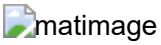
3.1 图像即矩阵(Image is a matrix)

数字图像分析是现代数值计算中的热门问题之一。常用的计算软件均提供了功能强大的图像处理工具箱。其中大多是集成了著名的图像处理开源软件：OpenCV.

常将数字图像理解成一个二元函数/矩阵:

- 采样：将连续的图像变换成离散点的操作。如横向M行和纵向N列的采样得到总像素为M × N的数字图像。其中，像素点(i,j)的四个相邻点分别为$(i-1,j)$、$(i+1,j)$、$(i,j-1)$、$(i,j+1)$;
- 量化：将像素点的灰度值分成2^8个等级，取值$0 \sim 255$
- 像素点P(i,j)与Q(m,n)之间的距离

$$dist(P,Q) = \sqrt{(i-m)^2 + (j-n)^2}$$



In [2]:

```
pkg load image
```

In [3]:

```
load testlabel.mat
```

In [6]:

```
groundTruthLabelingSession
```

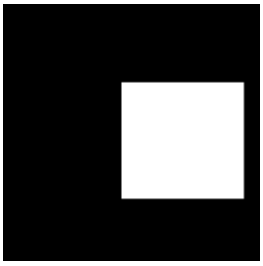
```
groundTruthLabelingSession =

  scalar structure containing the fields:

    MCOS =

      3707764736
               2
               1
               1
               1
              11

    driving.internal.videoLabeler.tool.Session =

error: octave_base_value::print (): wrong type argument '<unknown type>'
```

常用图像类型

- 二值图像

In [8]:

```
A = zeros(128); A(40:97, 60:120) = 1;imshow(A);
```



- 灰度图像

In [10]:

```
img = imread('figs/lenna.bmp'); imshow(img);
```

In [12]:

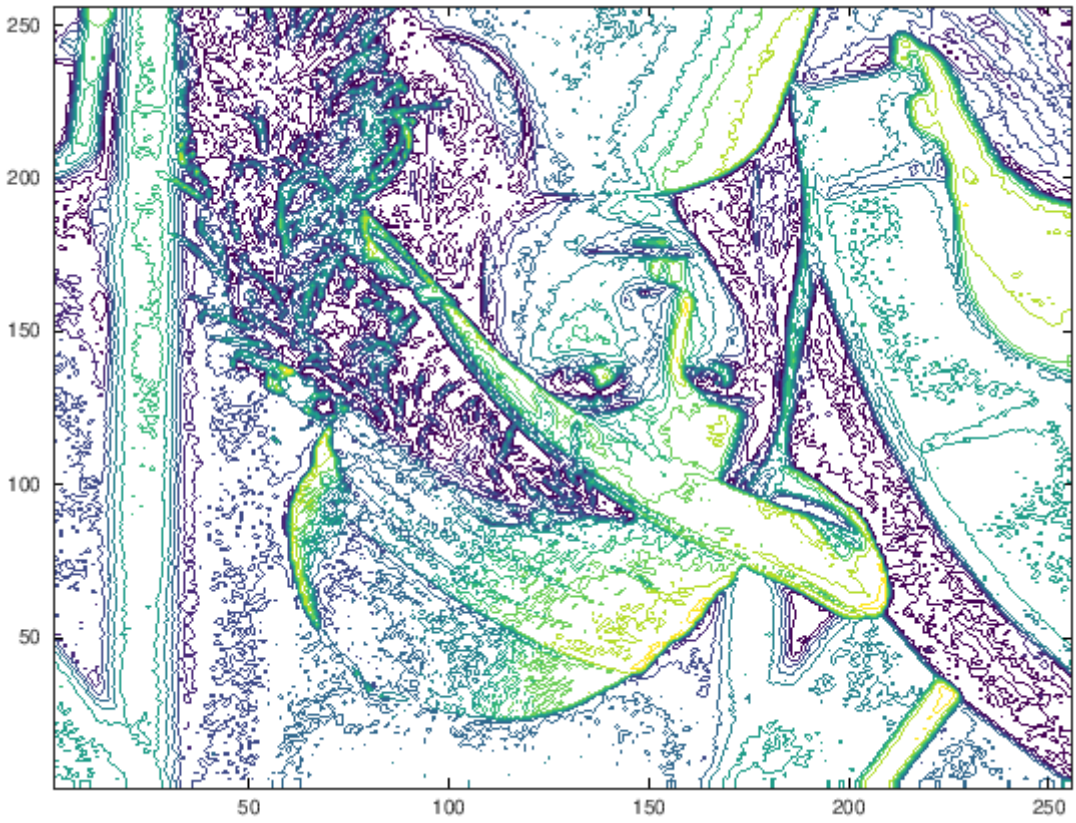
```
img(83:90,20:29)
```

```
ans =

  145   154   148   147   147   146   149   150   136   126
  145   142   149   148   146   148   145   146   134   123
  152   152   152   152   151   151   151   147   137   128
  148   149   151   144   152   147   146   145   135   130
  146   144   151   146   149   146   151   145   140   130
  143   141   152   149   147   152   152   142   146   128
  151   149   154   153   150   152   146   143   140   127
  149   155   146   151   153   152   147   148   135   133
```

- 和处理矩阵一样对图像做任意的处理:

In [13]:

```
contour(flipud(img));  % imshow(img);
```



- 彩色图像可以相应地理解为RGB三个通道组成的三个矩阵

In [8]:

```
img2 = imread('figs/lenna_aged.jpg'); imshow([img2, img2(:,:,[1 3 2]), img2(:,:,[1 1 1]) ]);
```

python中的OpenCV展示

pythoncv2

cifar、ImageNet等数据集的.m读写接口

In [ ]:

```
```

In [ ]:

```
```

Octave中图像基本运算

数字图像事实上是二元离散函数，可定义加减乘除.

- 加法运算

In [18]:

```
I = imread('figs/barbara.bmp');
J = imnoise(I,'gaussian', 0.2); % 加Gauss噪声  salt & pepper
```

In [19]:

```
imshow([I J])
```

In [19]:

```
imnoise     % please try different type of noise
```

```
error: Invalid call to imnoise.  Correct usage is:

 -- Function File: imnoise (A, TYPE)
 -- Function File: imnoise (..., OPTIONS)
 -- Function File: imnoise (A, "gaussian", MEAN, VARIANCE)
 -- Function File: imnoise (A, "poisson")
 -- Function File: imnoise (A, "salt & pepper", DENSITY)
 -- Function File: imnoise (A, "speckle", VARIANCE)

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

- 乘法运算

In [21]:

```
J = immultiply(I, 2);   imshow([I J])
```

In [24]:

```
[ I(1:8,1:8) zeros(8,1) J(124:131,124:131)]
```

```
ans =

 Columns 1 through 16:

  181   201   202   195   189   194   197   206     0   255   255   255   255   255   255   255
  171   198   201   192   190   193   197   207     0   255   255   255   255   255   255   255
  175   195   193   183   187   192   197   210     0   255   255   255   255   255   255   255
  184   201   192   180   188   195   200   213     0   255   255   255   255   255   255   255
  197   206   194   185   188   194   201   211     0   255   255   255   255   255   255   255
  200   202   190   188   194   197   204   212     0   255   255   255   255   255   255   255
  192   196   182   187   196   197   208   212     0   255   255   255   255   255   255   255
  196   189   181   189   199   203   210   211     0   255   255   255   255   255   255   255

 Column 17:

  255
  255
  255
  255
  255
  255
  255
  255
```

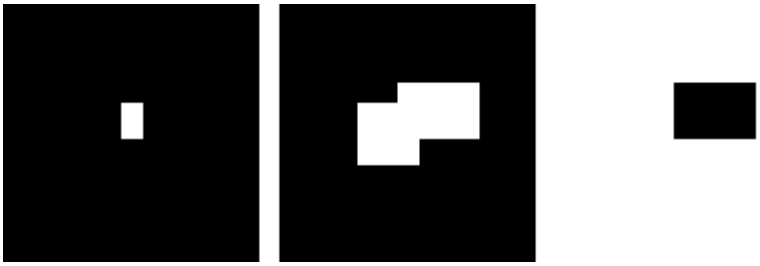
- 二值图像的逻辑运算(and、or、not等)

In [29]:

```
A = zeros(128); A(40:67, 60:100) = 1;
B = zeros(128); B(50:80, 40:70) = 1;
imshow([A, ones(128,10), B])
```

In [30]:

```
C = and(A,B); D = or(A,B); E = not(A);
imshow([C, ones(128,10), D, ones(128,10), E])
```



In [ ]:

```
```

图像变换

In [1]:

```
pkg load image;
```

图像的几何运算-仿射变换

变换

$$g(x,y) = f(u(x,y), v(x,y))$$

中, $u = u(x,y), v = v(x,y)$唯一确定了空间变换

- 平移(imtransform)

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

In [ ]:

```
```

- 镜像

$$\begin{pmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In [ ]:

```
```

- 旋转(imrotate)

$$\begin{pmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In [ ]:

```
```

- 缩放(imresize)

$$\begin{pmatrix} a & 0 & \delta x \\ 0 & a & \delta y \\ 0 & 0 & 1 \end{pmatrix}$$

In [6]:

```
imresize
```

```
error: Invalid call to imresize.  Correct usage is:

 -- Function File: imresize (IM, SCALE)
 -- Function File: imresize (IM, [M N])
 -- Function File: imresize (..., METHOD)

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

In [ ]:

```

```

图像尺寸变换更多细节和例子 (https://blog.csdn.net/u013165921/article/details/79054788?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.nonecase)

练一练： 请将上述例子自己动手做一遍，并记录你的结果和心得

In [ ]:

```

```

In [ ]:

```

```

图像特征提取

图像的特征，让我们分三个类型来展示： 全局特征、局部特征和其他特征（机器学习中各类算法）

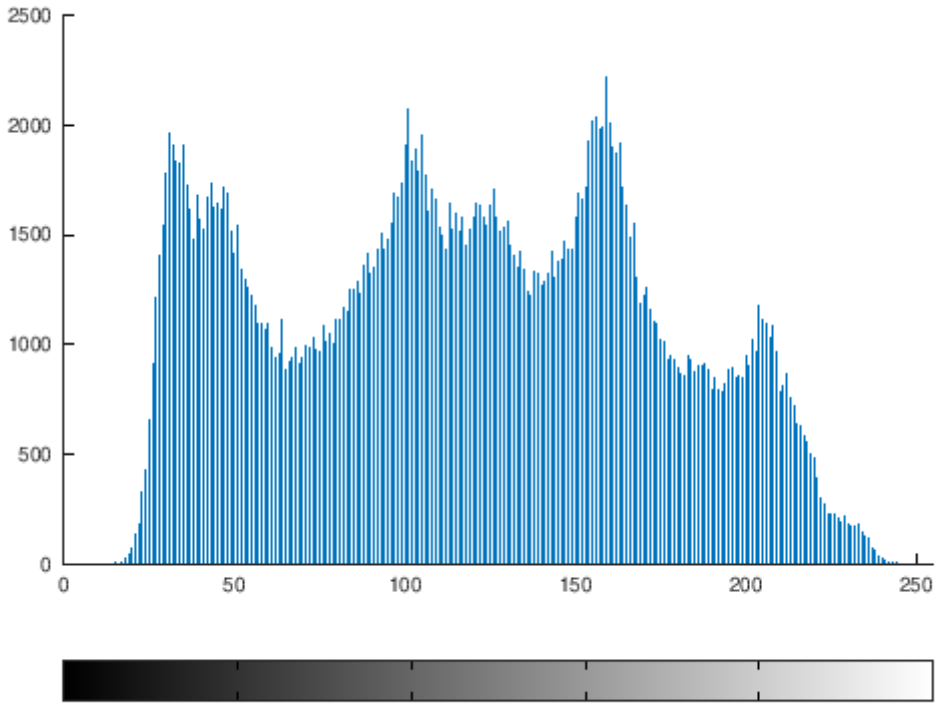
全局特征

- Intensity Histogram

这里 (https://zhuanlan.zhihu.com/p/143244712)是一个关于数字图像Intensity Transformations and Histogram的实验报告样本，

In [20]:

```
imhist(I)
```



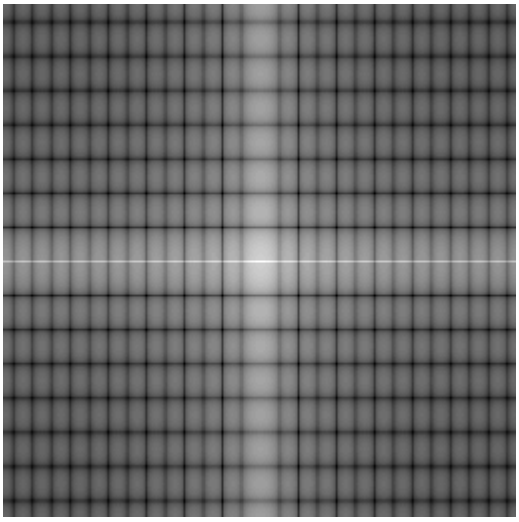
- Fourier变换/分析

In [55]:

```
I = imread('figs/lenna.bmp');  imshow(I)
```

In [66]:

```
J = fftshift(fft2(I));  imshow(log(abs(J)),[8,10]);
```

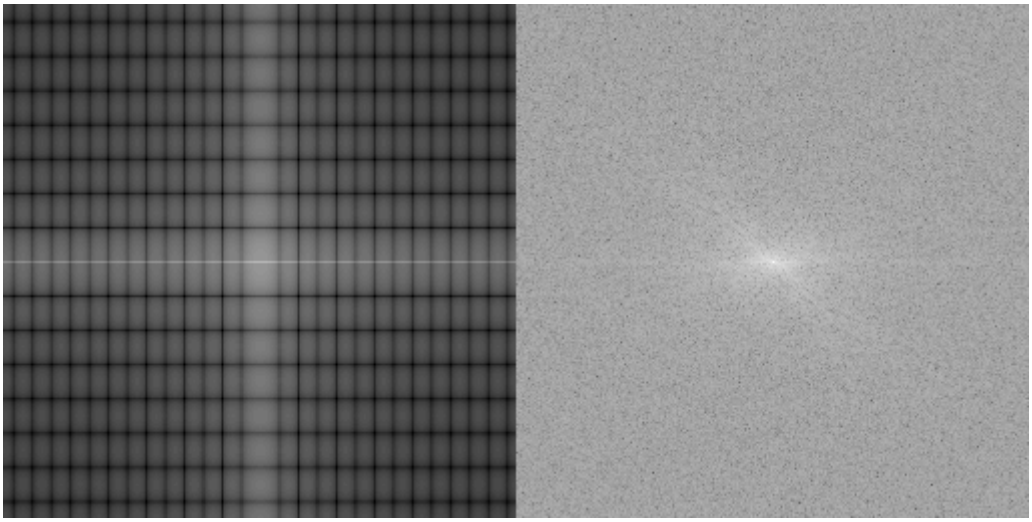


In [57]:

```
I_noise = imnoise(I,'gaussian',0,0.01);  imshow([I I_noise]);
```

In [68]:

```
J_noise = fftshift(fft2(I_noise)); imshow([log(abs(J)), log(abs(J_noise))],[8,10]);
```



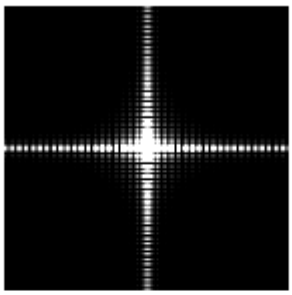
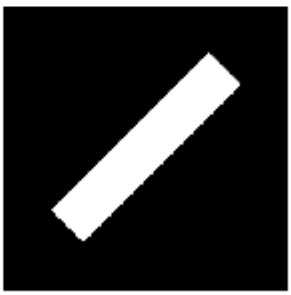
In [ ]:

```
```

- 旋转变换的Fourier分析

In [63]:

```
% the original image
I = zeros(256,256);
I(28:228,108:148) = 1;
subplot(2,2,1);imshow(I);
% the frequency of original image
J = fft2(I);
J1 = fftshift(abs(J));
subplot(2,2,2); imshow(J1,[5,50]);
% the rotated image
I = imrotate(I,315,'bilinear','crop');
subplot(2,2,3); imshow(I);
% the frequency of rotated image
J = fft2(I);
J1 = fftshift(abs(J));
subplot(2,2,4); imshow(J1,[5,50]);
```

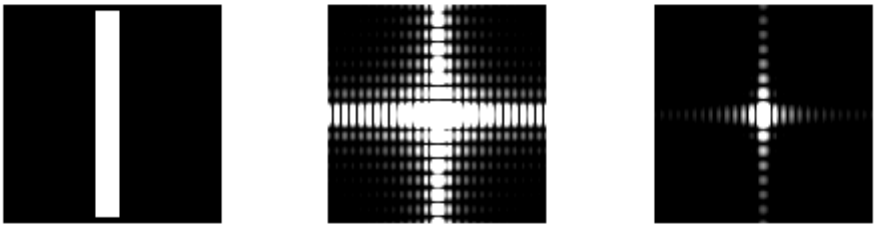

In [ ]:

```
```

- 尺度变换的Fourier性质

In [64]:

```
I = zeros(256,256); I(8:248,110:136) = 5;
subplot(1,3,1);imshow(I);
J = fft2(I); J1 = fftshift(abs(J));
subplot(1,3,2);imshow(J1,[5 30]); % frequency of original image
J = fft2(0.1*I);  J2 = fftshift(abs(J));
subplot(1,3,3);imshow(J2,[5 30]); % frequency of darked imaged
```



算法原理：离散Fourier变换(DFT)

N个数据点$x_l(l = 0, 1, \ldots, N-1)$的DFT是

$$y_m = \sum_{k=0}^{N-1} x_k \omega_N^{mk}, \quad m = 0, 1, \ldots, N-1.$$

易知其逆变换为

$$x_m = \frac{1}{N} \sum_{k=0}^{N-1} y_k \omega_N^{-mk}, \quad m = 0, 1, \ldots, N-1.$$

验证:当$N = 4$时，逆变换和正变换矩阵 $$ \frac{1}{4} \left[

$$\begin{matrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{matrix}$$

\right] \left[\begin{array}{cccc} 1 & 1 & 1 & 1\ 1 & \omega^1 & \omega^2 & \omega^3 \ 1 & \omega^2 & \omega^4 & \omega^6 \ 1 & \omega^3 & \omega^6 & \omega^9 \

\end{array}\right]

\left[

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

In [ ]:

```
```

局部特征

In [2]:

```
pkg load image
```

- 边缘检测 - 局部特征

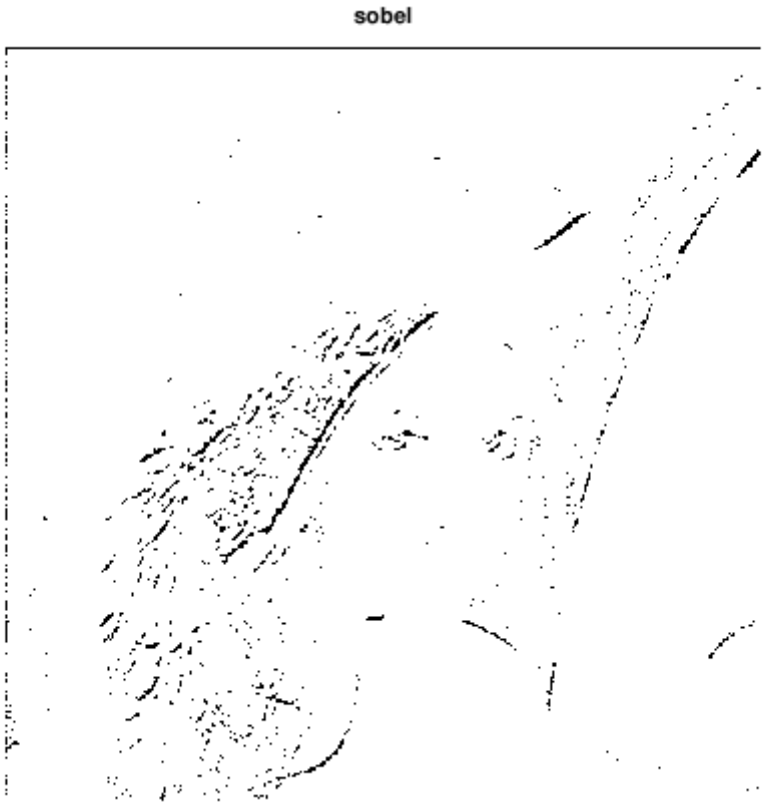
In [4]:

```
X = imread('figs/lena.jpg');
```

典型的边缘提取算子（一阶导数近似计算 / 卷积）

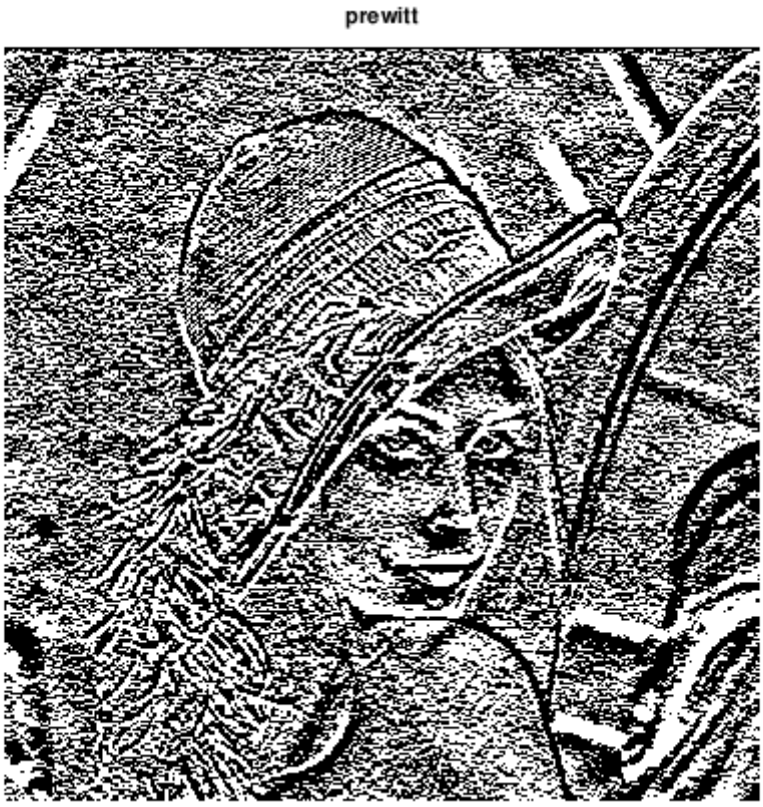
In [5]:

```
%h1 = fspecial('sobel');
h1 = [ 1  2  1 ;...
       0  0  0 ;...
       1 -2 -1 ];
X1 = filter2(h1, X); imshow(X1); title('sobel');
```



In [6]:

```
%h2 = fspecial('prewitt');
h2 = [ 1  1  1;...    % prewitt
       0  0  0 ;...
      -1 -1 -1 ];
X2 = filter2(h2, X);  imshow(X2); title('prewitt');
```



关于conv2、filter2、imfilter的更多区别，参考[这里 (https://www.ilovematlab.cn/thread-293710-1-1.html)](https://www.ilovematlab.cn/thread-293710-1-1.html)

- filter2、conv2将输入转换为double类型，输出也是double的，输入总是补零（zero padded), 不支持其他的边界补充选项，只能对二维图像（灰度图）进行空间滤波
- imfilter：不将输入转换为double，输出只与输入同类型，有灵活的边界补充选项,且可进行多维图像（RGB等）进行空间滤波

In [ ]:

```
```

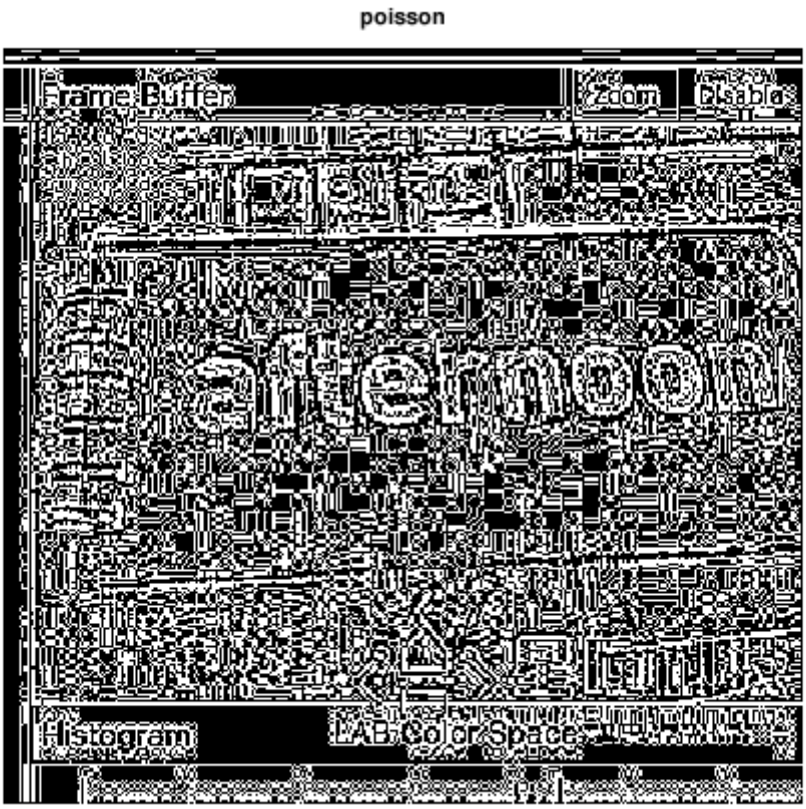
进一步地，我们换个不同类型的图像看看效果：

In [20]:

```
X = rgb2gray(imread('figs/character.jpg'));
```

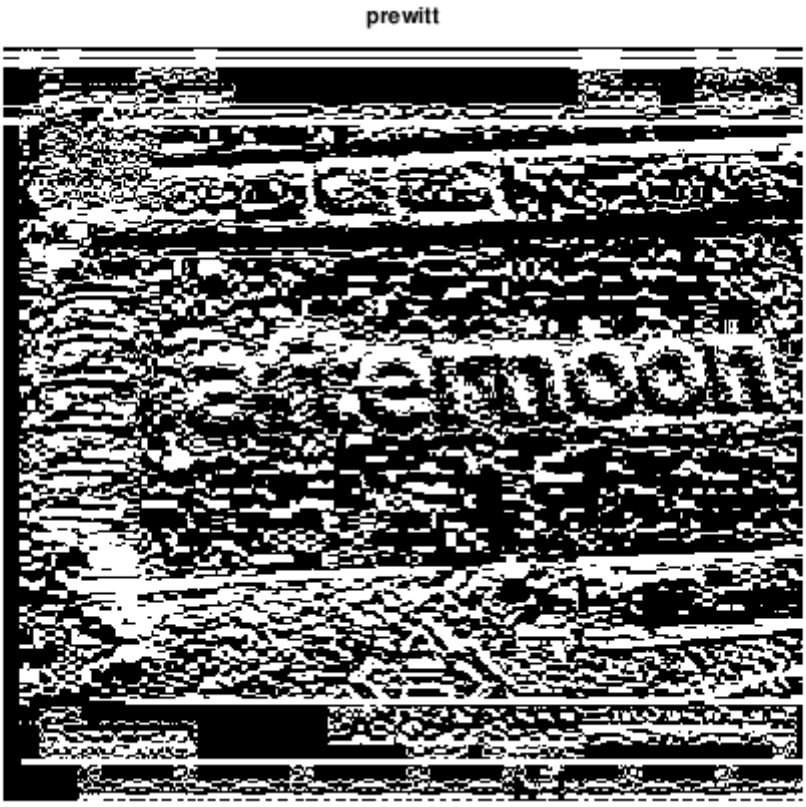
In [40]:

```
%h1 = fspecial('sobel');   % L a p l a c e算子的五点差分格式
h1 = [ 0  1  0 ;...
       1  -4  1 ;...
       0  1  0 ];
X1 = filter2(h1, X);imshow(X1); title('poisson');
```



In [38]:

```
%h2 = fspecial('prewitt');
h2 = [ 1  1  1;...    % prewitt
       0  0  0;...
      -1 -1 -1];
X2 = filter2(h2, X);  imshow(X2); title('prewitt');
```



更多的图像边界探测算子，参考edge-detect.m。可存档备用。

In [44]:

```
%load m/edge-detect.m
```

In [ ]:

```
function EdgeDetect(name,th,method)

%读取图像
SrcIm = imread(name);
%转换成灰度图
GrayIm = rgb2gray(SrcIm);

%uint8图像数据转换成单精度
GrayIm = double(GrayIm);


switch(method)
    case 'sobel'
        %高斯平滑滤波
        FiltedIm = imfilter(GrayIm, fspecial('gaussian'));
        result = grads(FiltedIm, fspecial('sobel'));
        if th ~= 0
            result = im2value(result,th);
        end
        result = ThinningImage(result);
        figure,imshow(result);
        title('Sobel edge detection');
    case 'prewitt'
        %高斯平滑滤波
        FiltedIm = imfilter(GrayIm, fspecial('gaussian'));
        result = grads(FiltedIm, fspecial('prewitt'));
        if th ~= 0
            result = im2value(result,th);
        end
        result = ThinningImage(result);
        figure,imshow(result);
        title('Prewitt edge detection');
    case 'roberts'
        %高斯平滑滤波
        FiltedIm = imfilter(GrayIm, fspecial('gaussian'));
        result = grads(FiltedIm, [-1 0;0 1]);
        if th ~= 0
            result = im2value(result,th);
        end
        result = ThinningImage(result);
        figure,imshow(result);
        title('Roberts edge detection');
    case 'LoG'
        result = LoGFilter(GrayIm);
        figure,imshow(result);
        title('LoG edge detection');
    case 'canny'
        result = CannyFilter(GrayIm);
        result = ThinningImage(result);
        figure,imshow(result);
        title('Canny edge detection');
end



%图像二值化函数
function result = im2value(array, th)
[y,x] = size(array);
result = false(y,x);
```

```
for i=1:y
    for j=1:x
        result(i,j) = ( array(i,j) > th );
    end
end

function result = th_im(array, th)
[y,x] = size(array);
result = zeros(y,x,'uint8');
for i=1:y
    for j=1:x
        if( array(i,j) < th )
            result(i,j) = 0;
        else
            result(i,j) = array(i,j);
        end
    end
end

%梯度检测
function result = grads(array, h)
[y,x] = size(array);
result = zeros(y, x, 'double');
%求横向梯度
Dx = imfilter(array, h, 'conv','replicate');
%求纵向梯度
Dy = imfilter(array, h', 'conv','replicate');
%求梯度的模
for i = 1:y
    for j = 1:x
        result(i,j) = round(sqrt(Dx(i,j)^2 + Dy(i,j)^2));
    end
end

%LoG边缘检测
function e = LoGFilter(array)
[m,n] = size(array);
fsize = ceil(2*3) * 2 + 1;  % choose an odd fsize > 6*sigma;
b = imfilter(array,fspecial('log',fsize,2),'replicate');
%创建输出矩阵
e = false(m,n);
%计算阈值
thresh = 0.75*mean2(abs(b));
rr = 2:m-1; cc=2:n-1;
%找出过零点
[rx,cx] = find( b(rr,cc) < 0 & b(rr,cc+1) > 0 & abs( b(rr,cc)-b(rr,cc+1) ) > thresh );   % [- +]
e((rx+1) + cx*m) = 1;
[rx,cx] = find( b(rr,cc-1) > 0 & b(rr,cc) < 0 & abs( b(rr,cc-1)-b(rr,cc) ) > thresh );   % [+ -]
e((rx+1) + cx*m) = 1;
[rx,cx] = find( b(rr,cc) < 0 & b(rr+1,cc) > 0 & abs( b(rr,cc)-b(rr+1,cc) ) > thresh);   % [- +]'
e((rx+1) + cx*m) = 1;
[rx,cx] = find( b(rr-1,cc) > 0 & b(rr,cc) < 0 & abs( b(rr-1,cc)-b(rr,cc) ) > thresh);   % [+ -]'
e((rx+1) + cx*m) = 1;
%清除孤立的噪声点
e = bwmorph(e, 'clean');

function result = CannyFilter(array)
[m,n] = size(array);
e = false(m,n);
width = 1;
sigma = 2;
```

```
%高斯平滑
t = (-width:width);
% 生成1维高斯滤波模板
H = exp(-(t.*t)/(2*(sigma^2)))/(2*pi*(sigma^2));
%生成2维高斯滤波模板
[x,y] = meshgrid(-width:width,-width:width);
H_2D = -x.*exp(-(x.*x+y.*y)/(2*(sigma^2)))/(pi*(sigma^2));
%H = fspecial('gaussian');
%两个方向进行高斯平滑
SmoothIm = imfilter(array,H,'conv','replicate');
SmoothIm = imfilter(SmoothIm,H','conv','replicate');
%计算边缘梯度的模
ax = imfilter(SmoothIm, H_2D, 'conv','replicate');
ay = imfilter(SmoothIm, H_2D', 'conv','replicate');
%mag = grads(SmoothIm, H_2D)
mag = sqrt((ax.*ax) + (ay.*ay));
%imshow(mag);
magmax = max(mag(:));
if magmax>0
    mag = mag / magmax;    % 对梯度幅值归一化
end
%求出双阈值法用到的高低阈值
counts=imhist(mag, 64);
highThresh = find(cumsum(counts) > 0.7*m*n,1,'first') / 64;
lowThresh = 0.4*highThresh;
thresh = [lowThresh highThresh];
%四个方向进行非最大值抑制
idxStrong = [];
for dir = 1:4
    idxLocalMax = cannyFindLocalMaxima(dir,ax,ay,mag);
    idxWeak = idxLocalMax(mag(idxLocalMax) > lowThresh);
    e(idxWeak)=1;
    idxStrong = [idxStrong; idxWeak(mag(idxWeak) > highThresh)];
end

rstrong = rem(idxStrong-1, m)+1;
cstrong = floor((idxStrong-1)/m)+1;
result = bwselect(e, cstrong, rstrong, 8);

%进行边缘细化
function result = ThinningImage(array)
result = bwmorph(array, 'clean');
result = bwmorph(result, 'thin', 1);

function idxLocalMax = cannyFindLocalMaxima(direction,ix,iy,mag)
[m,n] = size(mag);
%找出边缘点的坐标
switch direction
 case 1
  idx = find((iy<=0 & ix>-iy)  | (iy>=0 & ix<-iy));
 case 2
  idx = find((ix>0 & -iy>=ix)  | (ix<0 & -iy<=ix));
 case 3
  idx = find((ix<=0 & ix>iy) | (ix>=0 & ix<iy));
 case 4
  idx = find((iy<0 & ix<=iy) | (iy>0 & ix>=iy));
end

% Exclude the exterior pixels
if ~isempty(idx)
  v = mod(idx,m);
```

```
  extIdx = find(v==1 | v==0 | idx<=m | (idx>(n-1)*m));
  idx(extIdx) = [];
end

ixv = ix(idx);
iyv = iy(idx);
gradmag = mag(idx);

% 对结果进行线性内插
switch direction
 case 1
  d = abs(iyv./ixv);
  gradmag1 = mag(idx+m).*(1-d) + mag(idx+m-1).*d;
  gradmag2 = mag(idx-m).*(1-d) + mag(idx-m+1).*d;
 case 2
  d = abs(ixv./iyv);
  gradmag1 = mag(idx-1).*(1-d) + mag(idx+m-1).*d;
  gradmag2 = mag(idx+1).*(1-d) + mag(idx-m+1).*d;
 case 3
  d = abs(ixv./iyv);
  gradmag1 = mag(idx-1).*(1-d) + mag(idx-m-1).*d;
  gradmag2 = mag(idx+1).*(1-d) + mag(idx+m+1).*d;
 case 4
  d = abs(iyv./ixv);
  gradmag1 = mag(idx-m).*(1-d) + mag(idx-m-1).*d;
  gradmag2 = mag(idx+m).*(1-d) + mag(idx+m+1).*d;
end
idxLocalMax = idx(gradmag>=gradmag1 & gradmag>=gradmag2);
```