电工导 C 第十次实验报告

姓名: 宋士祥　　学号:521030910013　　班级:F2103001

2022 年 11 月 23 日

1　实验概览

本次实验简单学习了边缘检测的相关内容。图象的边缘指图象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。

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。

Canny 算法是一种很好的边缘检测算法，分为以下五个步骤：

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强度和方向。
3. 应用非极大值（Non-Maximum Suppression）抑制，以消除边缘检测带来的杂散响应。
4. 应用双阈值（Double-Threshold）检测来确定真实的和潜在的边缘。
5. 通过抑制孤立的弱边缘最终完成边缘检测。

在`openCV` 库中，我们有一个成熟的`Canny` 函数可以直接进行边缘检测，然而本次实验中我们会尝试自己写一个与之相仿的`Canny` 函数。

2　实验环境

本次实验采用所需的实验环境如下：

- Docker 中的`sjtumic/ee208` 镜像
- Python3（使用 VSCode 编译）
- `numpy` 扩展。
- `OpenCV` 环境。(均在 SJTUEE208 镜像中给出)

3　问题重述、实验原理与代码说明

本次实验我们需要完成 Canny 函数。为了方便我们最后的输出，我们将整个函数封装到一个大的`Canny` 类中便于我们后续进行实验。分为五个任务：

3.1 灰度化

灰度化的原理在实验九已经提及。我们采取直接灰色读入的方式完成即可。值得注意的是，openCV 在读取的时候采用是最常用的一种方法，公式如下：

$$\text{Gray} = 0.299R + 0.587G + 0.114B$$

相关代码如下：

```
# 任务一灰度读取:
def __init__(self, image_index):
    self.img = cv2.imread('dataset/{}.jpg'.format(str(image_index)), cv2.
                                                    IMREAD_GRAYSCALE)
```

3.2 高斯滤波

图像高斯滤波的实现可以用两个一维高斯核分别两次加权实现，也可以通过一个二维高斯核一次卷积实现。

对于二维的高斯函数，我们有 (ppt 中有误，不应当是 $*$ 的卷积符号)：

$$H(x,y) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

最后需要对其归一化处理。

确定参数 σ 就可以得到一维核向量与二维核向量（以一个 3×3 的二维卷积核为例）：

$$H = \begin{bmatrix} H(-1,-1) & H(0,-1) & H(1,-1) \\ H(-1,0) & H(0,0) & H(1,0) \\ H(-1,1) & H(0,1) & H(1,1) \end{bmatrix}$$

再将该核与图像求卷积即可。

ppt 中给出了一个 $\sigma = 1.4$ 的情形。

$$H = \begin{bmatrix} 0.0924 & 0.1192 & 0.0924 \\ 0.1192 & 0.1538 & 0.1192 \\ 0.0924 & 0.1192 & 0.0924 \end{bmatrix}$$

而高斯滤波在openCV 中已经有一个比较类似的函数，于是我们有对应函数：

```
def Guass_noise(self):
    Gause_img = cv2.GaussianBlur(self.img, (3, 3), 0)
    self.Gause_img = Gause_img
```

3.3 灰一阶偏导的有限差分来计算梯度的幅值和方向

关于图像灰度值得梯度可使用一阶有限差分来进行近似，这样就可以得图像在 x 和 y 方向上偏导数的两个矩阵。本次实验采用 Sobel 算子进行计算。

Sobel 算子在 x 方向和 y 方向的卷积核分别为：

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

卷积模板为该点前后左右 3×3 的点，最终形成一个 3×3 的卷积矩阵。分别对 x 方向，y 方向求卷积，得到 g_x, g_y，利用公式 $G(x,y) = \sqrt{g_x^2 + g_y^2}$，得到梯度矩阵。

同样，由于我们需要求后续的方向，我们同样需要放一个 $\tan\theta = \frac{g_y}{g_x}$ 的矩阵。但是考虑到读取等的问题，我们只是象征性的添加了该矩阵，在后续的任务中我们不会使用该矩阵。

与前两个任务类似，OpenCV 中有提供类似的函数模板。应当注意的是 ppt 中只提供了 x 方向的 sobel 模板，而对于 y 方向，需要修改对应的参数。相应代码如下：

```python
# 任务三灰一阶偏导的有限差分来计算梯度的幅值和方向:
def Guass_sobel_gradient(self):
    img_sobel_x = cv2.Sobel(self.Gause_img, cv2.CV_16S, 1, 0)
    img_sobel_y = cv2.Sobel(self.Gause_img, cv2.CV_16S, 0, 1)
    self.dx, self.dy = img_sobel_x, img_sobel_y
    theta = np.zeros(img_sobel_x.shape)
    gradient = np.zeros(img_sobel_x.shape)
    for i in range(img_sobel_x.shape[0]):
        for j in range(img_sobel_x.shape[1]):
            gradient[i][j] = math.sqrt((img_sobel_x[i][j]**2) +
                                       (img_sobel_y[i][j]**2))
            theta[i][j] = math.atan(img_sobel_y[i][j] / img_sobel_x[i][j])
    self.gradient = gradient
    # 这里按照要放Ppt但是操作中我们不用，求反三角再求回去多少会引入误差,
    self.theta = theta
```

3.4 梯度幅值进行非极大值抑制

图像梯度幅值矩阵中的元素值越大，说明图像中该点的梯度值越大。在 Canny 算法中，非极大值抑制是进行边缘检测的重要步骤，**是指寻找像素点梯度局部最大值，将非极大值点所对应的灰度值置为** 0，从而可以剔除掉一大部分非边缘点。

利用 3.3 部分的 dx 与 dy，我们可以大致判断梯度的方向。随后我们需要通过比较周边的梯度值 (利用权重 $\frac{g_y}{g_x}$ 或 $\frac{g_x}{g_y}$ 比较极值 (其实本来公式会复杂一点，这里并未采用 ppt 中所述的方法，具体参考:https://blog.csdn.net/kezunhai/article/details/11620357))，以确定该点是否为边界点（因为边界点梯度值会突变）。

同样，由于我们只选取了 3×3 的矩阵，只有几个固定方向，而真实的边界方向会更复杂，所以我们采用了插值的方法对其进行操作，其插值方法为dTemp1 = weight*g1 + (1-weight)*g2;dTemp2 = weight*g3 + (1-weight)*g4;。只有当该点同时大于这两个极大值才无需变动。

同时，我们还需要对梯度的方向判断，如方向一致我们取行列式主对角线方向，而若方向相反我们应当取行列式副对角线方向。对应代码如下：

```python
# 任务四对梯度幅值进行非极大值抑制:
def NMS(self):
    # 三个梯度
    dx, dy = self.dx, self.dy
```

```
    d = self.gradient
    width, height = d.shape
    # 一个过来copy但是不能直接做会使得内存错误,,
    NMS = np.copy(d)
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            # 如果当前梯度为，该点就不是边缘点0
            if d[i][j] == 0:
                NMS[i][j] = 0
            else:
                gradX = dx[i][j]  # 当前点 x 方向导数
                gradY = dy[i][j]  # 当前点 y 方向导数
                grad = d[i][j]  # 当前梯度点
                # 如果 y 方向梯度值比较大，说明导数方向趋向于 y 分量
                if abs(gradY) > abs(gradX):
                    weight = abs(gradX) / abs(gradY)
                    grad2 = d[i - 1][j]
                    grad4 = d[i + 1][j]
                    # 通过导数符号决定取得边界
                    if gradX * gradY > 0:
                        grad1 = d[i - 1][j - 1]
                        grad3 = d[i + 1][j + 1]
                    else:
                        grad1 = d[i - 1][j + 1]
                        grad3 = d[i + 1][j - 1]
                # 如果 x 方向梯度值比较大
                else:
                    weight = abs(gradY) / abs(gradX)
                    grad2 = d[i][j - 1]
                    grad4 = d[i][j + 1]
                    if gradX * gradY > 0:
                        grad1 = d[i + 1][j - 1]
                        grad3 = d[i - 1][j + 1]
                    else:
                        grad1 = d[i - 1][j - 1]
                        grad3 = d[i + 1][j + 1]
                # 由的公式进行插值运算ppt
                gradTemp1 = weight * grad1 + (1 - weight) * grad2
                gradTemp2 = weight * grad3 + (1 - weight) * grad4
                # 当前像素的梯度是局部的最大值，可能是边缘点，此时无需变动
                if grad >= gradTemp1 and grad >= gradTemp2:
                    continue
                else:
                    # 删掉~
                    NMS[i][j] = 0
    self._NMS = NMS
```

3.5 双阈值算法检测和连接边缘

Canny 算法中减少假边缘数量的方法是采用双阈值法。选择两个阈值，根据高阈值得到一个边缘图像，这样一个图像含有很少的假边缘，但是由于阈值较高，产生的图像边缘可能不闭合，为解决此问题采用了另外一个低阈值。对非极大值抑制图像作用两个阈值 th1 和 th2，两者关系 th1=0.4th2。

在高阈值图像中把边缘链接成轮廓，当到达轮廓的端点时，该算法会在断点邻域点中寻找满足低阈值的点，再根据此点收集新的边缘，直到整个图像边缘闭合。

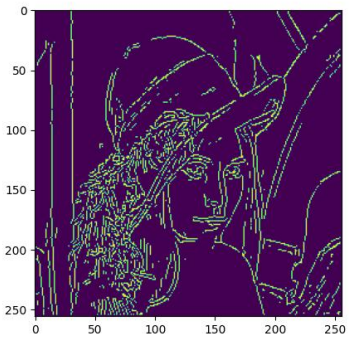
如果某一像素的梯度值高于高阈值，则保留；如果某一像素的梯度值低于低阈值，则舍弃；如果某一像素的梯度值介于高低阈值之间，则从该像素的 8 邻域的寻找像素梯度值，如果存在像素梯度值高于高阈值，则保留，如果没有，则舍弃。

对应代码如下：(不含 bonus，最终的版本会添加一些细节，这里仅仅预留了接口)

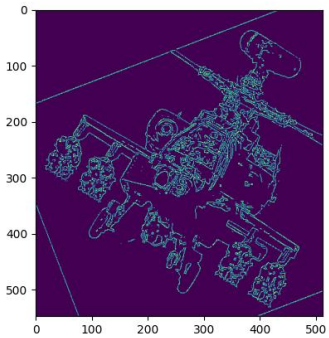
```
# 任务五: 双阈值算法检测和连接边缘
def double_threshold(self, type='Sobel', th=0.4, tl=0.1):
    self.Guass_noise()
    if type == 'Sobel':
        self.Guass_sobel_gradient()
    self.NMS()
    NMS = self._NMS
    width, height = NMS.shape
    DT = np.zeros(NMS.shape)
    # 定义高低阈值
    TH = th * np.max(NMS)
    TL = tl * np.max(NMS)  #这里不能直接用max
    # 作为测试用
    self.TH = TH
    self.TL = TL
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            # 双阈值选取
            if (NMS[i][j] < TL):
                DT[i][j] = 0
            elif (NMS[i][j] > TH):
                DT[i][j] = 1
        # 连接
            elif (NMS[i - 1][j - 1:j + 1] <TH).any() or NMS[i + 1][j - 1:
                j + 1].any() or NMS[i][j - 1] < TH or NMS[i][j + 1] < TH:
                DT[i][j] = 1
    return DT
```

4 结果展示

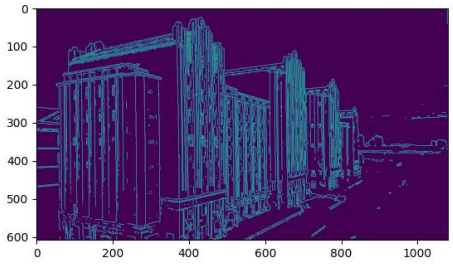
我们写的 canny 得到的边缘如下：



(a) img1

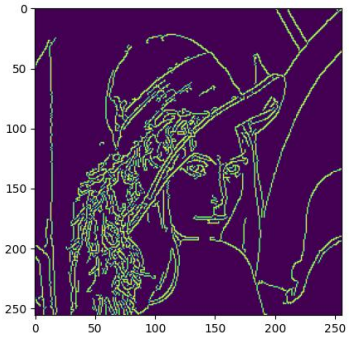


(b) img2

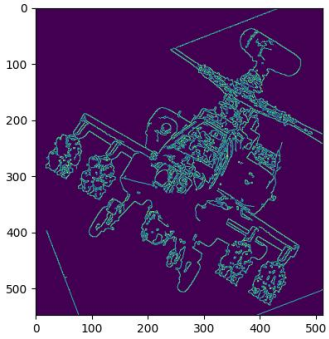


(c) img3

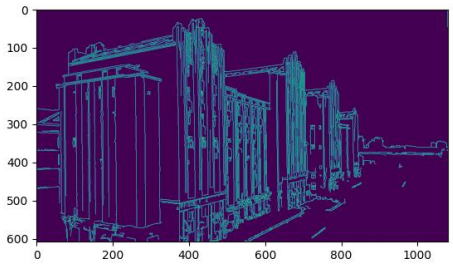
对比内置的 canny 函数获得的边缘：



(d) img1



(e) img2



(f) img3

可以看出结果并非完全一致，效果相比于 canny 函数较差，但总体反映出了图像的轮廓样式。整体而言基本达到预期。

5 问题分析与拓展思考

5.1 一些函数的结构

`any()` 函数是比较这一个`array` 中的全部内容。如果全为假，则返回假，有一为真即返回真。

`canny=double_threshold` 是给函数起一个别名，增强可读性。

`np.max()` 在于系统自带的`max` 函数不支持二维矩阵的最大值，因此我们需要调用`numpy` 的方法。

5.2 边沿点的取值

我们的最高阈值和最低阈值取得是梯度最高值的 0.4 倍和 0.1 倍，经过测试效果较好。我们使用的是动态取值而非静态设置梯度的最大或最小值，这样可以保证最终的结果基本是稳定的，不会存在过大或过小的情况出现。

5.3 高斯滤波的作用

高斯滤波是为了模糊图像之间明显的界限，否则图像会严重收到噪声的影响，最后造成我们提取的边缘严重失真，达不到我们最终希望看到的的结果。

5.4 双阈值算法检测背后的原理

双阈值本质是寻找合适的梯度点，排除梯度过低的情况。然而，当出现梯度值不高不低的情况时，我们就需要去验证是否处于边缘。一个简单的方法是验证周围是否有类似的边缘，因为经过高斯滤波后，梯度函数应当具有较好的连续性。这就是为什么直接查找周围的八个点即可基本确定位置。

6 Bonus: 换几个参数

我们在实验中固定选取了几个参数作为实验的变量。我们可以尝试改动几个参数来验证不同阈值和不同的梯度算法计算出的结果。

6.1 不同的梯度算法

我们在实验中使用了`sobel` 算子，而在 ppt 中介绍了多种计算差分梯度的方法。我们在这里尝试使用不同的算法处理。

6.1.1 Roberts 算子

Roberts 算子的卷积核为

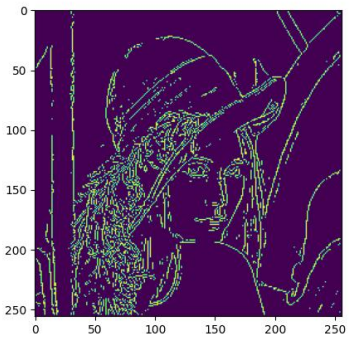
$$s_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad s_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

值得注意的是，`openCV` 也有一个可以直接使用 Roberts 算子的方法，对应代码如下：

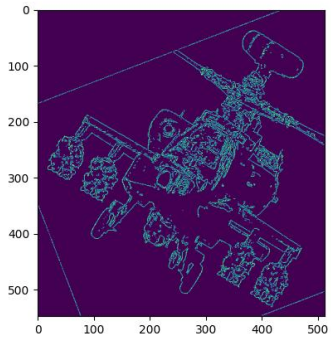
```
# 算子Roberts
def Roberts(self):
    kernelx = np.array([[-1, 0], [0, 1]], dtype=int)
    kernely = np.array([[0, -1], [1, 0]], dtype=int)
    x = cv2.filter2D(self.img, cv2.CV_16S, kernelx)
```

```
    y = cv2.filter2D(self.img, cv2.CV_16S, kernely)
    self.dx = x
    self.dy = y
    # 转转成uint8
    absX = cv2.convertScaleAbs(x)
    absY = cv2.convertScaleAbs(y)
    Roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)
    self.gradient = Roberts
```

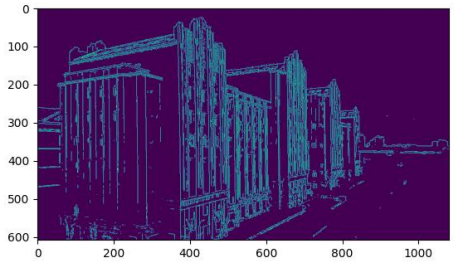
运行结果：



(g) img1



(h) img2



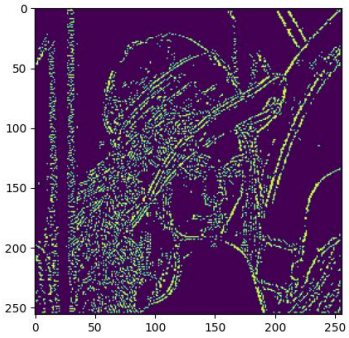
(i) img3

可以看出运行效果更差，噪点更多，这是由于 Roberts 算子的灵敏度较高，对应的结果显得早点较大。

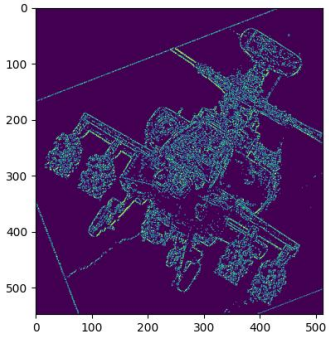
6.1.2 Prewitt 算子

把 Soberts 算子的 2、-2 换成 1、-1，其余不变。。。

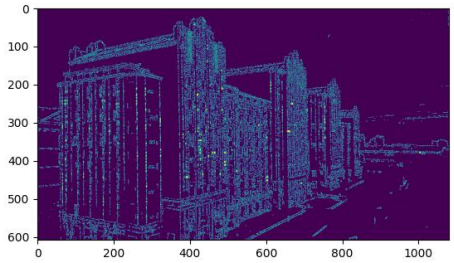
卷积需要的代码与 6.1.1 的代码也很类似，此处不再放出。运行结果：



(j) img1



(k) img2

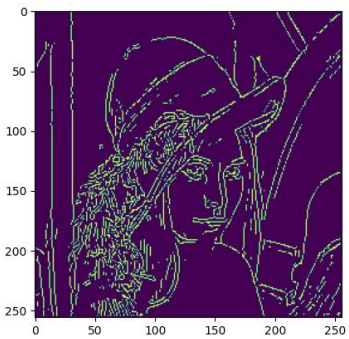


(l) img3

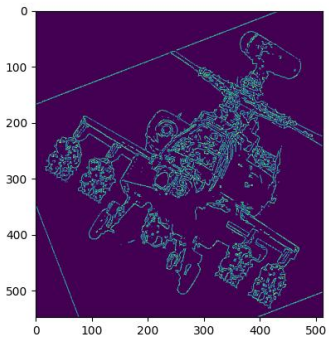
可以看出，Prewitt 的滤波效果并不好。

6.2 替换双阈值算法检测的参数

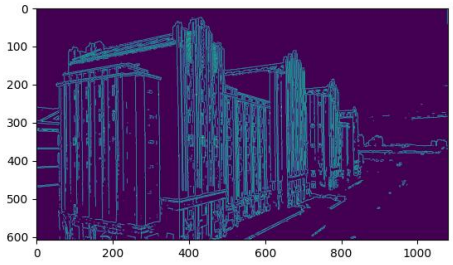
我们将高噪声提高到 0.5，对比发现相较于原来结果较好。



(m) img1



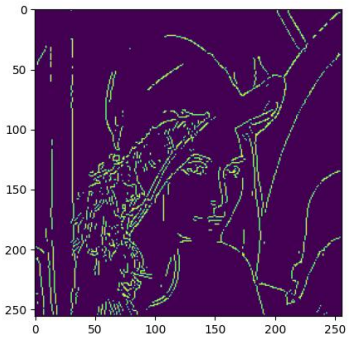
(n) img2



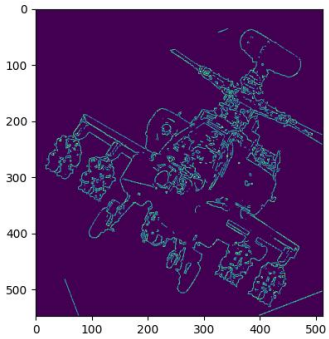
(o) img3

这似乎说明我们初始的取值可能有些偏小。但显然我们不可以将其调整的过大，比如我们将 th 调到 0.9，tl 调到 0.36 的话最终的图片效果会非常差（但对于理科楼这种对比度较为明显的图反而影响不大）。

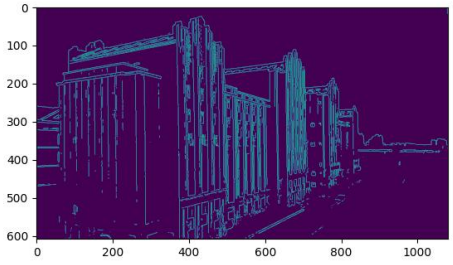
那么我们缩小空间是否有类似效果呢？我们尝试将区间缩小到 0.2-0.4 这一区间中，结果如图。



(p) img1



(q) img2



(r) img3

我们发现线条确实比较严重，这说明区间过小了。

事实上，该部分区间取值的重要意义在于避免出现过大的噪声或者丢失部分信息，这也是我们需要反复调整参数的意义所在。

部分代码参考来源

1. https://www.cnblogs.com/king-lps/p/8007134.html
2. https://cloud.tencent.com/developer/article/1408073
3. https://www.cnblogs.com/zhouzetian/p/13324627.html
4. https://www.runoob.com/python/python-func-any.html