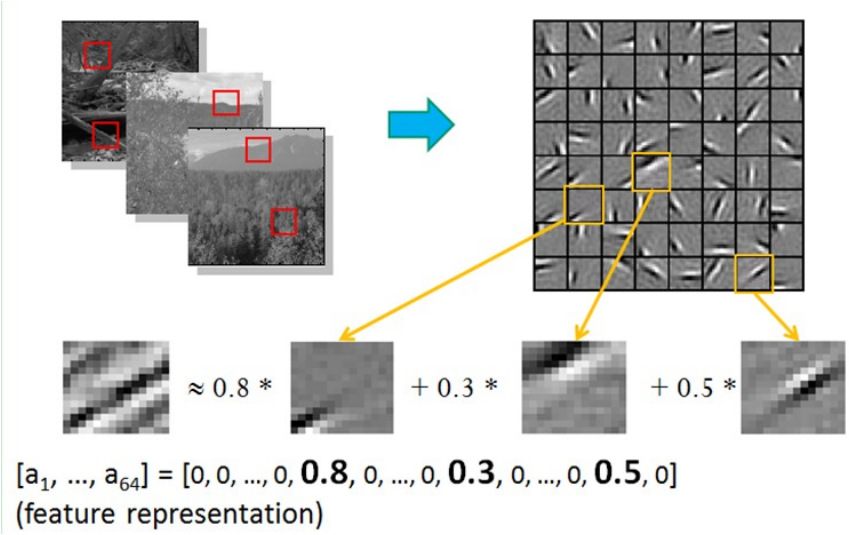
Canny 边缘检测

1.原理简述

图像的边缘指图像局部区域亮度变化显著的部分，该区域的灰度剖面一般可以看作是一个阶跃，即灰度值在很小的区域内急剧的变化。在计算机视觉中，图像边缘非常重要，用稀疏编码来表示图像可发现，经过迭代被遴选出的图片碎片基本都是不同物体的边缘线，也就是说复杂图形往往由一些基本结构组成，可以用正交的edges来线性表示。



边缘特征的拆分

实现图像的边缘检测，主要用离散化梯度逼近函数根据二维灰度矩阵梯度向量来寻找图像灰度矩阵的灰度跃变位置，然后在图像中将这些位置的点连起来就构成了所谓的图像边缘。

边缘检测的不同算法原理基本相同，只是采用的边缘检测算子有所区别。近20多年来随着计算机视觉的发展，边缘检测算子多种多样，这次尝试了较常见的Roberts，Sobel，Scharr，Laplacian，Prewitt和Canny算子等。

2.检测步骤

- **滤波**

边缘检测是基于图像强度的一阶和二阶导，但导数通常对噪声较为敏感，因此需要去噪处理。在这里我们使用了高斯滤波。

- **增强**

增强算法的目的是将图像灰度点邻域强度值有显著变化的点凸显出来。在这里我们首先使用边缘检测算子求图片的梯度，然后对梯度图进行非极大值抑制，只保留变化最显著的局部峰值点。

- **检测**

经过增强的图像，邻域中梯度值较大的很多，检测的目的是去除我们不需要的点。我们使用了双阈值算法减少假边缘的出现。

3.代码实现

- **读入灰度图并滤波**

```
img2=cv2.imread("/Users/markdana/Desktop/dataset/2.jpg",cv2.IMREAD_GRAYSCALE)
#默认是Gray=0.299R+0.587G+0.114B
Gauss_img=cv2.GaussianBlur(img2,(3,3),0)
```

需要注意的是，使用opencv自带的 `cv2.IMREAD_GRAYSCALE` 方式读入图片时，默认采用的是符合人眼生理特点的计算方法 Gray=0.299R+0.587G+0.114B；另外，采用高斯滤波时，过滤器的尺寸需要设成奇数。

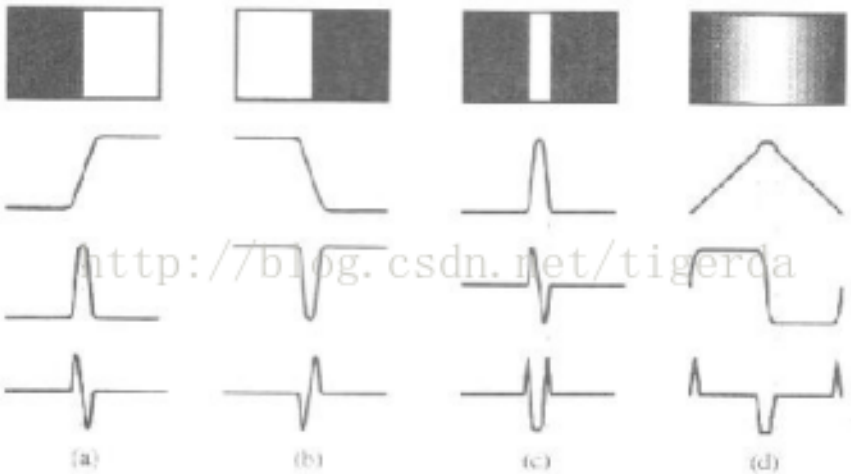
- **使用Canny算子求得图片x、y方向的梯度**

```
def cannyX(rawimg):
    img=rawimg.astype(np.float)
    res=np.zeros([img.shape[0]-1,img.shape[1]-1],float)
    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i][j]=(img[i][j+1]-img[i][j]+img[i+1,j+1]-img[i+1,j])/2
    return res

def cannyY(rawimg):
    img=rawimg.astype(np.float)
    res=np.zeros([img.shape[0]-1,img.shape[1]-1],float)
    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i][j]=(img[i][j]-img[i+1][j]+img[i,j+1]-img[i+1,j+1])/2
    return res
```

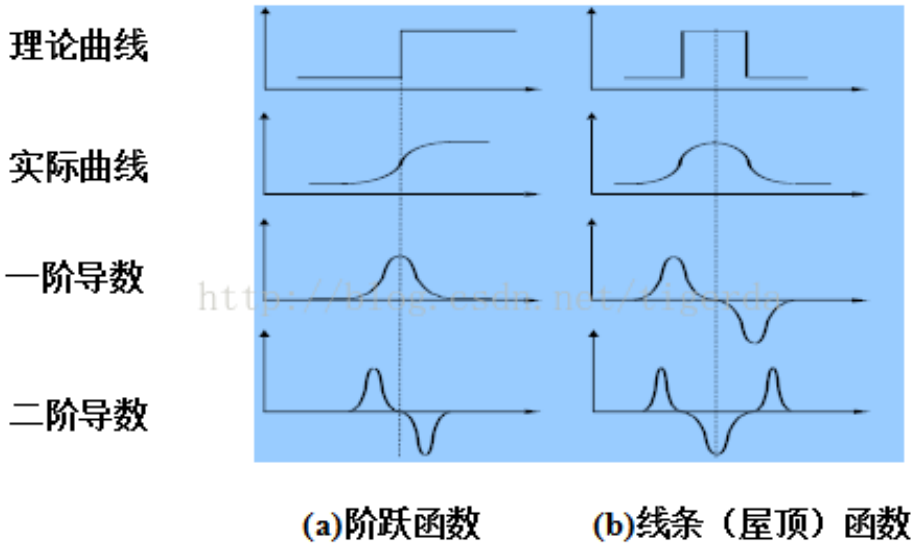
求梯度实际上可以认为是对原图在某方向上的求导。在以上代码中，为了处理图片边缘，输出的梯度图尺寸比原图尺寸小一圈，这样确保进行矩阵元素遍历求梯度时不会出现越界。如果苛求于保持图片尺寸不变，也可以使用卷积神经网络中卷积层的padding设置，也就是在进行卷积操作之前图片周围加一圈0（尺寸由卷积核尺寸、步幅等共同决定）；当然，也可以使用opencv自带的卷积操作，这里会自动padding以保持尺寸：

```python
def cannyX(rawimg):
    rawimg=rawimg.astype(np.float)
    filter=np.array([
        [-0.5, 0.5],
        [-0.5, 0.5]
    ],dtype=np.float64)
    res=cv2.filter2D(rawimg,-1,filter)
    return res

def cannyY(rawimg):
    rawimg=rawimg.astype(np.float)
    filter=np.array([
        [0.5, 0.5],
        [-0.5, -0.5]
    ],dtype=np.float64)
    res=cv2.filter2D(rawimg,-1,filter)
    return res
```

描述边缘需要两个属性：梯度的幅值和方向。边缘可以分为阶跃型、屋脊型、斜坡型、脉冲型。我们主要关注阶跃和屋脊型边缘，可通过微分算子来刻画。



边缘的分类



边缘的梯度函数特征

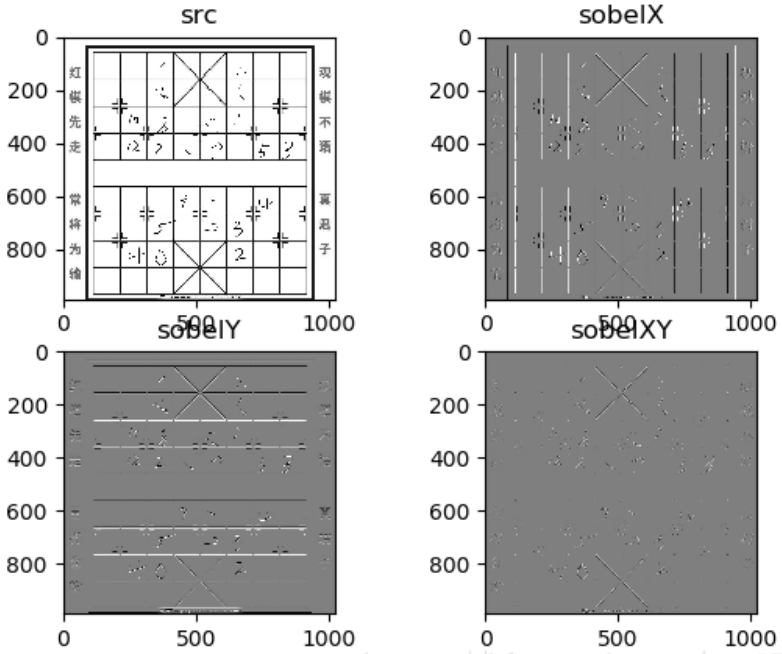
幅值可通过n2范数定义，直接使用numpy库的函数可算出。定义方向则通过x和y的梯度算出正切表示。实际处理中，为了防止分母为0，对分母采用加一个很小的bias操作，（根据前序步骤，分母不可能等于-bias）。

```
def cannyXY(imgx,imgy):
    return np.sqrt(imgx*imgx+imgy*imgy)
...
tan=absY[i][j]/(absX[i][j]+0.00000001)
```

换用其他的算子时方法基本相同，如Sobel算子：

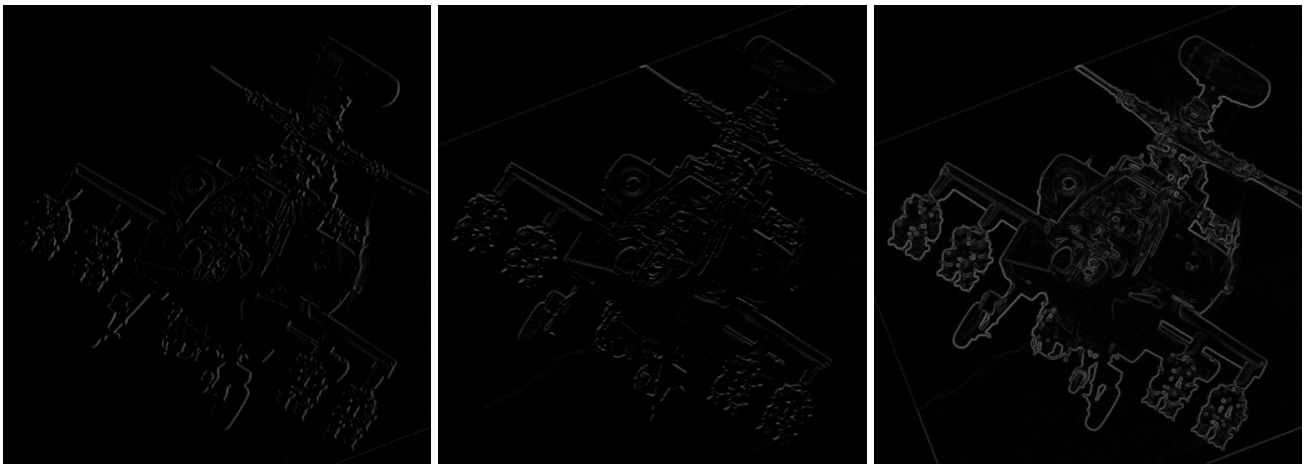
```
x = cv2.Sobel(img,cv2.CV_16S,1,0)
y = cv2.Sobel(img,cv2.CV_16S,0,1)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
absXY = np.sqrt(absX*absX+absY*absY)
```

要注意，传入参数的(0,1)和(1,0)分别表示y和x方向，但是传入(1,1)却并不是直接算出了两个方向并取幅值，也不是对角线方向的，而是另外的算法，此处用不到。



边缘的梯度函数特征

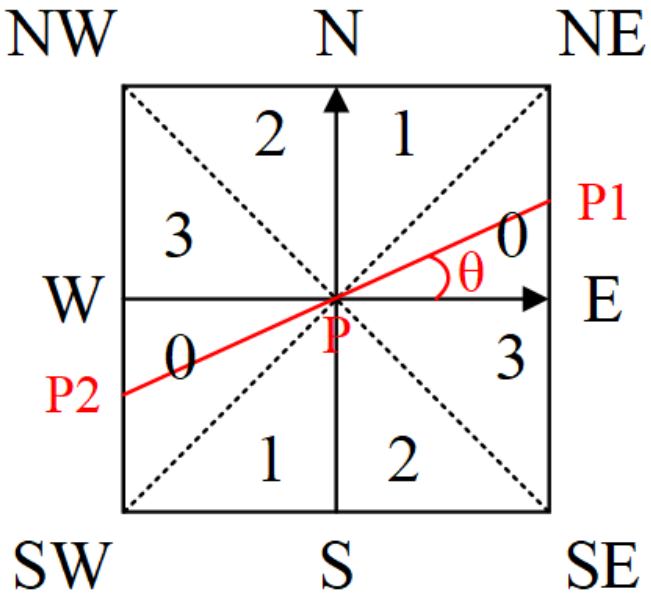
用Canny算子求梯度得到的结果如下(分别为x,y,幅值)：



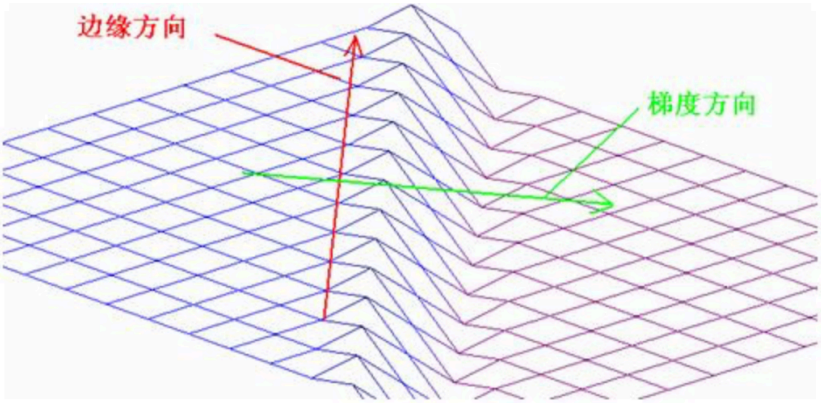
- **非极大值抑制**

```
for i in range(1,absXY.shape[0]-1):
    for j in range(1,absXY.shape[1]-1):
        tan=absY[i][j]/(absX[i][j]+0.00000001)
        if(tan>=0 and tan<=1):
            cmp1=tan*absXY[i-1][j+1]+(1-tan)*absXY[i][j+1]
            cmp2=tan*absXY[i+1][j-1]+(1-tan)*absXY[i][j-1]
            if(absXY[i][j]<cmp1 or absXY[i][j]<cmp2):
                absXY[i][j]=0
            continue

        if(tan>1):
            tan=1/tan
            cmp1=tan*absXY[i-1][j+1]+(1-tan)*absXY[i-1][j]
            cmp2=tan*absXY[i+1][j-1]+(1-tan)*absXY[i+1][j]
            if(absXY[i][j]<cmp1 or absXY[i][j]<cmp2):
                absXY[i][j]=0
            continue

        if(tan<-1):
            tan=-1/tan
            cmp1=tan*absXY[i-1][j-1]+(1-tan)*absXY[i-1][j]
            cmp2=tan*absXY[i+1][j+1]+(1-tan)*absXY[i+1][j]
            if(absXY[i][j]<cmp1 or absXY[i][j]<cmp2):
                absXY[i][j]=0
            continue

        if(tan<0 and tan>=-1):
            tan=-tan
            cmp1=tan*absXY[i-1][j-1]+(1-tan)*absXY[i][j-1]
            cmp2=tan*absXY[i+1][j+1]+(1-tan)*absXY[i][j+1]
            if(absXY[i][j]<cmp1 or absXY[i][j]<cmp2):
                absXY[i][j]=0
            continue
```

上一节的图片展示可见，边缘仍然十分模糊，这是由于没有选出局部的极大值，则相对来看不能凸显出边缘“线”。因此采用非极大值抑制，就是“瘦边”，边缘稀疏，使边缘只有一个准确的响应。以上的代码中我使用的是线性插值的比较方法，这种方法的连续性较好：

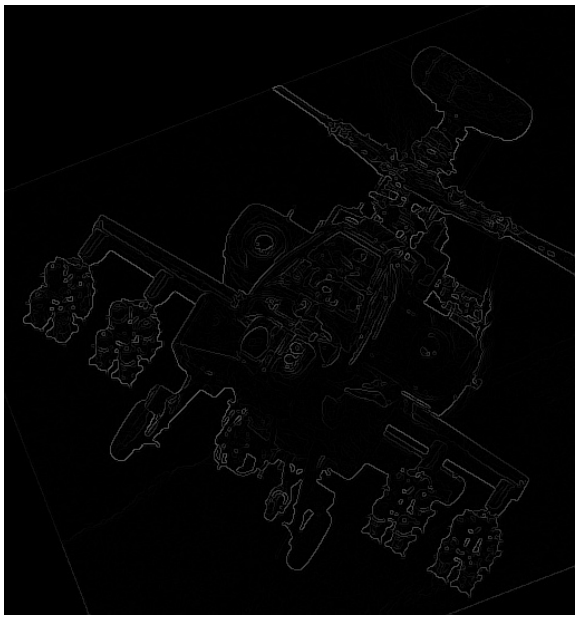


在刚开始我也并没有使用线性插值的方法，而是将图片的8邻域分为了8个区块，对应不同的角度值，22.5°、67.5°等。落在某个区间内的都对应到同一个index，即同一个梯度值。这种方法的确连续性不如后来采用的好。



1	2	3
8		4
7	6	5

相比上一节的结果，进行过非极大值抑制后的结果少了很多模糊区，边缘变得清晰了：



- **双阈值检测和连接**

如果直接对非极大值抑制后的图像（上图）进行二值化并输出，我们会发现，被划分为强边缘的像素点已经被确定为边缘并且是清晰的，即没有了连续的模糊区。但是，仍然存在弱边缘像素，其中有一些是与强边缘连续的、是实在的边缘；但也有一些是仅仅由于颜色变化或噪声引起的，他们离散在图中。怎样去除这些“假边缘呢”?

人们发明了双阈值算法。根据上一段的描述也可以知道算法的原理和实现：在强边缘点（大于高阈值的点）的周围的弱边缘点（介于高低阈值之间的点）才可以被判定为边缘，其余弱边缘点则被拍平。这样也能基本保证强边缘点不是离散的，可以连续起来。

代码实现如下：

```
highThreshold=np.zeros_like(absXY,dtype=float)
for i in range(1,absXY.shape[0]-1):
    for j in range(1,absXY.shape[1]-1):
        if (absXY[i][j]<=high):
            highThreshold[i][j]=0
        else:
            highThreshold[i][j]=255

lowThreshold=np.zeros_like(absXY,dtype=float)
for i in range(1,absXY.shape[0]-1):
    for j in range(1,absXY.shape[1]-1):
        if (absXY[i][j]<=low):
            lowThreshold[i][j]=0
        else:
            lowThreshold[i][j]=255

for i in range(1,absXY.shape[0]-1):
    for j in range(1,absXY.shape[1]-1):
        if(highThreshold[i][j]>0):
            highThreshold[i][j+1]=lowThreshold[i][j+1]
            highThreshold[i-1][j+1]=lowThreshold[i-1][j+1]
            highThreshold[i-1][j]=lowThreshold[i-1][j]
            highThreshold[i-1][j-1]=lowThreshold[i-1][j-1]
            highThreshold[i][j-1]=lowThreshold[i][j-1]
            highThreshold[i+1][j-1]=lowThreshold[i+1][j-1]
            highThreshold[i+1][j]=lowThreshold[i+1][j]
            highThreshold[i+1][j+1]=lowThreshold[i+1][j+1]
```

其中 `high` 和 `low` 为高低阈值。那么如何确定这两个阈值呢？显然与原图的梯度幅值有关，首先我考虑使用某个系数乘以梯度幅值的最大值，如

```
high=0.09*np.max(absXY)
low=0.03*np.max(absXY)
```

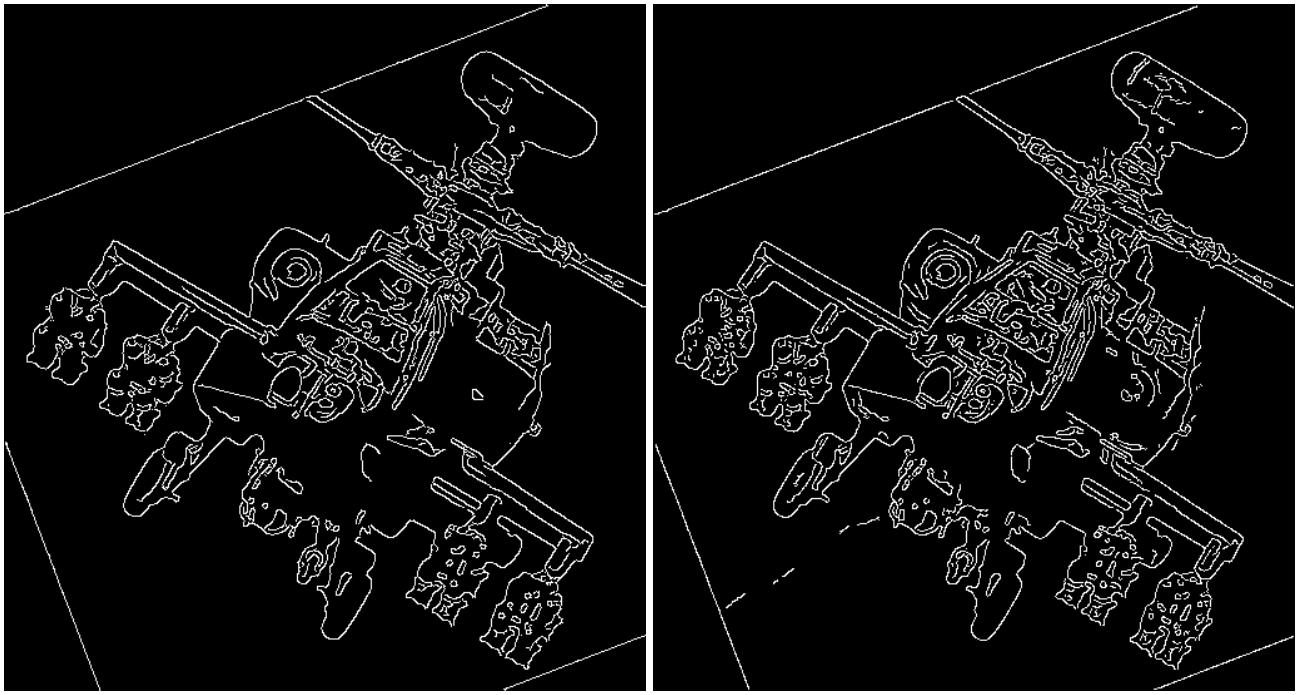
但后来我考虑到，假如梯度分布类似1,1,1,1,...,100呢？这种情况下max并不能很好的展示图片梯度幅值的总体特性。于是我想到了利用直方图，找到统计意义上占梯度幅值某个比例的双阈值。需要注意的是，由于有大量的0，这些甚至不是弱边缘点，则决定在做直方图时排除掉0，只考虑大于0的部分（用range参数实现）：

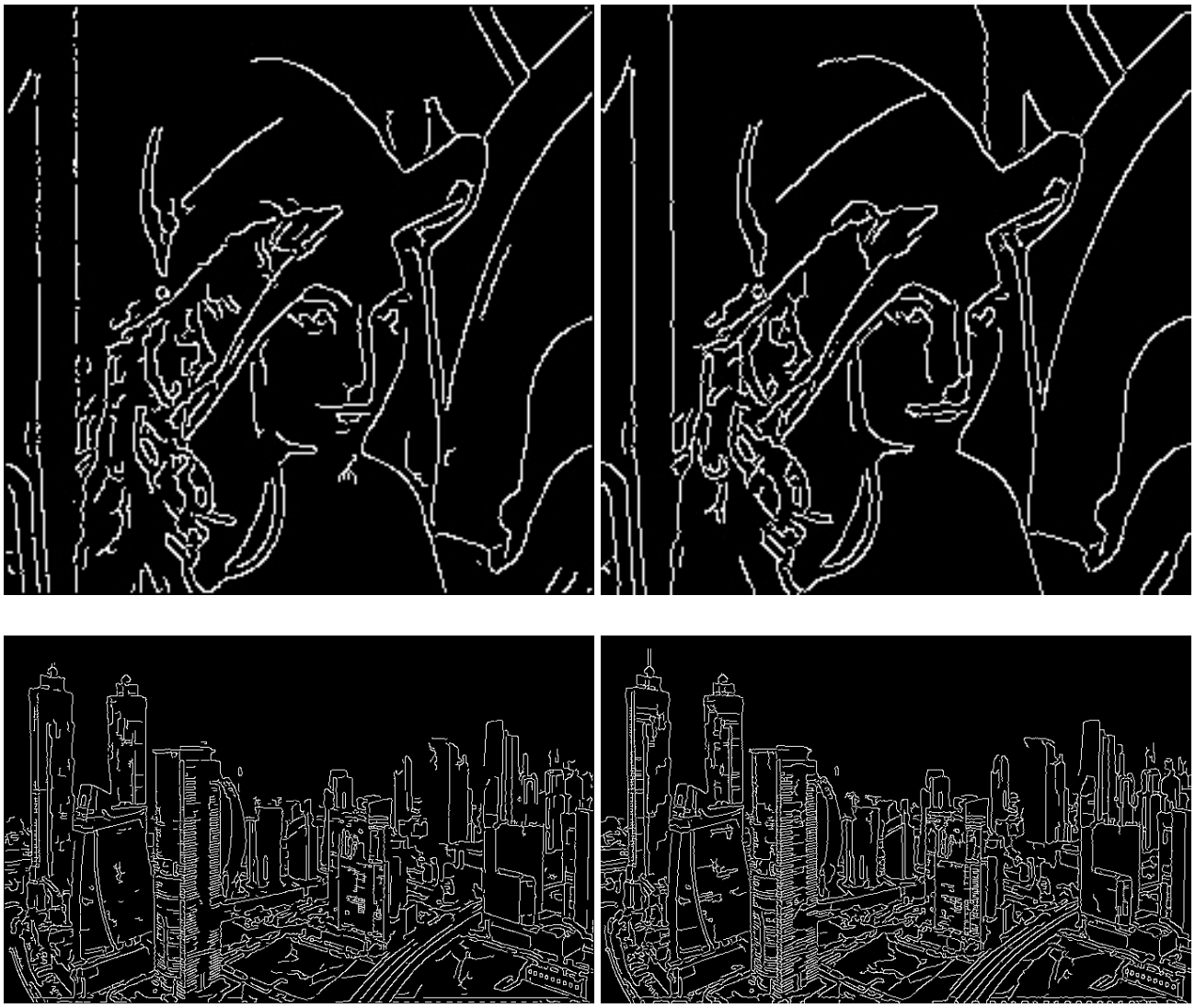
```
hist=np.histogram(absXY,bins=256,range=(0.01,256),normed=True)
low,high=0,0
sum=0
for i in range(0,len(hist[0])):
    if(sum<=0.25):
        sum+=hist[0][i]
    else:
        low=hist[1][i]
        break

sum=0
for i in range(0,len(hist[0])):
    if(sum<=0.70):
        sum+=hist[0][i]
    else:
        high=hist[1][i]
        break

print(low)
print(high)
```

当然，其中0.25和0.70是针对直升机图片调出来的经验值，也和网上的“high约为low的4倍“”这一说法不同。在我目前的测试集上，这个阈值算法表现良好，能够针对不同的图像有较清晰的边缘勾勒。

- **结果展示**

左边为我的程序跑出来的，右边为系统自带的函数跑出来的。（由于阈值也是试出来的，并不能完全等价）





- **完整代码**

```
#edge detection by Haoyue Dai @Nov/29/18
#前文中已有详细讲解，故此代码内不再注释
import cv2
import numpy as np

img2=cv2.imread("dataset/2.jpg",cv2.IMREAD_GRAYSCALE)
Gauss_img=cv2.GaussianBlur(img2,(7,7),0)

def Sobel(img):
    x = cv2.Sobel(img,cv2.CV_16S,1,0)
    y = cv2.Sobel(img,cv2.CV_16S,0,1)
    absX = cv2.convertScaleAbs(x)
    absY = cv2.convertScaleAbs(y)
    absXY = np.sqrt(absX*absX+absY*absY)
    return absX,absY,absXY

def scharr(img):
```

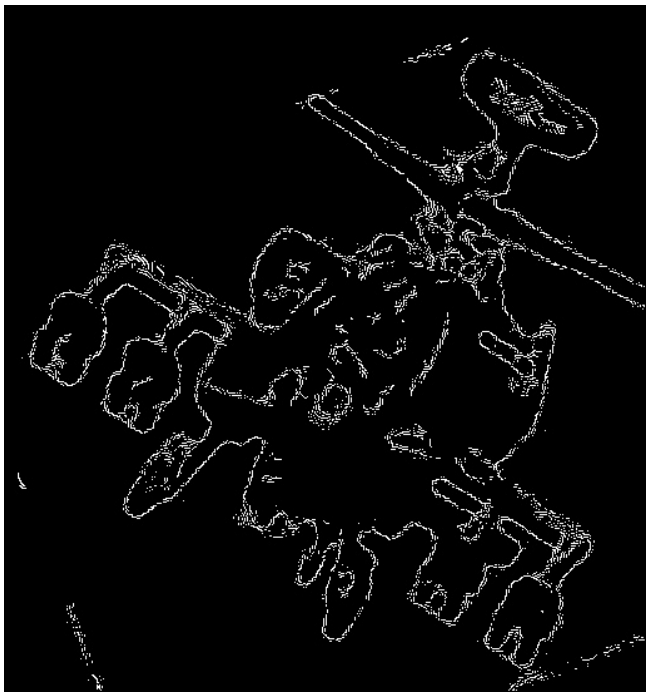
```
    x = cv2.Scharr(img,cv2.CV_16S,1,0)
    y = cv2.Scharr(img,cv2.CV_16S,0,1)
    absX = cv2.convertScaleAbs(x)
    absY = cv2.convertScaleAbs(y)
    absXY = np.sqrt(absX*absX+absY*absY)
    return absX,absY,absXY

def lap(img):
    img_lap=cv2.Laplacian(img,cv2.CV_64F)
    img_lap=cv2.convertScaleAbs(img_lap)
    return img_lap

'''
def cannyX(rawimg):
    img=rawimg.astype(np.float)
    res=np.zeros([img.shape[0]-1,img.shape[1]-1],float)
    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i][j]=(img[i][j+1]-img[i][j]+img[i+1,j+1]-img[i+1,j])/2
    return res

def cannyY(rawimg):
    img=rawimg.astype(np.float)
    res=np.zeros([img.shape[0]-1,img.shape[1]-1],float)
    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i][j]=(img[i][j]-img[i+1][j]+img[i,j+1]-img[i+1,j+1])/2
    return res
'''

def cannyX(rawimg):
    rawimg=rawimg.astype(np.float)
    filter=np.array([
        [-0.5, 0.5],
        [-0.5, 0.5]
    ],dtype=np.float64)
    res=cv2.filter2D(rawimg,-1,filter)
    return res

def cannyY(rawimg):
    rawimg=rawimg.astype(np.float)
    filter=np.array([
        [0.5, 0.5],
        [-0.5, -0.5]
    ],dtype=np.float64)
    res=cv2.filter2D(rawimg,-1,filter)
    return res

```

```
def cannyXY(imgx,imgy):
    return np.sqrt(imgx*imgx+imgy*imgy)

absX=cannyX(Gauss_img)
absY=cannyY(Gauss_img)
absXY=cannyXY(absX,absY)
suppression=np.zeros_like(absXY,dtype=float)
for i in range(1,absXY.shape[0]-1):
    for j in range(1,absXY.shape[1]-1):
        tan=absY[i][j]/(absX[i][j]+0.00000001)
        if(tan>=0 and tan<=1):
            cmp1=tan*absXY[i-1][j+1]+(1-tan)*absXY[i][j+1]
            cmp2=tan*absXY[i+1][j-1]+(1-tan)*absXY[i][j-1]
        elif(tan>1):
            tan=1/tan
            cmp1=tan*absXY[i-1][j+1]+(1-tan)*absXY[i-1][j]
            cmp2=tan*absXY[i+1][j-1]+(1-tan)*absXY[i+1][j]
        elif(tan<-1):
            tan=-1/tan
            cmp1=tan*absXY[i-1][j-1]+(1-tan)*absXY[i-1][j]
            cmp2=tan*absXY[i+1][j+1]+(1-tan)*absXY[i+1][j]
        else:
            tan=-tan
            cmp1=tan*absXY[i-1][j-1]+(1-tan)*absXY[i][j-1]
            cmp2=tan*absXY[i+1][j+1]+(1-tan)*absXY[i][j+1]
        if(absXY[i][j]>cmp1 and absXY[i][j]>cmp2):
            suppression[i][j]=absXY[i][j]


hist=np.histogram(suppression,bins=256,range=(0.01,256),normed=True)
low,high=0,0
sum=0
for i in range(0,len(hist[0])):
    if(sum<=0.25):
        sum+=hist[0][i]
    else:
        low=hist[1][i]
        break

sum=0
for i in range(0,len(hist[0])):
    if(sum<=0.70):
        sum+=hist[0][i]
    else:
        high=hist[1][i]
        break

print(low)
```

```
print(high)

#high=0.09*np.max(suppression)
#low=high/3

highThreshold=np.zeros_like(suppression,dtype=float)
for i in range(suppression.shape[0]):
    for j in range(suppression.shape[1]):
        if (suppression[i][j]<=high):
            highThreshold[i][j]=0
        else:
            highThreshold[i][j]=255

lowThreshold=np.zeros_like(suppression,dtype=float)
for i in range(suppression.shape[0]):
    for j in range(suppression.shape[1]):
        if (suppression[i][j]<=low):
            lowThreshold[i][j]=0
        else:
            lowThreshold[i][j]=255

for i in range(1,suppression.shape[0]-1):
    for j in range(1,suppression.shape[1]-1):
        if(highThreshold[i][j]>0):
            highThreshold[i][j+1]=lowThreshold[i][j+1]
            highThreshold[i-1][j+1]=lowThreshold[i-1][j+1]
            highThreshold[i-1][j]=lowThreshold[i-1][j]
            highThreshold[i-1][j-1]=lowThreshold[i-1][j-1]
            highThreshold[i][j-1]=lowThreshold[i][j-1]
            highThreshold[i+1][j-1]=lowThreshold[i+1][j-1]
            highThreshold[i+1][j]=lowThreshold[i+1][j]
            highThreshold[i+1][j+1]=lowThreshold[i+1][j+1]

cv2.imwrite("final.jpg",highThreshold)
system_canny=cv2.Canny(Gauss_img.astype(np.uint8), 40, 120)
cv2.imwrite("system_canny.jpg",system_canny)
```

- **一个思考**

整个边缘检测过程可以理解为卷积神经网络的第一层的工作。而关于高斯模糊，确实它能够降低噪声，但是同时也会降低边缘检测的敏感度，因为许多弱边缘也随着噪声被抑制了（尤其在高斯卷积核显著大于平均边缘直径时，如下图的特例）。

是否有优化的方案呢？我认为有，那就是不直接上高斯模糊，而是在高斯模糊的同时加入梯度检测（融合后面的过程）,若周围有连续的梯度方向，则判定某点是弱边缘，不被高斯模糊。算法正在实现中。