《计算机视觉与图形图像处理》

Programming Assignment 1: 图像边缘检测和霍夫变换

南京农业大学人工智能学院 黄君贤

截止日期：2024年11月22日 11:00PM

注：以下作业可以使用 numpy、scipy基础科学计算库完成，除特殊说明，**不能直接使用OpenCV等图像处理库的高阶API**

1. 边缘检测
2. 卷积滤波

编写函数，使用给定的滤波算子对图像进行卷积运算

def my\_image\_filter(img, h)

传入参数img为黑白灰度图像数据（可以是numpy数组），h为图像滤波算子矩阵（可以是numpy数组）

返回经过卷积计算后的图像数据，需要和原始图像同样的大小

注意：为了保证输出图像和输入图像尺寸保持一致，在运算之前需要对原始图像增加边缘像素，扩充的边缘像素一般使用原始图像的边缘像素值，扩充的边缘大小需要根据输入滤波矩阵大小动态调整

1. 边缘检测

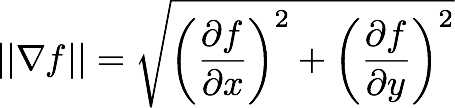
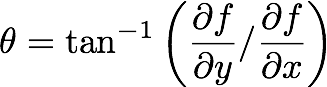
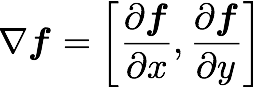
第一步，使用高斯滤波算子对图像进行平滑处理，该步骤一般作为边缘检测的前置操作用于降低噪点和弱化图像中的虚假边缘，关于高斯滤波算子的获取，可以使用如下的辅助函数：

def gauss2d\_smooth\_filter(shape=(3, 3), sigma=0.5):  
 m, n = [(ss - 1.) / 2. for ss in shape]  
 y, x = np.ogrid[-m:m + 1, -n:n + 1]  
 h = np.exp(-(x \* x + y \* y) / (2. \* sigma \* sigma))  
 h[h < np.finfo(h.dtype).eps \* h.max()] = 0  
 sumh = h.sum()  
 if sumh != 0:  
 h /= sumh  
 return h

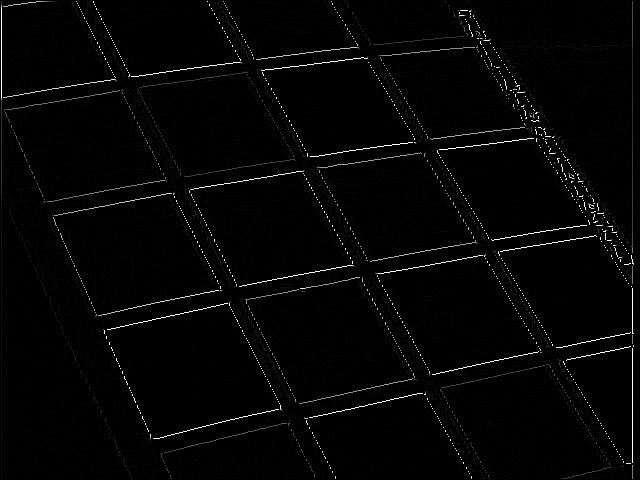
shape用于限定输出矩阵大小，此处可以做如下限制

shape = 2 \* ceil(3 \* sigma) + 1)

第二步，使用Sobel算子对第一步的输出图像进行边缘检测，你需要对x和y轴的边缘检测结果进行整合



第三步，观察第二步的输出结果，边缘线条比较粗，一般需要对边缘线条图像进行非极大值抑制(Non-Maximum Suppression, NMS)处理，即只保留局部最大值，关于NMS的介绍请参阅文档最后的介绍或者上网查阅相关资料，需要注意，此处NMS需要使用到第二步所示公式中的theta值，此处只比较关联角度上的邻居值，最终的结果应当类似下图所示，请保留该图像边缘结果，用于第二部分的直线检测



1. 霍夫变换
2. 编写函数，对图像边缘检测的结果做霍夫变换，此处只考虑直线检测

def hough\_line(img, value\_threshold=5)

传入参数img为图像数据，value\_threshold用于表示超过一定数值的像素认定为边缘像素点

返回三个参数

* hspace，二维数组，表示霍夫变换后的投票（累加）结果
* thetas，角度数据数组，采用弧度制(不限定数值全为正数)
* rhos，原点到线的“距离”ρ数据数组(不限定数值全为正数)

1. 编写函数，找出上一步霍夫变换结果中的峰值数据作为检测出的候选直线属性信息

def hough\_line\_peaks(hspace, thetas, rhos, threshold=None)

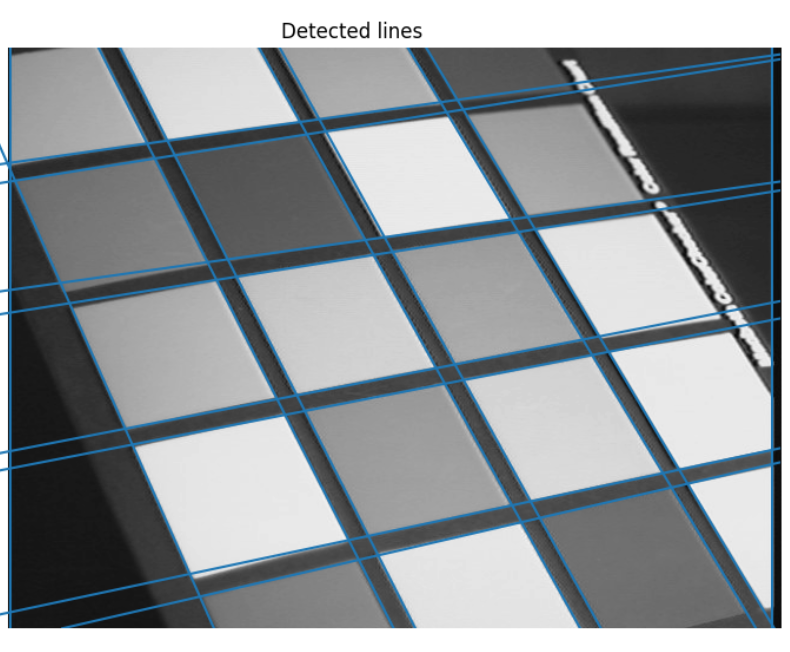
传入参数前三个对应上一步的计算结果，threshold用于限制最低投票（累加）要求

返回参数thetas, rhos表示最终的筛选结果

注：在筛选过程中应当考虑使用NMS，仅保留满足条件的最佳直线信息，此处NMS应当对一块区域的所有邻居进行比较取最大值

1. 将候选直线绘制于原图（或边缘检测图）中

最终结果应当类似于如下图所示



1. 将同样的算法应用于数据文件夹中的另一张图片，调整相关计算参数观察其对最终结果的影响，从而体会相关参数的意义所在

提交内容：

1、code文件夹中补全代码。

2、新建result文件夹，将边缘检测结果保存在文件夹内。

3、将霍夫变换的直线显示在原图片上，将显示直线的图像保存在result文件夹内。

4、将上述文件打包，命名为StudentName\_StudentID\_PA1.rar 或 .zip

NMS介绍

Non-maximum suppression (NMS) is an algorithm used to find local maxima using the property that the value of a local maximum is greater than its neighbors. To implement the NMS in 2D image, you can move a 3×3 (or 7×7, etc.) filter over the image. At every pixel, the filter suppresses the value of the center pixel (by setting its value to 0) if its value is not greater than the value of the neighbors (in the original image). If you find that your lines are very fat, this method can be used to thin out the line. However, note that it does not take into account the direction of the gradient.

To use NMS for edge thinning which preserves lines, you should compare the gradient magnitude of the center pixel with the neighbors along the gradient direction instead of all the neighbors. Remember that the gradient is perpendicular to the line in the image (that was the definition of an image gradient). To simplify the implementation, you can quantize the gradient direction into one of 0◦, 45◦, 90◦ , and 135◦, and compare the center pixel with two of the 8 neighbors in the 3×3 window, according to the gradient direction. The value of a center pixel is set to 0 if its gradient magnitude is less than either one or both of its neighbors’ values, along the gradient direction.

For example, if the gradient direction of a pixel is 45◦ (i.e., the image edge is oriented at 135◦), we compare its gradient magnitude with the north-east and south-west neighbors and suppress its magnitude if it’s not greater than these two neighbors. When the gradient direction at a pixel is 90◦, we compare the pixel to its north and south neighbors and suppress its magnitude if it’s not greater than these two neighbors. The same logic applies for a gradient direction of 0◦ and 135◦.