Python 图像处理基础 C13

信息科学与技术学院

胡俊峰

内容

- 图像数据格式与基本操作
- 特征统计与变换
- 卷积、角点
- 图像特征提取与匹配

```
import cv2   # 目前是opencv version 4
import matplotlib.pyplot as plt
cv2.__version__
```

```
: '4.9.0'
```

pip install opencv-python pip install opencv-python -i https://pypi.tuna.tsinghua.edu.cn/simple

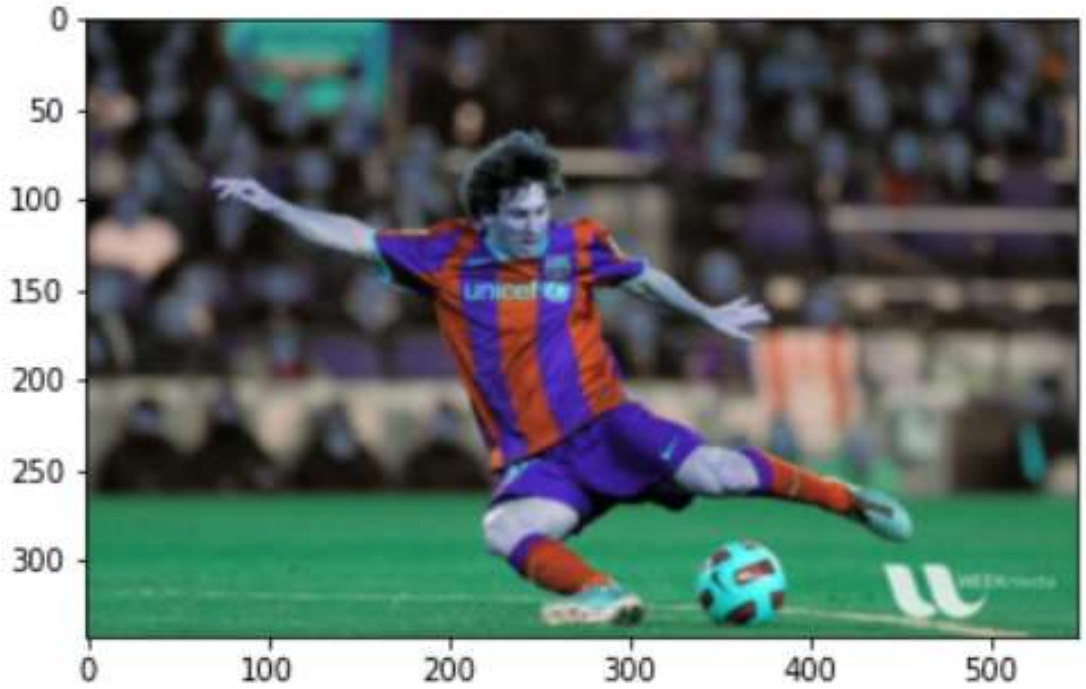
```
messi_gray = cv2.imread('common/data/messi.jpg', 0)  # 第二个参数为0表示单通道
plt.imshow(messi_gray,cmap ='gray')
messi_gray
```

```
: array([[ 43,  46,  48, ...,  55,  53,  50],
        [ 41,  46,  50, ...,  60,  58,  55],
        [ 46,  51,  56, ...,  64,  63,  60],
        ...,
        [120, 110, 107, ..., 113, 114, 124],
        [116, 119, 108, ..., 111, 122, 117],
        [107, 118, 129, ..., 104, 105, 104]], dtype=uint8)
```

```
messi_color = cv2.imread(img_dir+'data/messi.jpg') # default flag is 1 "color"
print(type(messi_color), messi_color.shape, messi_color.dtype)
my_show(plt.gca(), messi_color)
# 实际编码 GBR - 习惯 RGB
```

messi_rgb = cv2.cvtColor(messi_color, cv2.COLOR_BGR2RGB)

```
<class 'numpy.ndarray'> (342, 548, 3) uint8
```




```
# opencv is GBR; matplotlib is RGB.
my_show(plt.gca(), messi_color[:,:,::-1]) # walk last axis in opposite order (we'll never do this again!)
```

```
# we can also use scipy to read in images:
from scipy import ndimage
img = ndimage.imread(img_dir+'data/messi.jpg') # default is 1 RGB
#img1 = plt.imread(img_dir+'data/messi.jpg')
my_show(plt.gca(), img1)
img1.shape
```

```
C:\Users\hjf_p\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
(342, 548, 3)
```

三通道真彩色编码

Since messi_rgb is "just" a NumPy array, we can do NumPy array things:

```
1  print(messi_rgb[100,100],      # access a pixel
2        messi_rgb[300,:,:].shape) # sub-select a row; it's an array also.   take
```

```
[200 166 156] (548, 3)
```

一个三通道像素，每行的数据为548*3的矩阵

```
1  # pixels are people ... err ... arrays too
2  pixel = messi_rgb[100,100]
3  print(type(pixel),
4        pixel.shape,   # 1-D, scalar, array
5        pixel)
```

```
<class 'numpy.ndarray'> (3,) [200 166 156]
```

切片操作:

```
1  # massi's right wrist has a white spot!
2  messi_rgb[100:105,100:105]=[255,255,255] # white (note, our target pixel also had 3 spots
3  my_show(plt.gca(), messi_rgb)
```

```
1  ball_soi = messi_rgb[280:340, 330:390] # "soi" = square of interest :)
2  messi_rgb[273:333, 100:160] = ball_soi  # copy to new area
3  my_show(plt.gca(), messi_rgb)
```

对图片的简单像素操作

```
# 加载灰度图像
gray_lena = cv2.imread('lena.jpg', 0)  # 0表示加载灰度图像
print(gray_lena.shape)
plt.imshow(gray_lena)
plt.axis("off")
plt.show()

# 对图像取反
reverse_lena = 1 - lena
plt.imshow(reverse_lena)
plt.axis("off")
plt.show()

# 对图像像素做线性变换
new_lena = np.zeros_like(reverse_lena)
for i in range(reverse_lena.shape[0]):
    for j in range(reverse_lena.shape[1]):
        new_lena[i, j] = reverse_lena[i, j] * 1.2 + 0.1
new_lena = np.where(new_lena > 1.0, 1.0, new_lena)
plt.imshow(new_lena)
plt.axis("off")
plt.show()
```

```
(512, 512)
```

对图像设定颜色阈值

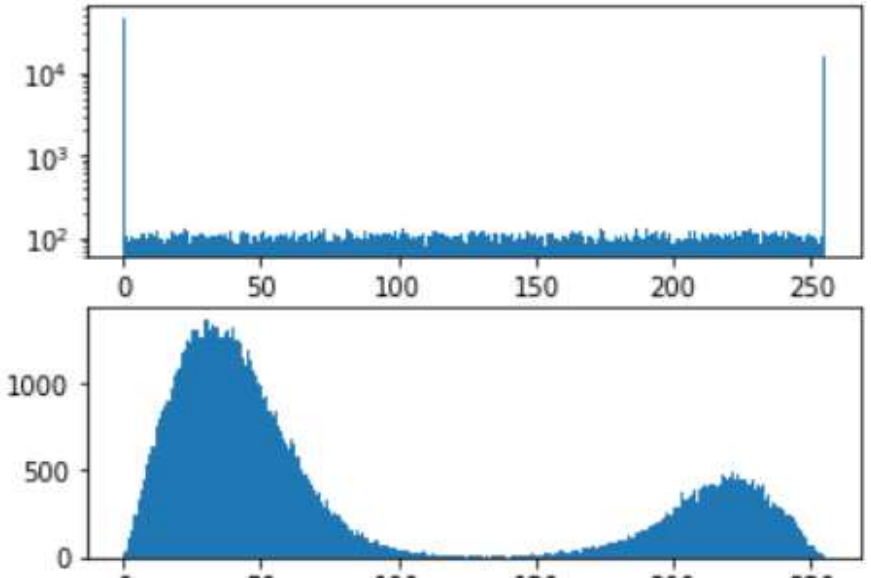
```
# 通过cv2.inRange()函数对图像设定颜色阈值
lena = cv2.imread('lena.jpg') # 读取和代码处于同一目录下的 lena.jpg
lower_bl = np.array([100, 100, 100])
upper_bl = np.array([130, 255, 255])
threshold_lena = cv2.inRange(lena, lower_bl, upper_bl)
plt.imshow(threshold_lena,'gray')
plt.axis("off")
plt.show()
```

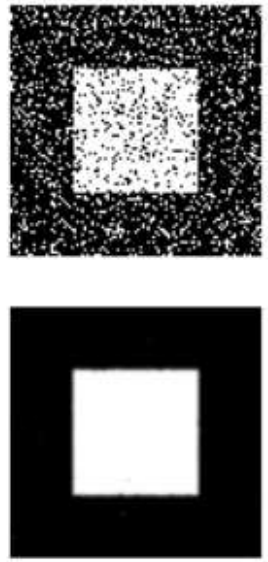
[46]:

```
fig,axes = plt.subplots(2,2,figsize=(12,4))
axes = axes.flat

# Otsu's thresholding
next(axes).hist(blurred.flatten(), 256, log=True)
# this seems weird:  shouldn't it be black and white?!?
thresh, th_otsu = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print("Optimal Thresh is", thresh)
my_show(next(axes), th_otsu, cmap='gray', interpolation=None) # force us to see what's there!

reblurred = cv2.GaussianBlur(blurred, (5,5), 0) # neighborhood, variance?
next(axes).hist(reblurred.flatten(), 256);
thresh, th_otsu = cv2.threshold(reblurred, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print("Optimal Thresh is", thresh)
my_show(next(axes), th_otsu, cmap='gray')# also: interpolation=None)
```

OTSU二值化-降噪

```
Optimal Thresh is 117.0
Optimal Thresh is 126.0
```





```python
# often we want to access color channels separately
# split to separate arrays per color (costly, prefer to access by indexing)
chans = r,g,b = cv2.split(messi_rgb)
restored = cv2.merge((r,g,b))
```

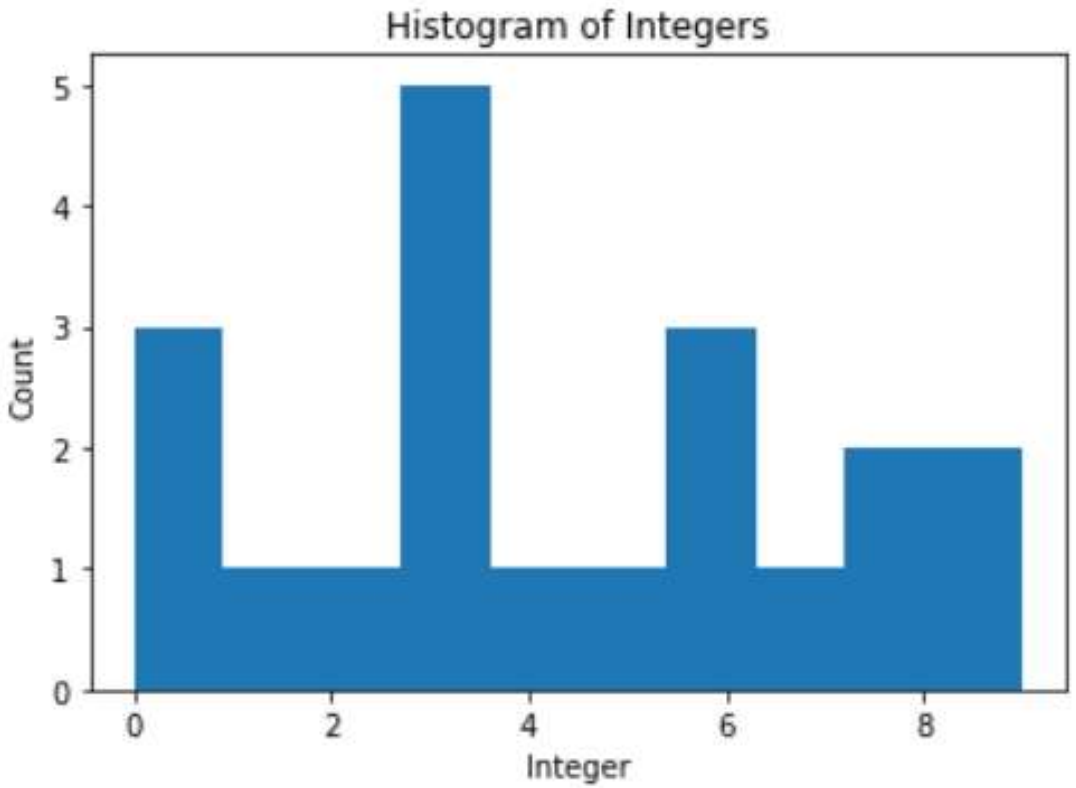
```python
fig, axes = plt.subplots(1,4,figsize=(12,3))
axes = axes.flat #  a numpy array of axes

# handle first as special case
first_axis = next(axes)
my_show(first_axis, messi_rgb)
first_axis.set_title("original")

# display per channel images
for ax, ch, name in zip(axes, chans, ["R", "G", "B"]):
    my_gshow(ax, g)
    ax.set_title("{} channel".format(name))
```

```python
ml   = my_read(img_dir+'/data/ml.png')
frog = my_read(img_dir+'/data/frog.jpg')
my_show(plt.gca(), ml)
min_r, min_c = (min(ml.shape[0], frog.shape[0]),
                min(ml.shape[1], frog.shape[1])) # 取得可叠加区域

# blending of two images:
# by:  img1 * wgt1 + img2 * wgt2 + wgt3
#      addWeights(img1, wgt1, img2, wgt2, wgt3)
dst = cv2.addWeighted( ml[:min_r, :min_c], 0.7,
                       frog[:min_r, :min_c], 0.3, 0) #加权混合
my_show(plt.gca(), dst)
```

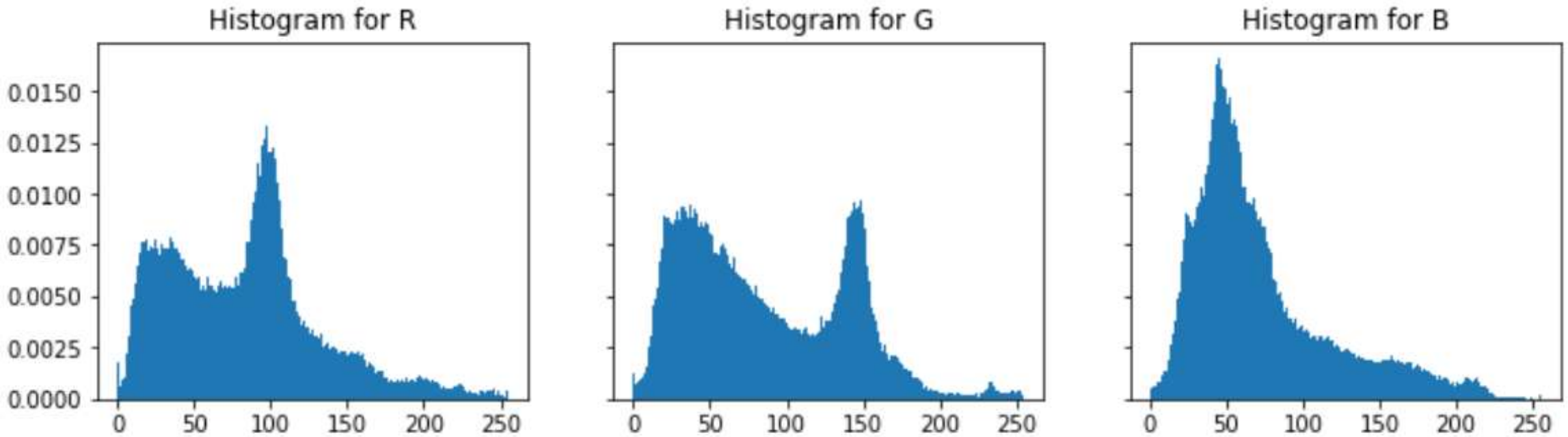
```
1  ax = plt.gca()
2  arr = npr.randint(0, 10, 20)
3  ax.hist(arr, bins=10)
4
5  ax.set_xlabel("Integer")
6  ax.set_ylabel("Count")
7  ax.set_title("Histogram of Integers")
8
9  import collections as co
10 print(co.Counter(arr)) # pure python counts of occurances
```

统计直方图

```
Counter({3: 5, 0: 3, 6: 3, 8: 2, 9: 2, 1: 1, 2: 1, 4: 1, 5: 1, 7: 1})
```



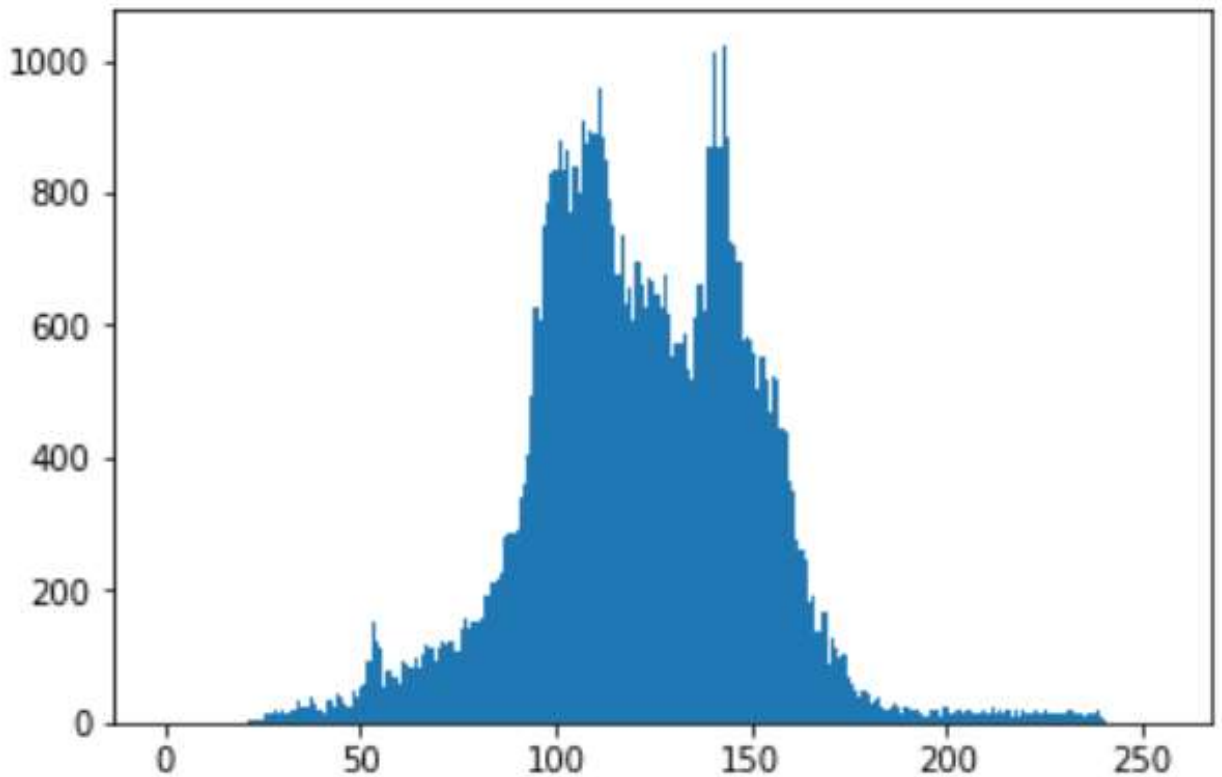
```python
# we can often just use indexing directly (see line 9)
# also show off matplotlib histograms

color_to_index = {"R":0, "G":1, "B":2}  # map strings to appropriate index in

fig, axes = plt.subplots(1,3,figsize=(12,3), sharey=True)
for ax, color in zip(axes, color_to_index):
    c = color_to_index[color]
    this_channel = messi_rgb[:,:,c].ravel() # 1D flat array view without copying

    ax.hist(this_channel, 256, normed=True)
    ax.set_title("Histogram for {}".format(color))
```



Histograms

```
1  apple = my_read_g(img_dir+'data/apple.png')
2  hist = cv2.calcHist([apple], [0], None, [256], [0,256]) # src imgs, color channels, mask
3                                                           # num bins, range
4  plt.hist(range(256), weights=hist, bins=256)
5  print(hist.shape)
```

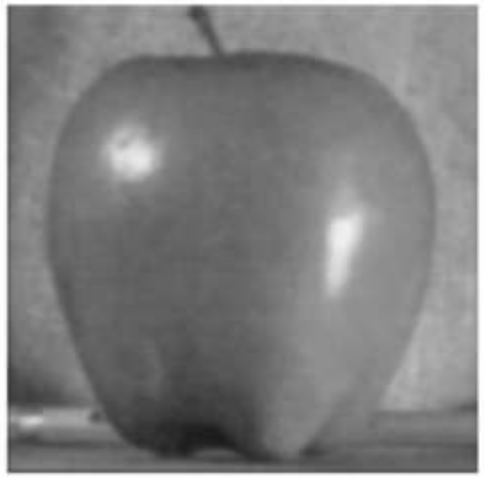
```
(256, 1)
```

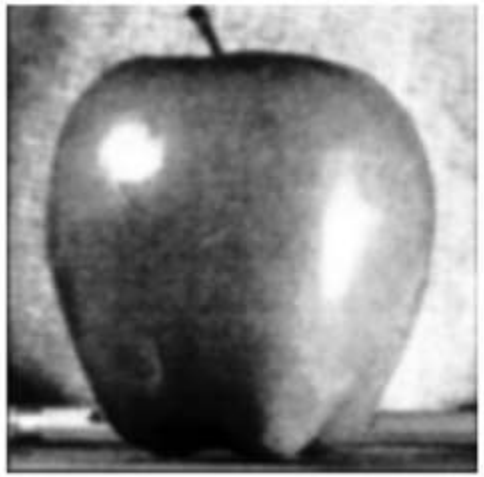


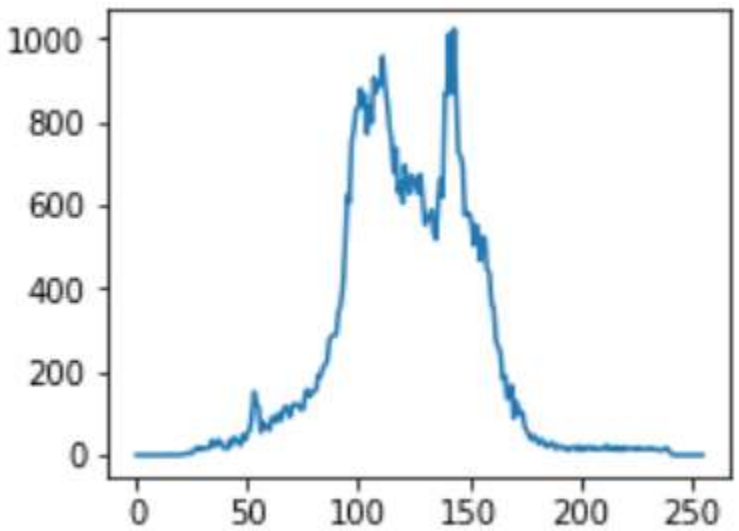
Histogram Equalization

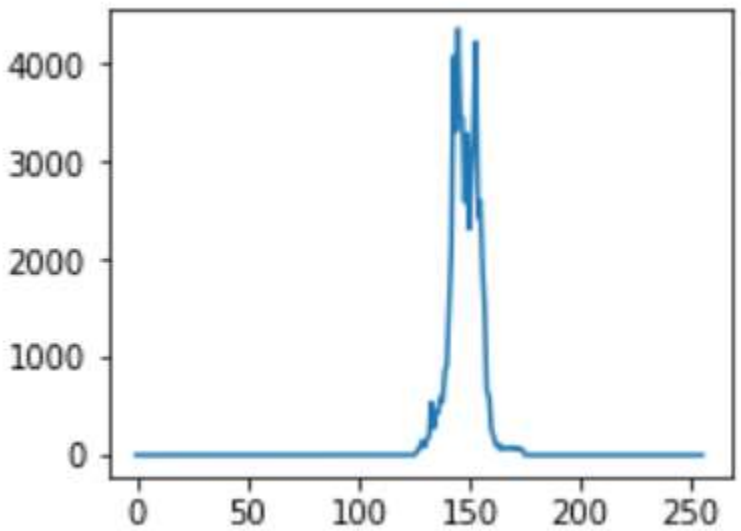
```
1  apple = my_read_g(img_dir+'data/apple.png')
2  # reduce contrast (squash intensities)
3  # new min:  100, new max: 175
4  new = np.interp(apple, [apple.min(), apple.max()], [125, 175]).astype(np.uint8)
5  # equalize using CDF technique
6  equalized = cv2.equalizeHist(new)
7  fig, axes = plt.subplots(2,3,figsize=(12,6))
8  for idx, an_apple in enumerate([apple, new, equalized]):
9      # vmin/vmax set enforced min/max gray scale values ... without them
10     # 125 -> 0 ... 175 --> 255 and linearly interpolated
11     print("min: {} max: {}".format(an_apple.min(), an_apple.max()))
12     my_gshow(axes[0, idx], an_apple, vmin=0, vmax=255)
13     hist = cv2.calcHist([an_apple], [0], None, [256], [0,256])
14     axes[1,idx].plot(hist)
```

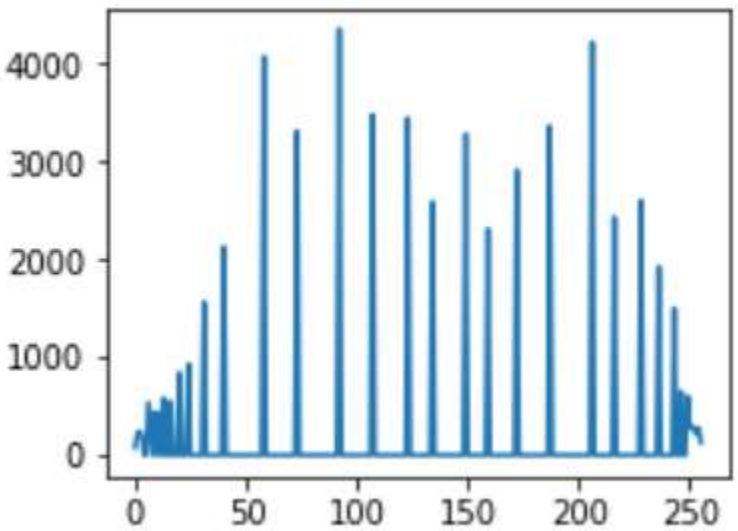
```
min: 16 max: 241
min: 125 max: 175
min: 0 max: 255
```






1000
800
600
400
200
0
0
50
100
150
200
250


4000
3000
2000
1000
0
0
50
100
150
200
250


4000
3000
2000
1000
0
0
50
100
150
200
250

卷积与卷积变换

- 边缘增强-检测
- 卷积，卷积变换

Image enhancement based on gradient

- gradient: first-o

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- Here the magnitude of the gradient vector is considered as “gradient”

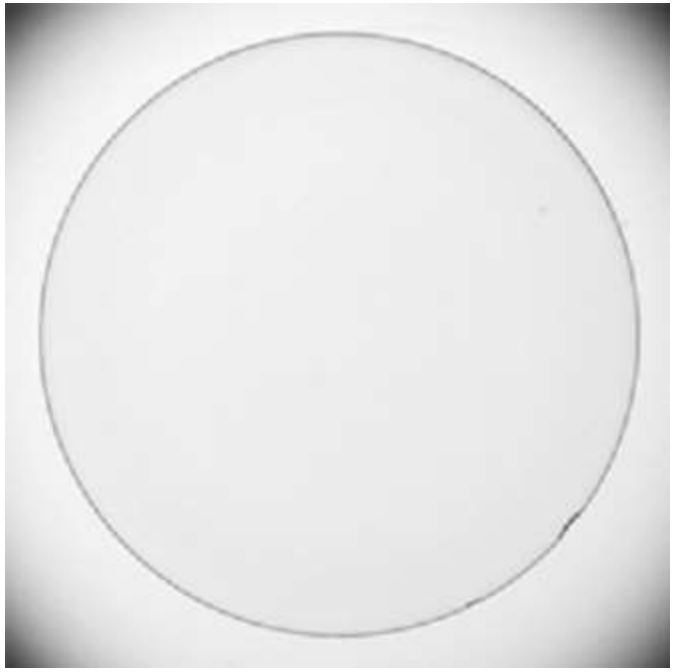
$$\begin{aligned} \nabla f &= \text{mag}(\nabla \mathbf{f}) \\ &= \left[G_x^2 + G_y^2\right]^{1/2} \\ &= \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \end{aligned}$$

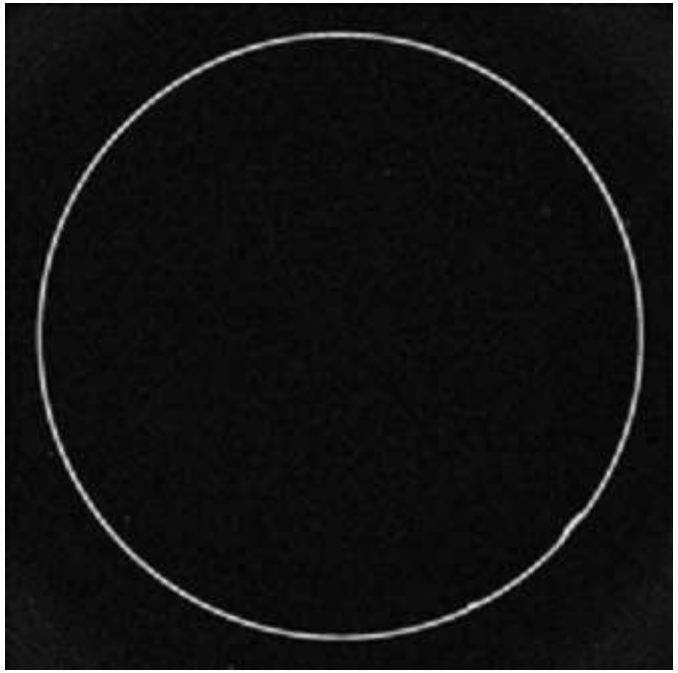
Image enhancement based on gradient

- Substitute absolute with square and square root
- take a example, we $\nabla f \approx |G_x| + |G_y|$ gradient of Z_5
- digital approximation

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Examples



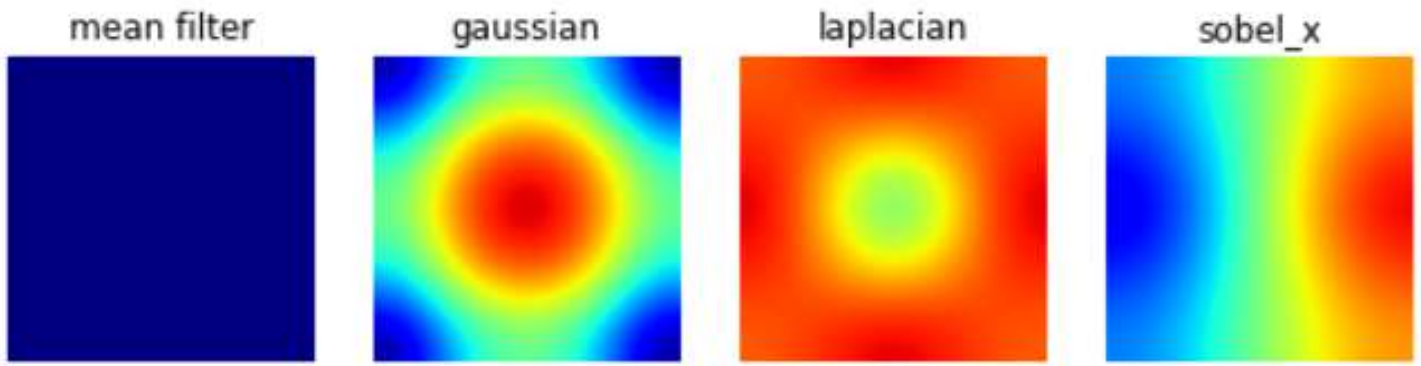


使用Canny算法进行图像边缘检测

```
import matplotlib.pyplot as plt # plt 用于显示图片
import cv2
import numpy as np
lena = cv2.imread('lena.jpg') # 读取和代码处于同一目录下的 lena.jpg
canny_image = cv2.Canny(lena, 50, 150)
plt.imshow(canny_image)
plt.axis("off")
plt.show()
```

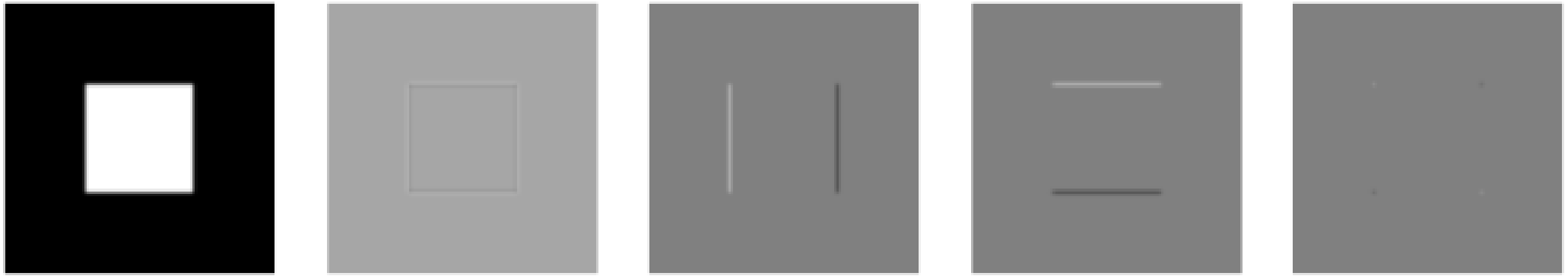
卷积算子:

```
mean_filter = np.ones((3,3))       # 均值算子
gk = cv2.getGaussianKernel(5,10) # kernal size, sigma
gaussian = gk*gk.T                 # 2D 高斯算子
                                   # laplacian算子
laplacian = np.array([[0, 1, 0],
                      [1,-4, 1],
                      [0, 1, 0]])
                                   # 横向sobel，边缘检测
sobelx = np.array([[-1,0,1],
                   [-2,0,2],
                   [-1,0,1]] )


filters = [mean_filter, gaussian, laplacian,sobelx]
filter_names = ['mean filter', 'gaussian', 'laplacian','sobel_x']

fig, axes = plt.subplots(1,4,figsize=(9,3))
for name, filt, ax in zip(filter_names, filters, axes.flat):

    my_show(ax, filt, cmap='jet')
    ax.set_title(name)
```



Sobel算子

```
# this code is written simply, but if we have to update anything, ugh
laplacian = cv2.Laplacian(box, cv2.CV_64F, ksize=5)
sobel_x   = cv2.Sobel(box, cv2.CV_64F, 1, 0, ksize=5)
sobel_y   = cv2.Sobel(box, cv2.CV_64F, 0, 1, ksize=5)
sobel_xy  = cv2.Sobel(box, cv2.CV_64F, 1, 1, ksize=5)

fig, axes = plt.subplots(1,5,figsize=(12,3))
for ax, smoother in zip(axes.flat,
                        [box,laplacian, sobel_x, sobel_y, sobel_xy]):
    my_gshow(ax, smoother)
```

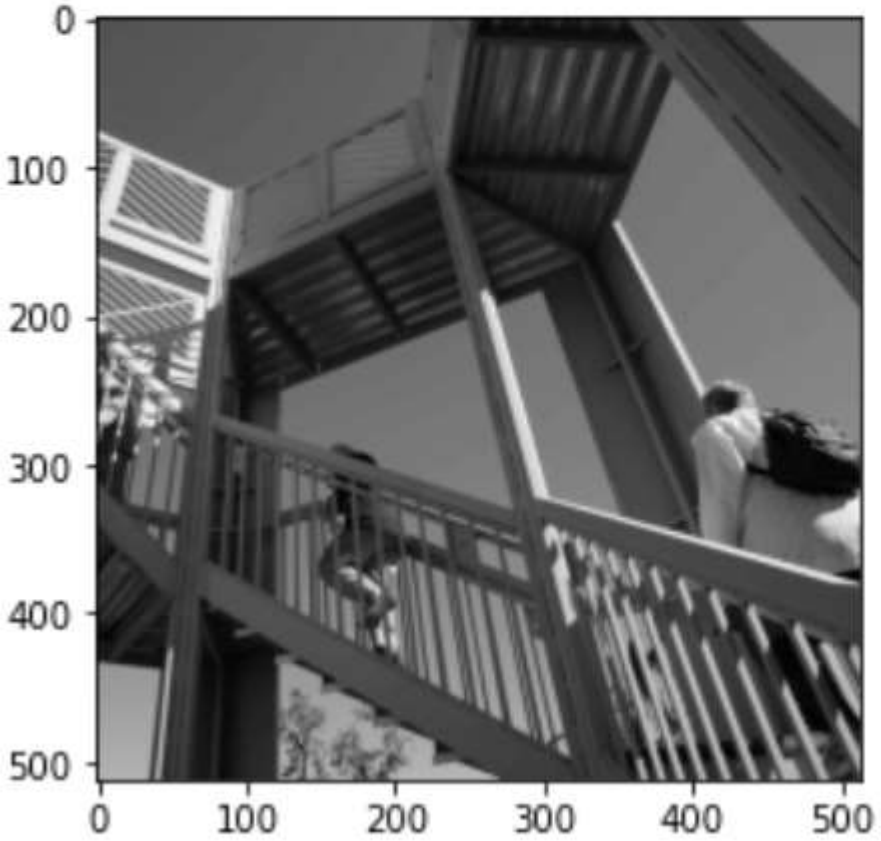


Filters and Convolutions

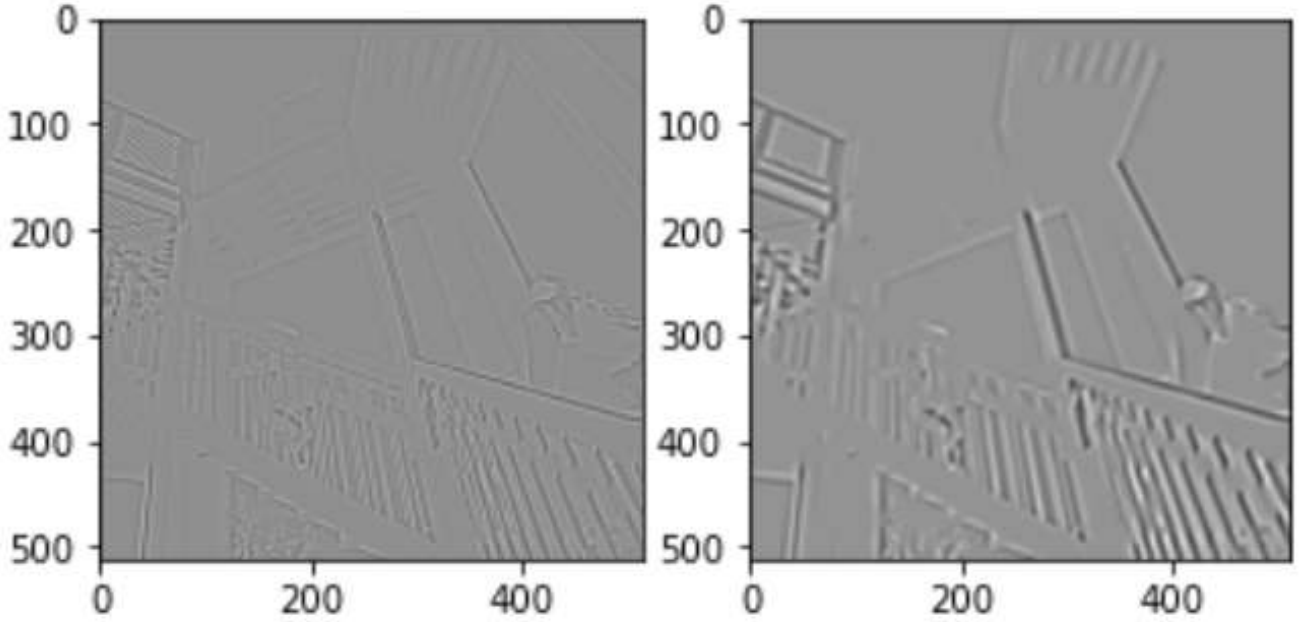
```
# simple averaging filter without scaling parameter
mean_filter = np.ones((3,3))

# creating a guassian filter
gk = cv2.getGaussianKernel(5,10)
gaussian = gk*gk.T

# different edge detecting filters
# laplacian
laplacian=np.array([[0, 1, 0],
                    [1,-4, 1],
                    [0, 1, 0]])

filters = [mean_filter, gaussian, laplacian, sobel_x, sobel_y, scharr]
filter_names = ['mean filter', 'gaussian', 'laplacian',
                'sobel x', 'sobel y', 'scharr x']

fig, axes = plt.subplots(2,3,figsize=(8,5))
for name, filt, ax in zip(filter_names, filters, axes.flat):
    # interesting variations:
    # cmap='gray', 'jet', default; interpolation = None
    my_show(ax, mag_fft(filt), cmap='jet')
    ax.set_title(name)
```

```
from scipy import ndimage, misc
import matplotlib.pyplot as plt
ascent = misc.ascent()
plt.imshow(ascent)
```

```
<matplotlib.image.AxesImage at 0x1e334596580>
```




```
fig = plt.figure()
plt.gray()  # show the filtered result in grayscale
ax1 = fig.add_subplot(121)  # left side
ax2 = fig.add_subplot(122)  # right side
result = ndimage.gaussian_laplace(ascent, sigma=1)
ax1.imshow(result)
result = ndimage.gaussian_laplace(ascent, sigma=3)
ax2.imshow(result)
plt.show()
```



梯度相关的算子

Sobel算子常用于获得图像的一阶梯度，形式上图像I关于x和y的一阶梯度分别可以用下式中的G_x和G_y来近似（这里以$k=3$的Sobel算子为例），对应cv2.Sobel()：

$$G_x=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}*I,\quad G_y=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}*I,$$

其中$*$表示卷积运算（为简化描述，之后我们仅描述卷积核，忽略卷积图像的部分）。

细心的同学可能会发现，应用Sobel算子的结果理应是近似一阶梯度的若干倍，但由于我们一般仅关注梯度绝对值的*相对大小*，这里可以忽略这个倍数的关系。

Laplacian算子被定义为$\Delta I=\frac{\partial^2 I}{\partial x^2}+\frac{\partial^2 I}{\partial y^2}$，我们一般可以用以下两种核去卷积图像得到近似的结果，对应cv2.Laplacian()：

$$\begin{bmatrix}0&-1&0\\-1&4&-1\\0&-1&0\end{bmatrix},\quad\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$$

Hessian算子通过求$\det\left(\begin{bmatrix}I_{xx}&I_{xy}\\I_{xy}&I_{yy}\end{bmatrix}\right)$来检测极值点，

请用**除了opencv中直接实现的函数以外的方法**，例如用cv2.filter2D()、numpy中的函数等实现以下功能（?分）

- $k=3$的Sobel算子
- Laplacian算子（其中Laplacian算子选用第一种卷积核）
- （逐像素地）求Hessian矩阵的行列式值（基于$k=3$的Sobel算子）
- 根据$\frac{\partial I}{\partial x}$和$\frac{\partial I}{\partial y}$（基于$k=3$的Sobel算子）求梯度方向和梯度强度（求L2 norm，不用课件里的近似）

图像开运算与闭运算

图像开运算与闭运算与膨胀和腐蚀运算有关，由膨胀和腐蚀两个运算的复合与集合操作（并、交、补等）组合成的运算构成。开运算与闭运算依据腐蚀和膨胀演变而来。

1）开运算：先对图像腐蚀后膨胀。

$A \circ S = (A \ominus S) \oplus S$

作用：用来消除小的物体，平滑形状边界，并且不改变其面积。可以去除小颗粒噪声，断开物体之间的粘连。

2）闭运算：先对图像膨胀后腐蚀

$A \bullet S = (A \oplus S) \ominus S$

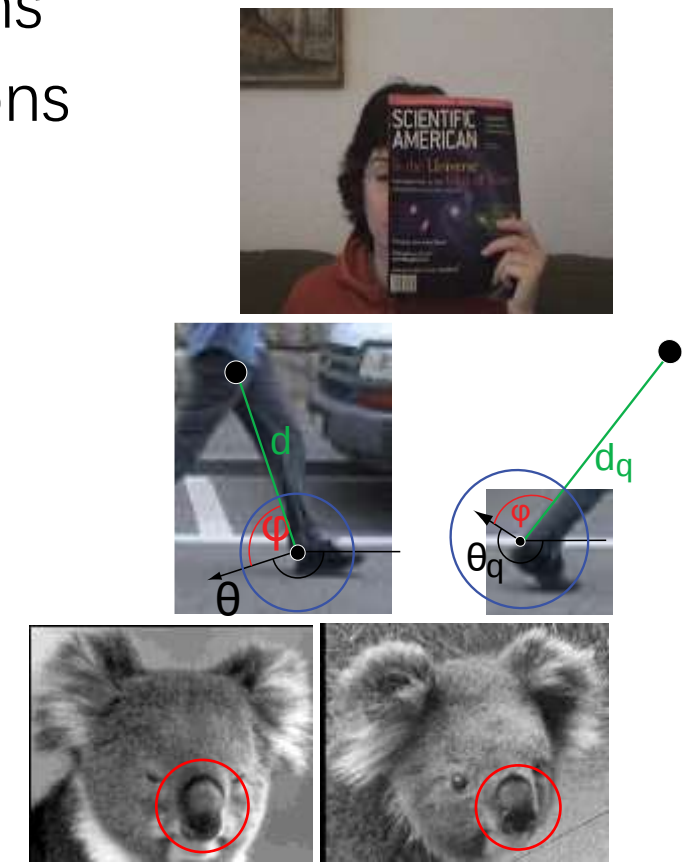
作用：用来填充物体内的小空洞，连接邻近的物体，连接断开的轮廓

Local Invariant Features: Detection & Description

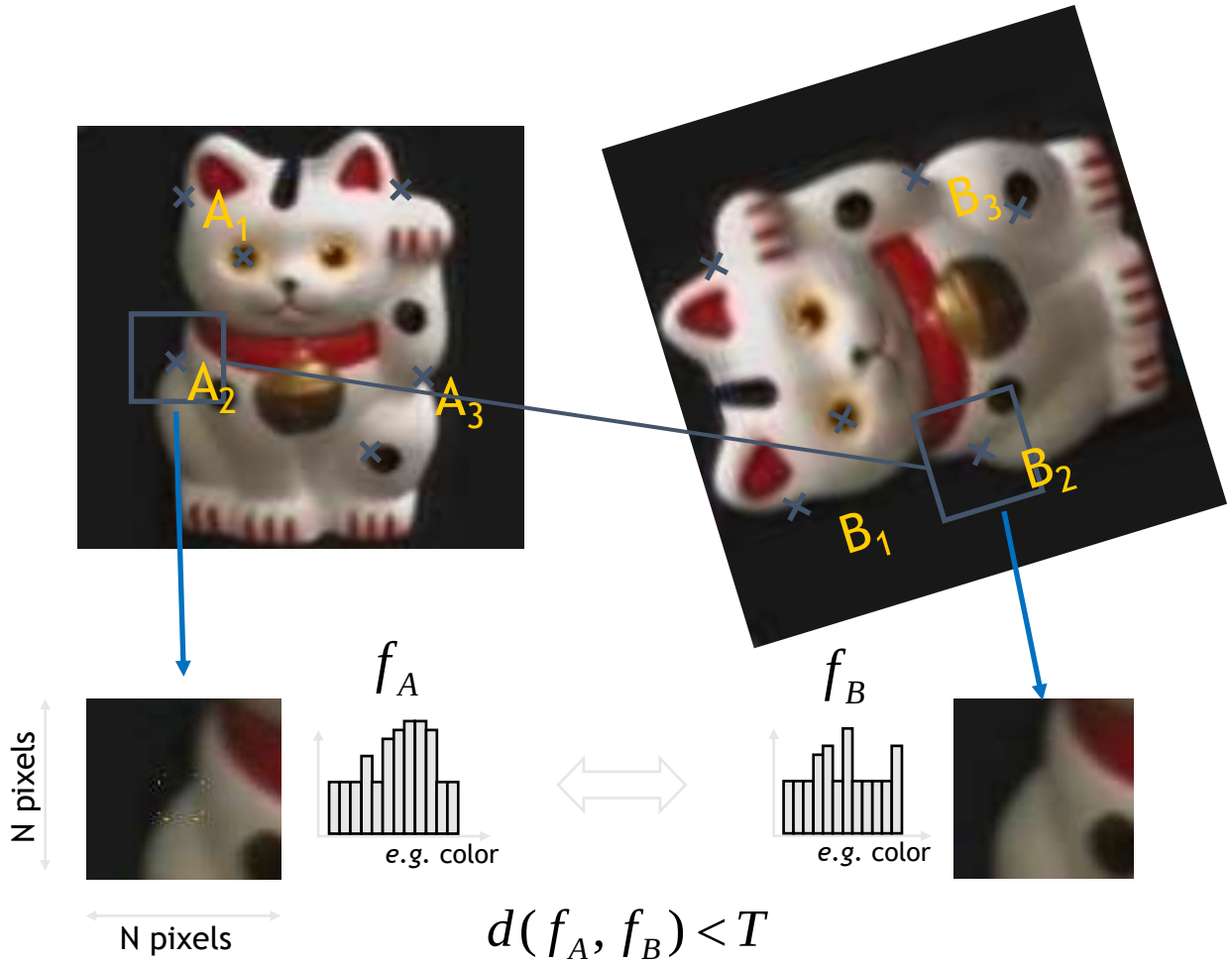
Slides credit: K. Grauman, B. Leibe

Motivation

- Global representations have major limitations
- Instead, describe and match only local regions
- Increased robustness to
  - Occlusions
  - Articulation
  - Intra-category variations



Approach



1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

Requirements

- Region extraction needs to be repeatable and precise
  - Translation, rotation, scale changes
  - (Limited out-of-plane (≈affine) transformations)
  - Lighting variations
- We need a sufficient number of regions to cover the object
- The regions should contain “interesting” structure

Many Existing Detectors Available

- Harris-/Hessian-Affine
- EBR and IBR
- Salient Regions
- Others…

[Beaudet ‘78], [Harris ‘88]

[Lindeberg ‘98], [Lowe 1999]

[Mikolajczyk & Schmid ‘01]

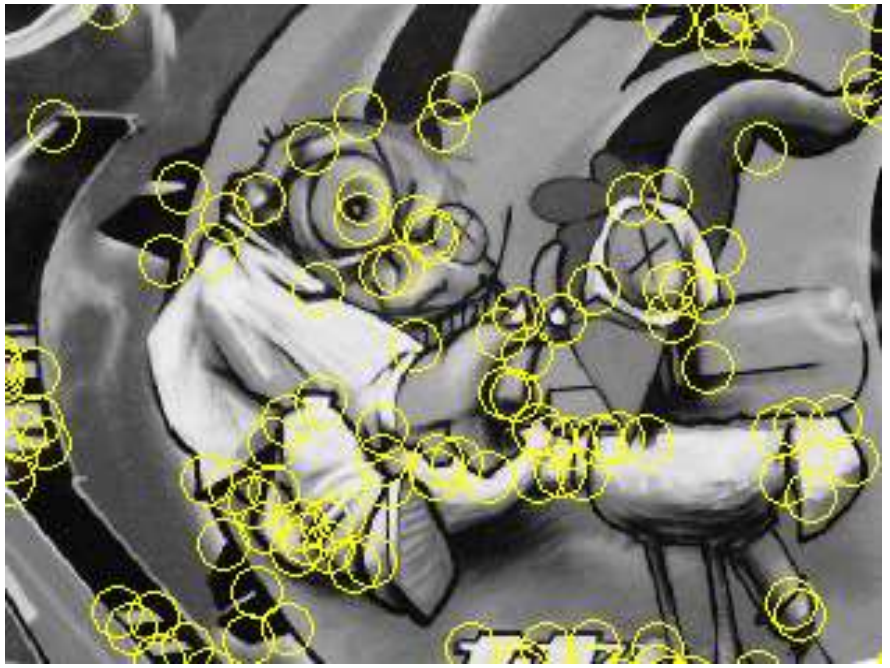
[Mikolajczyk & Schmid ‘04]

[Tuytelaars & Van Gool ‘04]

[Matas ‘02]

[Kadir & Brady ‘01]

Keypoint Localization



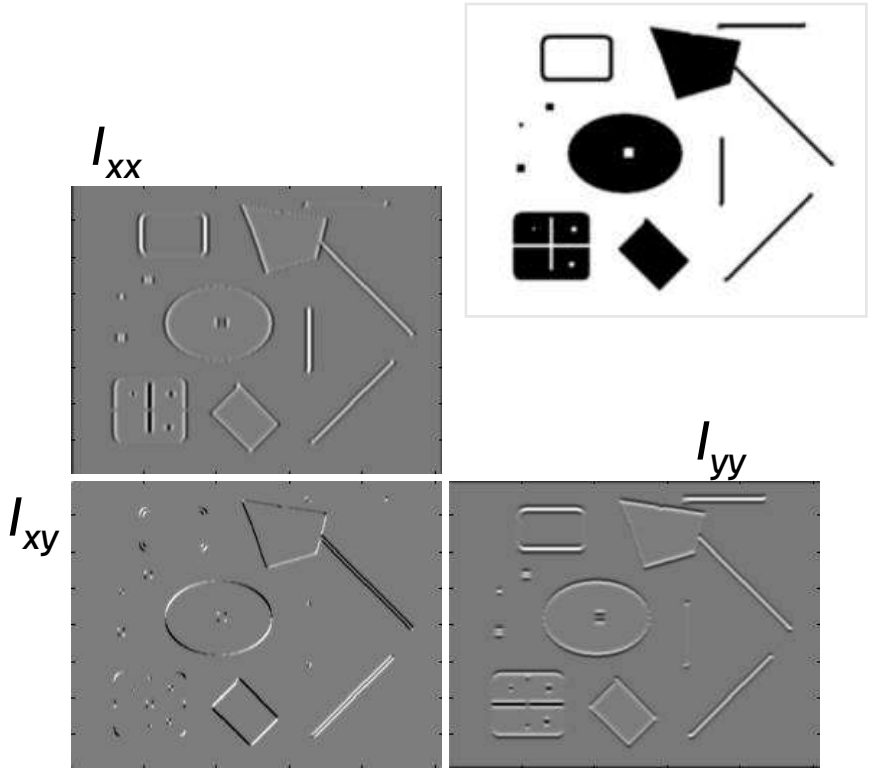
- Goals:
  - Repeatable detection
  - Precise localization
  - Interesting content

$\Rightarrow$ *Look for two-dimensional signal changes*

Hessian Detector [Beaudet78]

- Hessian determinant

$$Hessian\,(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$



Intuition: **Search for strong derivatives in two orthogonal directions**

Hessian matrix

In mathematics, the **Hessian matrix** (or simply the **Hessian**) is the square matrix of second-order partial derivatives of a function; that is, it describes the local curvature of a function of many variables. The Hessian matrix was developed in the 19th century by the German mathematician Ludwig Otto Hesse and later named after him. Hesse himself had used the term "functional determinants".

Given the real-valued function

$$f(x_1, x_2, \ldots, x_n),$$

if all second partial derivatives of f exist, then the Hessian matrix of f is the matrix

$$H(f)_{ij}(x) = D_i D_j f(x)$$

where $x = (x_1, x_2, \ldots, x_n)$ and D_i is the differentiation operator with respect to the ith argument and the Hessian becomes

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Some mathematicians[1] define the Hessian as the determinant of the above matrix.

Hessian matrix

二元函数极值存在的充分条件

观察二元函数极值存在的充分条件.

设 z=f(x,y)在(x0,y0)的一个邻域内所有二阶偏导数连续， 且,

记 .

那么，海赛矩阵.

（1）若 A>0，detH=AC-B2>0，则 H 正定，从而(x0,y0)是 f(x,y)的极小点.

（2）若 A<0，detH=AC-B2>0，则 H 负定，从而(x0,y0)是 f(x,y)的极大点.

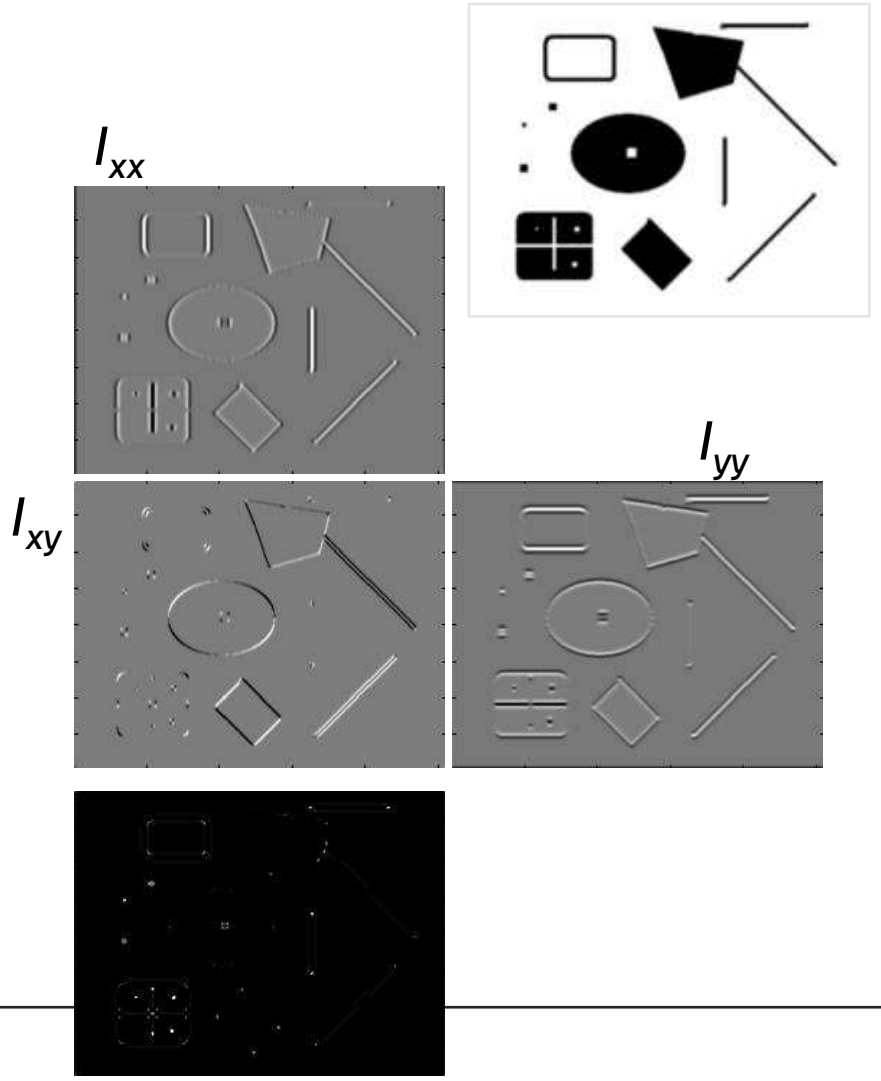
（3）若 detH=AC-B2<0，则 H 的特征根有正有负，从而(x0,y0)不是 f(x,y)的极值点.

（4）若 detH=AC-B2=0，则不能判定(x0,y0)是否为 f(x,y)的极值点.

Hessian Detector [Beaudet78]

- Hessian determinant

$$Hessian\,(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$

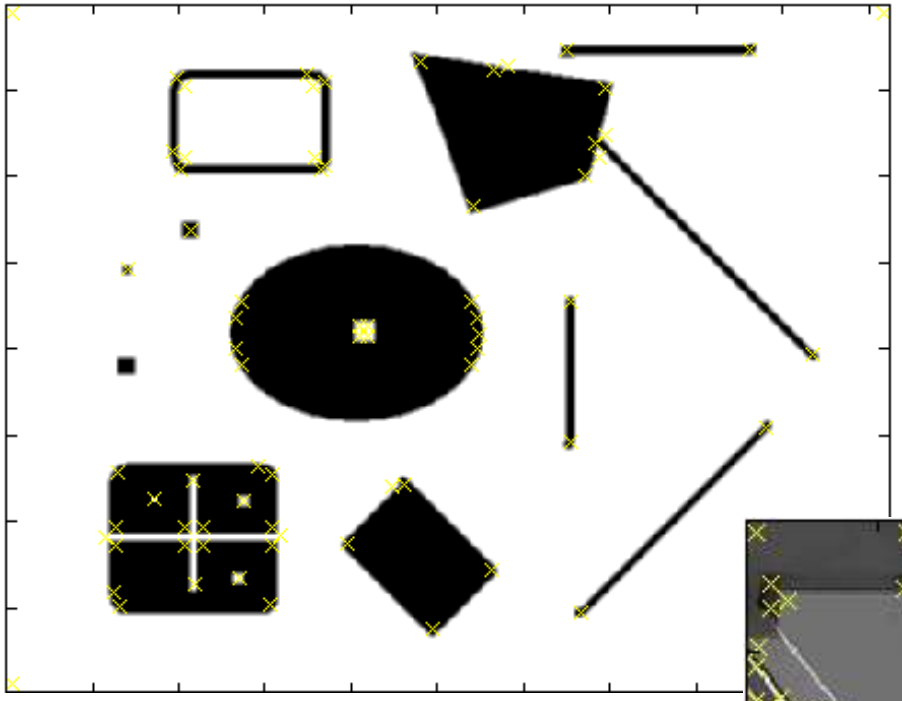


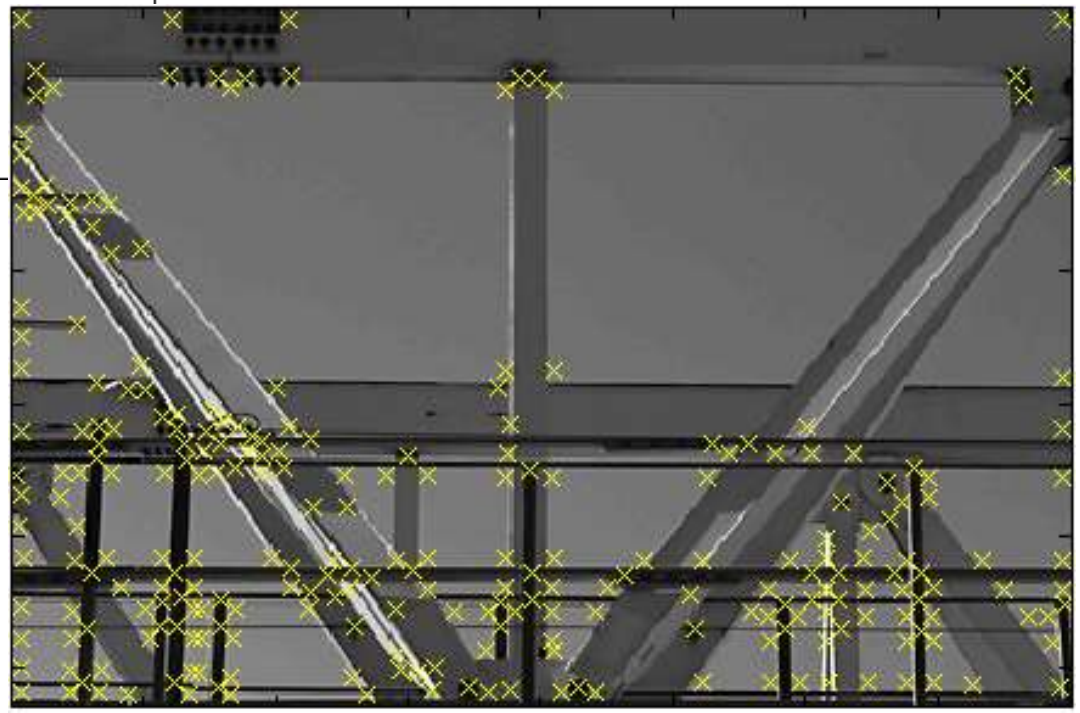
$$\det(Hessian(I)) = I_{xx}I_{yy} - I_{xy}^2$$

In Matlab:

$$I_{xx}.*I_{yy} - (I_{xy})\text{^}2$$

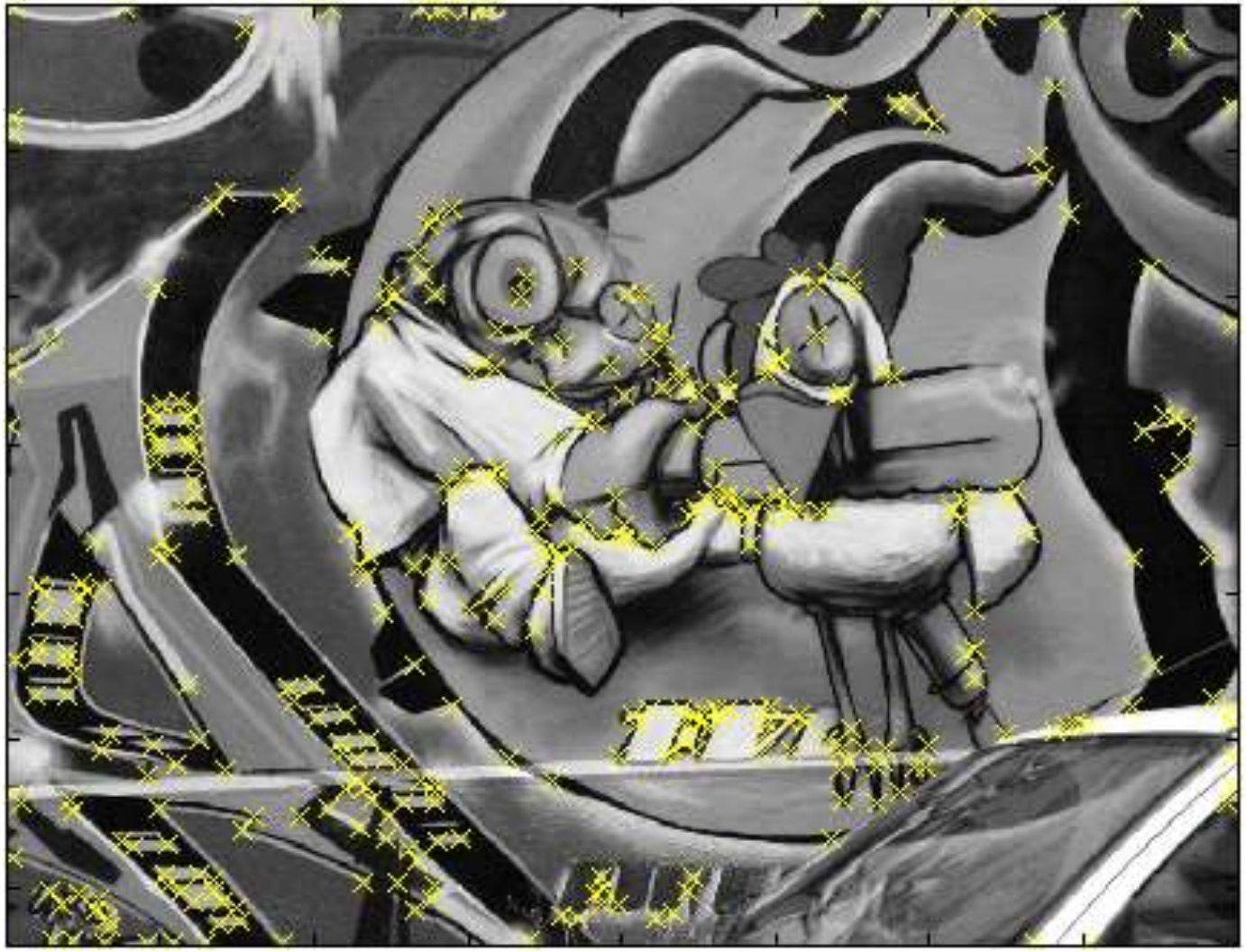
Hessian Detector – Responses [Beaudet78]



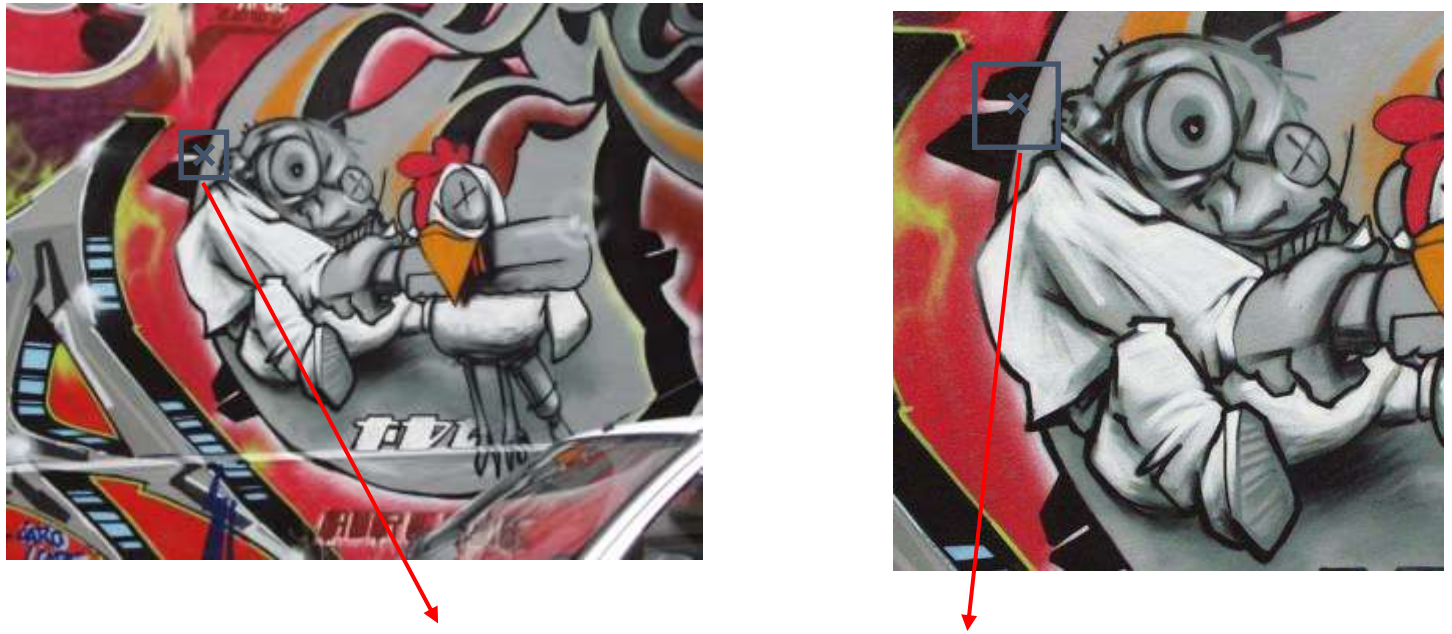


Effect: **Responses mainly on corners and strongly textured areas.**

Hessian Detector – Responses [Beaudet78]



Automatic Scale Selection： SIFT提出



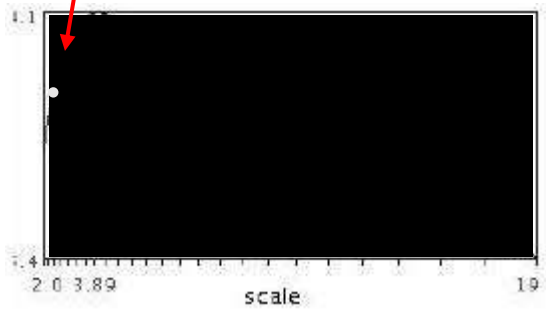
$$f(I_{i_1\ldots i_m}(x,\sigma)) = f(I_{i_1\ldots i_m}(x',\sigma'))$$

Same operator responses if the patch contains the same image up to scale factor

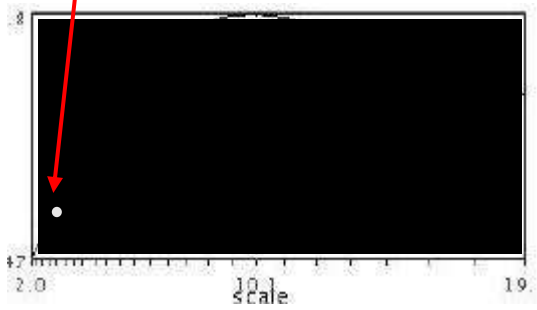
How to find corresponding patch sizes?

Automatic Scale Selection

- Function responses for increasing scale (scale signature)



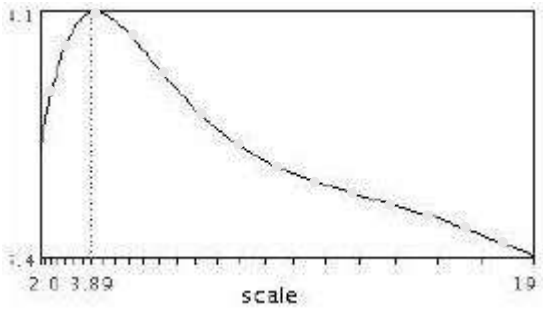
$f(I_{i_1 \ldots i_m}(x,\sigma))$



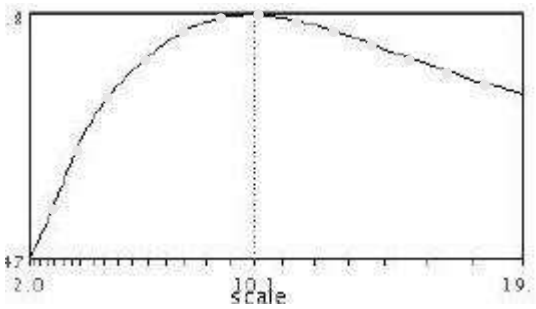
$f(I_{i_1 \ldots i_m}(x',\sigma))$

特征尺度自适应 Automatic Scale Selection

- Function responses for increasing scale (scale signature)



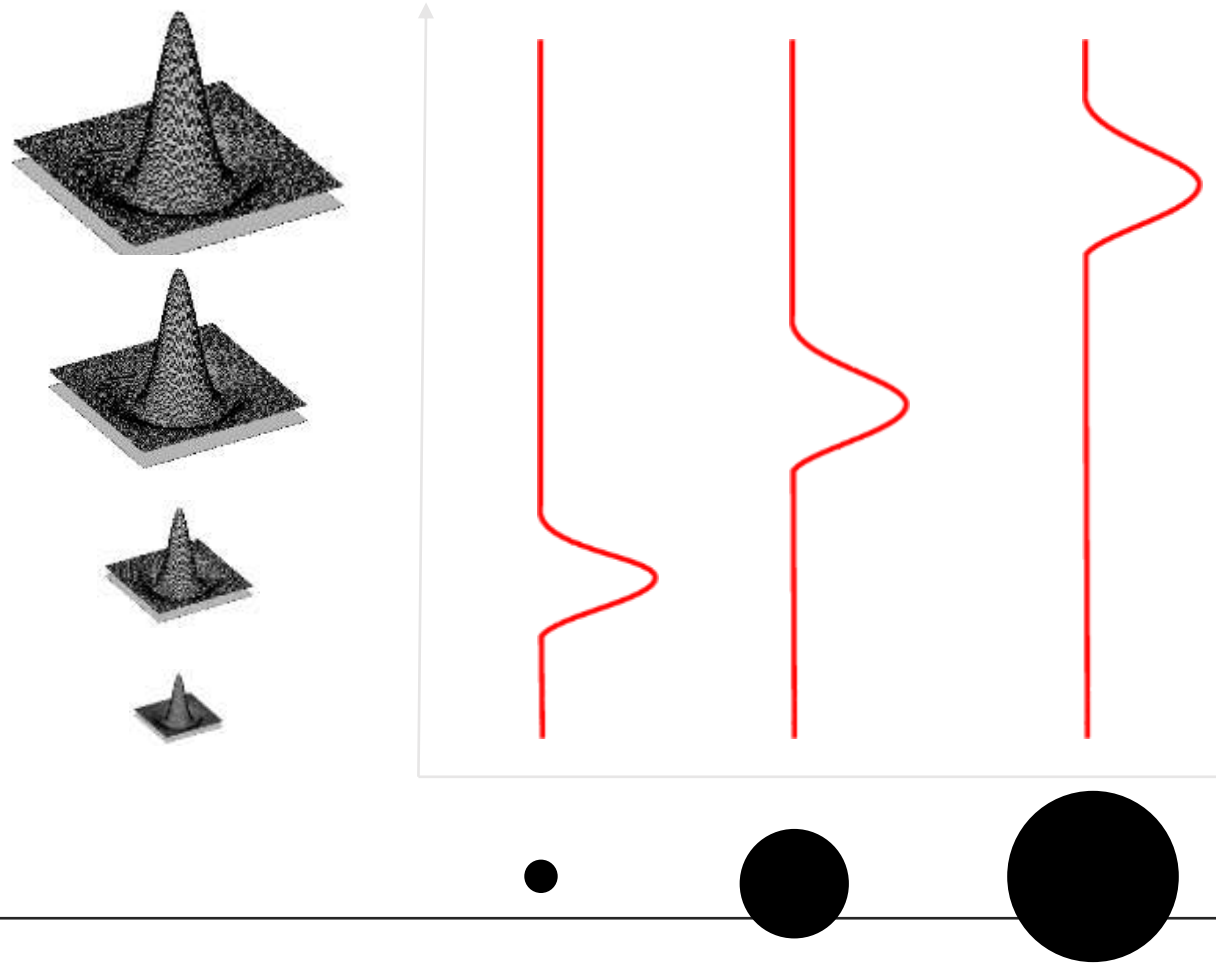
$f(I_{i_1 \ldots i_m}(x,\sigma))$



$f(I_{i_1 \ldots i_m}(x',\sigma'))$

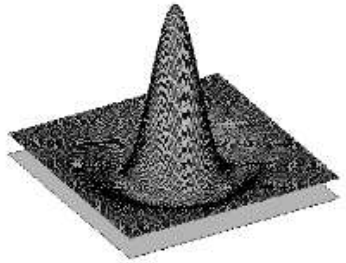
What Is A Useful Signature Function?

- Laplacian-of-Gaussian = “blob” detector

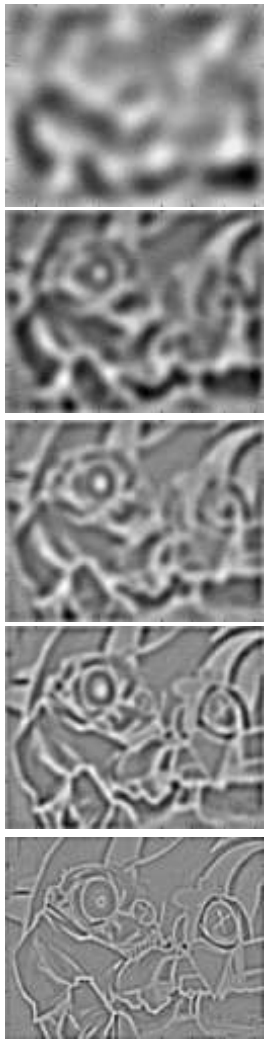


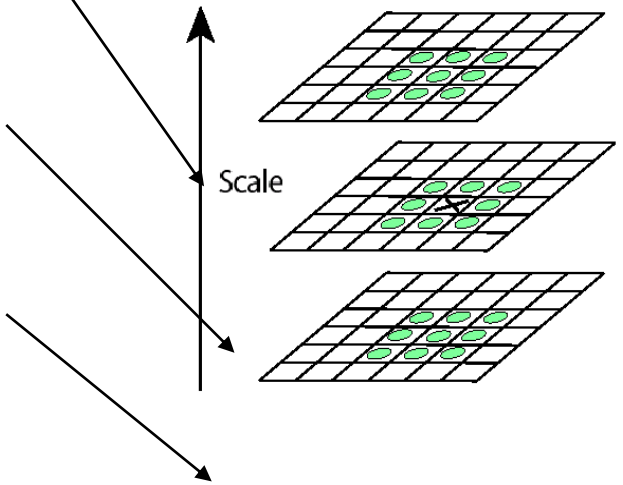
高斯拉普拉斯变换 Laplacian-of-Gaussian (LoG)

- Local maxima in scale space of Laplacian-of-Gaussian



$$L_{xx}(\sigma)+L_{yy}(\sigma)$$

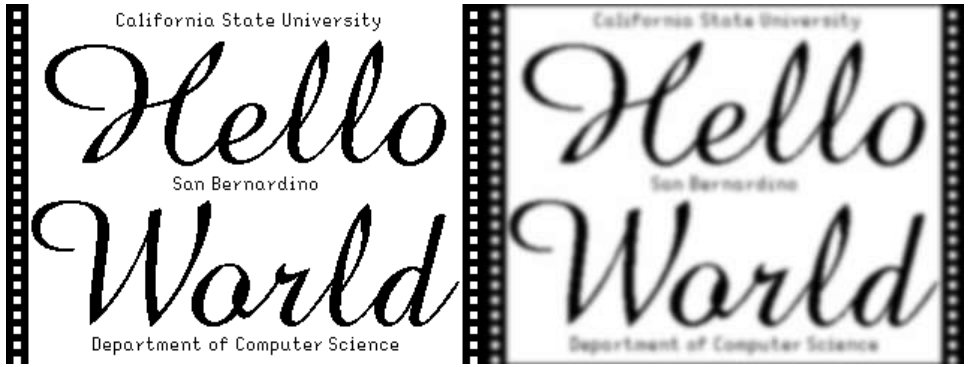


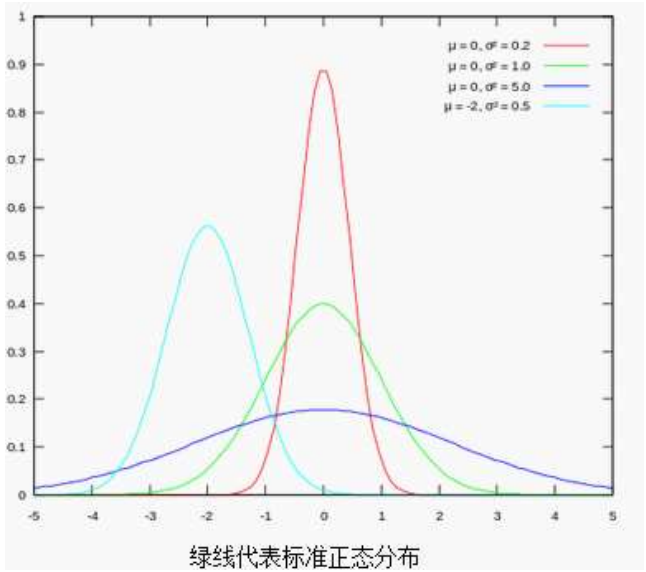


⇒ **List of**
(x, y, s)

高斯模糊与DOG

高斯模糊是在Adobe Photoshop等图像处理软件中广泛使用的处理效果，通常用它来减小图像噪声以及降低细节层次。这种模糊技术生成的图像的视觉效果是好像经过一个半透明的屏幕观察图像。





绿线代表标准正态分布

高斯模板大小的选择

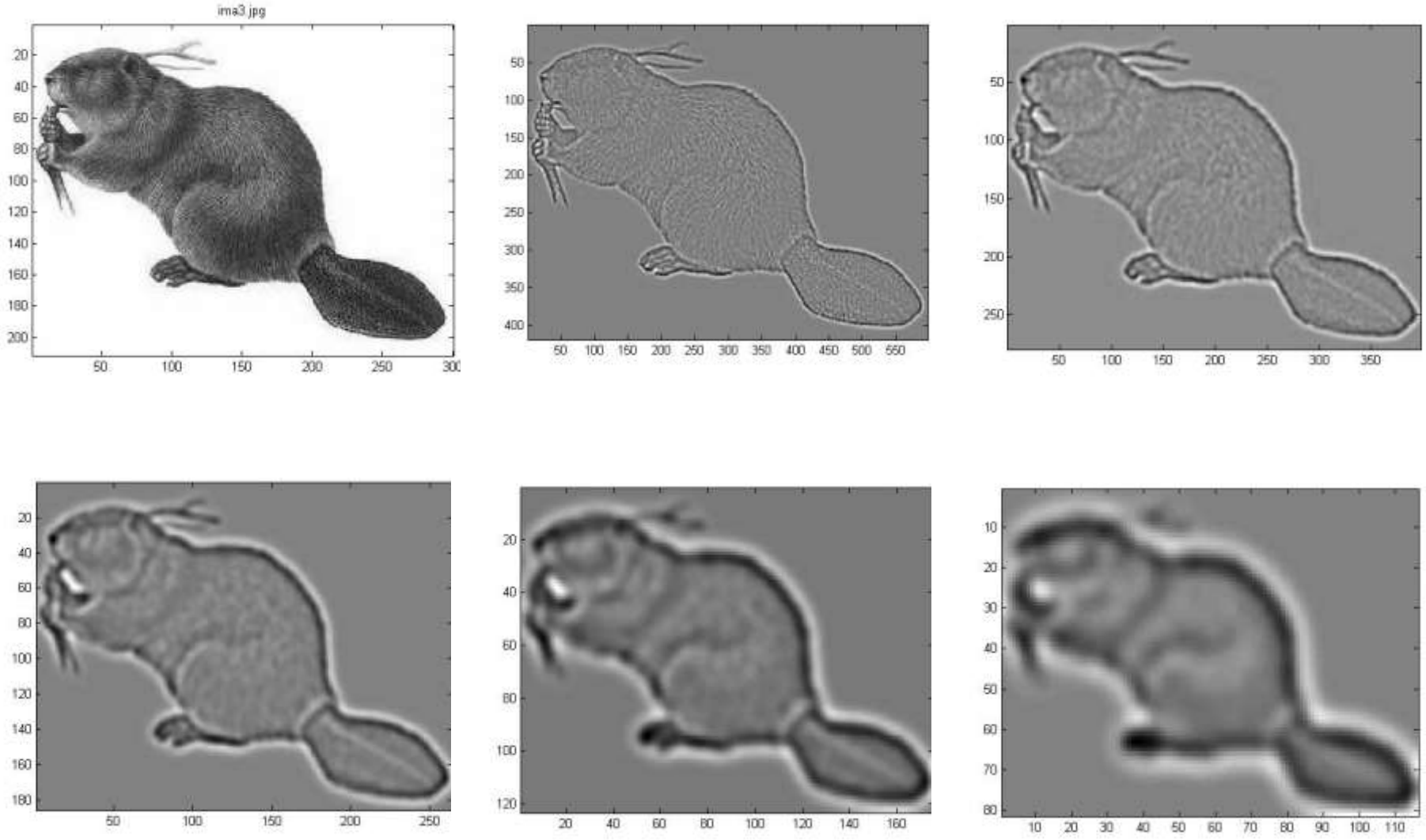
高斯模板

0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00038771	0.01330373	0.11098164	**0.22508352**	0.11098164	0.01330373	0.00038771
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067

- **在实际应用中，在计算高斯函数的离散近似时，在大概3σ距离之外的像素都可以看作不起作用，这些像素的计算也就可以忽略。**

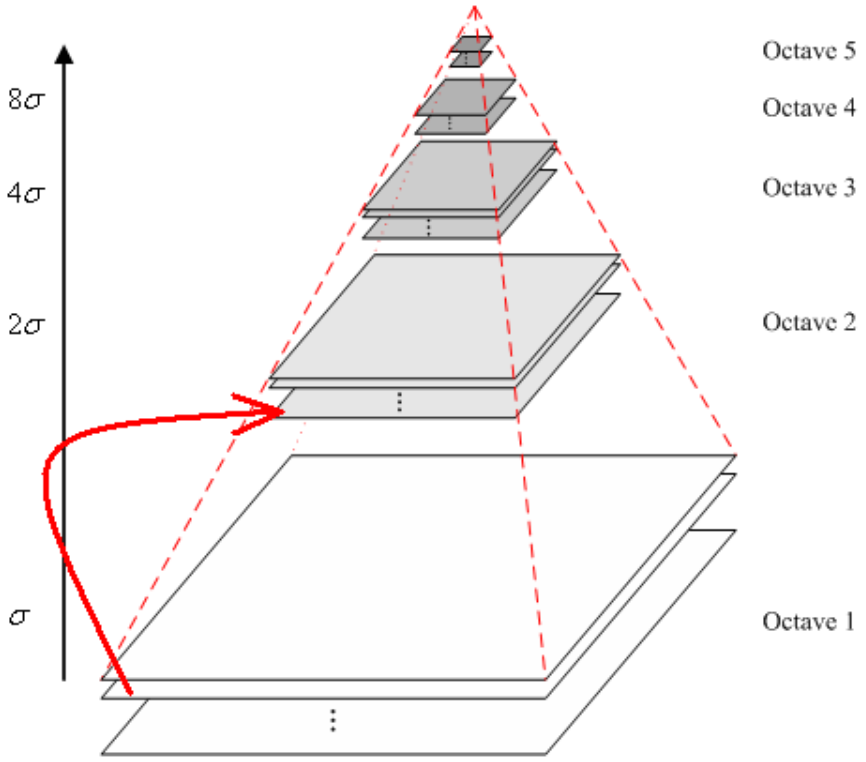
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

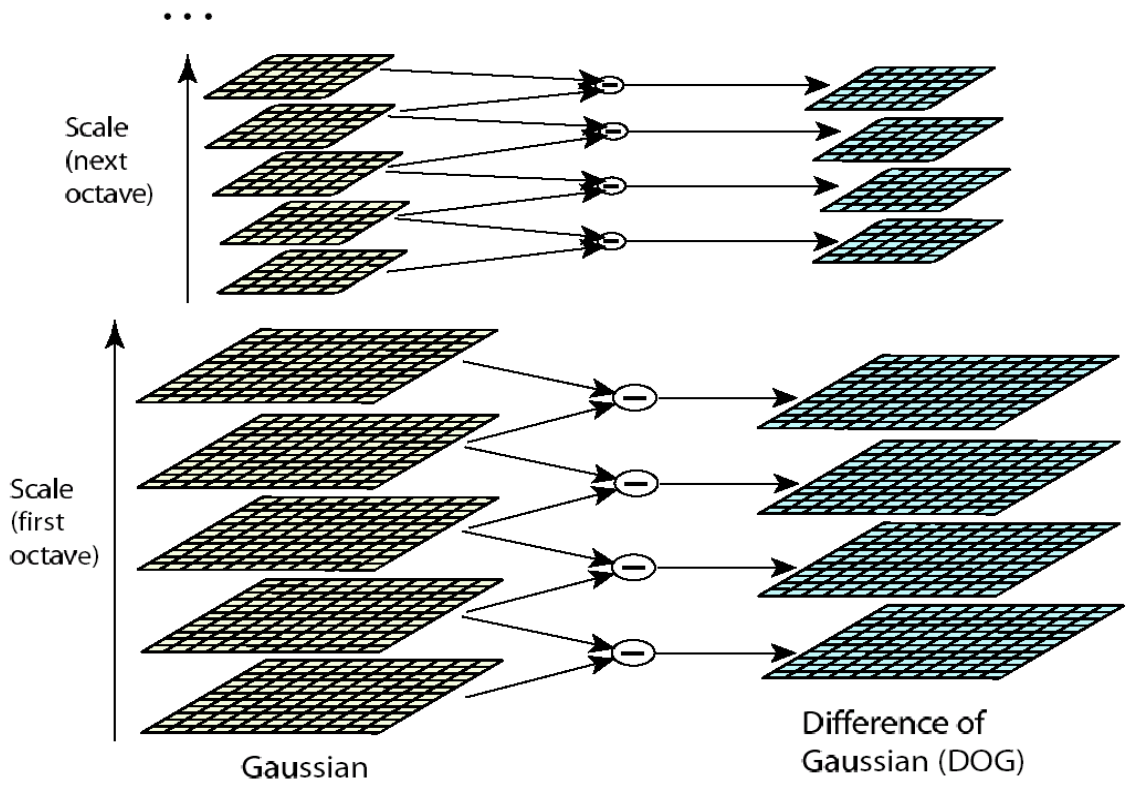
不同尺度的DOG对边缘和BLOB进行增强，类似高斯拉普拉斯变换



- 每层图像的底层是由前一层图像的隔点采样生成。这样可以保持尺度变换的连续性。

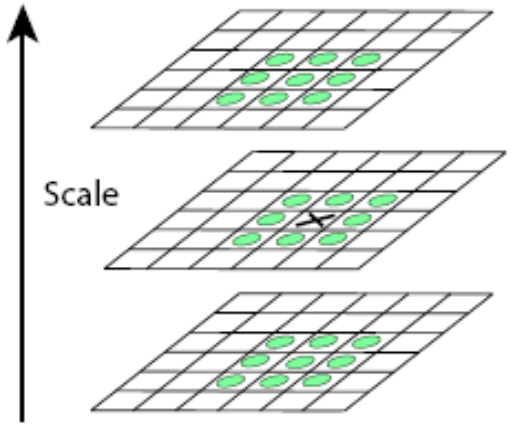
不同层之间的高斯变换的差形成差分金字塔（DOG）





DoG局部极值检测

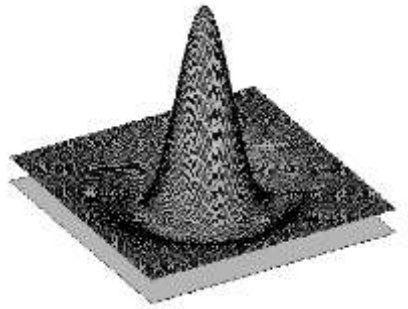
关键点是由DOG空间的局部极值点组成的。为了寻找DoG函数的极值点，每一个像素点要和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小。



中间的检测点和它同尺度的8个相邻点和上下相邻尺度对应的9×2个点共26个点比较，以确保在尺度空间和二维图像空间都检测到极值点。

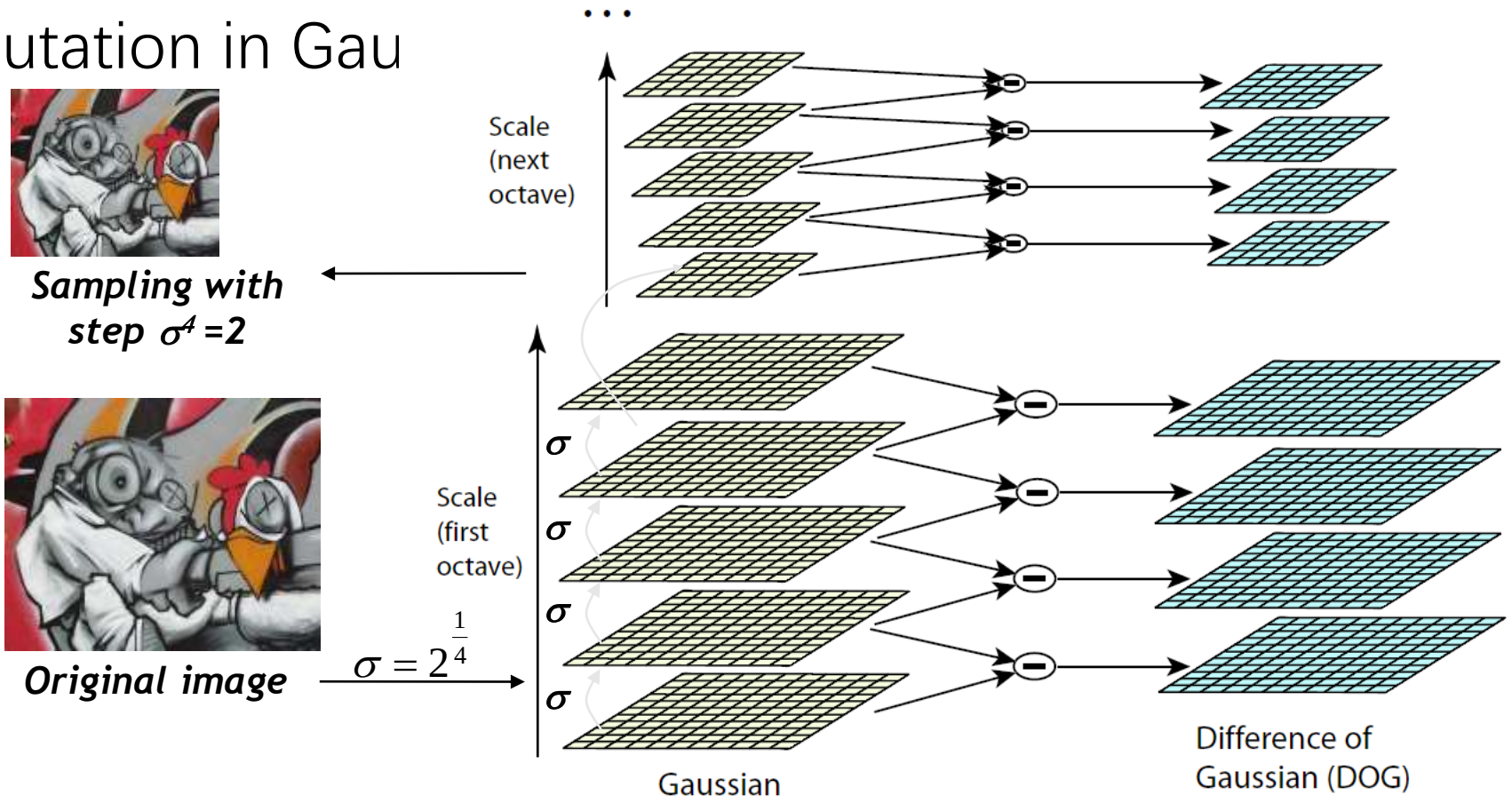
高斯差 Difference-of-Gaussian (DoG)

- Difference of Gaussians as approximation of t
  Laplacian-of-Gaussian

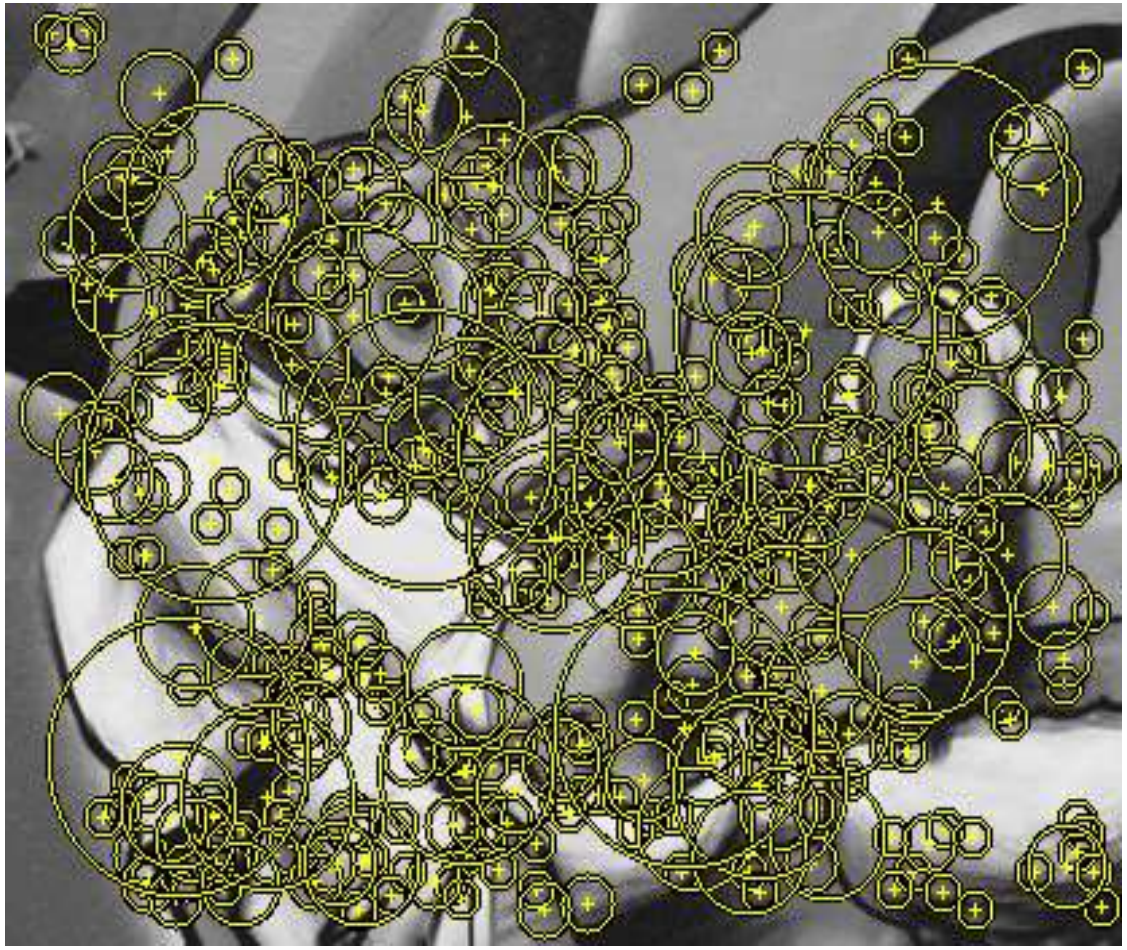


DoG – Efficient Computation

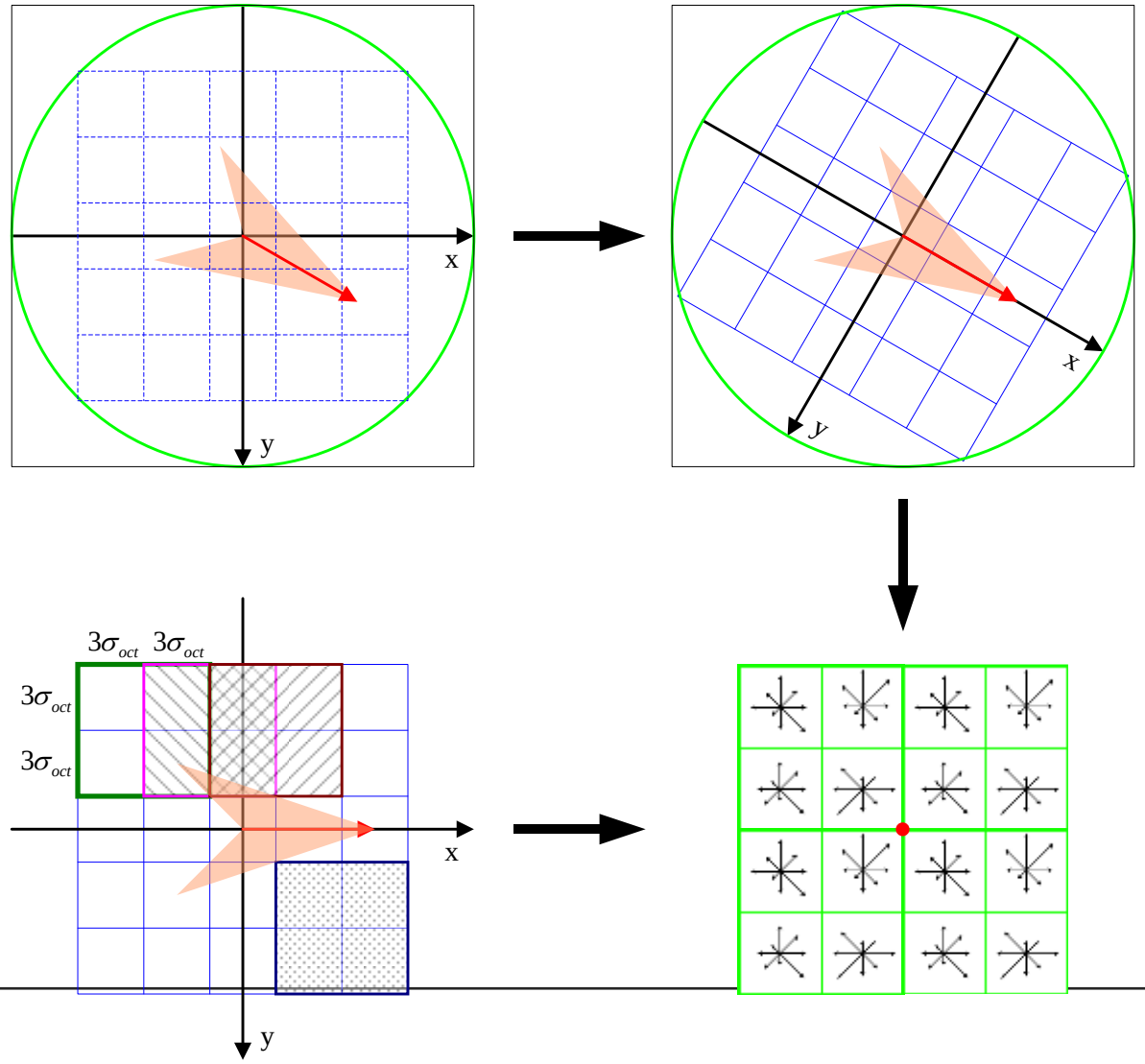
- Computation in Gau



Results: Laplacian-of-Gaussian



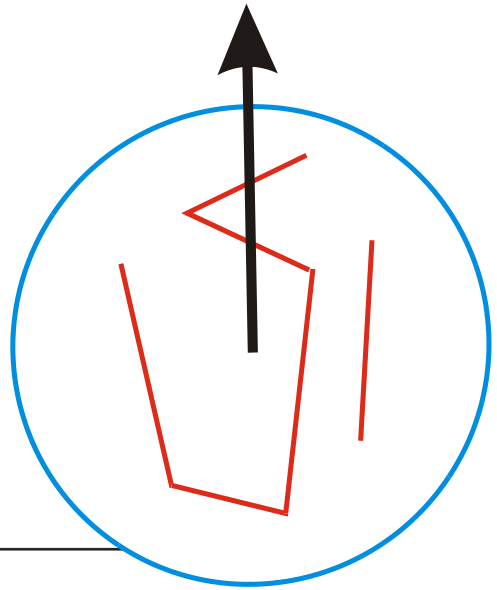
关键点特征描述

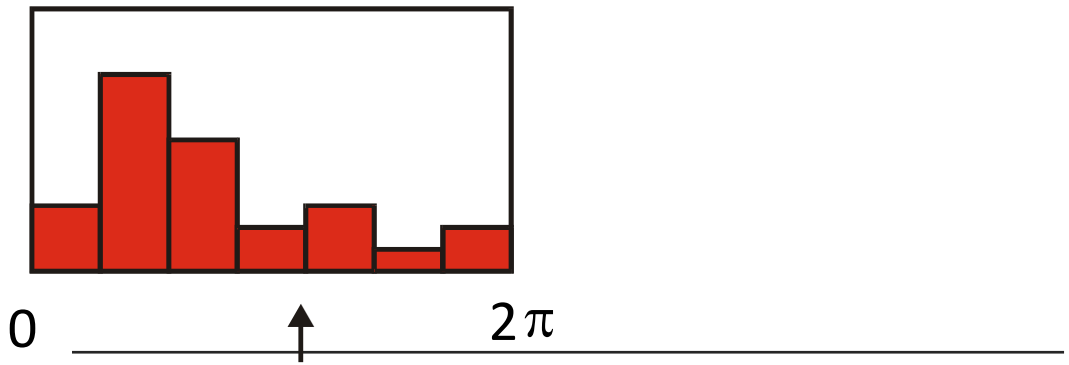


Lowe实验结果表明：描述子采用4×4×8＝128维向量表征，综合效果最优（不变性与独特性）。

Orientation Normalization
（关键点方向正则化）

- Compute orientation histogram 计算方向的直方图分布
- Select dominant orientation 选择主方向
- Normalize: rotate to fixed orientation 旋转到特定的正则化方向



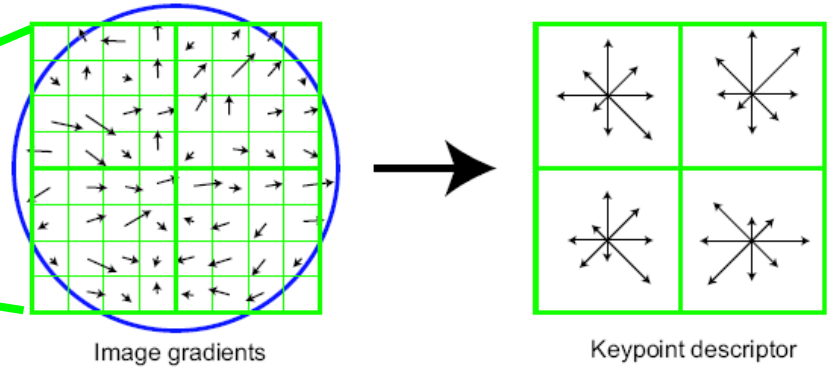


Local Descriptors（局部特征表达向量）

- The ideal descriptor should be
  - Repeatable
  - Distinctive
  - Compact
  - Efficient

- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color still relatively seldomly used
    (more suitable for homogenous regions)

Local Descriptors: SIFT Descriptor



Histogram of oriented gradients

- **Captures important texture information**
- **Robust to small translations / affine deformations**

[Lowe, ICCV 1999]

numpy二维图像数据处理

- 按numpy数据格式读入图像数据

```
1  from skimage import io
2
3  pic = io.imread("pic/children.jpg")
4  plt.imshow(pic)
5  print(pic.shape)
```

```
(322, 660, 3)
```


```
1  gry = np.mean(pic, axis = 2)
2  print(gry.shape)
3  gry[100:200, 20:30] = 256
4  plt.imshow(gry,cmap='gray')
```

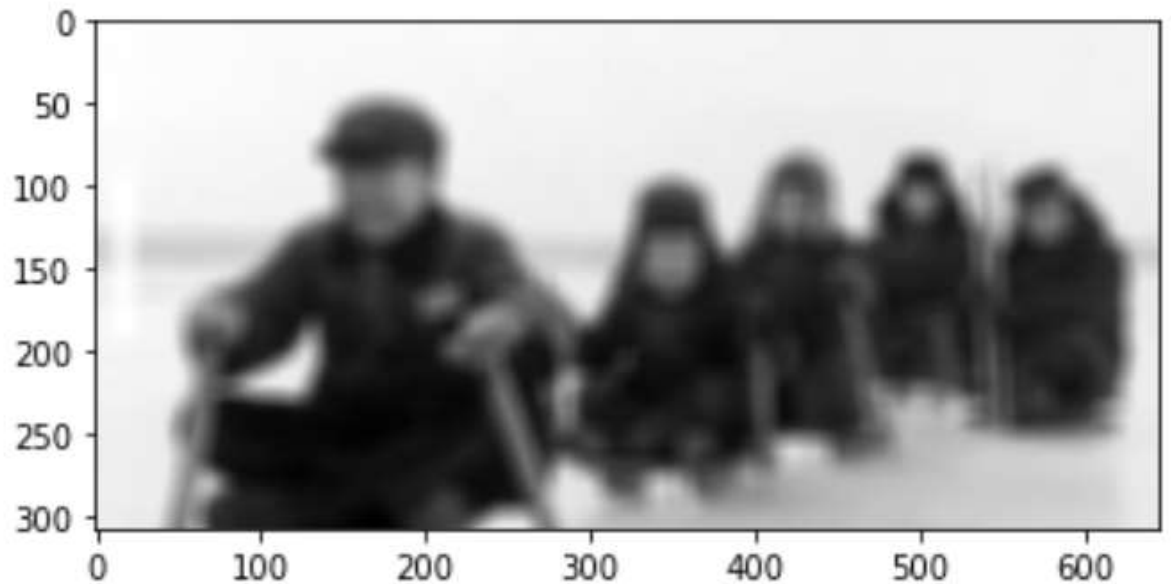
```
(322, 660)
```

```
1  print(gry[:, 130])  # 一行
```

```
[247. 247. 247. 247. 246. 246. 246. 246. 247. 247. 248. 248.
 248. 248. 246. 246. 246. 246. 246. 246. 246. 246. 247. 247.
 246. 246. 246. 246. 246. 246. 246. 246. 246. 246. 246. 246.
 246. 246. 246. 246. 245. 245. 243. 243. 243. 244. 244. 244.
 246. 246. 246. 247. 247. 247. 246. 246. 246. 244. 243. 242.
 243. 242. 242. 242. 242. 242. 241. 241. 241. 241. 241. 241.
 241. 241. 241. 241. 241. 241. 241. 241. 241. 241. 241. 241.
 242. 241. 241. 241. 240. 240. 240. 240. 239. 239. 239. 239.
 240. 240. 240. 240. 240. 239. 239. 239. 239. 239. 239. 240.
 239. 235. 242. 228. 235. 204. 137. 120. 120.  84.  86.  87.
```

```python
size = 15    # 滑窗大小 15*15
m, n = gry.shape
mm, nn = m - size + 1, n - size + 1

patch_means = np.empty((mm, nn))

# 滑窗
for i in range(mm):
    for j in range(nn):
        patch_means[i, j] = gry[i: i+size, j: j+size].mean() # 平均

plt.imshow(patch_means,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x20d4d7299d0>
```



numpy.lib.stride_tricks.sliding_window_view

lib.stride_tricks.sliding_window_view(*x, window_shape, axis=None, *, subok=False, writeable=False*) [source]

Create a sliding window view into the array with the given window shape.

Also known as rolling or moving window, the window slides across all dimensions of the array and extracts subsets of the array at all window positions.

```
1  x = np.arange(6)
2  print(x.shape)
3
4  v = np.lib.stride_tricks.sliding_window_view(x, 3)
5  print(v.shape)
6  v
```

滑窗视图

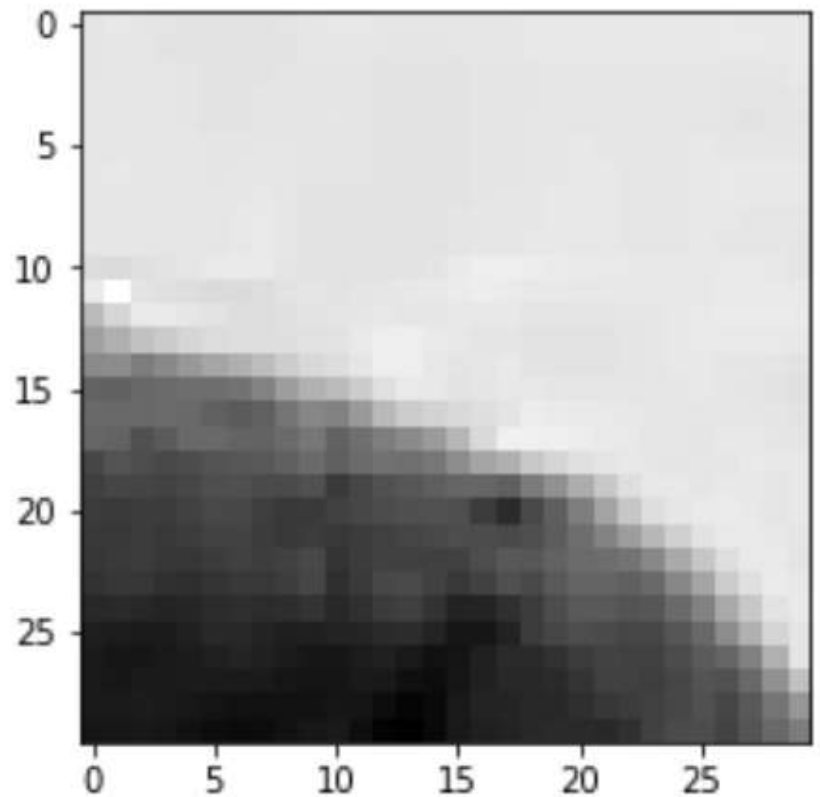
```
(6,)
(4, 3)

array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```

```
1 shape =(30,30)
2 win_gry = np.lib.stride_tricks.sliding_window_view(gry, shape)
3 win_gry.shape
```

(293, 631, 30, 30)

```
1 plt.imshow(win_gry[110,230], cmap='gray')
```

<matplotlib.image.AxesImage at 0x20d4ed0fa60>

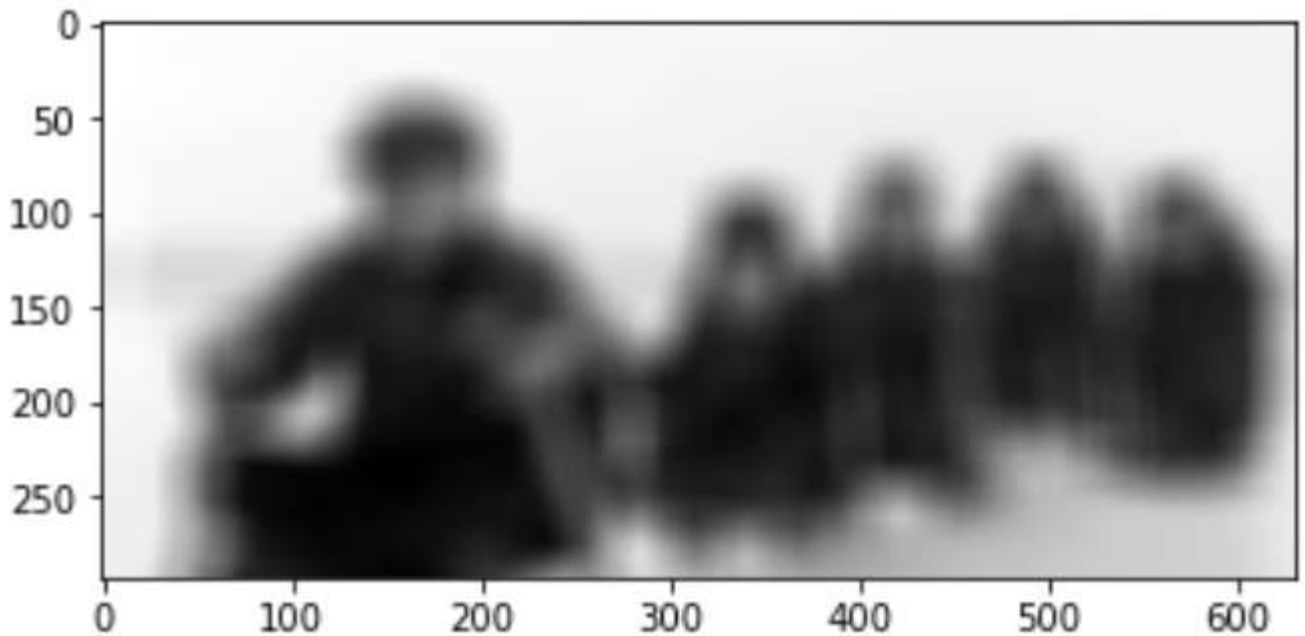


```
win_gry_mean = win_gry.mean((2,3))
print(win_gry_mean.shape)

win_gry_max = win_gry.max((2,3))
```

(293, 631)

```
plt.imshow(win_gry_mean,cmap='gray')
```

<matplotlib.image.AxesImage at 0x20d4ed630a0>

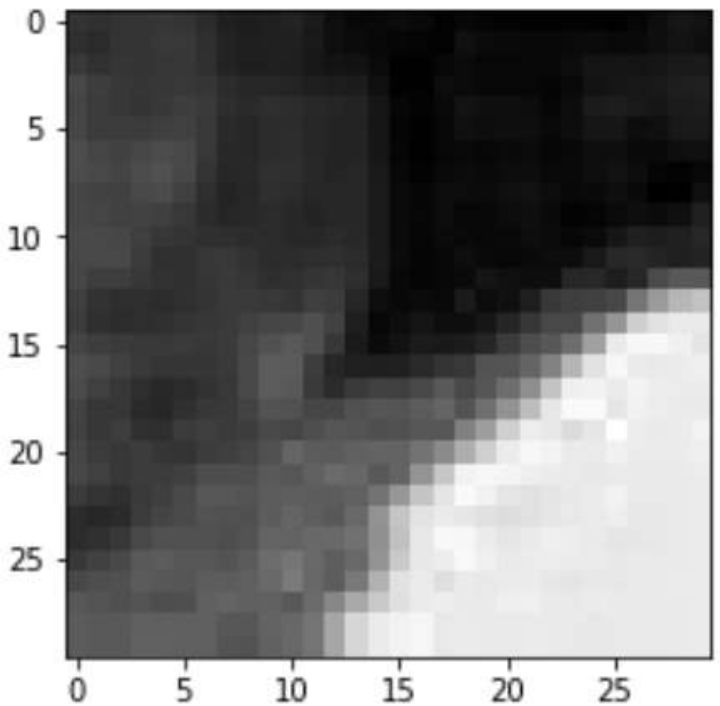


```
contra = win_gry_max - win_gry_mean
print(contra.shape)
pos = np.where(contra > 150)  # 差异大
print(pos)
```

```
(293, 631)
(array([ 51,  51,  51, ..., 292, 292, 292], dtype=int64), array([162, 163, 164, ..., 222, 223, 224], dtype=int64))
```

```
plt.imshow(win_gry[292,222], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x20d4f3c66d0>
```



numpy.lib.stride_tricks.as_strided

`lib.stride_tricks.as_strided(x, shape=None, strides=None, subok=False, writeable=True)`

Create a view into the array with the given shape and strides. [source]

> **Warning**
>
> This function has to be used with extreme care, see notes.

Parameters: **x** : *ndarray*

Array to create a new.

shape : *sequence of int, optional*

The shape of the new array. Defaults to `x.shape`.

strides : *sequence of int, optional*

The strides of the new array. Defaults to `x.strides`.

subok : *bool, optional*

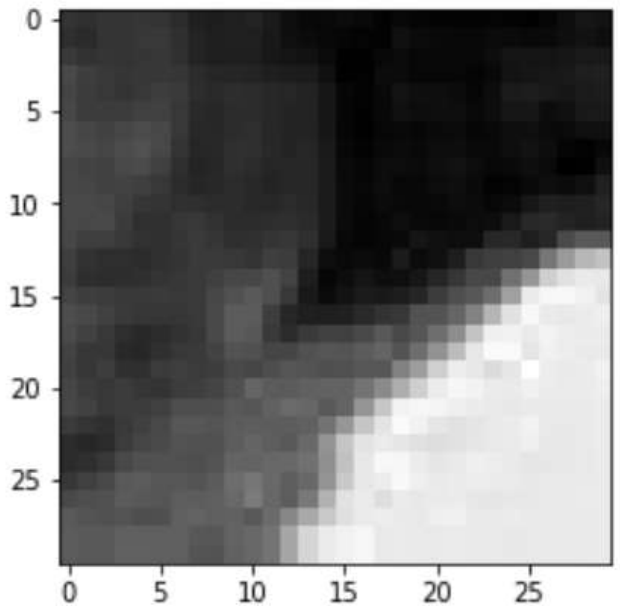
:
```
step = 30
shape = (gry.shape[0] - step + 1, gry.shape[1] - step + 1, step, step)
strides = 2 * gry.strides

print(shape)
print(strides)
```

```
(293, 631, 30, 30)
(5280, 8, 5280, 8)
```

:
```
slide_win2 = np.lib.stride_tricks.as_strided(gry, shape=shape, strides=strides)
print(slide_win2.shape)
plt.imshow(slide_win2[292,222], cmap='gray')
```

```
(293, 631, 30, 30)
```

: <matplotlib.image.AxesImage at 0x20d50830b50>



HOG特征：

方向梯度直方图（Histogram of Oriented Gradient, HOG）通过计算和统计图像局部区域的梯度方向直方图来构成特征。具备尺度不变性的特点。但不具备旋转不变性。常用于行人、车辆这类物体的检测以及人脸识别等

提取流程：

1）灰度化（将图像看做一个x,y,z（灰度）的三通道图像）；

2）采用Gamma校正法对输入图像进行颜色空间的标准化（归一化）；（类似灰度直方图变换的一个幂函数映射）

3）计算图像每个像素的梯度（包括大小和方向）；主要是为了捕获轮廓信息，同时进一步弱化光照的干扰。

4）将图像划分成小cells（例如6*6像素/cell）；

5）统计每个cell的梯度直方图（不同梯度的个数），即可形成每个cell的descriptor；

6）将每几个cell组成一个block（例如3*3个cell/block），在block级别再做一次对比度均衡。一个block内所有cell的特征descriptor串联起来便得到该block的HOG特征

7）将图像image内的所有block的HOG特征descriptor串联起来就可以得到该image（你要检测的目标）的HOG特征descriptor及可供分类使用的特征向量

1 读取图像数据

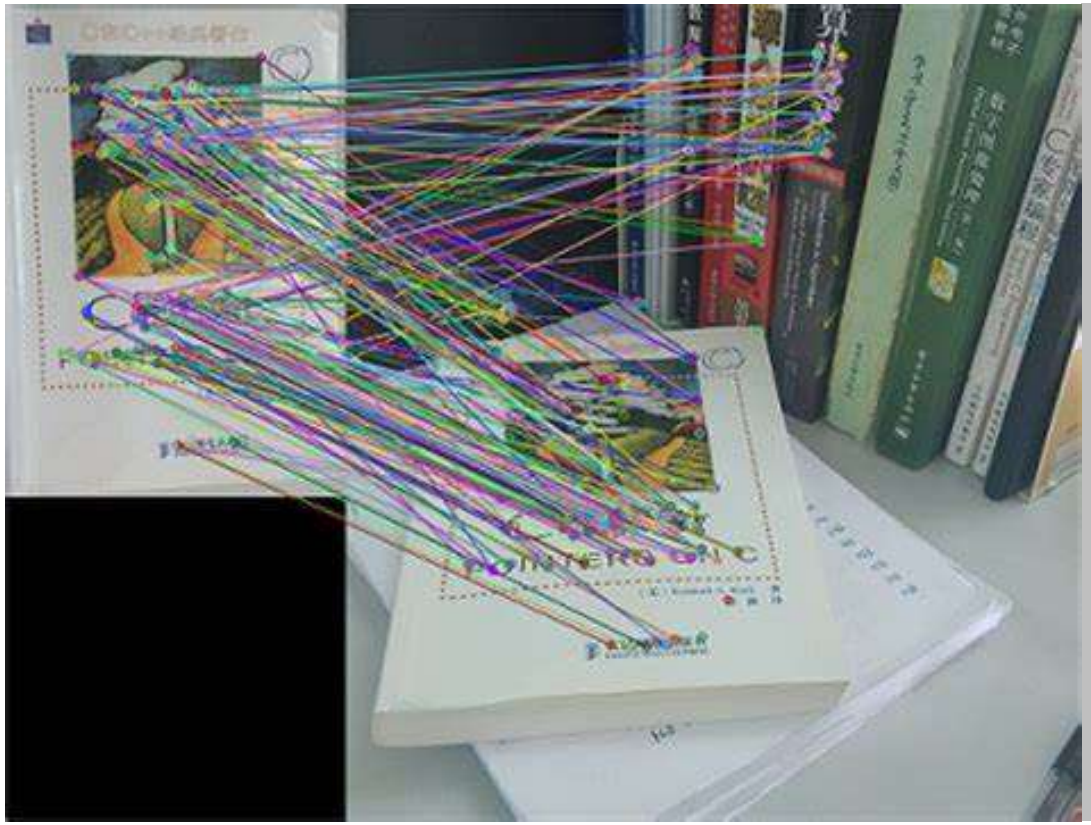
```python
import cv2
import numpy as np

SZ = 20
CLASS_N = 10

def split2d(img, cell_size, flatten=True):
    h, w = img.shape[:2]    # 1000*2000
    sx, sy = cell_size      # 20
    cells = [np.hsplit(row, w//sx) for row in np.vsplit(img, h//sy)] # 分割成20*20矩阵的list
    cells = np.array(cells)
    if flatten:
        cells = cells.reshape(-1, sy, sx)  # reshape(X, sy, sx), X自动算出
    return cells

def load_digits(fn):
    digits_img = cv2.imread(fn, 0)
    digits = split2d(digits_img, (SZ, SZ))
    labels = np.repeat(np.arange(CLASS_N), len(digits)/CLASS_N) # 做 Y repeat arange xi
    return digits, labels

digits, labels = load_digits('images/digits.png')
```

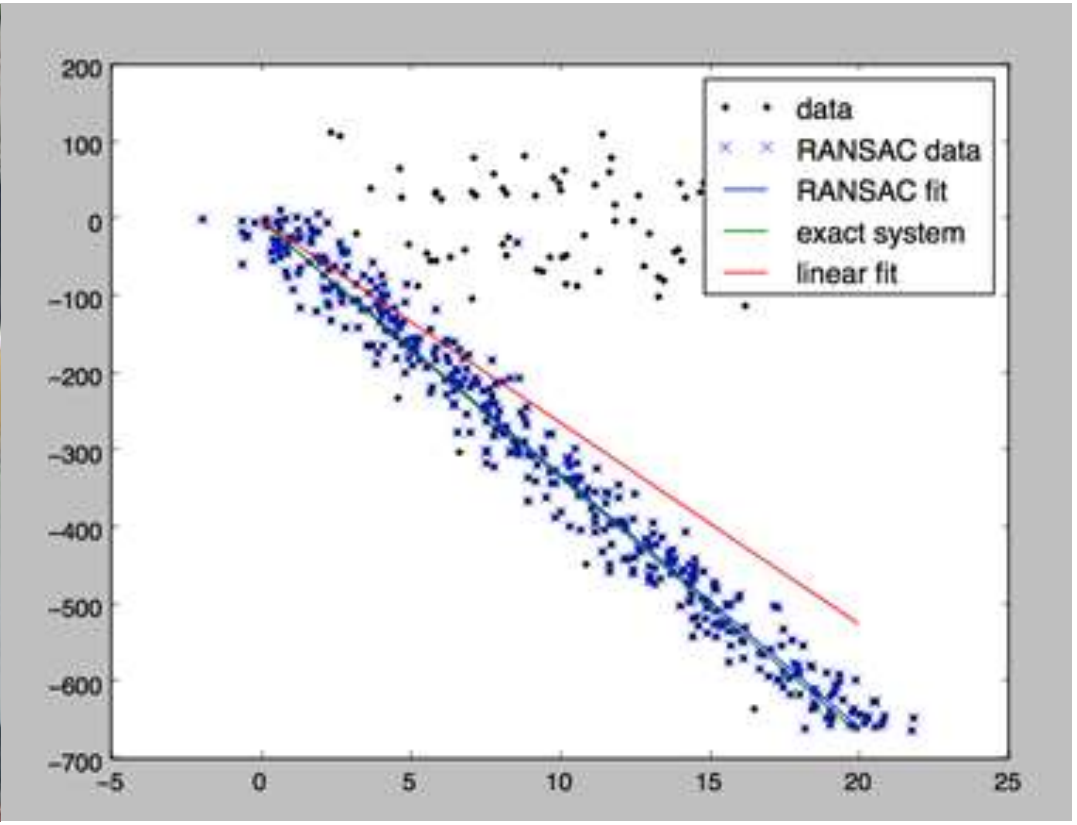
提取图像特征描述符

```
# 预设HOG特征参数
winSize = (20,20)
blockSize = (8,8)      #
blockStride = (4,4)
cellSize = (8,8)
nbins = 9
derivAperture = 1
winSigma = -1.
histogramNormType = 0
L2HysThreshold = 0.2
gammaCorrection = 1
nlevels = 64
signedGradient = True

hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,\
            cellSize,nbins,derivAperture,winSigma,\
            histogramNormType,L2HysThreshold,gammaCorrection,\
            nlevels,  signedGradient)

hog_descriptors = []
for img in digits:
    hog_descriptors.append(hog.compute(img))

print(hog_descriptors[0].shape)
hog_descriptors = np.squeeze(hog_descriptors) #squeeze 函数：从数组的形状中删除单维度条目，即把shape中为1的维度去掉

print(hog_descriptors.shape)
```

```
(144, 1)
(5000, 144)
```

特征匹配与RANSAC算法





仿射变换

- 仿射变换（Affine Transformation 或Affine Map）是一种二维坐标（x，y）到二维坐标（u，v）的线性变换，其数学表达式形式如下：

$$\begin{cases} u = a_1 x + b_1 y + c_1 \\ v = a_2 x + b_2 y + c_2 \end{cases}$$

- 对应的齐次坐标矩阵表示形式为：

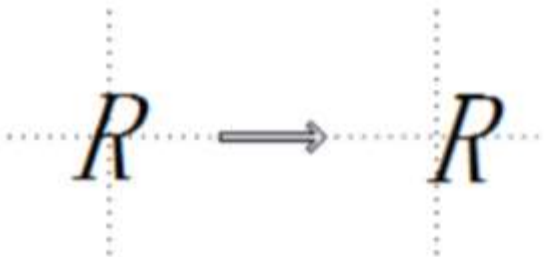
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 仿射变换保持了二维图形的“平直性”（直线经仿射变换后依然为直线）和“平行性”（直线之间的相对位置关系保持不变，平行线经仿射变换后依然为平行线，且直线上点的位置顺序不会发生变化）。非共线的三对对应点确定一个唯一的仿射变换。

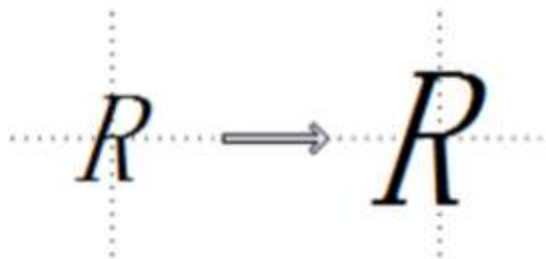
原子变换

- 仿射变换通过一系列原子变换复合实现，具体包括：平移（Translation）、缩放（Scale）、旋转（Rotation）、翻转（Flip）和错切（Shear）。

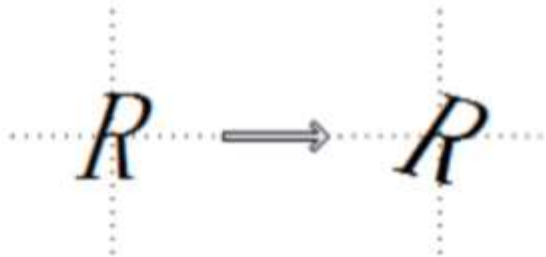
a. 平移



b. 缩放



c. 旋转



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

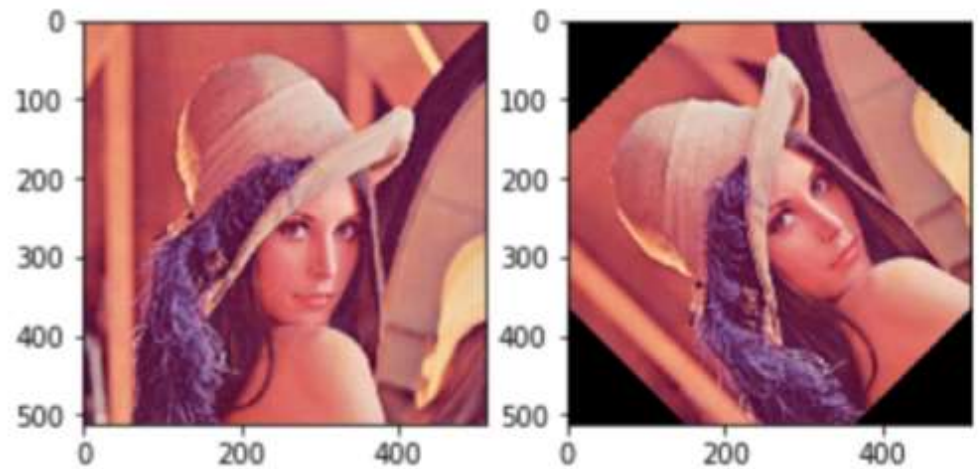
https://blog.csdn.net/liuweiyuxiang/article/details/82799999

```python
# refer https://blog.csdn.net/liuweiyuxiang/article/details/82799999
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```python
def get_img():
    img = cv2.imread('lena.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img
```

```python
img = get_img()
rows,cols = img.shape[:2]
#第一个参数旋转中心，第二个参数旋转角度，第三个参数：缩放比例
M = cv2.getRotationMatrix2D((cols/2,rows/2),45,1)
#第三个参数：变换后的图像大小
res = cv2.warpAffine(img,M,(rows,cols))
```

```python
# show
plt.subplot(121)
plt.imshow(img)
plt.subplot(122)
plt.imshow(res)
plt.show()
print('M: {}'.format(M))
```

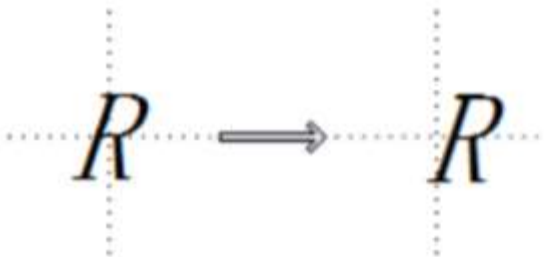



```
M: [[   0.70710678    0.70710678 -106.03867197]
 [  -0.70710678    0.70710678  256.        ]]
```

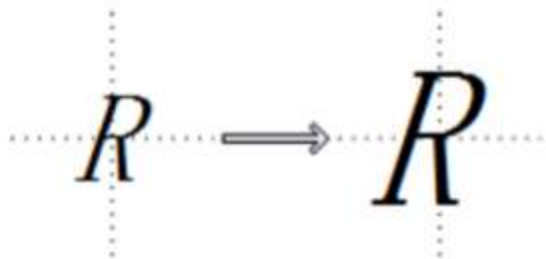
原子变换

- 仿射变换通过一系列原子变换复合实现，具体包括：平移（Translation）、缩放（Scale）、旋转（Rotation）、翻转（Flip）和错切（Shear）。

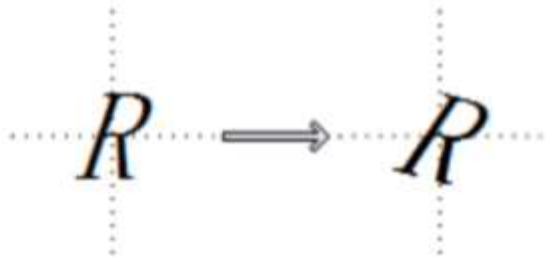
a. 平移



b. 缩放



c. 旋转



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

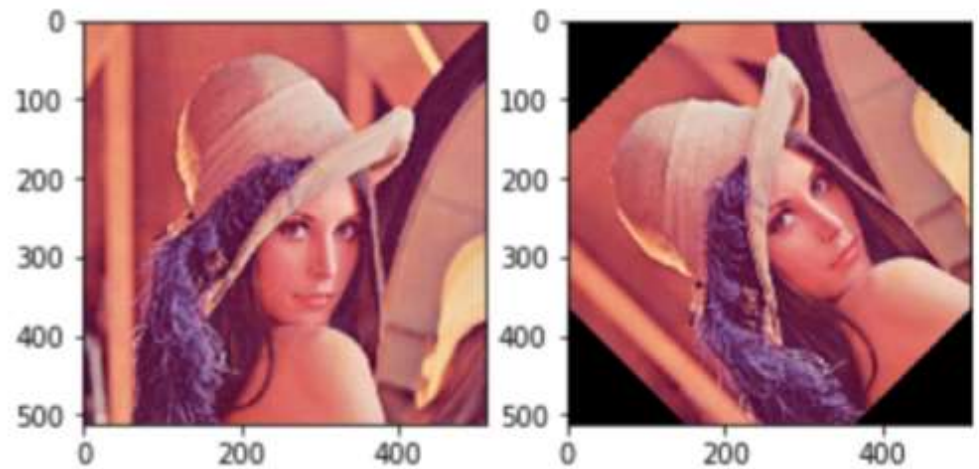
https://blog.csdn.net/liuweiyuxiang/article/details/82799999

```python
# refer https://blog.csdn.net/liuweiyuxiang/article/details/82799999
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```python
def get_img():
    img = cv2.imread('lena.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img
```

```python
img = get_img()
rows,cols = img.shape[:2]
#第一个参数旋转中心，第二个参数旋转角度，第三个参数：缩放比例
M = cv2.getRotationMatrix2D((cols/2,rows/2),45,1)
#第三个参数：变换后的图像大小
res = cv2.warpAffine(img,M,(rows,cols))
```

```python
# show
plt.subplot(121)
plt.imshow(img)
plt.subplot(122)
plt.imshow(res)
plt.show()
print('M: {}'.format(M))
```




```
M: [[   0.70710678    0.70710678 -106.03867197]
 [  -0.70710678    0.70710678  256.        ]]
```