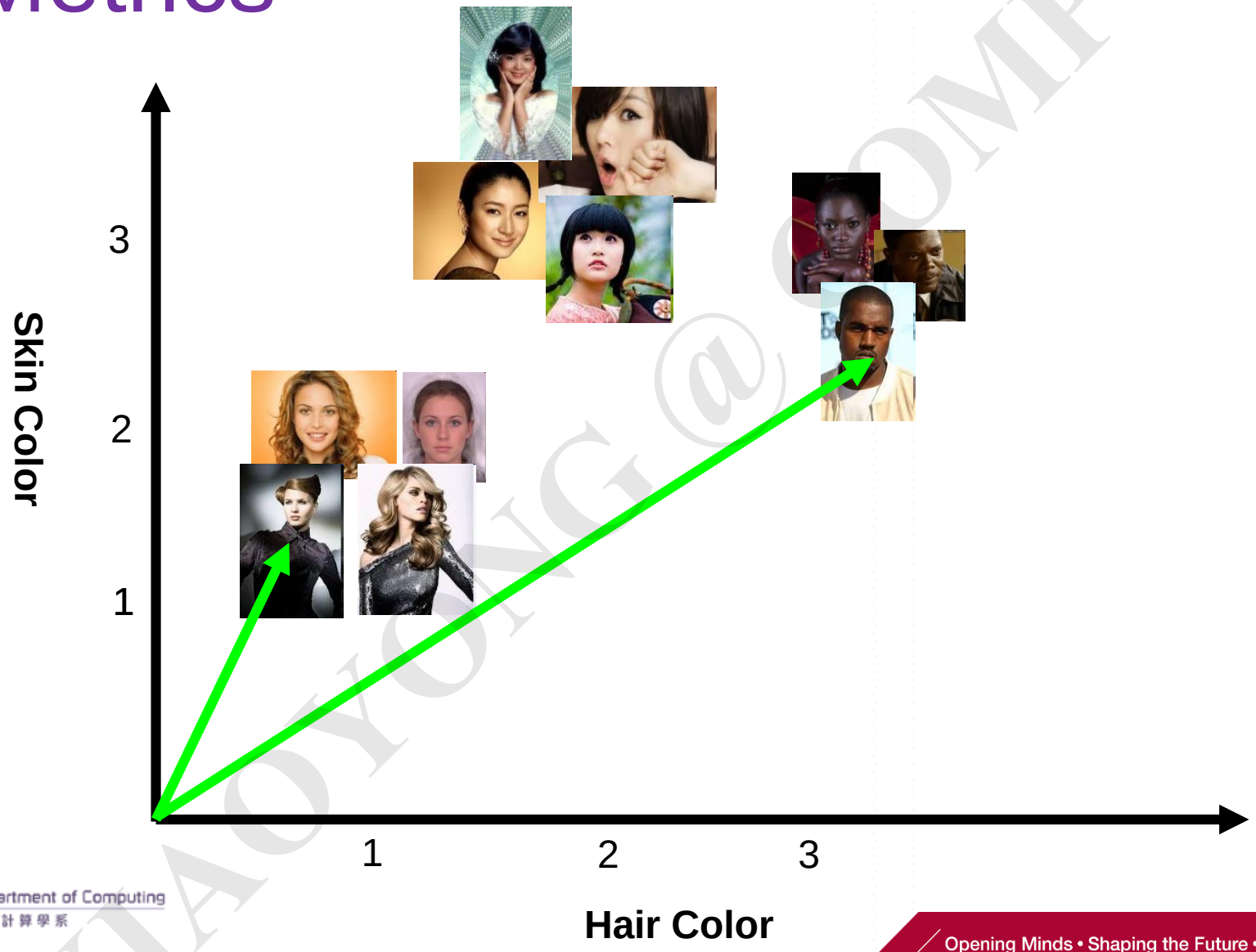
Can we teach computers to draw circles for grouping?

GOOD GUESS

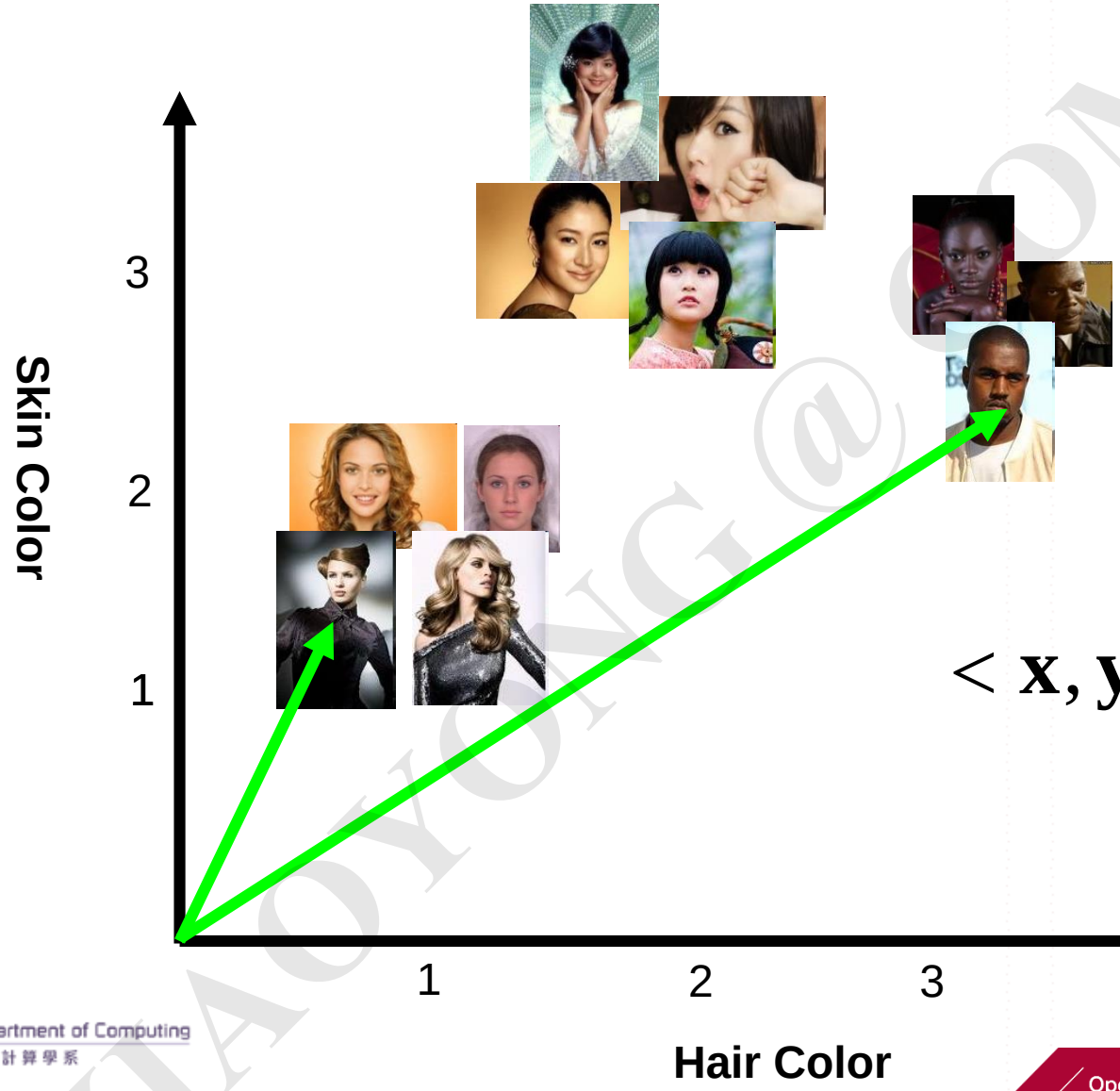
This is what we teach in this course!

But let's start from the most straightforward way by teaching computers to tell the similarity/difference.

Metrics

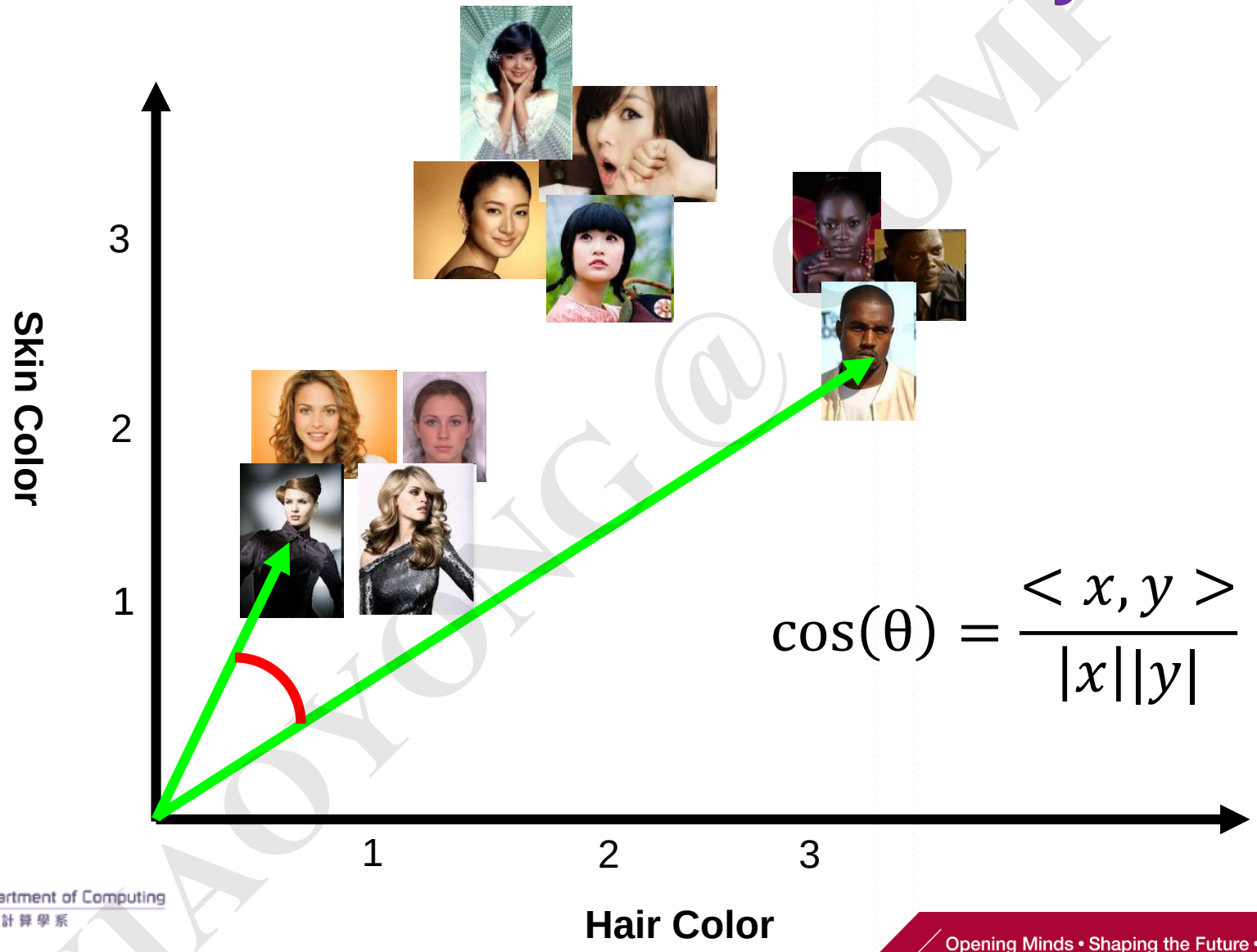


Metrics – Inner Product



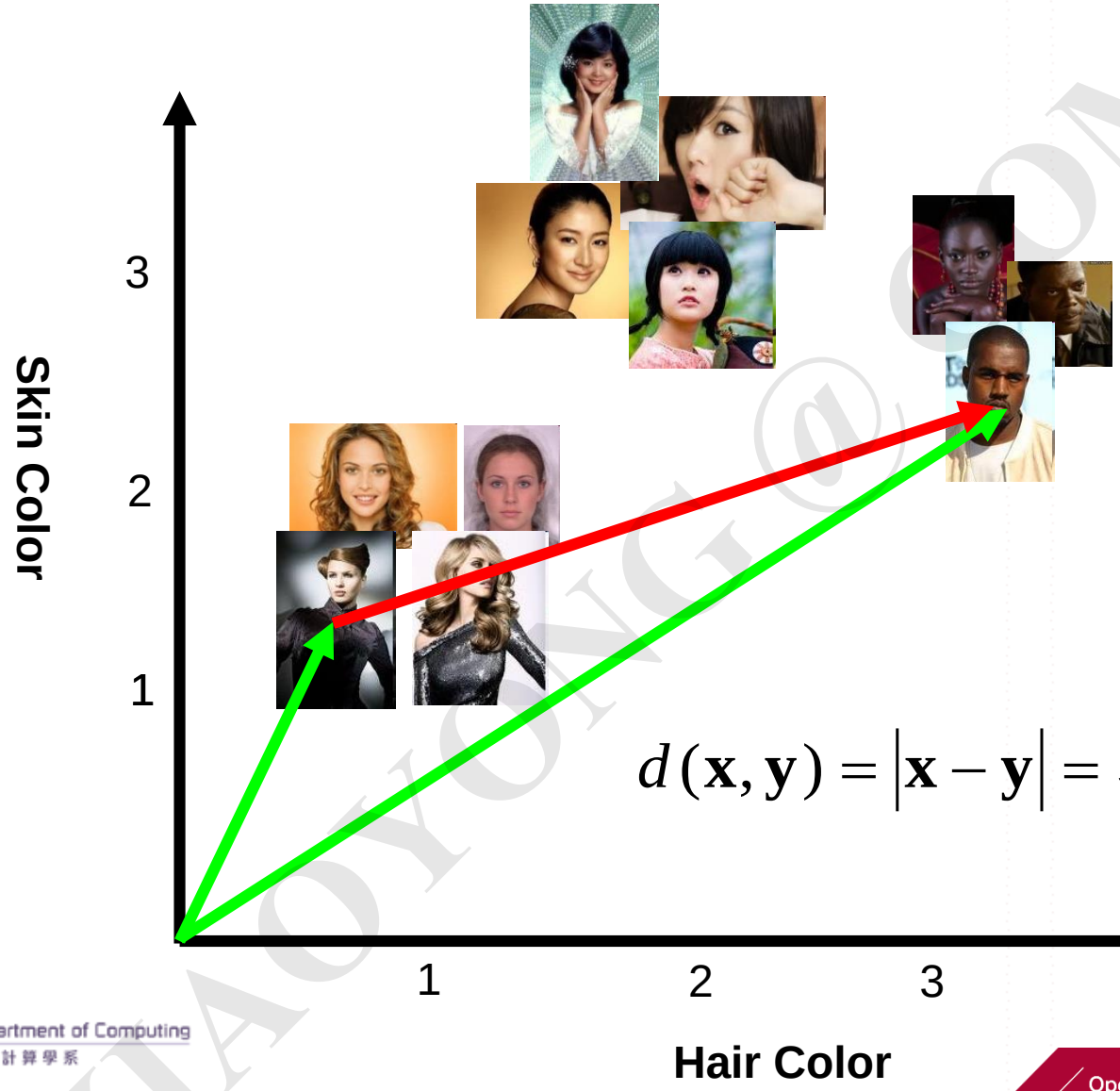
$$< \mathbf{x}, \mathbf{y} >= \sum_i x_i y_i$$

Metrics – Cosine Similarity



$$\cos(\theta) = \frac{< x, y >}{|x||y|}$$

Metrics – Euclidian Distance



$$d(\mathbf{x},\mathbf{y}) = |\mathbf{x}-\mathbf{y}| = \sqrt{<\mathbf{x}-\mathbf{y},\mathbf{x}-\mathbf{y}>}$$

I see!

We can tell if two images are from the same group by measuring the similarity/distance?

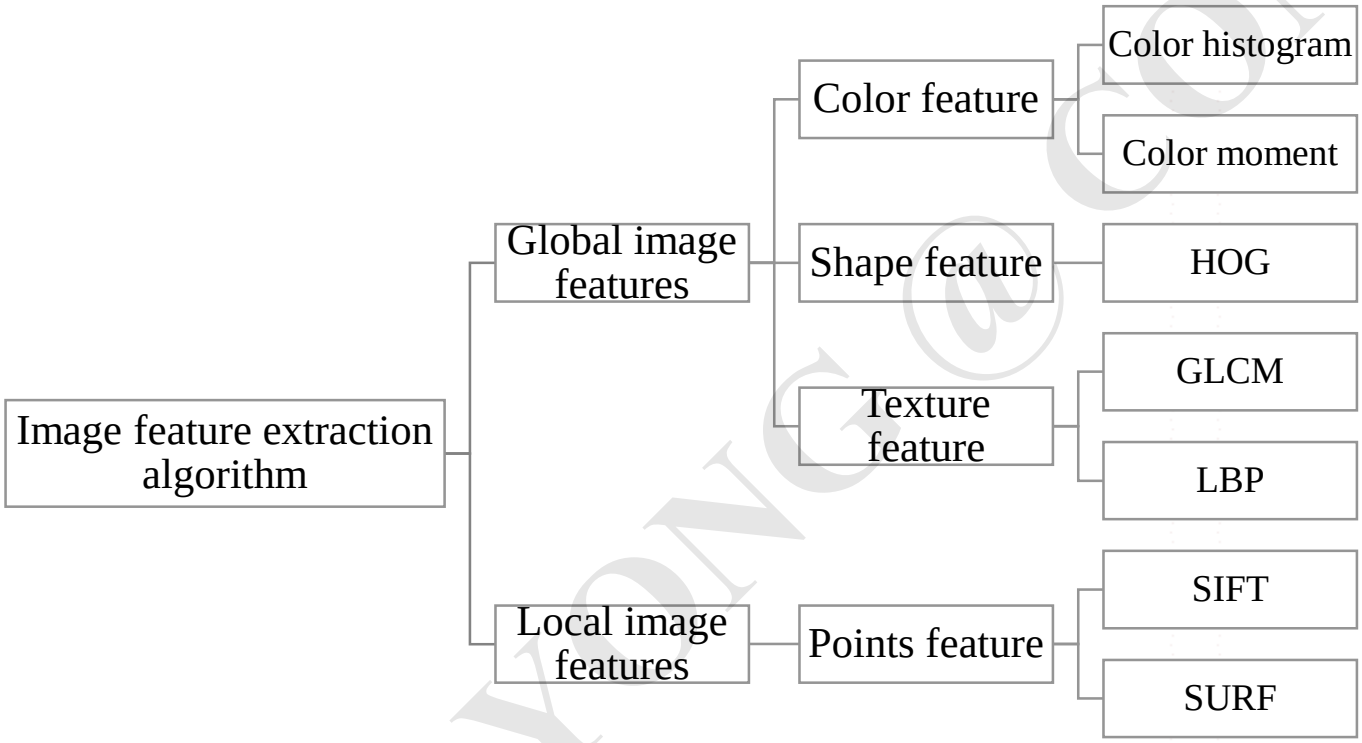
It's too early!

How can we obtain those numbers?

	Hair Color	Skin Color
	Yellow **(2.135)**	White **(1.05)**
	Black **(3.324)**	Yellow **(2.008)**
	Black **(2.945)**	Yellow **(2.355)**
	Yellow **(2.035)**	White **(1.035)**
	Black **(2.994)**	Black **(3.208)**

Feature Extraction

Features



Histogram of Oriented Gradient, HOG

Gray-Level Co-occurrence Matrix, GLCM

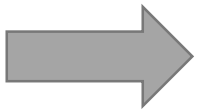
Local Binary Pattern, LBP

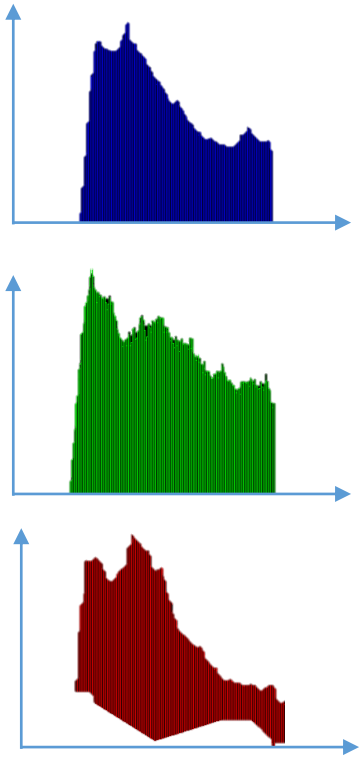
Scale-invariant feature transform, SIFT

Speeded Up Robust Features, SURF

Color Histograms

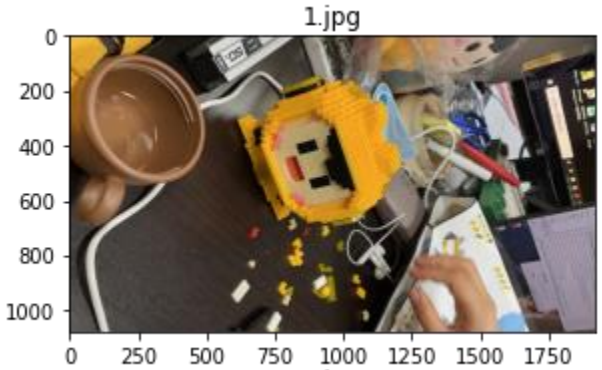
A color histogram of an image represents the distribution of the composition of colors in the image. It shows different types of colors appeared and the number of pixels in each type of the colors appeared.

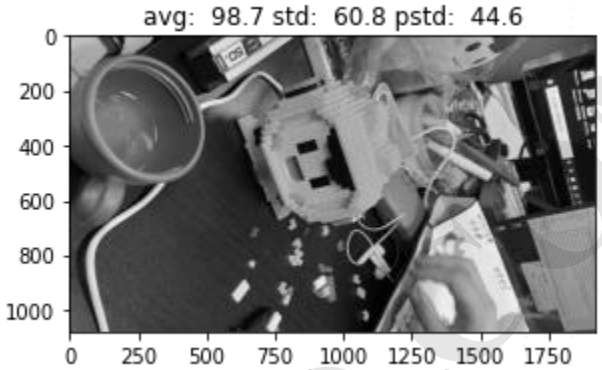


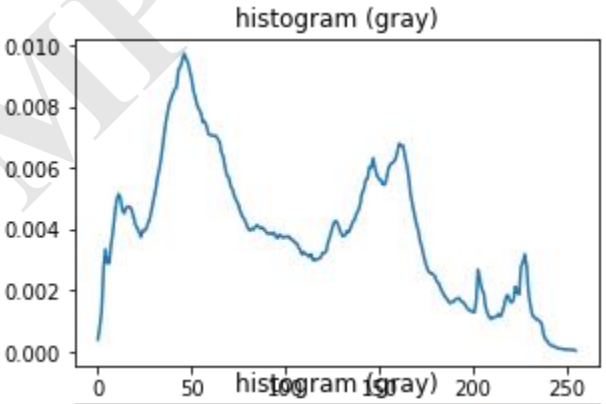


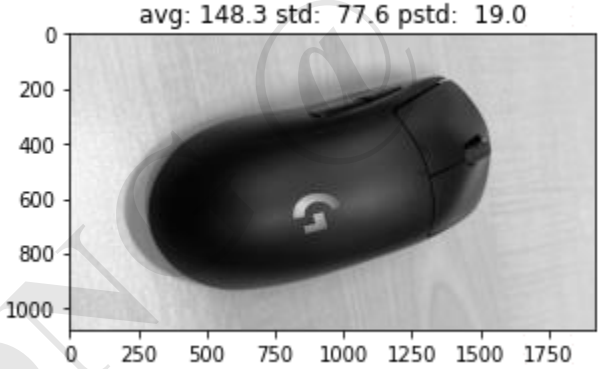
https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/index.html

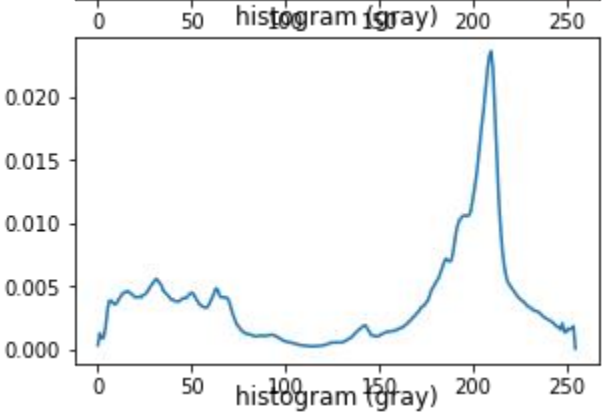
Examples from IMHere

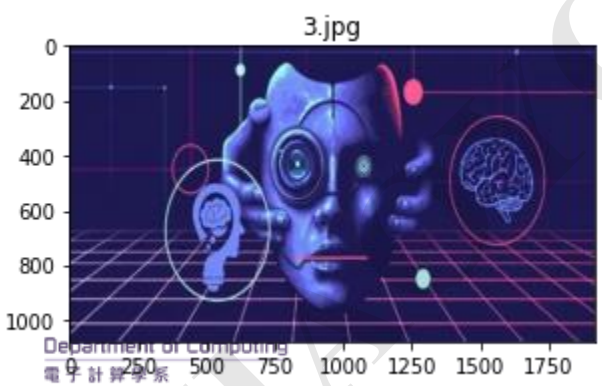


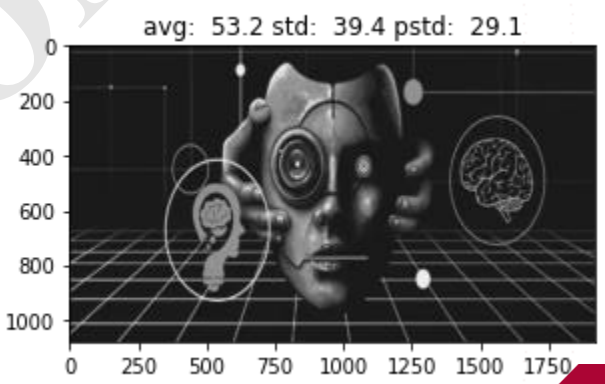


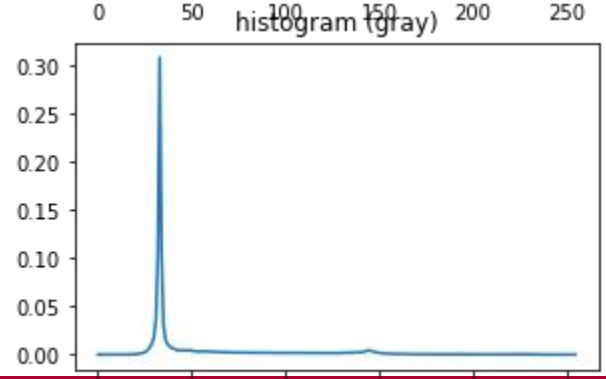












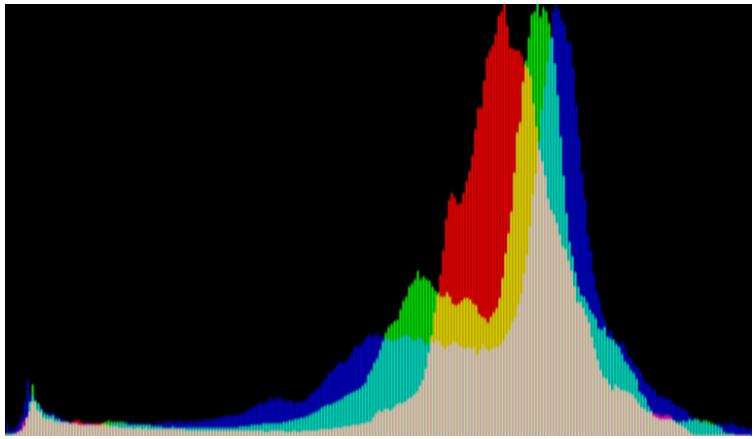
Color Histogram Extraction

To calculate the color histogram, the color space needs to be divided into several small color intervals, and each cell becomes a bin of the histogram. This process is called color quantization. Then, the color histogram can be obtained by calculating the number of pixels in each cell.

A picture of a cat

Red	Green	Blue	Pixel Count
Bin 0	Bin 0	Bin 0	7414
Bin 0	Bin 0	Bin 1	230
Bin 0	Bin 0	Bin 2	0
Bin 0	Bin 0	Bin 3	0
Bin 0	Bin 1	Bin 0	8
Bin 0	Bin 1	Bin 1	372
Bin 0	Bin 1	Bin 2	88
Bin 0	Bin 1	Bin 3	0
Bin 0	Bin 2	Bin 0	0
Bin 0	Bin 2	Bin 1	0
Bin 0	Bin 2	Bin 2	10
Bin 0	Bin 2	Bin 3	1
Bin 0	Bin 3	Bin 0	0
Bin 0	Bin 3	Bin 1	0

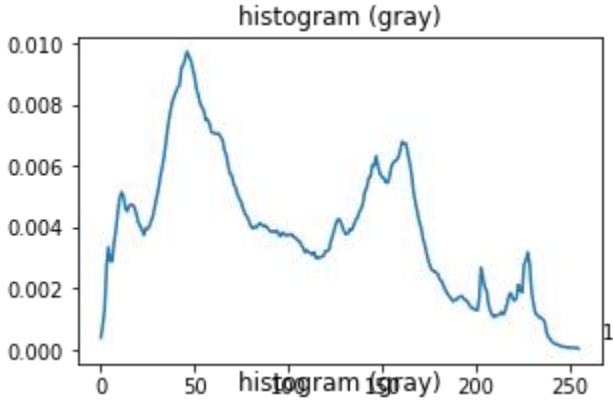
• • •

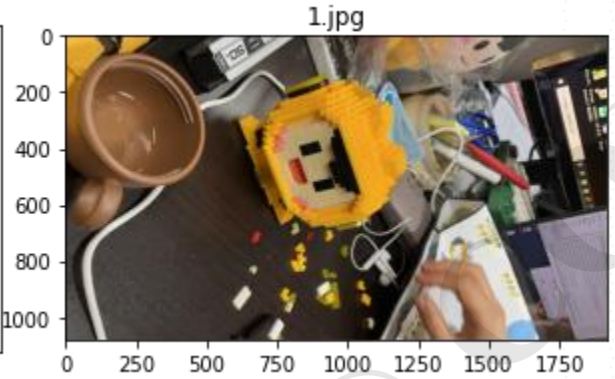


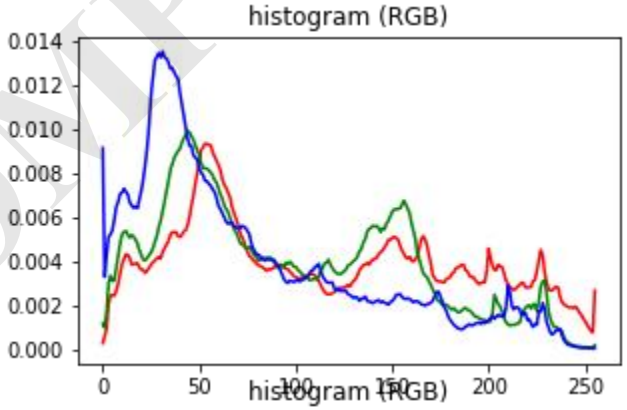
Color histogram of the cat picture with x-axis being RGB and y-axis being the frequency.

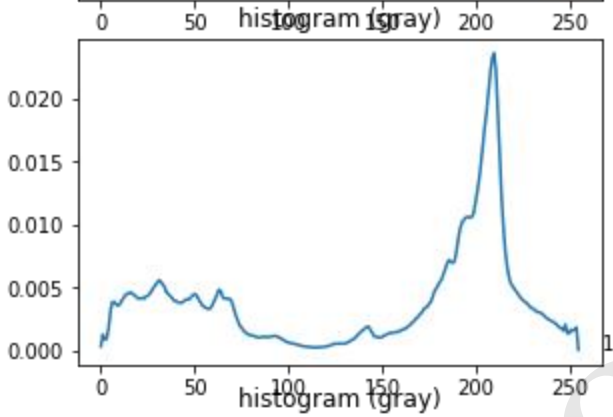
https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/index.html
https://en.wikipedia.org/wiki/Color_histogram

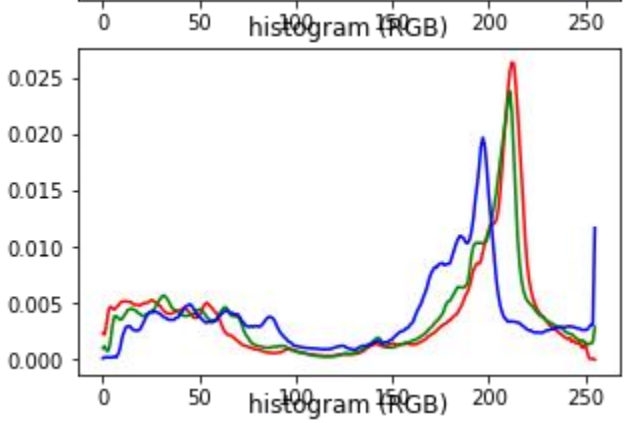
Examples from IMHere

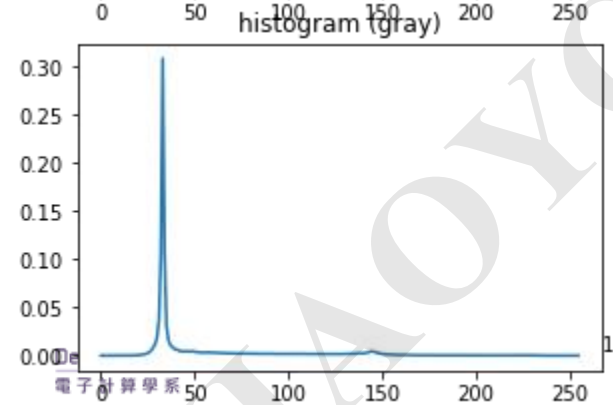


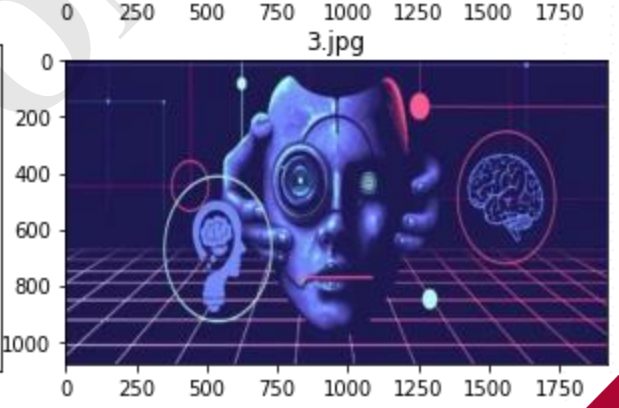


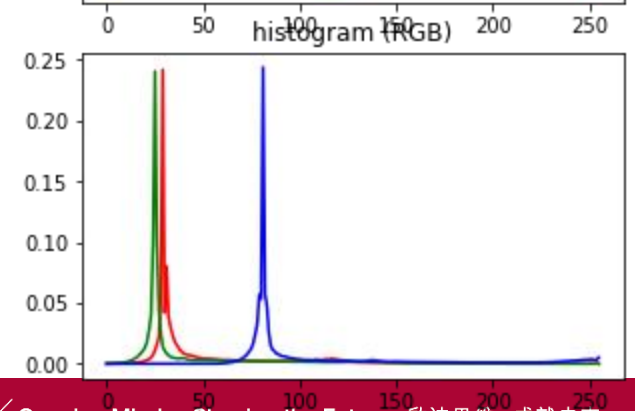






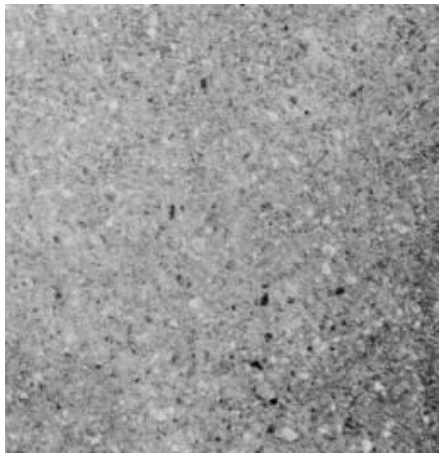


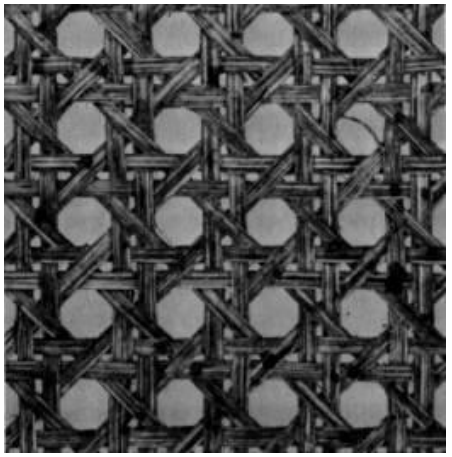




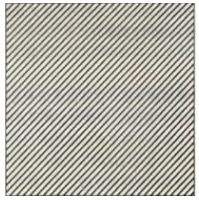
Texture Features

Texture feature refers to the common internal characteristics of the object surface, which contains the important information of the structure and arrangement of the object surface and its relationship with the surrounding objects.





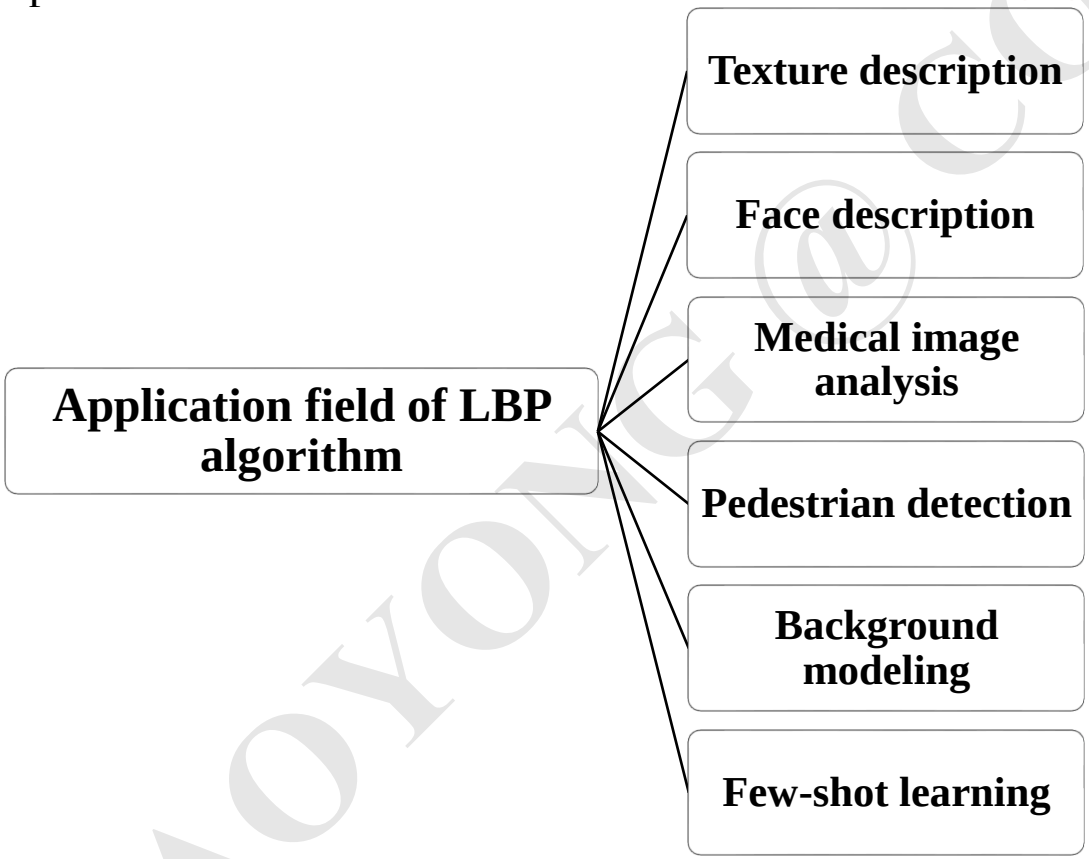




Local Binary Patterns (LBP)

LBP algorithm was originally proposed by Timo Ojala and Matti pietikäinen of machine vision and signal analysis center of Oulu University in Finland in 1996 for texture feature description.



Matti Pietikäinen
IEEE、IAPR Fellow
CMVSA of Oulu University

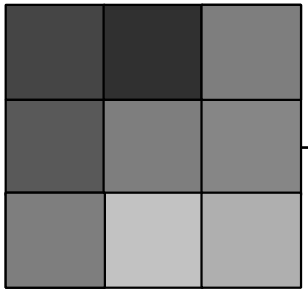
Local Binary Patterns (LBP)

Gray-scale image

LBP feature map

http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

Local Binary Patterns (LBP)



30	23	45
40	45	47
49	60	55

0	0	1
0		1
1	1	1

0	0	1	1	1	1	1	0

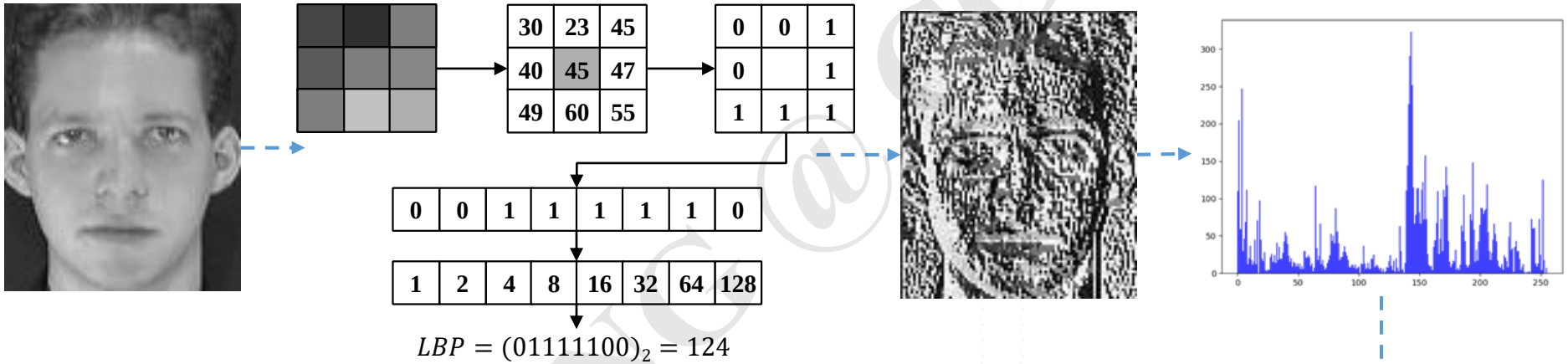
1	2	4	8	16	32	64	128

$$LBP = (01111100)_2 = 124$$

$$LBP_{P,R} = \sum_{p=1}^{P} S(g_p - g_c) \times 2^{p-1}$$

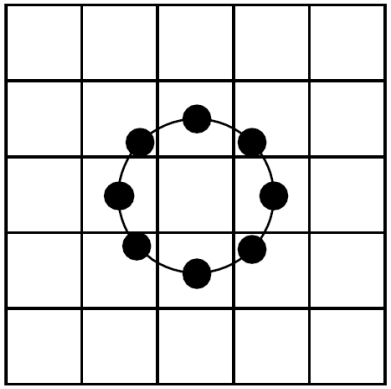
$$S(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Local Binary Patterns (LBP)

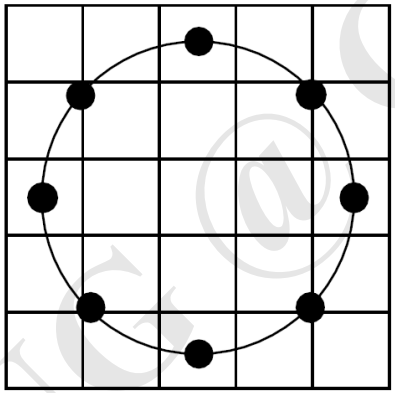


Feature vector $X = [x_1, x_2, x_3, \ldots, x_K]$

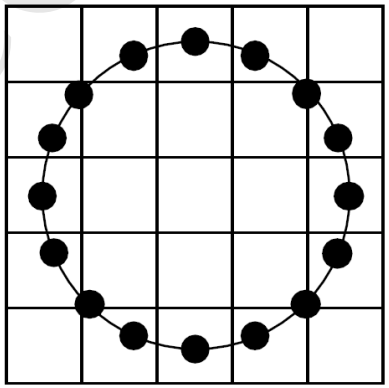
Circular LBP



R=1 P=8

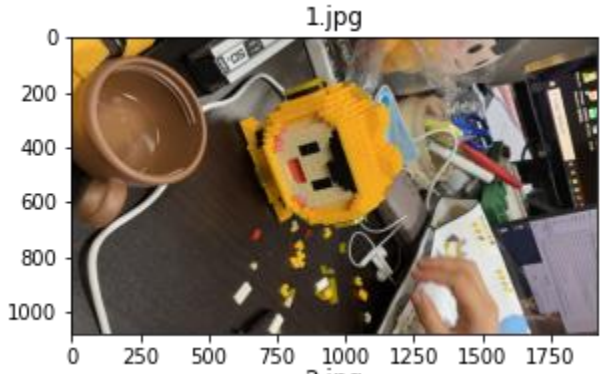


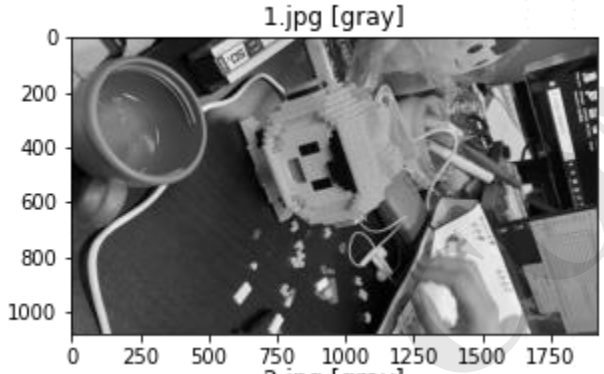
R=2 P=8

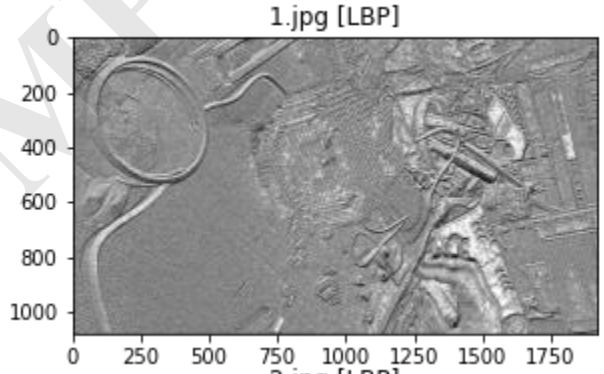


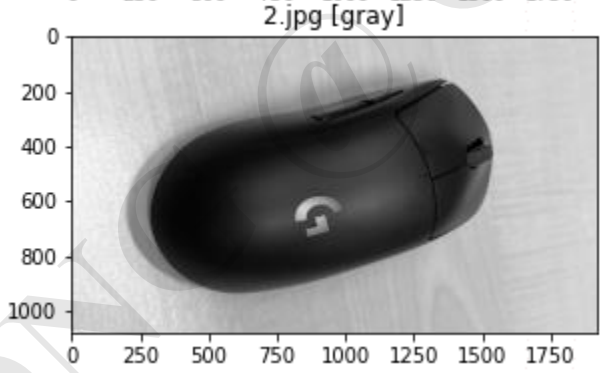
R=2 P=16

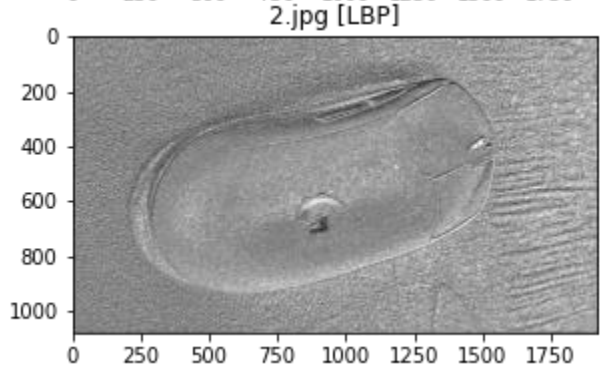
Examples from IMHere

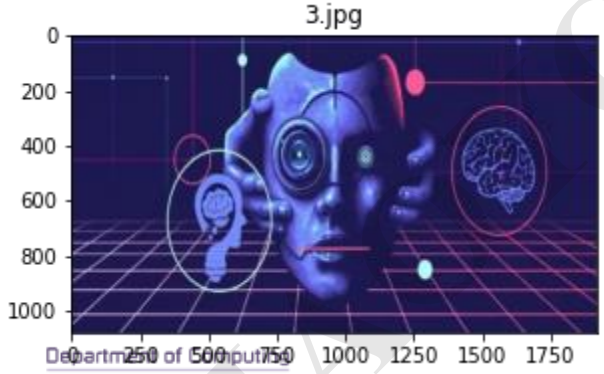


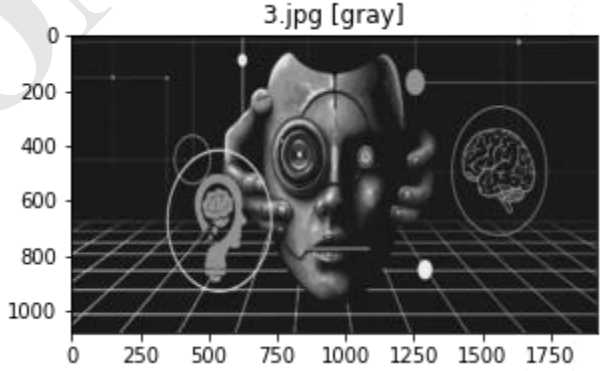


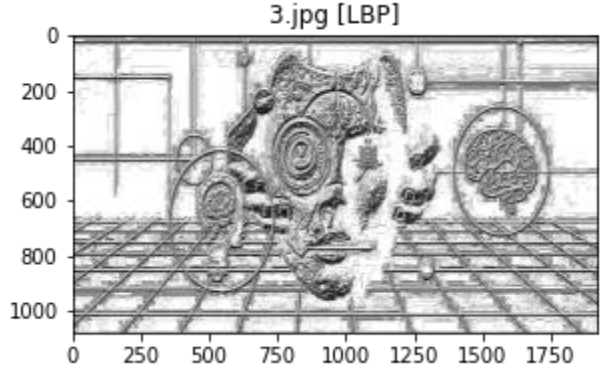








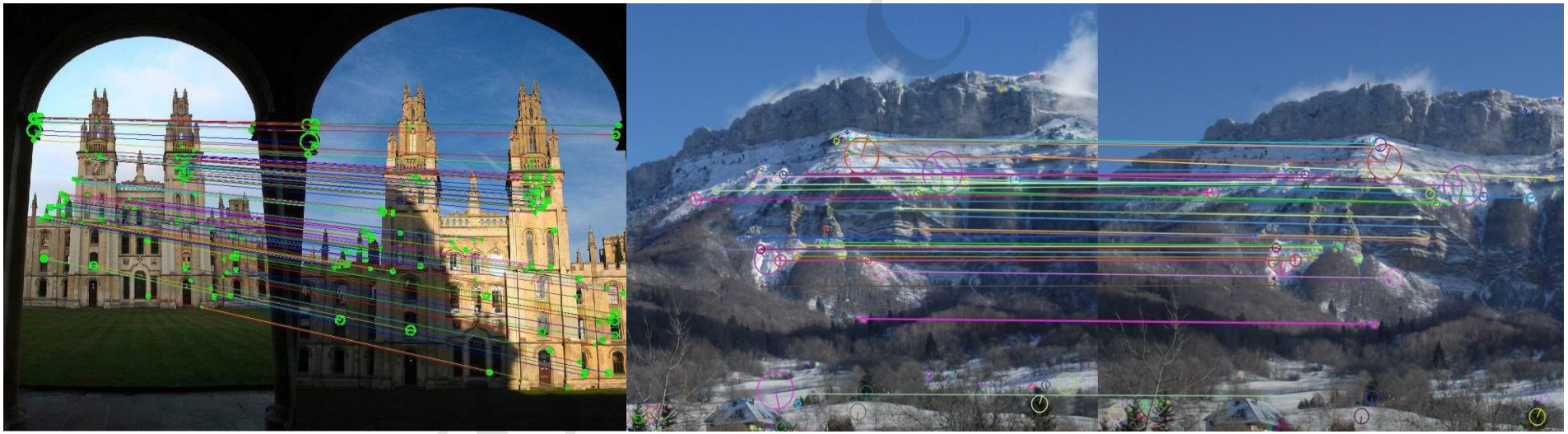




Can computers “see” the difference using color histograms or LBP?

Scale-invariant feature transform (SIFT)

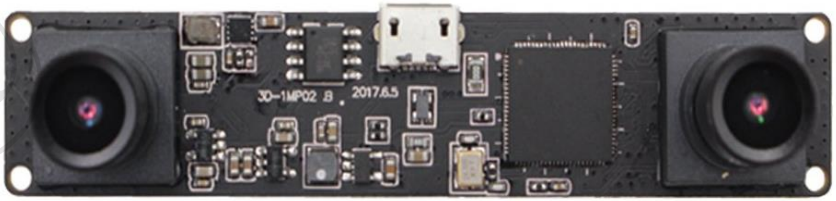
SIFT is a milestone work in the research field of local image feature descriptor. SIFT was first proposed in 1999 and was improved in 2004. SIFT is invariant to image changes such as scale, rotation, certain viewing angle and illumination changes.

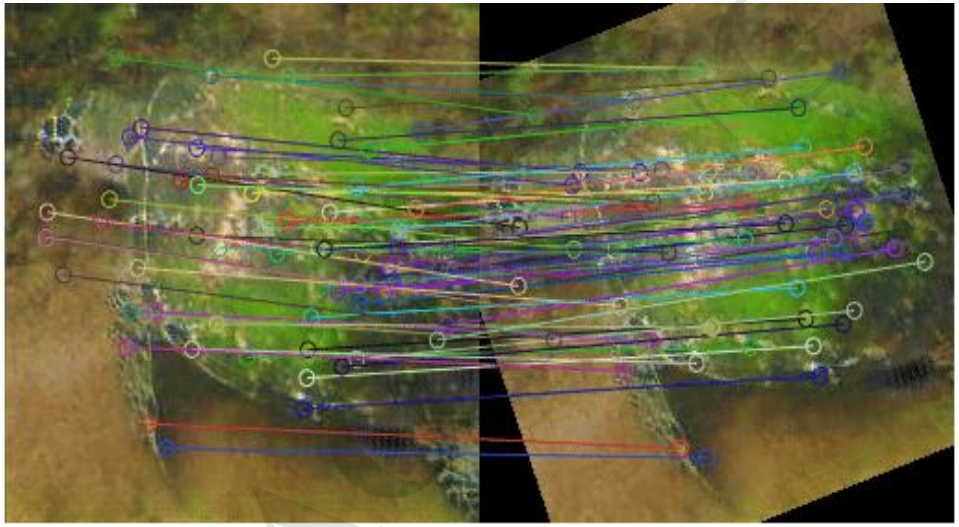


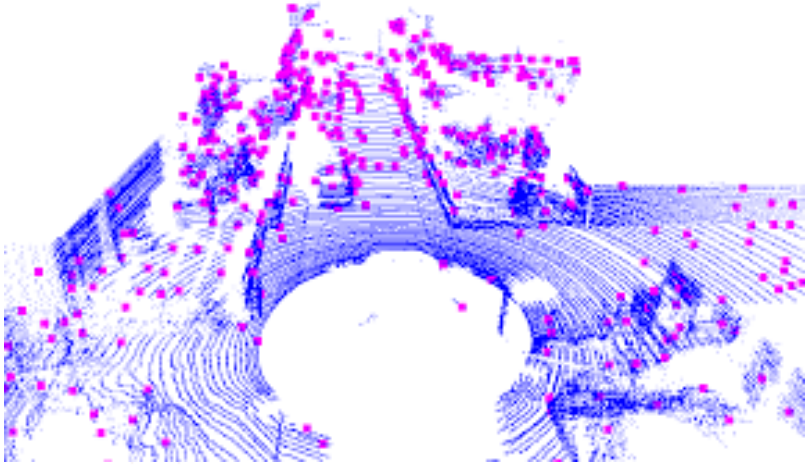
http://5b0988e595225.cdn.sohucs.com/images/20200318/bacc3866c2eb47b58cde6afd3b6512b6.jpeg

Scale-invariant feature transform (SIFT)

Scale-invariant feature transform (SIFT)







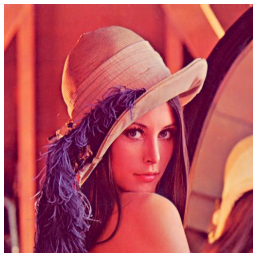
Scale-invariant feature transform (SIFT)

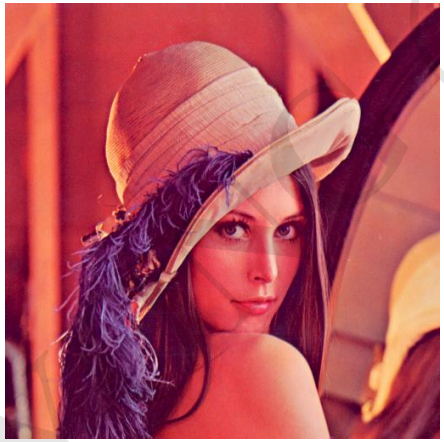
SIFT Detection & Description Steps

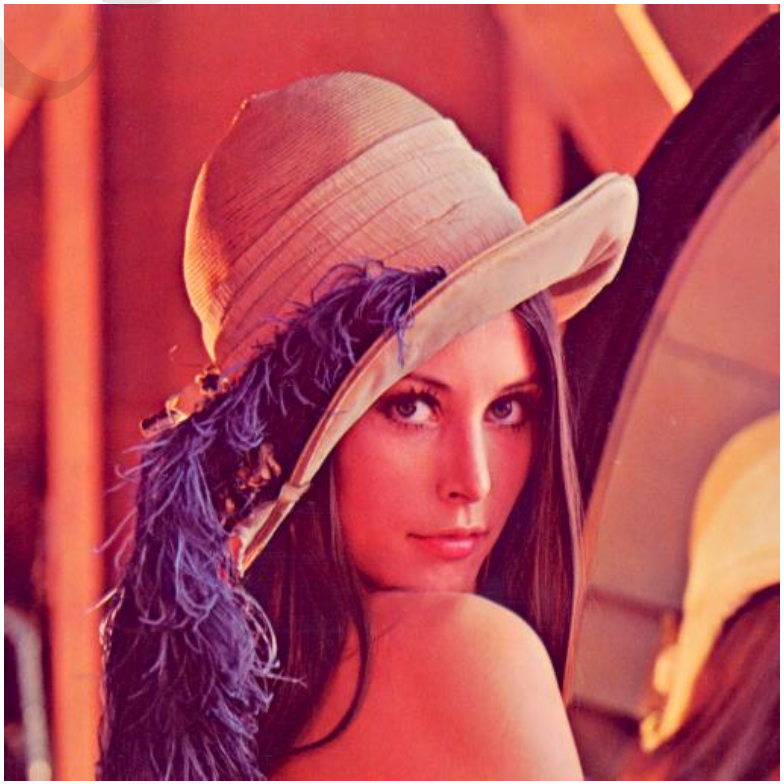
- **Candidate Localization**: Find Scale-invariant candidates
- **Refinement**: Keypoint filtering
- **Orientation Assignment**: Estimate orientations for keypoints
- **Description Generation**: Build description

Step 1: Candidate Localization

>What is a scale?

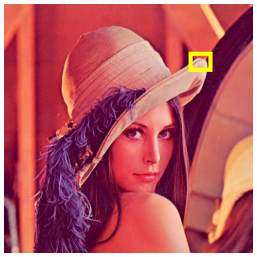


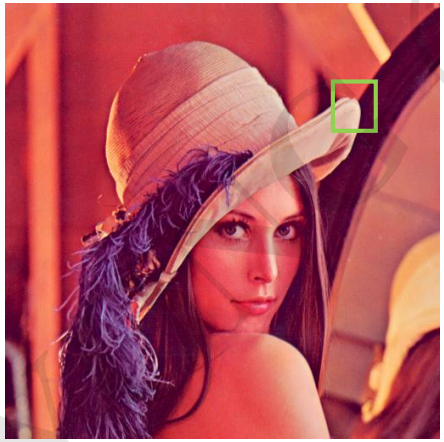


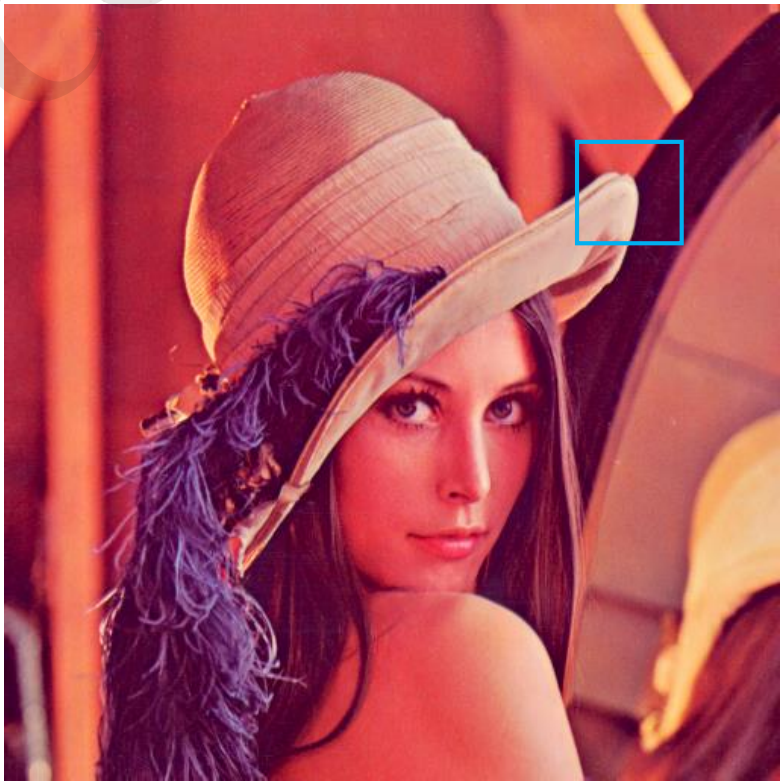


Step 1: Candidate Localization

>What is a scale?

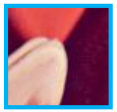






Step 1: Candidate Localization

>What is a scale?

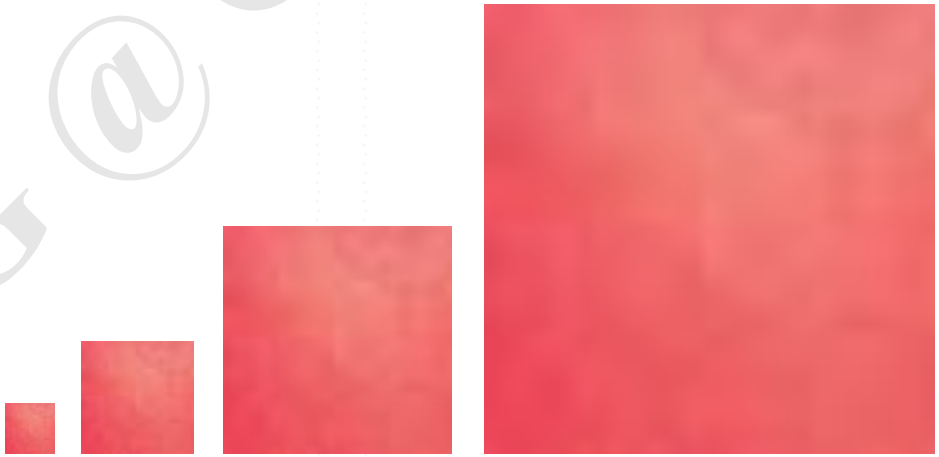


Step 1: Candidate Localization

>What is a scale?

Step 1: Candidate Localization

>What is a scale?



Step 1: Candidate Localization

>What is scale-invariant?

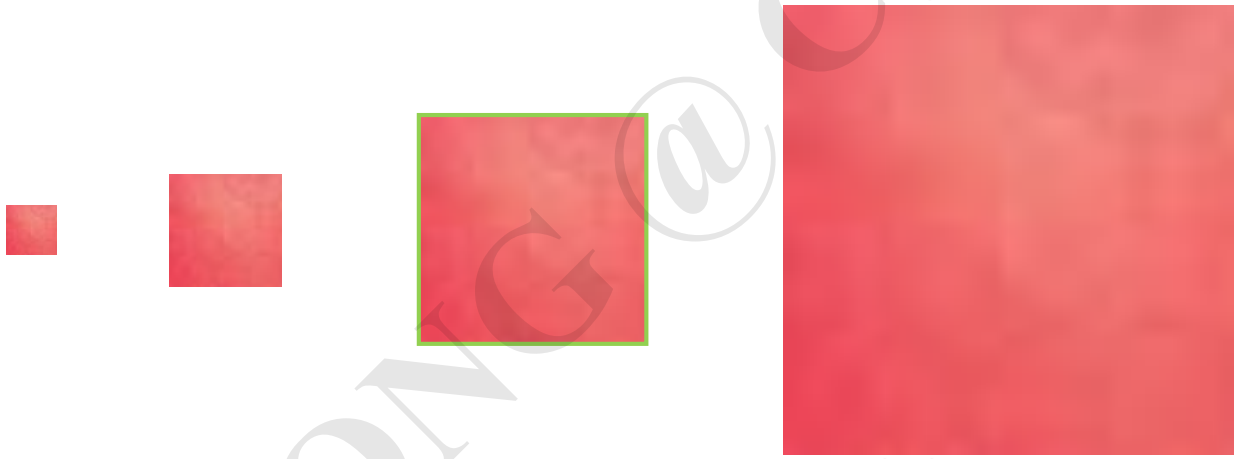
Step 1: Candidate Localization

>What is scale-invariant?

To find the scale at which a pattern is represented the **best** at its center (the key point)

Step 1: Candidate Localization

>What is scale-invariant?

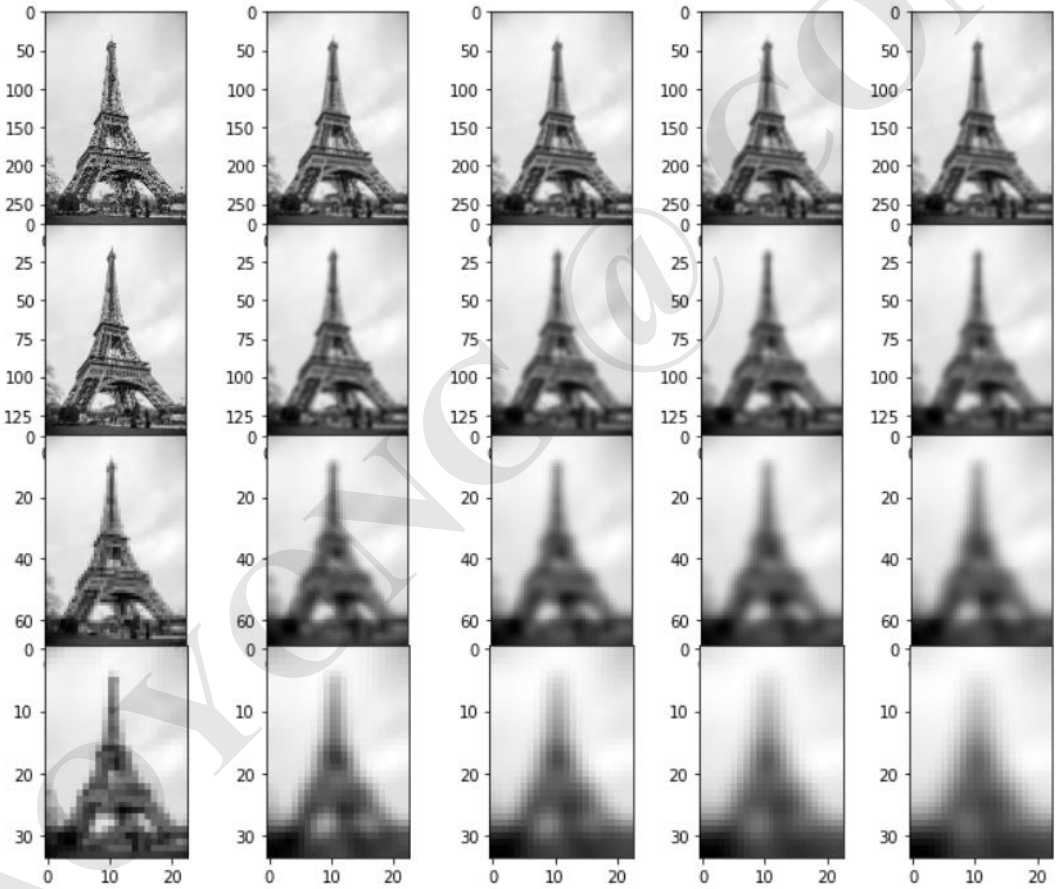


Scale-invariance: no matter what the original scale of the image is, the same **best scale** will be found

How can we do it?

Step 1: Candidate Localization

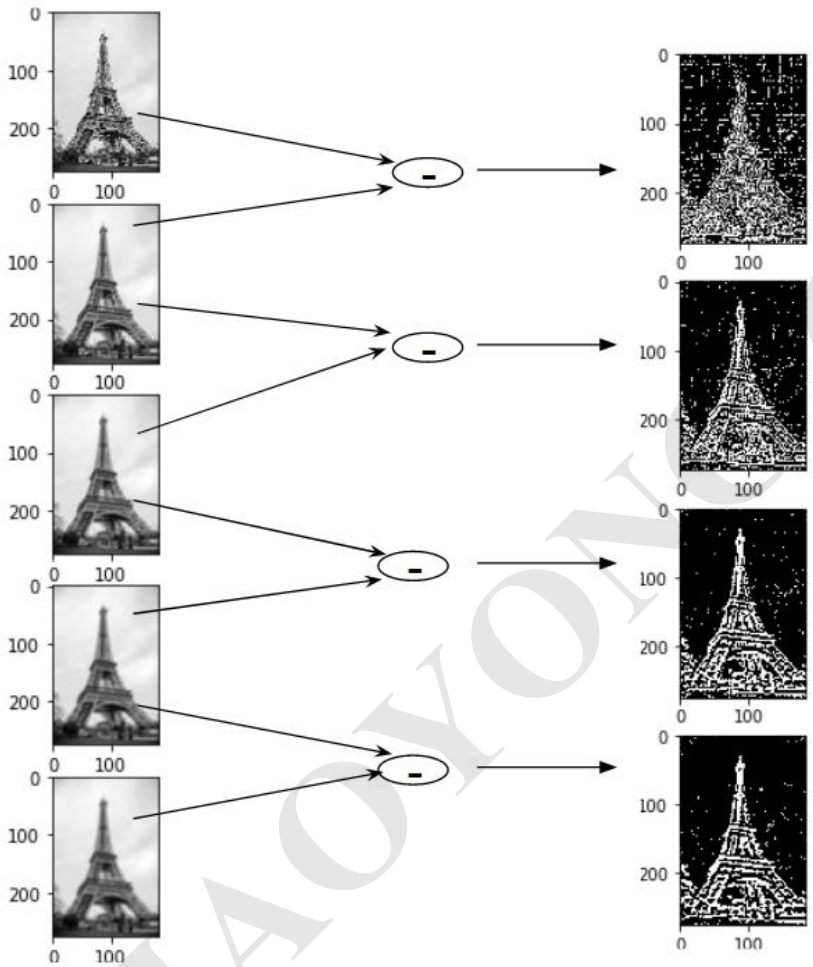
>Rescale and blur

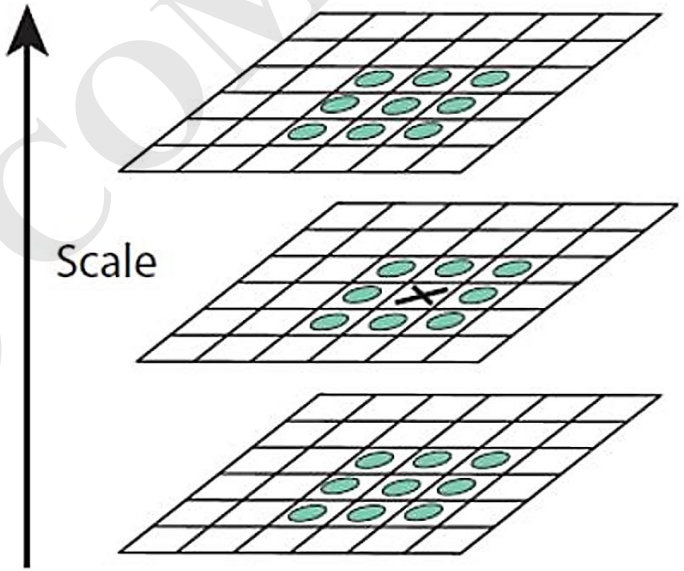


Credit: https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

Step 1: Candidate Localization

> Difference of Gaussian



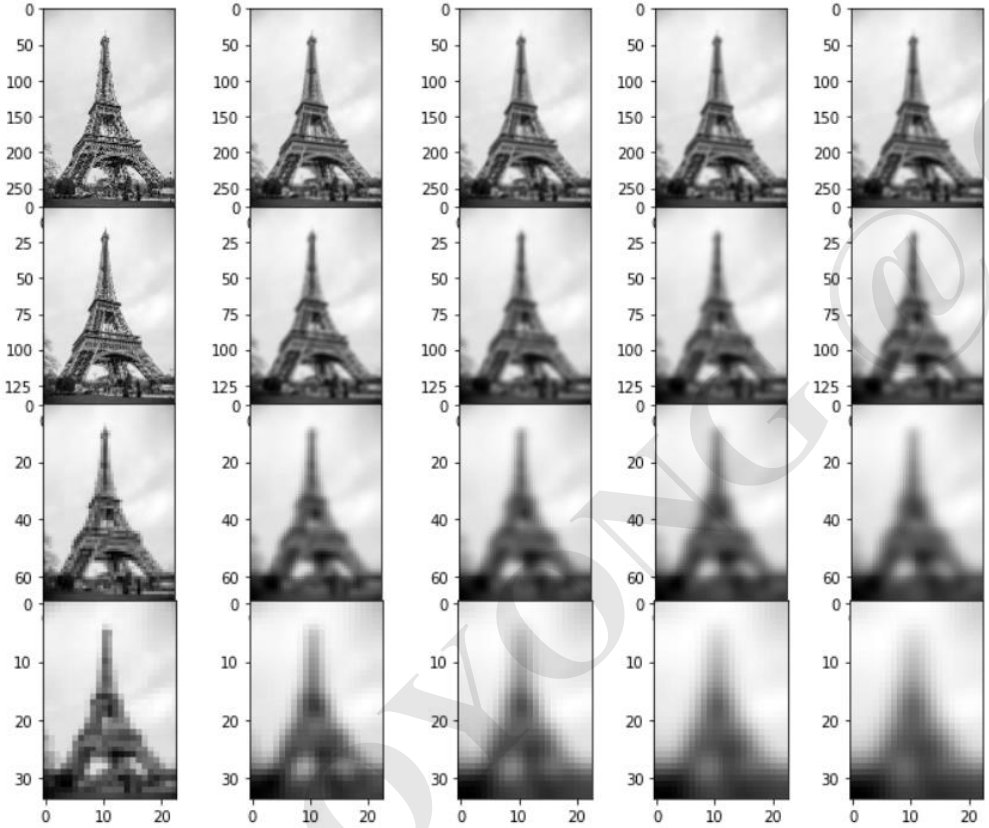


Local maxima and minima are those stand out at neighborhood

Sounds good but less efficient.
We need a little math here.

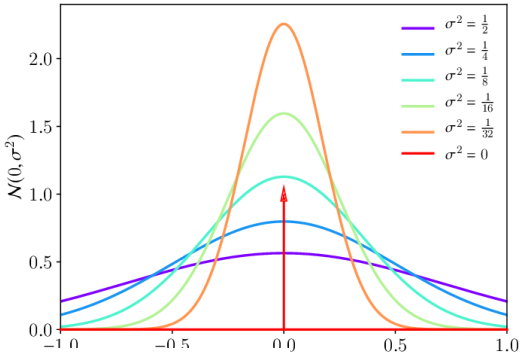
Step 1: Candidate Localization

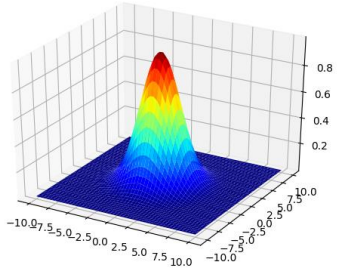
>Rescale and blur

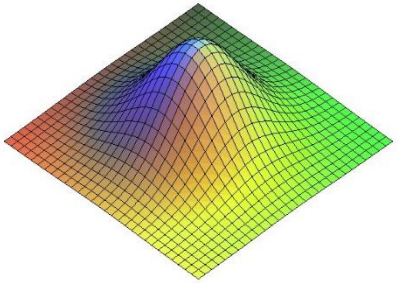


$$G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$



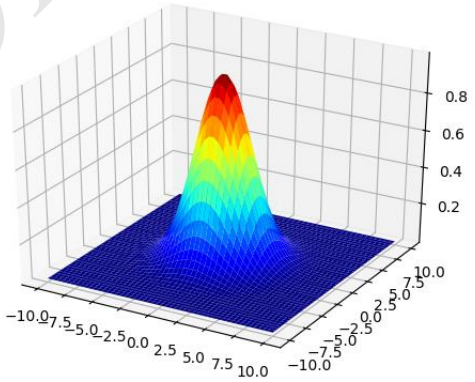


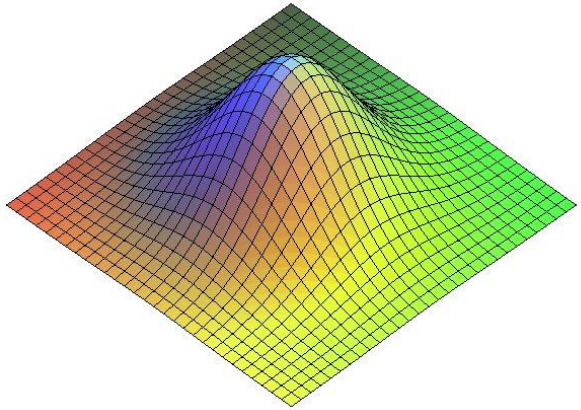


Step 1: Keypoint Localization

0.0049	0.0092	0.0134	0.0152	0.0134	0.0092	0.0049
0.0092	0.0172	0.0250	0.0283	0.0250	0.0172	0.0092
0.0134	0.0250	0.0364	0.0412	0.0364	0.0250	0.0134
0.0152	0.0283	0.0412	**0.0467**	0.0412	0.0283	0.0152
0.0134	0.0250	0.0364	0.0412	0.0364	0.0250	0.0134
0.0092	0.0172	0.0250	0.0283	0.0250	0.0172	0.0092
0.0049	0.0092	0.0134	0.0152	0.0134	0.0092	0.0049

This is a 7 x 7 (truncated) Gaussian mask with a mean of zero and a standard deviation $\sigma = 2$





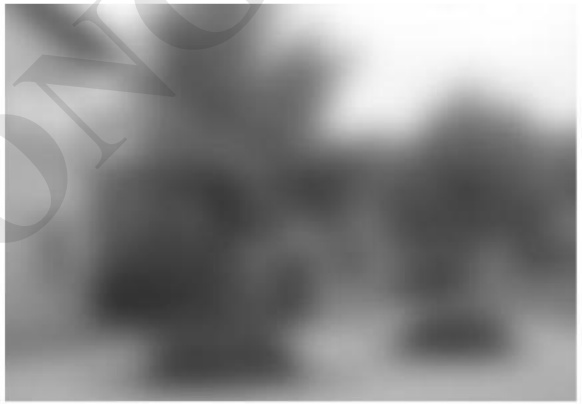
Step 1: Keypoint Localization

Scale = 0

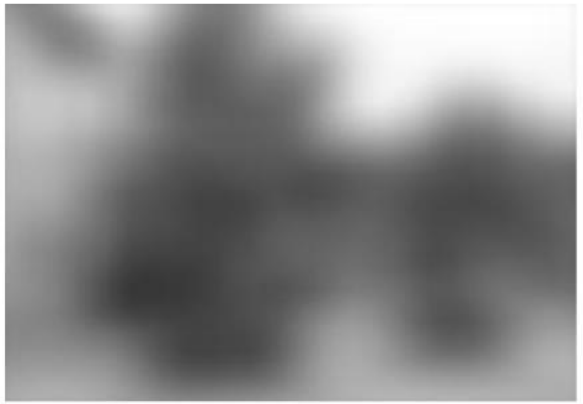
Scale = 1

Scale = 4

Scale = 16



Scale = 64

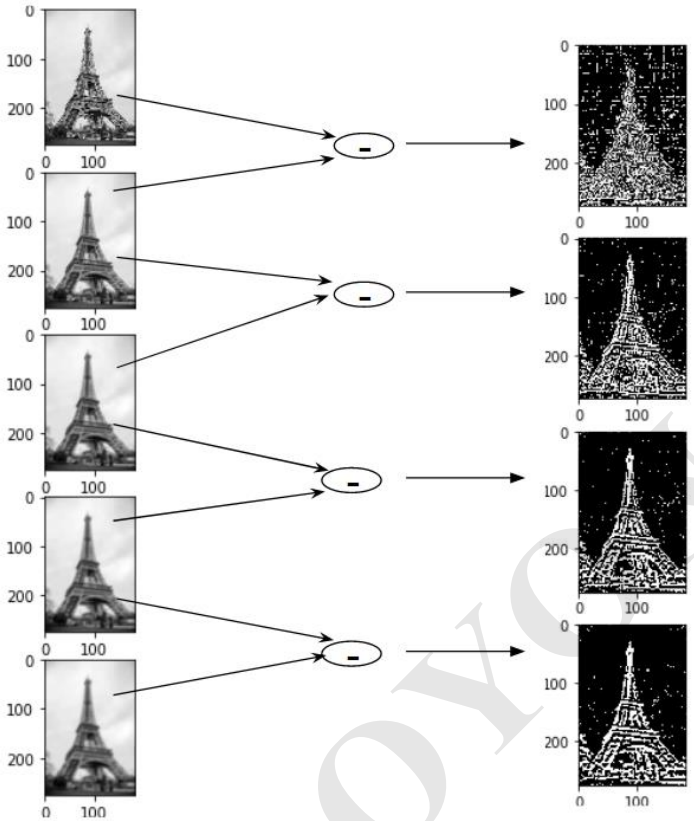


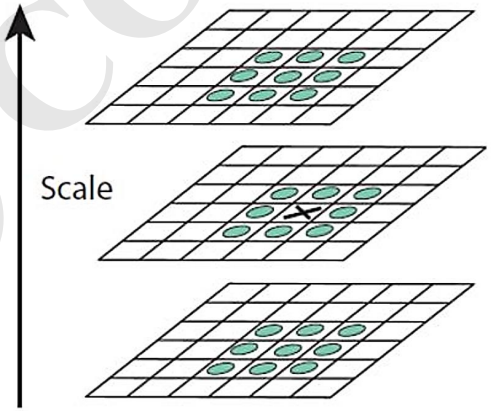
Scale = 256

http://en.wikipedia.org/wiki/Scale_space

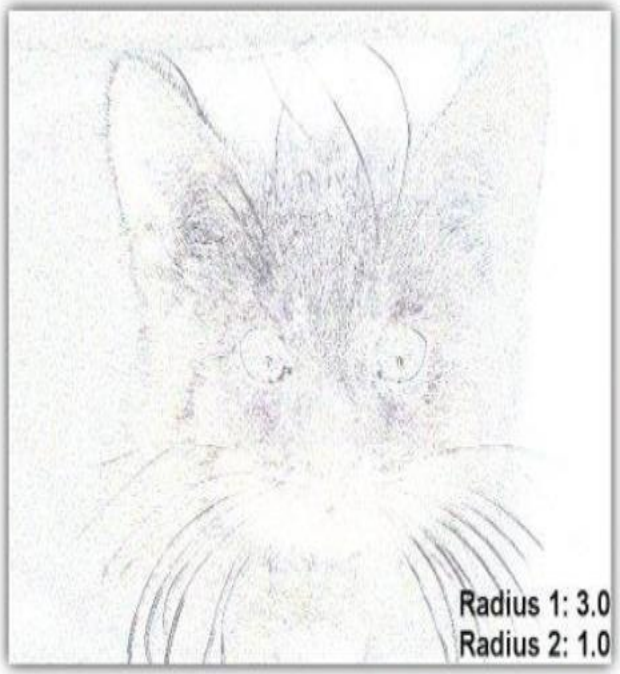
Step 1: Candidate Localization

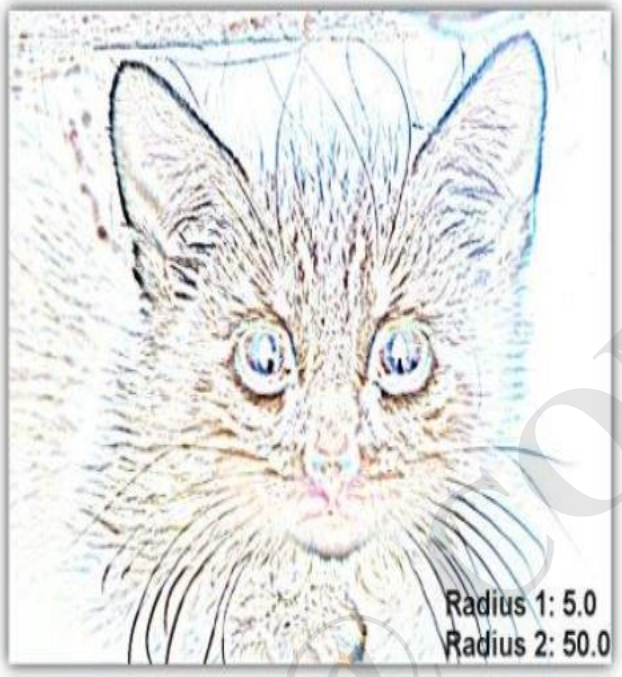
>Difference of Gaussian

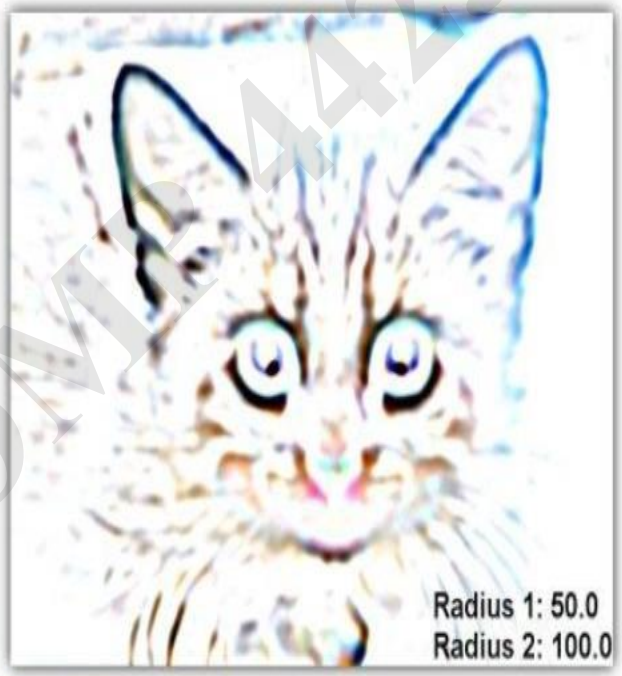


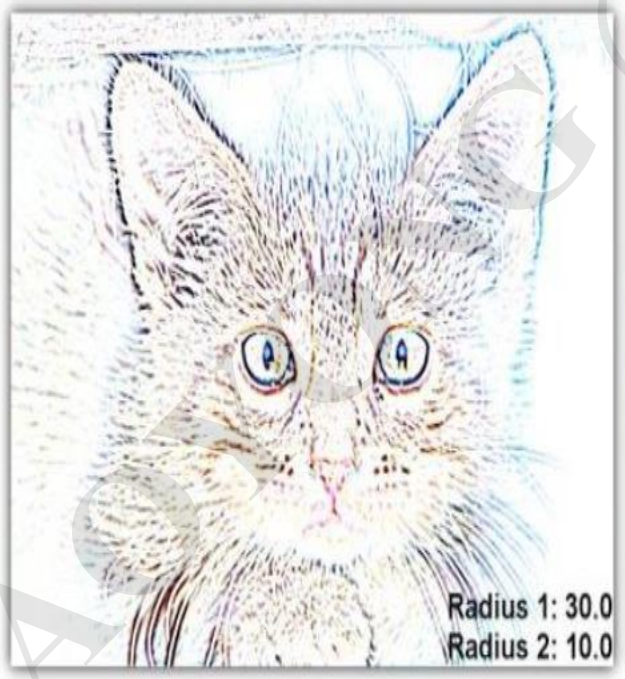


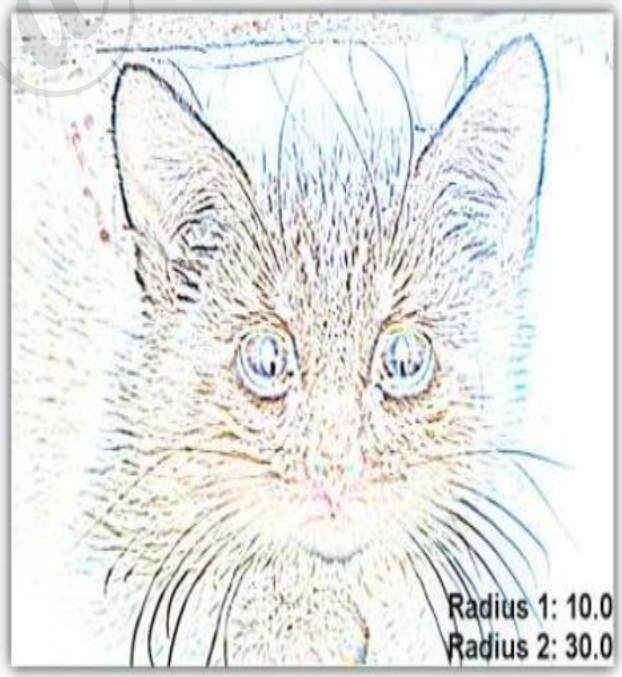
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$
$$D(x, y, \sigma) = L(x, y, \sigma) - L(x, y, k\sigma)$$
$$= \big(G(x, y, \sigma) - G(x, y, k\sigma)\big) * I(x, y)$$


Radius 1: 3.0
Radius 2: 1.0

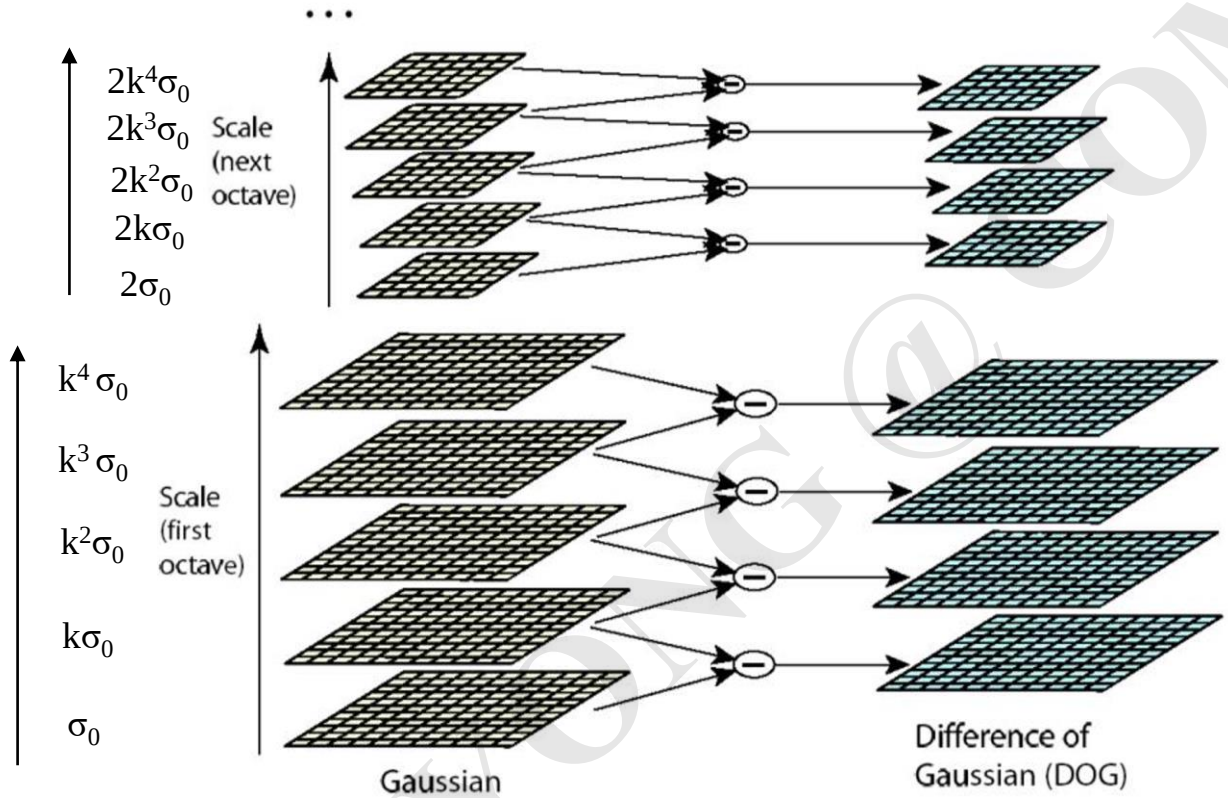

Radius 1: 5.0
Radius 2: 50.0


Radius 1: 50.0
Radius 2: 100.0


Radius 1: 30.0
Radius 2: 10.0


Radius 1: 10.0
Radius 2: 30.0

Step 1: Candidate Localization



$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

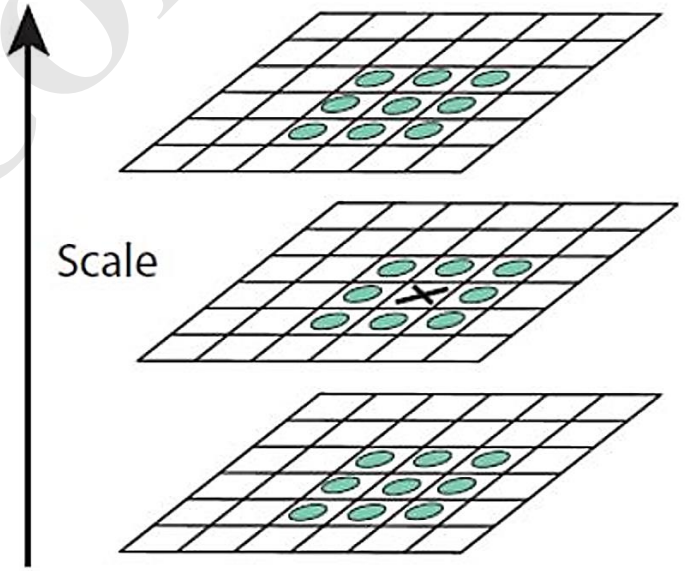
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Such a sequence of images convolved with Gaussians of increasing σ_0 constitute a so-called **scale space**.

Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 1: Candidate Localization

- The keypoints are maxima or minima in the “scale-space-pyramid”.
- Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3 × 3 regions at the current and adjacent scales (marked with circles)



Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

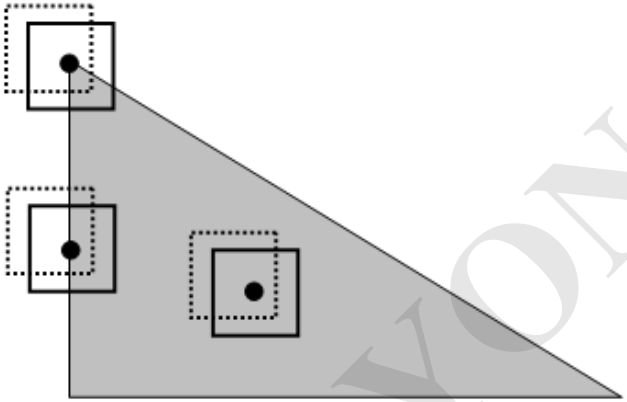
Step 1: Candidate Localization

Keypoint candidates

http://upload.wikimedia.org/wikipedia/commons/4/44/Sift_keypoints_filtering.jpg

Step 2: Refinement

Idea: keypoint candidates at low-contrast positions or on edges are not stable enough. We have to remove them.

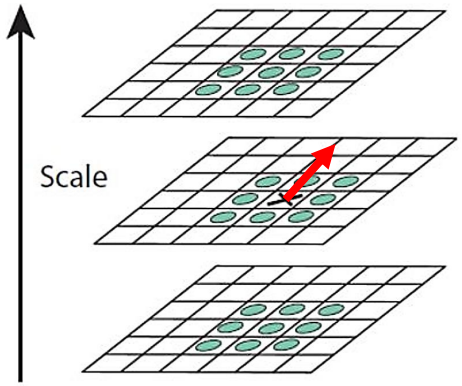


http://upload.wikimedia.org/wikipedia/commons/4/44/Sift_keypoints_filtering.jpg

Point-wise calculation is straightforward but inefficient. We need to introduce a little math again

Step 2: Refinement

Let us denote the DoG function in a small 3D neighborhood around a keypoint by a second-order Taylor-series:



$$\mathbf{x} = (x, y, \sigma)^T$$

Offset to the origin rather than the coordinates

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

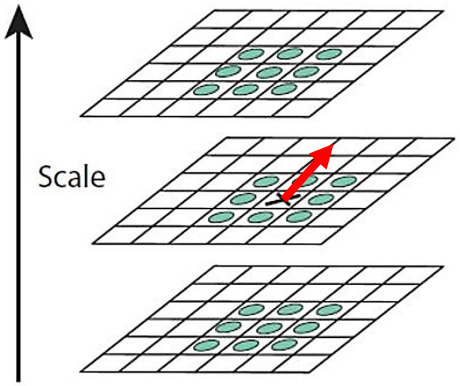
Gradient Vector

Hessian Matrix

By setting this to zero, we will have a $\hat{x}$, at which we can find the **theoretical** local extremum.
We discard current candidate if $\hat{x}$>0.5. Why?

Step 2: Refinement

Let us denote the DoG function in a small 3D neighborhood around a keypoint by a second-order Taylor-series:



$$\mathbf{x} = (x, y, \sigma)^T$$

Offset to the origin rather than the coordinates

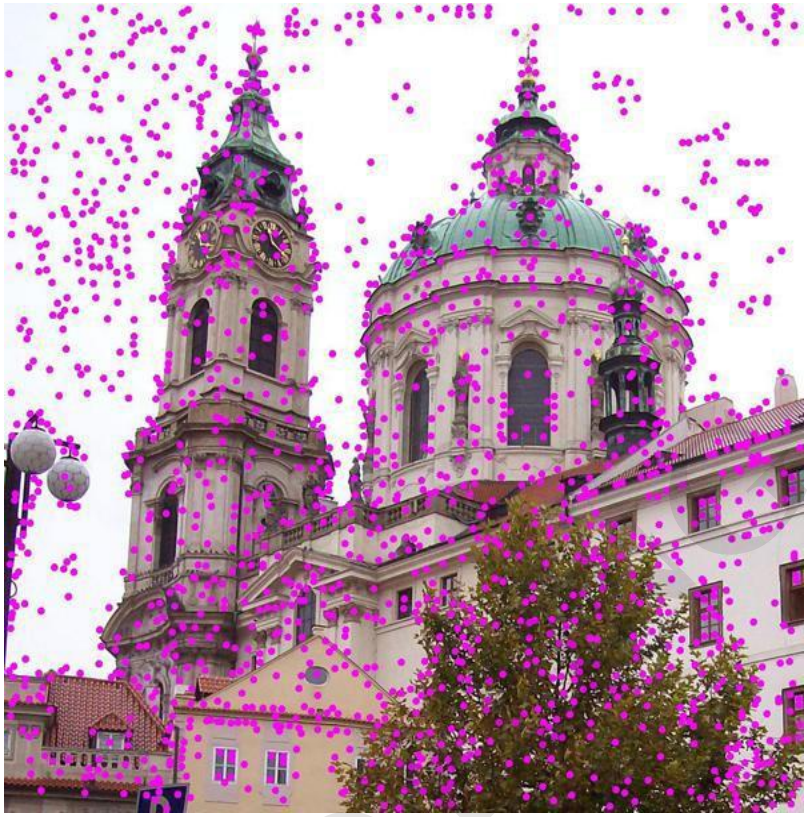
$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Gradient Vector

Hessian Matrix

To remove the low-contrast candidates, we calculate the second order Taylor at $\hat{x}$ and discard the candidate if the result less than 0.03. Why?

Step 2: Refinement



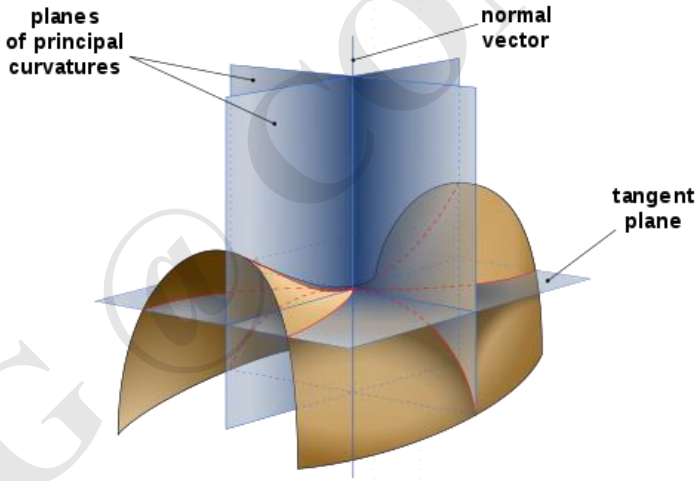
Removal of low-contrast keypoints

http://upload.wikimedia.org/wikipedia/commons/4/44/Sift_keypoints_filtering.jpg

Step 2: Refinement

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

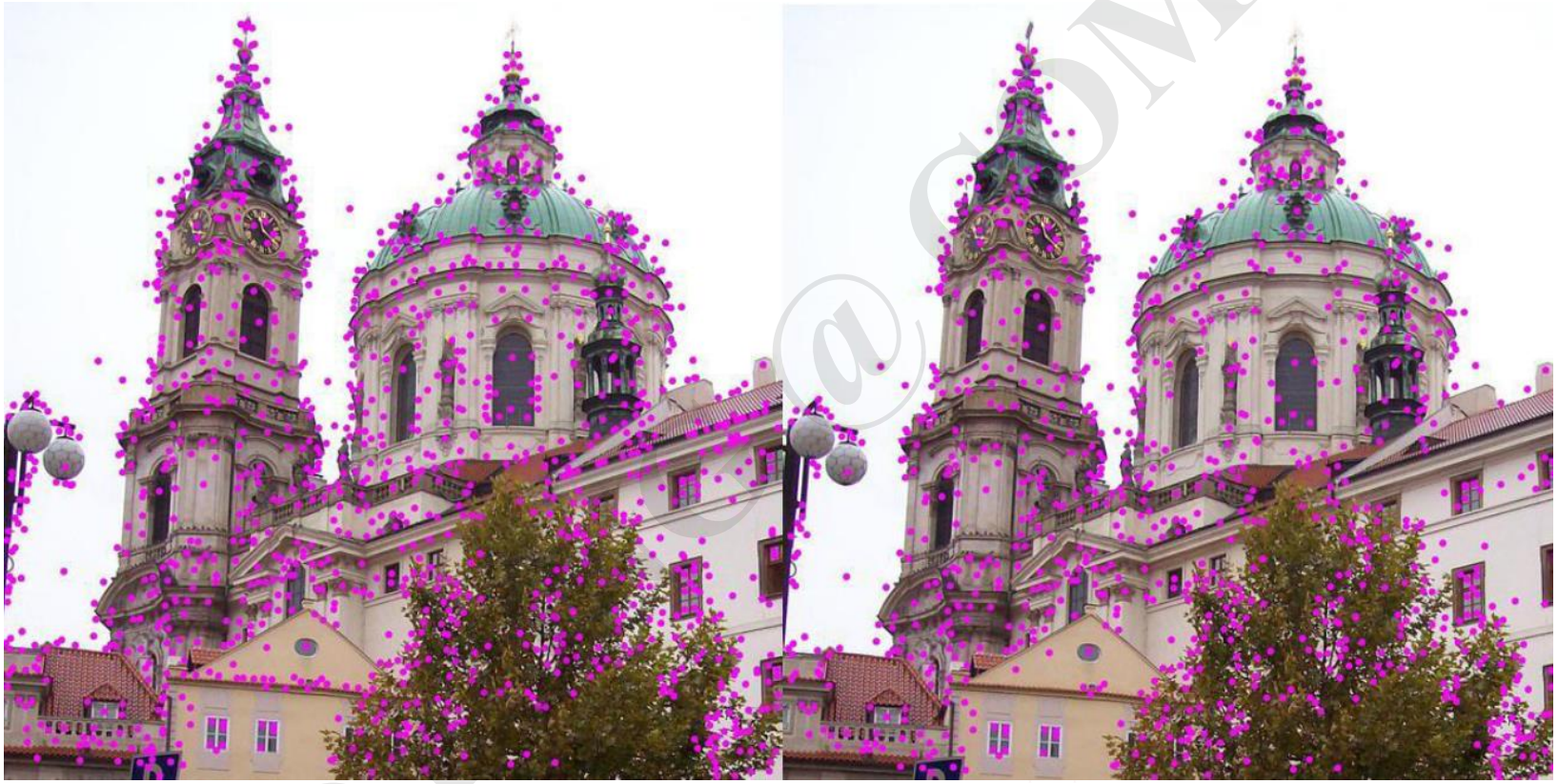
Hessian Matrix



At an edge candidate, the principal curvature across the edge is much larger than that along the edge. To remove the edge candidates, we calculate **Hessian Matrix**. The eigenvalues of H are proportional to the principal curvatures of D.

https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

Step 2: Refinement



Removal of high-contrast keypoints residing on edges

http://upload.wikimedia.org/wikipedia/commons/4/44/Sift_keypoints_filtering.jpg

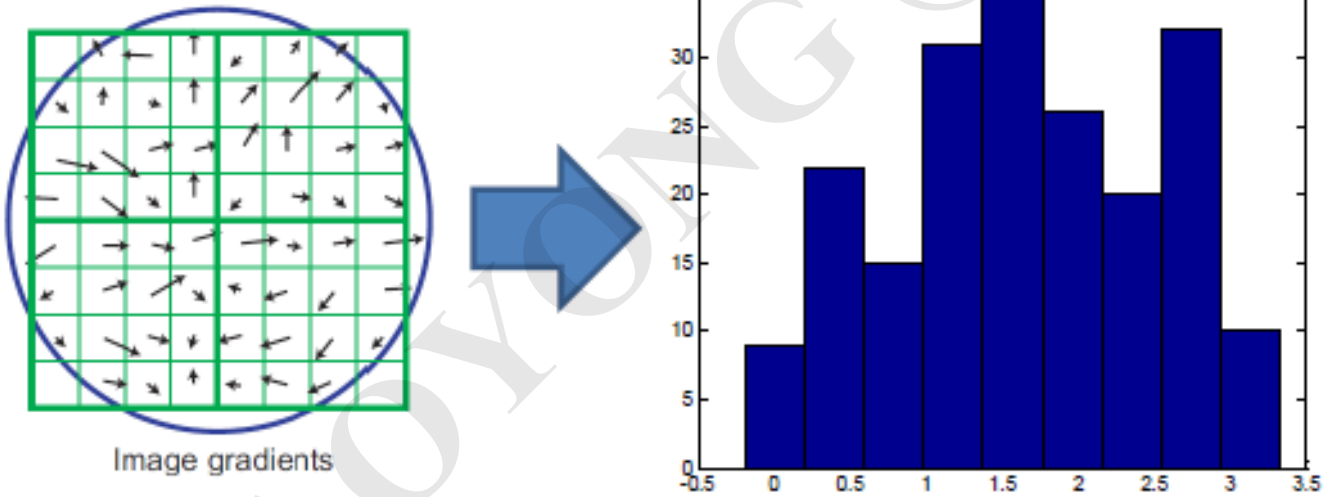
Step 3: Orientation Assignment

- Compute the gradient magnitudes and orientations in a small window around the keypoint – at the appropriate scale.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$



Histogram of gradient orientation – the bin-counts are weighted by gradient magnitudes and a Gaussian weighting function. Usually, 36 bins are chosen for the orientation.

Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 3: Orientation Assignment

- Assign the **dominant orientation** as the orientation of the keypoint.
- In case of multiple peaks or histogram entries more than 0.8 times the peak, create a **separate** descriptor for **each** orientation (they will all have the same scale and location)

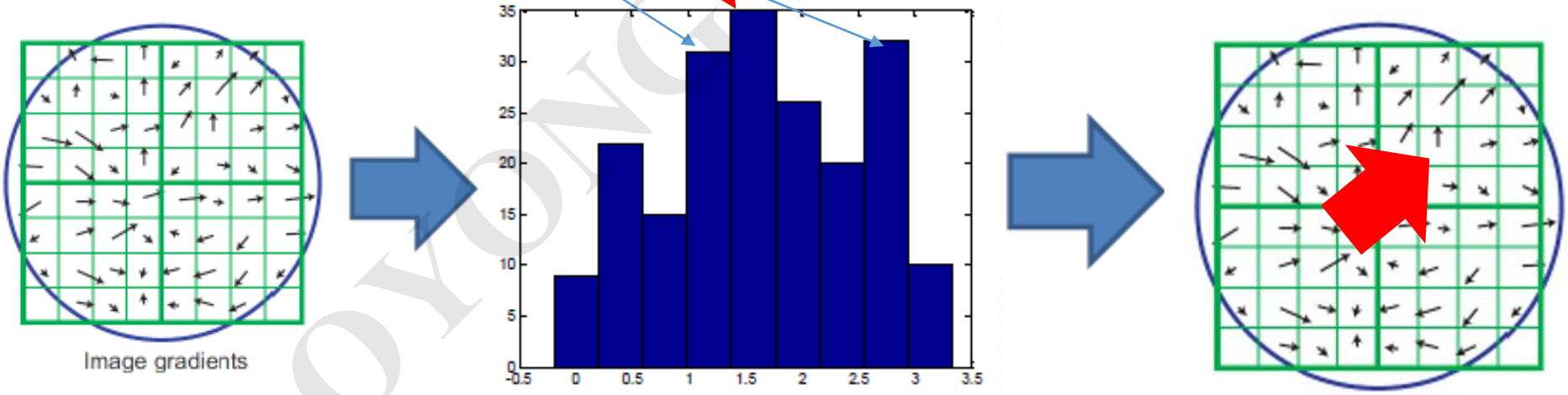


Image gradients

Assigning orientation

Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 3: Orientation Assignment

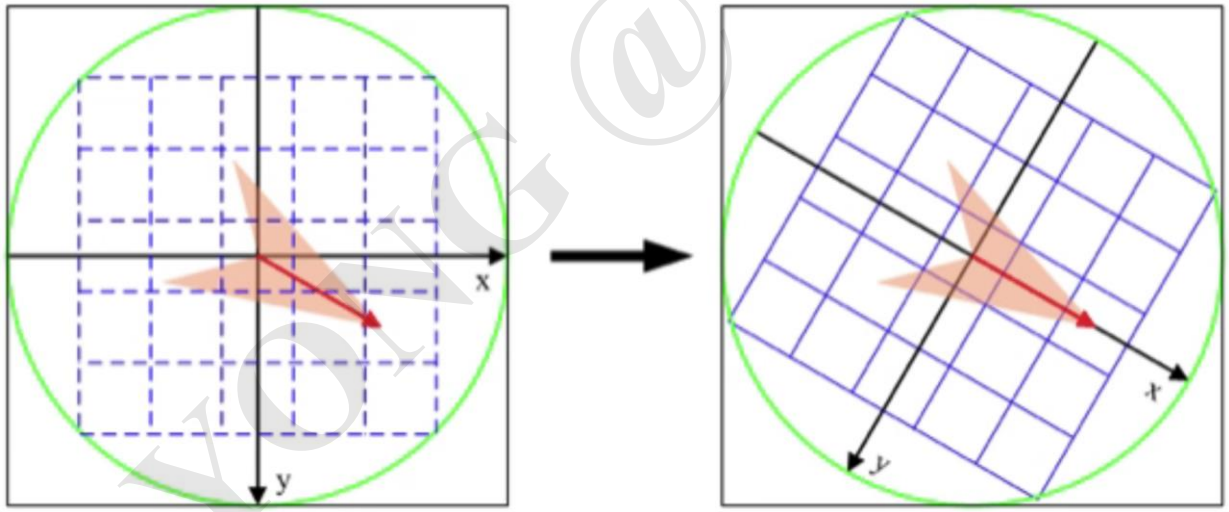
Original image

Keypoints and their orientations

https://www.vlfeat.org/overview/sift.html

Step 4: Build descriptions

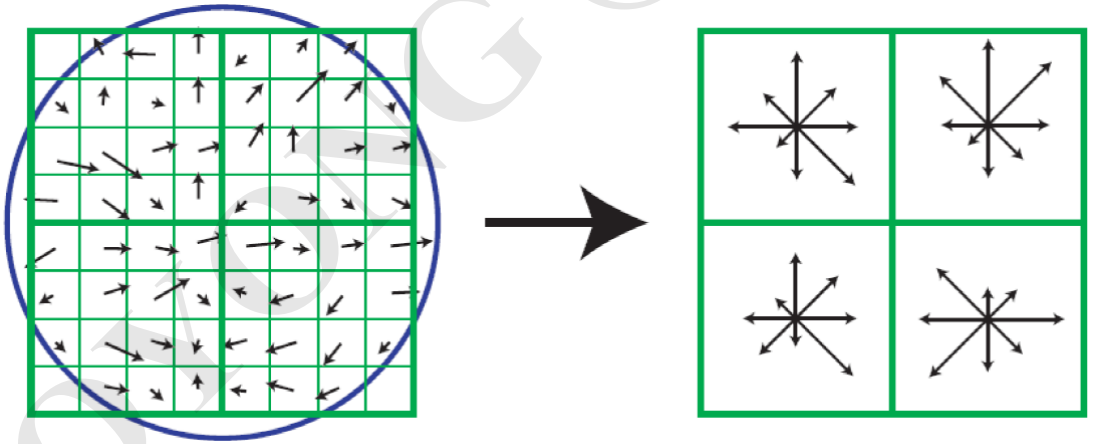
- In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation



Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 4: Build descriptions

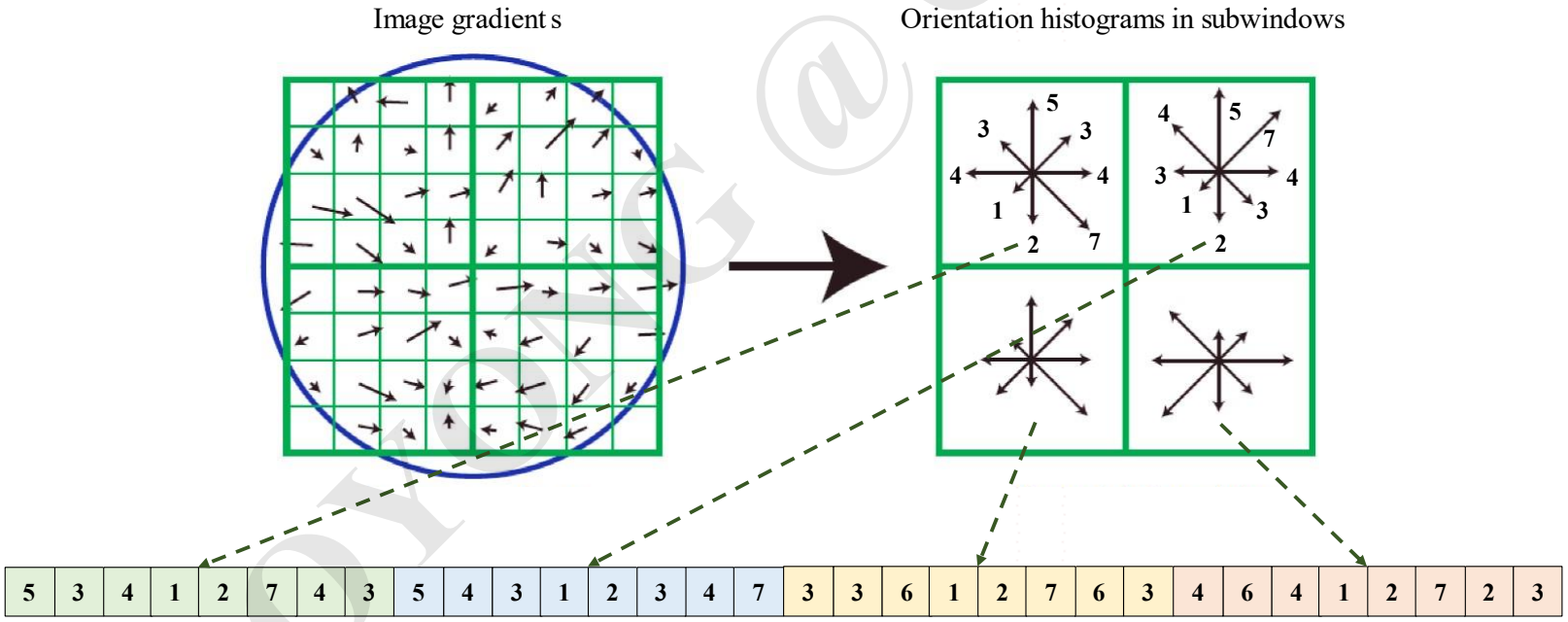
- Consider a small region around the keypoint. Divide it into *n* x *n* cells (for example, *n* = 2, Each cell is of size 4 x 4).
- Build a **gradient orientation histogram** in each cell. Each histogram entry is weighted by the *gradient magnitude* and a *Gaussian weighting function* with σ = 0.5 times window width.



Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 4: Build descriptions

- We now have a descriptor of size rn^2 if there are r bins in the orientation histogram.
- Typical case used in the SIFT paper: $r = 8$, $n = 4$, so length of each descriptor is 128.

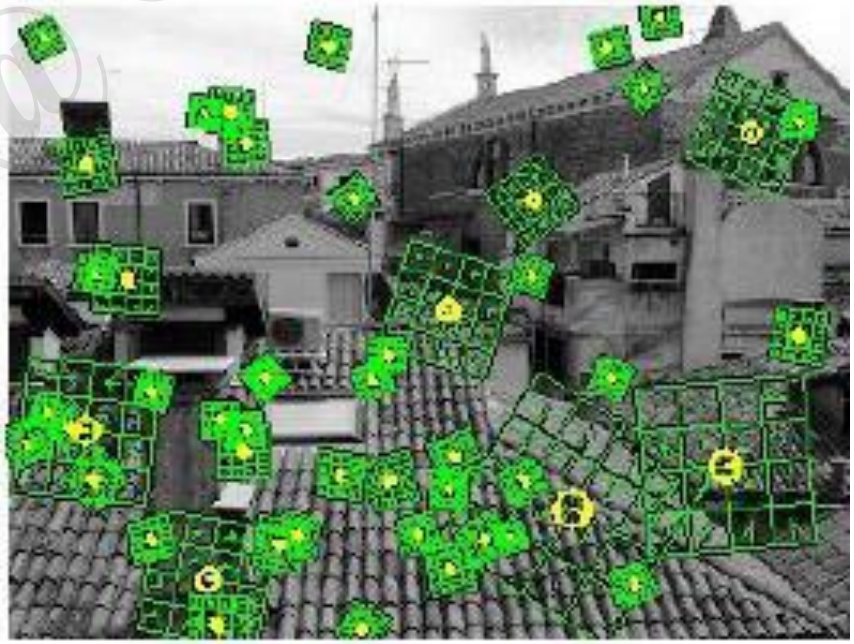


128-element SIFT feature vector

Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Step 4: Build descriptions

- To achieve scale-invariance, the size of the window should be adjusted as per scale of the keypoint. Larger scale = larger window.

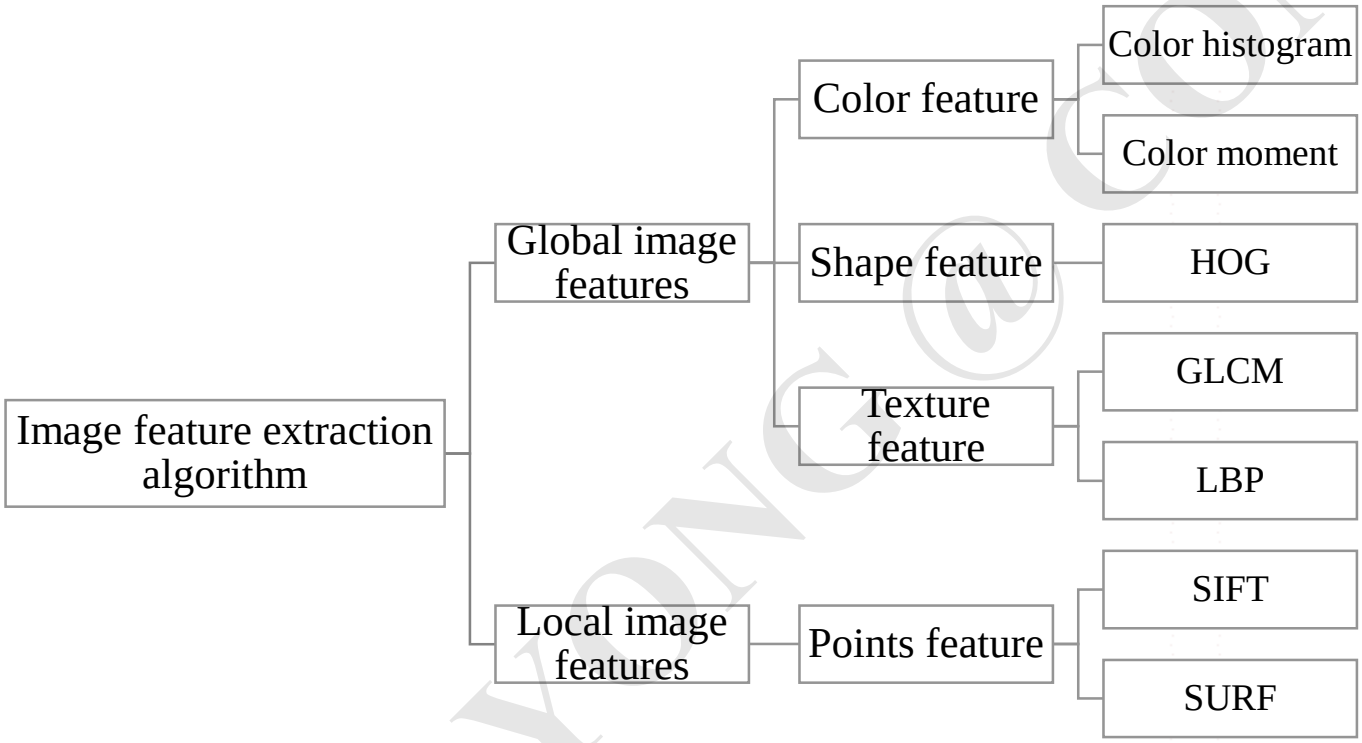


Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

Examples from IMHere

Examples from IMHere

Features



Histogram of Oriented Gradient, HOG

Gray-Level Co-occurrence Matrix, GLCM

Local Binary Pattern, LBP

Scale-invariant feature transform, SIFT

Speeded Up Robust Features, SURF

The New Toy

The New Toy

```python
import cv2,numpy as np,copy
# machter reference https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

# declare a video capture object
vid = cv2.VideoCapture(0)
inversion_on=False
keypoints_on=False

sift = cv2.SIFT_create()
detector=sift

bf_matcher=cv2.BFMatcher_create()
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=100)
flann = cv2.FlannBasedMatcher(index_params, search_params)
matcher=flann
kp_old,des_old=None,None
traces={}

def distance(p1, p2):
    dis = ((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2) ** 0.5
    return dis

```

The New Toy

```
25  while True:
26      # capture the video frame by frame
27      ret, frame_raw = vid.read()
28      frame = cv2.flip(frame_raw, 1) # optional
29      # get dimensions of the frame
30      h, w, c = frame.shape
31      print(h,w,c)
32
33      #
34      gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
35      kp, des = detector.detectAndCompute(gray, None)
36      #print(type(kp))
37
38      if kp_old!=None:
39          matches = matcher.knnMatch(des,des_old, k=2)
40          good = []
41          for m,n in matches:
42              if m.distance < 0.8*n.distance:
43                  good.append(m)
44          #matches=bf_matcher.match(des,des_old)
45
46          new_traces={}
47          for match in good:
48              pt1,pt2=kp[match.queryIdx].pt,kp_old[match.trainIdx].pt
49              pt1,pt2=[int(pt1[0]),int(pt1[1])],[int(pt2[0]),int(pt2[1])]
50              if distance(pt1,pt2)>200: continue
```

The New Toy

```
52              if (match.trainIdx in traces):
53                  len_pts=len(traces[match.trainIdx])
54                  new_traces[match.queryIdx]=[pt1]+copy.deepcopy(traces[match.trainIdx])[:20 if le
55                  pts=np.array(new_traces[match.queryIdx],dtype='int32')
56                  #print(pts)
57                  cv2.polylines(frame,[pts],False,(255, 255, 0),1)
58              else:
59                  cv2.line(frame,pt1,pt2,(255, 255, 0),1)
60                  new_traces[match.queryIdx]=[pt1,pt2]
61          traces=new_traces
62
63      kp_old,des_old=kp,des
64      if keypoints_on: cv2.drawKeypoints(frame, kp, frame, (255, 255, 0), cv2.DRAW_MATCHES_FLAGS_DI
65
66      # display the resulting frame
67      cv2.imshow('COMP 4423', 255-frame if inversion_on else frame)
68
69      key=cv2.waitKey(1) & 0xFF
70      # quit when 'q' is pressed
71      if  key== ord('q'):
72          break
73      if key==ord('i'):
74          inversion_on=not inversion_on
75      if key==ord('k'):
76          keypoints_on=not keypoints_on
77
```

Thank you!