Experiment 11

517030910356 谢知晖

目录

1 实验准备 **2**
1.1 实验环境 . . . 2
1.2 实验目的 . . . 2
1.3 实验原理 . . . 2
1.3.1 卷积运算 . . . 2
1.3.2 灰度化 . . . 3
1.3.3 高斯模糊 . . . 3
1.3.4 梯度计算 . . . 4
1.3.5 非极大值抑制 . . . 5
1.3.6 双阈值检测及连接边缘 . . . 6

2 实验过程 **6**
2.1 "MyCanny" 类的构造 . . . 6
2.2 灰度化 . . . 6
2.3 高斯模糊 . . . 7
2.4 梯度计算 . . . 7
2.5 非极大值抑制 . . . 7
2.6 双阈值检测及连接边缘 . . . 8
2.7 实验扩展: 参考阈值 . . . 8

3 实验结果 **9**
3.1 图一 . . . 9
3.2 图二 . . . 10
3.3 图三 . . . 11
3.4 结果分析 . . . 11

4 实验感想 **11**
4.1 阈值选择 . . . 11
4.2 效率分析 . . . 11
4.3 数值范围 . . . 12

5 源代码 **13**

1 实验准备

1.1 实验环境

本次实验环境依旧为 Ubuntu 平台 (版本号 18.04) 下的 Python(版本号 2.7)。
另外，实验中还用到的 Python 库有:

1. OpenCV(版本号 3.4.3)
2. NumPy(版本号 1.15.4)

1.2 实验目的

本次实验在上一次实验的基础上开展了更加有趣但也更加困难的内容。

实验要求我们复现 Canny 检测的全过程，包括灰度化、高斯模糊、计算梯度、非极大值抑制、双阈值检测及连接边缘五个步骤。每一步骤中所采用的方法对应着边缘检测技术的一些基本操作，而通过自己实现这些方法，我们能够逐步理解边缘检测的原理，并形成对于图像处理的许多宝贵直觉。

1.3 实验原理

边缘检测技术作为图像处理和计算机视觉的基本问题，在近年来一直处在快速的发展当中，也衍生出了深度学习等方法分支，但 Canny 算法仍未过时。它虽然早在 1986 年就被 John F. Canny 提出，却仍作为边缘检测的一种标准算法而被广泛使用。因此，我在这里重点介绍这个经典的算法原理。

正如之前所提到的，Canny 算法的五大步骤为:

1. 灰度化
2. 高斯模糊
3. 计算梯度
4. 非极大值抑制
5. 双阈值检测及连接边缘

其中的中心环节是围绕着梯度进行的。而在介绍这五大步骤之前，我觉得有必要先介绍在实验中多次用到的核心操作--卷积。

1.3.1 卷积运算

卷积本质上是一种数学运算。其离散的定义为:

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau) \tag{1}$$

其实际意义可以归结为一个函数在另一个函数上的加权叠加。如同将一条毛巾卷起来，毛巾在特定位置上的部分得到了重叠，而卷积正类似于这种重叠累加的操作。

这样的特性也让卷积在许多领域得到应用，如在统计学中的加权滑动平均，在信号与系统学中信号经过一个线性系统以后发生的变化。而更著名的例子，则是将卷积运算作为基本

操作的卷积神经网络。但我们本次主要关注的是卷积在数字图像处理中的应用，而不探讨在神经网络中使用卷积操作的基本动机。

数字图像是一个二维的离散信号，对数字图像做卷积操作其实就是利用卷积核（卷积模板）在图像上滑动，将图像点上的像素灰度值与对应的卷积核上的数值相乘，然后将所有相乘后的值相加作为卷积核中间像素对应的图像上像素的灰度值，并最终滑动完所有图像的过程。简单的来说，我们的两个卷积对象卷积对象分别为数字图像中某一区域内的像素以及卷积核。通过两者的卷积以达到特定的目的。

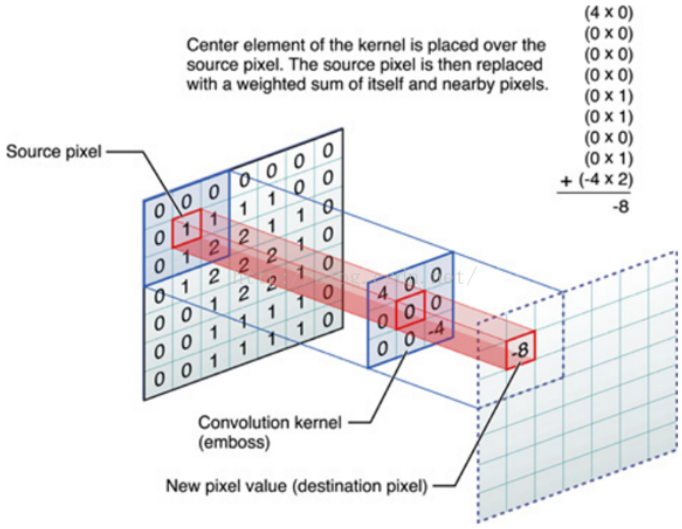


Fig 1: 图像处理中的卷积

需要注意的是，若要保持卷积前后的图像尺寸不变，则需要在卷积前进行边界的 Padding 操作。常见的方法有补零或者拓展边界。

另外，卷积核的选取也尤为重要。为了实现不同的功能，我们需要应用不同的卷积核，以得到不同的特征[1]。

1.3.2 灰度化

边缘检测的实质是采用某种算法来提取出图像中对象与背景间的交界线，而彩色图像由于容易受到光照等因素的影响，并且灰度化可以在不损失精确度的情况下很好地保留其所包含的像素强度信息，我们通常在进行正式的图像识别工作前都会预先对图像进行灰度化。

一般的灰度化方法分为两种:

1. 平均值法:

$$Gray = \frac{R+G+B}{3} \tag{2}$$

2. 加权平均值法:

$$Gray = 0.299R + 0.587G + 0.114B \tag{3}$$

在本次实验中，我采用的方法是加权平均值法，这更符合人眼的观感。

1.3.3 高斯模糊

由于边缘检测很容易受到噪声的影响，这些噪声也集中于高频信号，因此很容易被识别为伪边缘。我们应用高斯模糊技术去除这些潜在的噪声，以降低伪噪声的检测。

[1]具体的例子将在之后介绍高斯模糊时给出

高斯模糊运用了正态分布的密度函数，具体表现为高斯核的确定上。

高斯核将正态分布扩展到了的二维 (实际上适用于所有高维空间)，得到以下计算公式:

$$f(\vec{x}) = \frac{1}{(\sqrt{2\pi}\sigma)^2} e^{-(\vec{x}-\vec{u})^2/2\sigma^2} \tag{4}$$

其中 $\vec{x} = (x, y)$，即卷积核作用区域内任一点的坐标，$\vec{u}$ 为区域中心的坐标。(统计学中称为均值，即坐标的均值)

以尺寸为 3×3 的高斯核为例，对于图像中的任意一点，它周围的坐标为:

(x - 1, y - 1)	(x, y - 1)	(x + 1, y - 1)
(x - 1, y)	(x, y)	(x + 1, y)
(x - 1, y + 1)	(x, y + 1)	(x + 1, y + 1)

则每点的 $(\vec{x} - \vec{\mu})^2$ 为:

2	1	2
1	0	1
2	1	2

若取 $\sigma = 1$，则根据式 (4) 并经过归一化可得到最终的高斯核:

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

接下来的模糊操作即为简单的卷积。

可以看出，高斯模糊的实现是十分简单的，我们需要确定的参数仅有高斯核的大小以及标准差 σ。但高斯模糊的功能也是十分强大的。通过高斯模糊，我们能够有效地缓和图像低频到高频的变化，同时不丢失图像中局部的变化趋势，即 Canny 算法中的"Good Localization"。这使得高斯模糊能够有效地去噪，达到滤波的目的。

1.3.4 梯度计算

梯度计算的核心也是卷积操作，原理并不复杂，我们所主要关注的同样是核的选取。

在梯度计算中用到的卷积核通常被称为边缘检测算子。常见的边缘检测算子有"Sobel"、"Prewitt"、"Canny" 等。

以最简单的"Canny" 算子为例，其 x 方向上的算子为:

$$s_x = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \tag{5}$$

y 方向上的算子为:

$$s_y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \tag{6}$$

可以直观的看出，s_x 对应着 x 方向的变化。如果该像素点右边的灰度值较左边的大，则表现为梯度上的正值，反之则为负值。并且两者的差值越大，s_x 的绝对值也就越大，这样就反映出了图像灰度值的变化情况。

于是，对于图像上任意一点，若 $P[i,j]$ 为该点 x 方向梯度，$Q[i,j]$ 为该点 y 轴方向梯度，则梯度幅值很自然地被定义为了:

$$M[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2} \tag{7}$$

而梯度方向为:

$$\theta[i,j] = \arctan(Q[i,j]/p[i,j]) \tag{8}$$

这样，图像的边缘特性得到提取。我们已经得到了最初步的边缘分布。

1.3.5 非极大值抑制

非极大值抑制是一种边缘稀疏技术，其主要作用在于" 瘦" 边。

对图像进行梯度计算后，仅仅基于梯度值提取的边缘仍然很模糊。其原因在于在物体的边缘可能有很多像素的梯度值都处在高位，使得边缘看起来很粗。而我们不希望得到这样的结果。我们预期的是用一个准确的相应来描述边缘的具体位置，这就需要用到非极大值抑制来将局部极大值之外的点剔除掉。

算法关注每个像素点的邻域，通过判断该像素点是否为邻域内梯度幅值极大来屏蔽掉非边缘点。具体而言，我们需要判断在该梯度方向上的邻域内是否有比更大的梯度幅值点。

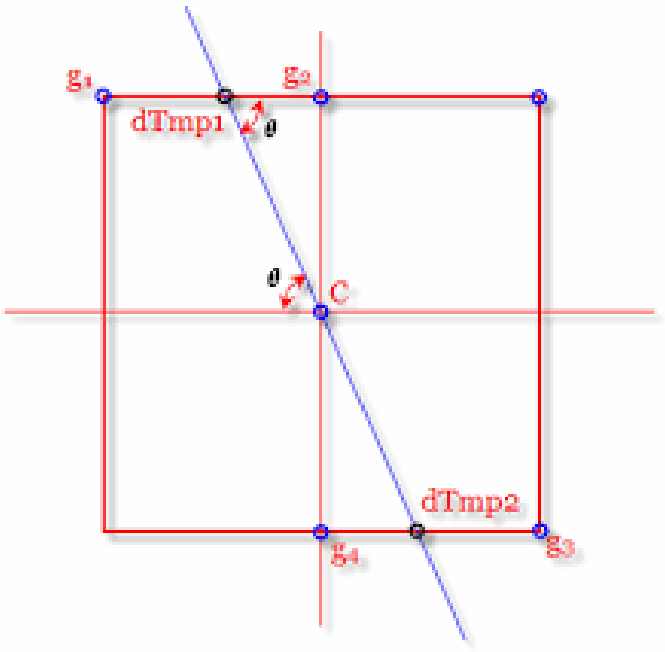


Fig 2: 局部最大值判断

由于图像的像素点是非连续的，我们有必要通过插值来得到所需的邻域点，公式为:

$$dTmp1 = g1 * weight + g2 * (1 - weight) \tag{9}$$

$$dTmp2 = g3 * weight + g4 * (1 - weight) \tag{10}$$

其中的 weight 可由梯度方向得到。如果中心梯度幅值不是这三点中的极大值，则可直接抑制为 0，否则保留。

1.3.6 双阈值检测及连接边缘

在完成非极大值抑制之后，我们得到了去除大量伪边缘的结果，但还是会有许多由于噪声或颜色变化引起的小的梯度值，对结果产生不利影响。

一般的边缘检测算法用一个阈值来滤除这些噪声，而保留大的梯度值。但 Canny 算法应用双阈值，即一个高阈值和一个低阈值来区分边缘像素。如果边缘像素点梯度值大于高阈值，则被认为是强边缘点。如果边缘梯度值小于高阈值，大于低阈值，则标记为弱边缘点。小于低阈值的点则被抑制掉。

为得到精确的结果，噪声或颜色变化引起的弱边缘点应该去掉。我们通常可以通过设计深度优先算法来实现。算法搜索所有连通的弱边缘，如果一条连通的弱边缘的任何一个点和强边缘点连通，则保留这条弱边缘，否则抑制这条弱边缘。

2 实验过程

根据之前的实验原理，通过实现这五个步骤，我们应该能够得到一个比较好的边缘检测算法。下面我逐步介绍我的实现算法，而各步的实验结果将在最后加以讨论。

2.1 "MyCanny" 类的构造

首先我定义了一个类"MyCanny" 来实现我们的检测。它的构造函数接受 6 个参数，其中只有读入的彩色图片'image' 没有默认值，其他的 5 个参数，包括梯度算子类型'kernel'，所选取的高斯模糊参数'gauss_size'、'gauss_sigma'，以及高低阈值'low_th'、'high_th'，都设有默认值。

其具体的构造函数如下:

```
def __init__(self, image, kernel='canny',
             gauss_size=(3, 3), gauss_sigma=0.5, low_th=20, high_th=30):
    self.__img_ori = image
    self.__x = image.shape[0]
    self.__y = image.shape[1]
    self.__low = low_th
    self.__high = high_th
    self.__kernel = kernel
    self.__g_size = gauss_size
    self.__g_sigma = gauss_sigma

    self.__graying()
    self.__gaussian_blur()
    self.__get_grad()
```

2.2 灰度化

本步骤实现函数为"graying"。

作为图片的预处理工作，灰度化的实现并不复杂。对于读入的彩色图像，我通过加权平均算法，即 $Gray = 0.299R + 0.587G + 0.114B$ 对图像进行处理，得到灰度图。

```
def __graying(self):
    self.__img = np.dot(self.__img_ori[..., :3], [0.114, 0.587, 0.299])
    self.__img = cv2.convertScaleAbs(self.__img)
```

这里我通过 OpenCV 的"convertScaleAbs" 方法对浮点数进行自动地类型转换，在之后的步骤中我也会多次使用到这个函数来调节数据。得到的灰度图存储在另一个私有成员变量"img" 中。

2.3 高斯模糊

本步骤实现函数为"gaussian_blur"。

由于 OpenCV 中带有高斯模糊函数，我可以很方便地调用得到去除潜在噪声的图像。函数"GaussianBlur" 的参数为我们之前接受的用户参数。

```
def __gaussian_blur(self):
    self.__img_gauss = cv2.GaussianBlur(self.__img, self.__g_size, self.__g_sigma)
```

2.4 梯度计算

本步骤实现函数为"get_grad"。

梯度的计算较前两步而言稍许复杂，但好在同样可以直接调用 OpenCV 的函数"filter2D"，省去了不少计算上的麻烦。

首先，对于不同的梯度算子，我定义好它们在两个方向上的卷积核，并直接调用"filter2D" 计算两个方向上的梯度。以 Canny 算子为例:

```
if self.__kernel == 'sobel':
    kernel_x = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])

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

    x_der = cv2.filter2D(gauss_float, -1, kernel_x)
    y_der = cv2.filter2D(gauss_float, -1, kernel_y)
```

值得一提的是，"filter2D" 的第二个参数 “ddepth" 我设置为-1，这保证卷积前后图像的尺寸不发生改变，显然其中存在默认的 Padding 操作，而由于我们并不关心边界的情况，具体是哪种操作也无需探讨。

接着，我根据梯度的两个分量计算梯度幅值以及方向。由于在之后直接用到的是角度的正切值，我仅计算出 y 方向梯度与 x 方向梯度的比值。而为了避免除零的情况发生，除式的分母被加上了一个极小正数。

```
self.__grad_mag = np.power(np.add(np.power(x_der, 2), np.power(y_der, 2)), 0.5)
self.__grad_mag = cv2.convertScaleAbs(self.__grad_mag)
self.__grad_dir_tan = np.divide(y_der, x_der + 0.0000001)
```

2.5 非极大值抑制

本步骤实现函数为"non_max_suppress"。

根据非极大值抑制原理，我们对于每一个像素的梯度方向进行判断。分别计算出应用插值算法的邻域点的梯度值，与中心位置的梯度进行比较。

我将这一步的结果保存在变量"suppress" 中。变量首先复制"img" 的灰度，然后在每次遇到非极大值时将该像素的灰度置零。

该部分的代码细节将附在报告最后。

2.6 双阈值检测及连接边缘

本步骤实现函数为"link_all"。

本步骤是实验中的关键步骤，通过这一步我们能够对之前孤立点进行清除。

边缘连接直接的实现思想是通过寻找连线断点，并从这个断点出发拓展我们的线条。由于断点位置未知，我们需要遍历图像的所有像素点。但我们不想重复对之前探索过的路线进行重复探索，因此我设置了一个矩阵存储已被探索过的高阈值点。同时，我用 preserve 矩阵来存储最终保留的点的位置。

```
visited = np.zeros((self.__x, self.__y))
preserve = np.zeros((self.__x, self.__y))
```

接下来便是遍历过程。对于 (i, j) 位置上的点，我首先判断它的梯度值是否高于阈值，如果高于，则将这个点设为保留点。然后，如果这个点为高阈值点或者在之前的探索中被设为保留点，并且它并没有被探索过，我们便可以在这个点周围进行探索。

具体而言，我使用的探索算法是广度优先的算法。首先对于上述没探索过的点，我将它的坐标添加到 visited 当中，并且建立一个列表 to_link 用作队列去存储之后需要搜索的点，(i, j) 是队列中最开始的元素。我每次从队列头拿出一个元素，并对其周围的八个邻域点 (i + m, j + n) 进行检测，如果 (i + m, j + n) 的梯度值大于低阈值，并且它没有被探索过，则把这个位置设为保留点，并将它标记为 visited，加入到 to_link 当中。

最终，通过矩阵 elementwise 的乘法，我得到了最终的结果。具体的代码附在报告最后。

```
self.__img_final = self.__suppress * preserve
```

2.7 实验扩展: 参考阈值

作为实验的扩展内容，我对于阈值的选取进行了进一步分析。

首先，不难发现在应用不同算子时梯度的大小是十分不同的，我们通常在更换算子时很难直接寻找到合适的阈值。为此，根据统计学的思想，我抛开了算子的差异性，设计了一个简单的参考阈值算法。

该算法是通过计算梯度直方图来实现寻找阈值的目的的。显然，应用双阈值的一大基础前提便是这两个阈值能够帮助我们过滤掉相当比例的低梯度点，这很符合统计学的原理。于是我将高阈值的选取定为梯度直方图的前 90%，即高阈值能够过滤掉 90% 的点，而低阈值设为高阈值的一半。

```
def get_adaptive_th(self):
    max_grad = np.max(self.__grad_mag)
    hist = np.histogram(self.__grad_mag, bins=range(max_grad))[0]
    mag_sum = np.sum(hist)
    high_level = mag_sum * 0.9
    count = 0
    high_th = 0

    for i in range(256):
        if count >= high_level:
```

```
            break

        else:
            count += hist[i]
            high_th = i

    return high_th, high_th / 2
```

3 实验结果

我应用三种算子在不同阈值下进行了实验，得到了以下结果。标准图选用

3.1 图一

标准图

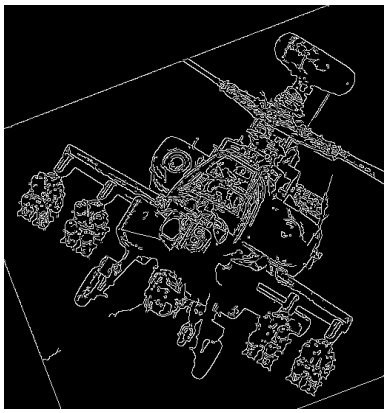
Prewitt 算子

Canny 算子

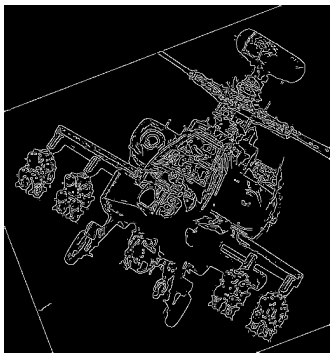
Sobel 算子

Fig3: 图一实验结果

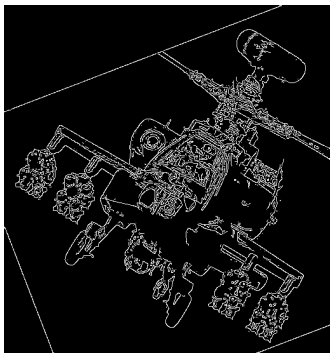
3.2 图二



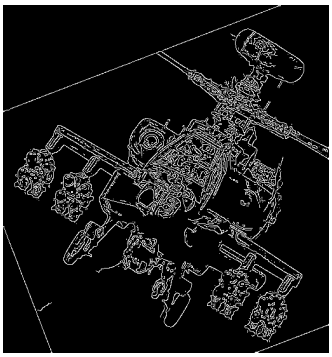
标准图



Prewitt 算子



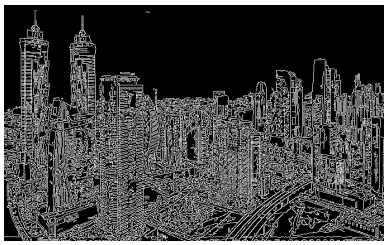
Canny 算子



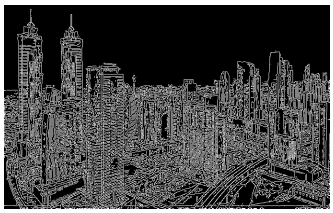
Sobel 算子

Fig4: 图二实验结果

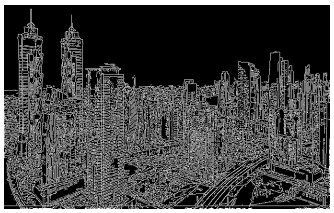
3.3 图三



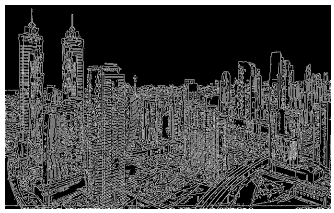
标准图



Prewitt 算子



Canny 算子



Sobel 算子

Fig4: 图三实验结果

3.4 结果分析

可以发现，我的实现结果还是比较令人满意的。不过，图像上仍然有少量散点没有被去除干净，这说明我的实现方法还有能够完善之处。

就三种算子而言，Sobel 的结果往往最好，而 Canny 和 Prewitt 算子存在一定的不稳定性，对于梯度的变化有时无法很好地体现，进而造成噪声和真边缘的混淆。

4 实验感想

本次实验的流程较多，处理也较为繁琐，我在实现过程中遇到了不少问题。下面我将对我遇到的几个主要困难进行讨论。

4.1 阈值选择

在上文中我提到了我的参考阈值算法。这样得出的阈值一般能够适应不同的算子，且相较于对梯度进行标准化实现也更为简单。

但值得注意的是，该技巧仅仅能够满足寻找阈值大致范围的功能，并不能得到精确的合理取值。这一方面归因于我的算法的粗劣，一方面也体现了图像识别的困难之处。

4.2 效率分析

在进行步骤" 双阈值检测及连接边缘" 时，我的早期版本程序执行时间极久，经常花费几分钟才能处理完成。分析以后，由于图像的像素只有有限个，且元素被探索过后即被标记，我断定问题并不出在算法上。

经检查，我发现我使用的数据结构存在很大问题。之前我是通过在 list 中存储坐标元组来实现探索过元素的标记，而由于 Python 的 list 是通过数组实现的，每次检查都需要从头遍历，时间复杂度为 O(n)，这样的效率显然是极其低下的。

为此，我将原本使用的 list 改为 Numpy 中的矩阵，通过标记矩阵对应元素来实现 O(1) 的时间复杂度。最终的运行时间也确实大大缩小，每次处理仅用时 5s 左右。

4.3 数值范围

在实验中，我曾经遇到无论怎么修改阈值，甚至将高阈值调到 250 都对结果没有变化的情况。

经分析，这是因为在计算梯度时数值已经超过 255，而矩阵中使用的数据类型是 uint8，这就导致了很多点的梯度值集中在 255，从而造成了误差。

为此，我在梯度计算时额外缩小了相应的倍数，使其保持在 255 以内。

5 源代码

```
import numpy as np
import cv2


class MyCanny(object):
    def __init__(self, image, kernel='canny',
                 gauss_size=(3, 3), gauss_sigma=0.5, low_th=20, high_th=30):
        self.__img_ori = image
        self.__x = image.shape[0]
        self.__y = image.shape[1]
        self.__low = low_th
        self.__high = high_th
        self.__kernel = kernel
        self.__g_size = gauss_size
        self.__g_sigma = gauss_sigma

        self.__graying()
        self.__gaussian_blur()
        self.__get_grad()

    def __graying(self):
        self.__img = np.dot(self.__img_ori[..., :3], [0.114, 0.587, 0.299])
        self.__img = cv2.convertScaleAbs(self.__img)

    def __gaussian_blur(self):
        self.__img_gauss = cv2.GaussianBlur(self.__img, self.__g_size, self.__g_sigma)

    def __get_grad(self):
        gauss_float = self.__img_gauss.astype(float)

        if self.__kernel == 'canny':
            kernel_x = np.array([[-1, 1],
                                 [-1, 1]])
            kernel_y = np.array([[1, 1],
                                 [-1, -1]])

            x_der = cv2.filter2D(gauss_float, -1, kernel_x) / 2
            y_der = cv2.filter2D(gauss_float, -1, kernel_y) / 2

        if self.__kernel == 'sobel':
            kernel_x = np.array([[-1, 0, 1],
                                 [-2, 0, 2],
                                 [-1, 0, 1]])

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

            x_der = cv2.filter2D(gauss_float, -1, kernel_x) / 4
```

```
            y_der = cv2.filter2D(gauss_float, -1, kernel_y) / 4

        if self.__kernel == 'prewitt':
            kernel_x = np.array([[-1, 0, 1],
                                 [-1, 0, 1],
                                 [-1, 0, 1]])

            kernel_y = np.array([[1, 1, 1],
                                 [0, 0, 0],
                                 [-1, -1, -1]])

            x_der = cv2.filter2D(gauss_float, -1, kernel_x) / 3
            y_der = cv2.filter2D(gauss_float, -1, kernel_y) / 3

        self.__grad_mag = np.power(np.add(np.power(x_der, 2), np.power(y_der, 2)), 0.5)
        self.__grad_mag = cv2.convertScaleAbs(self.__grad_mag)
        self.__grad_dir_tan = np.divide(y_der, x_der + 0.0000001)

    def get_adaptive_th(self):
        max_grad = np.max(self.__grad_mag)
        hist = np.histogram(self.__grad_mag, bins=range(max_grad))[0]
        mag_sum = np.sum(hist)
        high_level = mag_sum * 0.9
        count = 0
        high_th = 0

        for i in range(256):
            if count >= high_level:
                break

            else:
                count += hist[i]
                high_th = i

        return high_th, high_th / 3

    def __non_max_suppress(self):
        self.__suppress = self.__img * 1

        for i in range(1, self.__x - 1):
            for j in range(1, self.__y - 1):
                tan_grad = self.__grad_dir_tan[i, j]

                if tan_grad >= 1:
                    dtmp1 = (1 - 1 / tan_grad) * self.__grad_mag[i - 1, j] + 1 / tan_grad *
                                                                             self.__grad_mag[i - 1, j +
                                                                             1]
                    dtmp2 = (1 - 1 / tan_grad) * self.__grad_mag[i + 1, j] + 1 / tan_grad *
                                                                             self.__grad_mag[i + 1, j -
                                                                             1]


                elif 0 <= tan_grad < 1:
```

```
                dtmp1 = (1 - tan_grad) * self.__grad_mag[i, j - 1] + tan_grad * self.
                                                        __grad_mag[i + 1, j - 1]
                dtmp2 = (1 - tan_grad) * self.__grad_mag[i, j + 1] + tan_grad * self.
                                                        __grad_mag[i - 1, j + 1]

            elif -1 <= tan_grad < 0:
                dtmp1 = (1 + tan_grad) * self.__grad_mag[i, j + 1] - tan_grad * self.
                                                        __grad_mag[i + 1, j + 1]
                dtmp2 = (1 + tan_grad) * self.__grad_mag[i, j - 1] - tan_grad * self.
                                                        __grad_mag[i - 1, j - 1]

            else:
                dtmp1 = (1 + 1 / tan_grad) * self.__grad_mag[i - 1, j] - 1 / tan_grad *
                                                        self.__grad_mag[i - 1, j -
                                                        1]
                dtmp2 = (1 + 1 / tan_grad) * self.__grad_mag[i + 1, j] - 1 / tan_grad *
                                                        self.__grad_mag[i + 1, j +
                                                        1]

            if self.__grad_mag[i, j] < dtmp1 or self.__grad_mag[i, j] < dtmp2:
                self.__suppress[i, j] = 0

    def __get_high_low(self):
        self.__img_high = self.__img * (self.__grad_mag > self.__high)
        self.__img_low = self.__img * (self.__grad_mag < self.__low)

    def __link_all(self):
        visited = np.zeros((self.__x, self.__y))
        preserve = np.zeros((self.__x, self.__y))

        for i in range(1, self.__x - 1):
            for j in range(1, self.__y - 1):
                if self.__grad_mag[i, j] > self.__high:
                    preserve[i, j] = 1

                if preserve[i, j] == 1 and visited[i, j] != 1:
                    visited[i, j] = 1
                    to_link = [(i, j)]

                    while to_link:
                        (x, y) = to_link.pop(0)

                        if 0 < x < self.__x - 1 and 0 < y < self.__y - 1:
                            for (m, n) in [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1
                                                                    ), (1, 0), (1, 1)]:
                                if self.__grad_mag[x + m, y + n] > self.__low and visited[x +
                                                                        m, y + n] != 1:
                                    to_link.append((x + m, y + n))
                                    preserve[x + m, y + n] = 1
                                    visited[x + m, y + n] = 1

        self.__img_final = self.__suppress * preserve
```

```
    def set_th(self, high, low):
        self.__high = high
        self.__low = low

    def get_result(self):
        self.__graying()
        self.__gaussian_blur()
        self.__get_grad()

        self.__get_high_low()
        self.__non_max_suppress()
        self.__link_all()

        cv2.imshow("final", self.__img_final)
        cv2.imshow("Image", cv2.Canny(self.__img, 50, 150))

        k = cv2.waitKey(0)
        if k == 'q':
            cv2.destroyAllWindows()

        return self.__img_final


def main():
    img = cv2.imread("dataset/1.jpg")

    my_canny = MyCanny(img, kernel='prewitt', high_th=85, low_th=11)
    print(my_canny.get_adaptive_th())

    my_canny.get_result()


if __name__ == '__main__':
    main()
```

参考文献

[1] 卷积: https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF

[2] 高斯模糊: https://www.zhihu.com/question/54918332