目录

1 实验介绍 **2**

1.1 实验内容 2

1.2 实验环境 2

2 实验过程 **3**

2.1 Canny 边缘检测原理 3

2.2 灰度化及高斯滤波 3

2.3 计算梯度 4

2.4 非极大值抑制 5

2.5 双阈值检测 6

2.6 抑制孤立弱边缘点 6

3 检测效果与参数设置 **8**

3.1 完整检测流程 8

3.2 高斯核参数 8

3.3 双阈值检测参数 9

4 总结与分析 **11**

1 实验介绍

1.1 实验内容

对 dataset 文件夹中的图片进行 Canny 边缘检测, 并与 OpenCV 库中自带 Canny 检测结果进行对比.

可选取不同梯度幅值算子与阈值获得更优的检测性能.

1.2 实验环境

操作系统: Ubuntu 18.04 LTS
Python 版本: 3.7
NumPy: 1.14.5
OpenCV-Python: 3.4.3.18

2 实验过程

2.1 Canny 边缘检测原理

Canny 边缘检测是一种目前被广泛使用的图像边缘检测算法, 于 1986 年由 John F. Canny 提出.Canny 边缘检测算法是一种多阶段算法, 可以分为以下 5 个步骤:

1. 使用高斯滤波器, 以平滑图像, 滤除噪声.
2. 计算图像中每个像素点的梯度 (包含强度和方向).
3. 应用非极大值抑制, 消除边缘检测带来的杂散响应.
4. 应用双阈值算法来确定真实的边缘, 潜在的边缘, 以及非边缘.
5. 抑制孤立的弱边缘最终完成边缘检测.

2.2 灰度化及高斯滤波

对于一张彩色图片, 以 RGB 格式读入之后, 需要将其进行灰度化处理. 为了方便调节灰度化的具体参数, 自行编写的处理函数如下:

```
def rgb_to_grayscale(rgb_img):
    ratios = [0.114, 0.587, 0.299]
    result = np.zeros(rgb_img.shape[:-1], dtype="float32")
    for i in range(3):
        result += ratios[i] * rgb_img[:, :, i]
    result = result.astype("uint8")
    return result
```

在经过灰度化处理之后, 为了滤除图像中的噪声, 避免对后续边缘检测造成影响, 可采用高斯滤波的方法, 平滑图像. 大小为 $(2k+1)*(2k+1)$ 的高斯滤波器核的生成方式如下:

$$K_{ij} = \frac{1}{2\pi\sigma^2} e^{-\frac{(i-(k+1))^2+(j-(j+1))^2}{2\sigma^2}}, 1 \leq i, j \leq 2k+1$$

事实上, 上式即为二维正态分布概率密度公式. 但在生成了高斯滤波器核之后, 还需要对其进行归一化处理, 使其中元素之和为 1. 完整代码如下:

```
def get_gaussian_kernel(size=3, sigma=1):
    kernel = np.zeros((size, size))
    k = int((size-1)/2)
    for x in range(-k, k+1):
        for y in range(-k, k+1):
            kernel[k+x][k+y] = np.exp(-(x*x+y*y)/(2*sigma*sigma)) /
            ↪  (2*np.pi*sigma*sigma)
    kernel = kernel / np.sum(kernel)
    return kernel
```

在得到高斯核之后, 可调用 OpenCV 函数, 对图像进行卷积处理:

```
gaussian_blur_img = cv2.filter2D(img, -1, kernel)
```

需要注意的是, 高斯卷积核大小的选择将影响 Canny 检测器的性能: 尺寸越大,Canny 检测器对噪声的敏感度就越低, 但是边缘检测的定位误差也将略有增加.

2.3 计算梯度

关于图像灰度值的梯度, 可使用一阶有限差分来进行近似计算. 常见的梯度算子有 Roberts 算子,Sobel 算子,Prewitt 算子等. 不同的梯度算子有着不同的效果, 影响 Canny 边缘检测算法的最终结果. 在本次实验中, 我选择了 Sobel 算子.

Sobel 算子的 x,y 方向卷积模板为:

$$s_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, s_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

对每个像素点, 分别在 x 方向和 y 方向进行卷积之后, 得到梯度的 x 方向和 y 方向上的分量 g_x, g_y. 则梯度的强度和方向定义为:

$$M(i,j) = \sqrt{g_x^2 + g_y^2}$$

$$\theta(i,j) = \arctan \frac{g_y}{g_x}$$

需要注意的是, 考虑到梯度的方向可能为 $\pm\pi/2$, 此时 $g_x = 0$, 为避免除法出错, 需要将计算梯度方向的公式略作修改:

$$\theta(i,j) = \arctan \frac{g_y}{g_x + \epsilon}$$

故完整的代码如下:

```
def sobel(img):
    result = np.zeros_like(img)
    direction = np.zeros_like(img)
    width, height = img.shape
    s_y = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1], ])
    s_x = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1], ])
    for i in range(1, width-1):
        for j in range(1, height-1):
            x = np.sum(img[i-1:i+2, j-1:j+2] * s_x)
            y = np.sum(img[i-1:i+2, j-1:j+2] * s_y)
            result[i][j] = np.sqrt(x * x + y * y)
```

```
            direction[i][j] = np.arctan(y/(x+1e-5))
    result = result.astype("uint8")
    return result, direction
```

2.4 非极大值抑制

在 Canny 边缘检测算法中, 需要抑制那些梯度强度非极大的像素点.

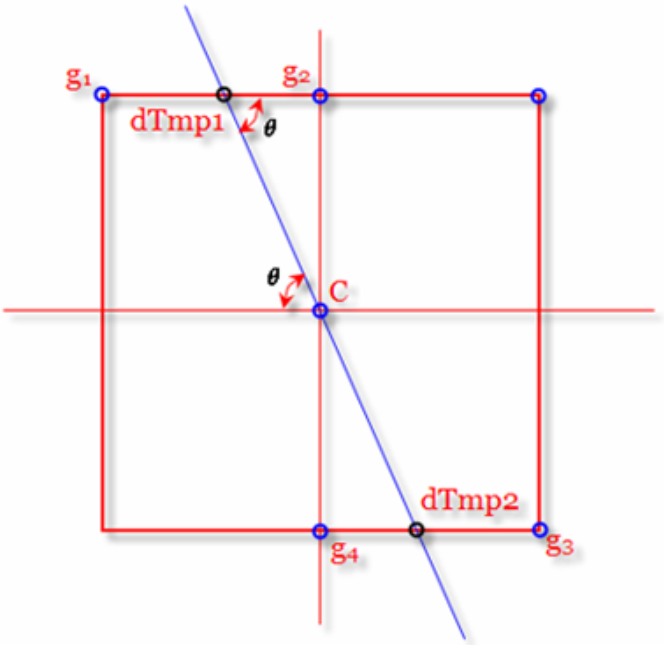


图 1: 插值法

如图所示, 对于点 C, 沿着其梯度方向, 与图像的像素网格产生了两个最近的交点 $dTmp1$ 和 $dTmp2$, 为确保 C 是梯度的极大值点, 需要满足 C 点的梯度强度比这两点都要大, 否则就将其抑制. 而由于 $dTmp1, dTmp2$ 这两点并非图像本身的像素点, 其梯度值只能通过插值法得到.

简便起见, 本次实验中我采用了线性插值法. 以 $dTmp1$ 点为例, 其梯度值介于 g_1 和 g_2 之间, 则根据 $dTmp1$ 到两者的距离比例, 计算出该点的梯度值即可.

在程序实现中, 根据梯度方向不同, 分为了 $|\theta| < \pi/4$ 和 $|\theta| \geq \pi/4$ 两类进行处理, 具体如下

```
def non_maximum_suppression(gradient, direction):
    result = np.zeros_like(gradient)
    width, height = gradient.shape
    for i in range(1, width-1):
        for j in range(1, height - 1):
            if -np.pi/4 < direction[i][j] < np.pi/4:
                x1, y1 = i + 1, j + np.tan(direction[i][j])
                x2, y2 = i - 1, j - np.tan(direction[i][j])
                temp1 = (y1 - int(y1))*gradient[x1][min(height-1, int(y1)+1)]
```

```
                    + (int(y1) + 1 - y1)*gradient[x1][int(y1)]
                temp2 = (y2 - int(y2))*gradient[x2][min(height-1, int(y2)+1)]
                    + (int(y2) + 1 - y2)*gradient[x2][int(y2)]
            else:
                x1, y1 = i + 1/np.tan(direction[i][j]), j + 1
                x2, y2 = i - 1/np.tan(direction[i][j]), j - 1
                temp1 = (x1 - int(x1))*gradient[min(width-1, int(x1)+1)][y1]
                    + (int(x1) + 1 - x1)*gradient[int(x1)][y1]
                temp2 = (x2 - int(x2))*gradient[min(width-1, int(x2)+1)][y2]
                    + (int(x2) + 1 - x2)*gradient[int(x2)][y2]
            if(gradient[i][j] >= temp1 and gradient[i][j] >= temp2):
                result[i][j] = gradient[i][j]
    return result
```

2.5 双阈值检测

在施加非极大值抑制之后, 为了减少假边缘,Canny 边缘检测算法采取了双阈值检测的做法. 即设定高阈值和低阈值, 对于每一个像素点的梯度强度, 如果高于高阈值, 则确定其为图像的边缘像素点; 如果低于低阈值, 则认为其不为边缘像素点, 将其抑制; 如果处于高低阈值之间, 则将其标记为弱边缘像素点, 留待进一步处理.

在程序实现中, 为了方便, 将弱边缘像素点的值设置为 1, 强边缘像素点的值设置为 255.

```
def double_threshold(img, low, high):
    result = np.zeros_like(img)
    width, height = img.shape
    for i in range(0, width):
        for j in range(0, height):
            result[i][j] = 254*(img[i][j] >= high) + 1*(img[i][j] >= low)
    return result
```

2.6 抑制孤立弱边缘点

弱边缘点可能是真的边缘, 也可能是由噪声引起的误差. 对于前者, 我们需要保留, 而后者, 则需要将其抑制.Canny 边缘检测算法认为, 如果一个弱边缘点周围的 8 个领域像素中存在强边缘点, 那么它是一个边缘像素点, 否则便是需要被抑制的噪声.

在我的程序实现中, 为了更好的处理那些不确定是否为边缘的点, 多次尝试, 直到完全确定其不为边缘点为止.

```
def track_edge(img):
    result = np.zeros_like(img)
    uncertain_points = list()
    width, height = img.shape
```

```
    for i in range(1, width-1):
        for j in range(1, height-1):
            if img[i][j] == 255:
                result[i][j] = 255
            elif img[i][j] == 1:
                result[i][j] = 0
                doubt = False
                for x in range(i-1, i+2):
                    for y in range(j-1, j+2):
                        if img[x][y] == 255:
                            result[i][j] = 255
                        elif img[x][y] == 1:
                            doubt = True
                if(result[i][j] == 0 and doubt):
                    uncertain_points.append((i, j))
    prev_len = 0
    while(prev_len != len(uncertain_points)):
        prev_len = len(uncertain_points)
        print(prev_len)
        still_uncertain = list()
        for point in uncertain_points:
            for x in range(point[0]-1, point[0]+2):
                for y in range(point[1]-1, point[1]+2):
                    if result[x][y] == 255:
                        result[point[0]][point[1]] = 255
            if result[point[0]][point[1]] == 0:
                still_uncertain.append(point)
        uncertain_points = still_uncertain
    return result
```

3 检测效果与参数设置

3.1 完整检测流程

图2展示了 Canny 边缘检测的完整流程, 并给出了 OpenCV 库中的 Canny() 函数的处理结果, 以供对比.

(a) 原始图片　(b) 灰度图片　(c) 高斯滤波　(d) 计算图像梯度

(e) 非极大值抑制　(f) 双阈值检测　(g) 抑制孤立弱边缘点 (最终结果)　(h) OpenCV.Canny 函数处理结果

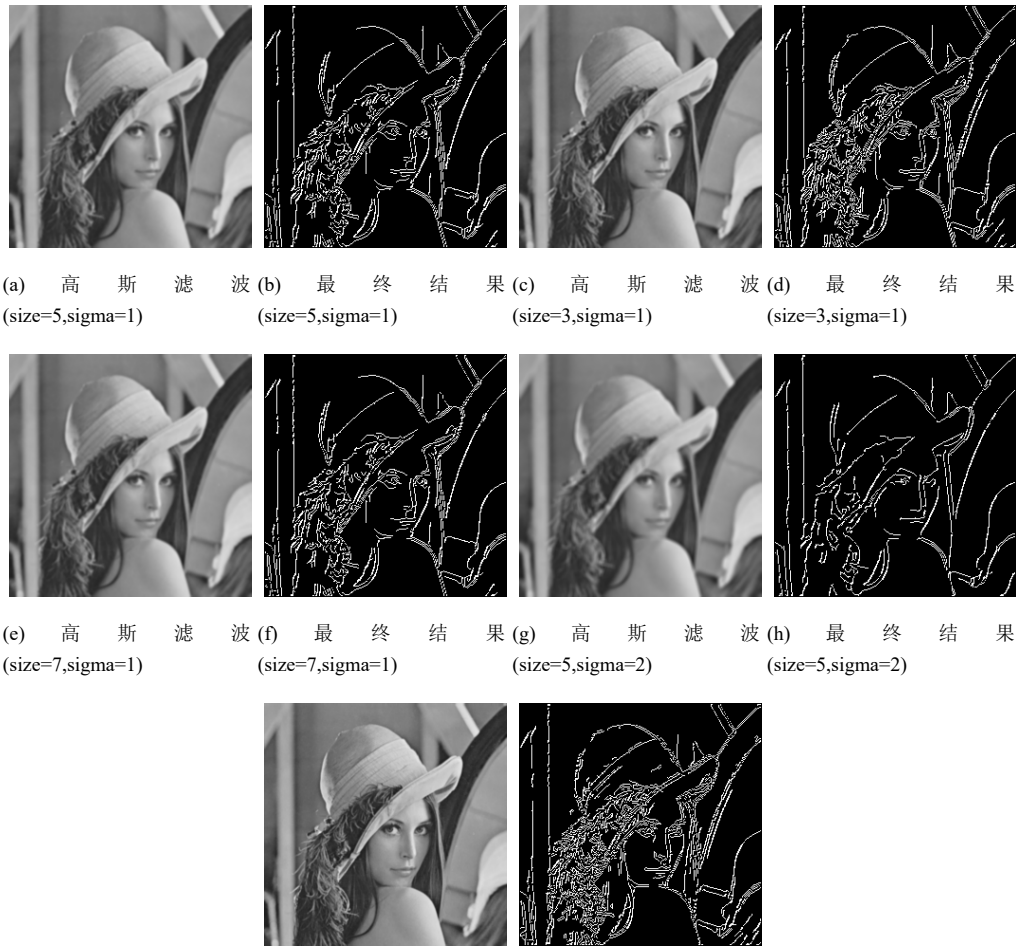
图 2: Canny 边缘检测流程 (size=3,sigma=1,low=50,high=100)

3.2 高斯核参数

根据2.2, 可知在进行高斯滤波处理时, 主要有两个可以调节的参数: 高斯核的尺寸 (size) 与高斯分布函数的标准差 (sigma).

直观上分析, 当高斯核的尺寸增大时, 高斯模糊的效果增强, 从而能够更好的抵御图像中的噪声, 但也不可避免地会失去一部分细节, 从而导致最终的检测结果精度下降; 而当标准差增大时, 高斯分布趋于平缓, 从而同样的使得高斯模糊的效果增强.

按不同的高斯滤波参数进行实验, 得到的结果如组图3所示 (各组实验的 low,high 均为 50,100), 分析边缘检测的效果, 与上述直观的想法基本一致. 同时也选定了较优的参数设定:size=3,sigma=1.



(a) 高斯滤波(size=5,sigma=1) (b) 最终结果(size=5,sigma=1) (c) 高斯滤波(size=3,sigma=1) (d) 最终结果(size=3,sigma=1)

(e) 高斯滤波(size=7,sigma=1) (f) 最终结果(size=7,sigma=1) (g) 高斯滤波(size=5,sigma=2) (h) 最终结果(size=5,sigma=2)

(i) 高斯滤波(size=5,sigma=0.5) (j) 最终结果(size=5,sigma=0.5)

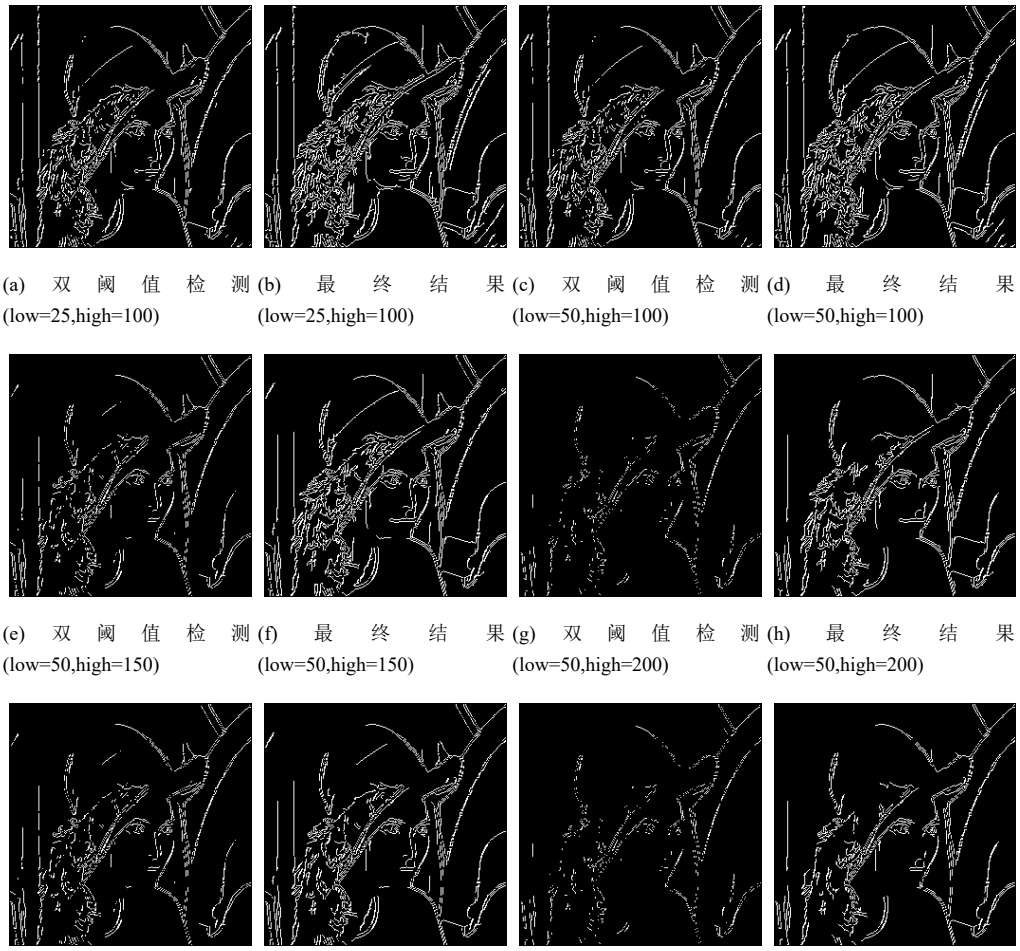
图 3: Canny 边缘检测效果与高斯滤波参数

3.3 双阈值检测参数

由2.5中的原理可知, 高低阈值影响着 Canny 算法对图像中噪声与细节的相互取舍, 不同的阈值必然会产生不同的边缘检测效果.

依旧是先从直观上进行分析: 若高阈值和低阈值都设定的很高, 那么会失去很多细节; 若高阈值和低阈值都设定的很低, 那么会受到噪声的极大干扰, 最终产生的边缘将是很不平滑的; 若高阈值很高, 低阈值很低, 那么大多数点将标记为弱边缘点, 依赖于最后一步的孤立弱边缘点抑制, 很可能导致边缘不完整; 若高阈值很低, 低阈值很高, 那么仅产生很少的弱边缘点,Canny 算法发生了退化, 不能充分发挥其效果.

按不同的阈值参数进行实验, 得到的结果如组图4所示 (各组实验中 size,sigma 均为 3,1), 分析结果, 基本上与上述直观分析相符合. 同时也发现,low=50,high=100 是一组相对较好的阈值设定.



(a) 双阈值检测 (low=25,high=100)　(b) 最终结果 (low=25,high=100)　(c) 双阈值检测 (low=50,high=100)　(d) 最终结果 (low=50,high=100)

(e) 双阈值检测 (low=50,high=150)　(f) 最终结果 (low=50,high=150)　(g) 双阈值检测 (low=50,high=200)　(h) 最终结果 (low=50,high=200)

(i) 双阈值检测 (low=100,high=150)　(j) 最终结果 (low=100,high=150)　(k) 双阈值检测 (low=100,high=200)　(l) 最终结果 (low=100,high=200)

图 4: Canny 边缘检测效果与双阈值参数

4 总结与分析

本次实验中, 我实现了基本的 Canny 边缘检测算法, 并探索设定了其中的部分参数, 以取得更好的边缘检测效果. 在这个过程中, 我对图像处理的许多概念有了较为清晰的认知: 譬如滤波, 噪声处理, 卷积等.

但在我看来, 不可否认 Canny 算法能够比较好的实现图像的边缘检测, 但其效果依赖于对具体图像的参数设定, 一旦参数设定不当, 其效果远不能使人满意. 而近年来的计算机视觉领域不断发展, 带来了许多新的基于深度学习的图像处理方法, 取得了更好的效果, 或许将来能够真正地基于内容对图像进行边缘检测, 而非机械地基于图像特征处理.