_____计算机视觉第三次实验_____

1611640305__温吉祥

实验内容：图像边缘检测

1. **Prewitt算子**
2. **Sobel算子**
3. **使用第三方库实现常见算子**

1 . Prewitt算子

1. **实现原理： Prewitt算子是一种一阶微分算子的边缘检测，利用像素点上下、左右邻点的灰度差，在边缘处达到极值检测边缘，去掉部分伪边缘，对噪声具有平滑作用 。其原理是在图像空间利用两个方向模板与图像进行邻域卷积来完成的，这两个方向模板一个检测水平边缘，一个检测垂直边缘。**

   垂直方向：

$$F_1 = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

   水平方向：

$$F_2 = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

2. **主要步骤及核心代码**

- 卷积函数：传入图像矩阵和检测算子，输出卷积后的矩阵

```
# 卷积函数
def imgConvolve(image_array, suanzi):
```

```
    '''
    :param image: 图片矩阵
    :param saunzi: 检测算子
    :return:卷积后的矩阵
    '''
    image = image_array.copy()      # 原图像矩阵的深拷贝
    dim1,dim2 = image.shape
    # 对每个元素与算子进行乘积再求和(忽略最外圈边框像素)
    for i in range(1,dim1-1):
        for j in range(1,dim2-1):
            image[i,j] = (image_array[(i-1):(i+2),(j-1):(j+2)]*suanzi).sum()
    # 由于卷积后灰度值不一定在0-255之间，统一化成0-255
    image = image*(255.0/image.max())
    # 返回结果矩阵
    return image
```

- Prewitt算子函数

```
# Prewitt Edge
def prewittEdge(image, prewitt):
    '''
    :param image: 图片矩阵
    :param prewitt: 检测方向
    :return:处理后的矩阵
    '''
    return imgConvolve(image, prewitt)
```

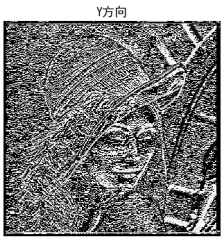
- 定义算子

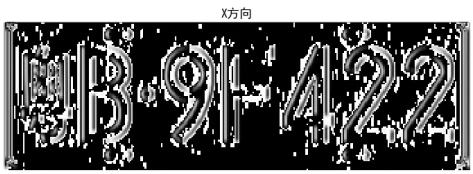
```
# prewitt 算子
prewitt_1 = 1/3*np.array([[-1, 0, 1],
                      [-1, 0, 1],
                      [-1, 0, 1]])

prewitt_2 = 1/3*np.array([[-1, -1, -1],
                      [0, 0, 0],
                      [1, 1, 1]])
```

- 传入图像并存储结果

```
img_prewitt1 = prewittEdge(image, prewitt_1)
cv2.imwrite('prewitt_x_lena.jpg',img_prewitt1)
img_prewitt2 = prewittEdge(image, prewitt_2)
cv2.imwrite('prewitt_y_lena.jpg',img_prewitt2)
image_xy = np.sqrt(img_prewitt1**2+img_prewitt2**2)
# 梯度矩阵统一到0-255
image_xy = (255.0/image_xy.max())*image_xy
cv2.imwrite('prewitt_xy_lena.jpg',image_xy)
```

- 结果图





---

2 . Sobel算子

1. **实现原理： Sobel算子是像素图像边缘检测中最重要的算子之一，在机器学习、数字媒体、计算机视觉等信息科技领域起着举足轻重的作用。在技术上，它是一个离散的一阶差分算子，用来计算图像亮度函数的一阶梯度之近似值。在图像的任何一点使用此算子，将会产生该点对应的梯度矢量或是其法矢量。**

两个方向的算子为：

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

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

1. **主要步骤及核心代码**

- Sobel算子函数

```
def sobelEdge(image, sobel):
    '''
    :param image: 图片矩阵
    :param sobel: 滤波窗口
    :return:Sobel处理后的矩阵
    '''
    return imgConvolve(image, sobel)
```

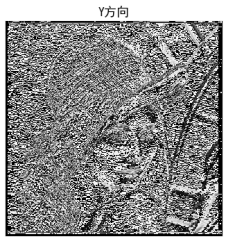
- Sobel算子

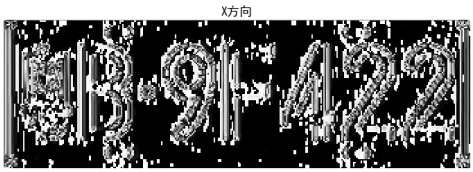
```
# sobel 算子
sobel_1 = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

sobel_2 = np.array([[-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]])
```

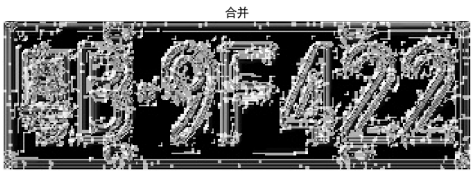
- 传入图像并存储结果

```
image=cv2.imread('lena.jpg',cv2.IMREAD_GRAYSCALE)
img_spbel1 = sobelEdge(image, sobel_1)
cv2.imwrite('sobel_x_lena.jpg',img_spbel1)
img_spbel2 = sobelEdge(image, sobel_2)
cv2.imwrite('sobel_y_lena.jpg',img_spbel2)
image_xy = np.sqrt(img_spbel1**2+img_spbel2**2)
# 梯度矩阵统一到0-255
image_xy = (255.0/image_xy.max())*image_xy
cv2.imwrite('sobel_xy_lena.jpg',image_xy)
```

- 结果对比图







3.使用第三方库实现常见算子

```
from skimage.filters import roberts,prewitt
```

```python
import cv2

image=cv2.imread('lena.jpg',cv2.IMREAD_GRAYSCALE)

gx = cv2.Scharr(image, ddepth=cv2.CV_16S, dx=1, dy=0)
gy = cv2.Scharr(image, ddepth=cv2.CV_16S, dx=0, dy=1)
gx_abs = cv2.convertScaleAbs(gx)
gy_abs = cv2.convertScaleAbs(gy)
scharr = cv2.addWeighted(src1=gx_abs, alpha=0.5, src2=gy_abs, beta=0.5, gamma=0)

canny = cv2.Canny(image,100,200)

sobel = cv2.Sobel(image,cv2.CV_8U, 1 , 1)

laplacian = cv2.Laplacian(image, -1)

robert = roberts(image)

prewitt = prewitt(image)
```