桂林电子科技大学

**实验3 图像锐化**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **图像锐化** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计算机科学与技术 | | |
| 学 号 | 2000301320 | | | 姓名 | | 靳志凌 | | |
| 实验日期 | 2022 | 年 | 11 | | 月 | | 11 | 日 |
|  |  | | | | | | | |

## 一、实验目的

掌握图像锐化算法的基本原理。能根据提供的图像数据，利用图像锐化算法的基本原理，对图像进行锐化处理。

## 二、实验内容与要求

1. 实现图像锐化的算法。
2. 对给定的图像分别用一阶梯度算子（罗伯特算子、索贝尔算子和普瑞维特算子）和二阶梯度算子（拉普拉斯算子）进行边缘提取，进行比较。
3. 调出MATLAB自带罗伯特算子和拉普拉斯算子函数，与你编写代码的效果进行比较。

## 三、实验环境

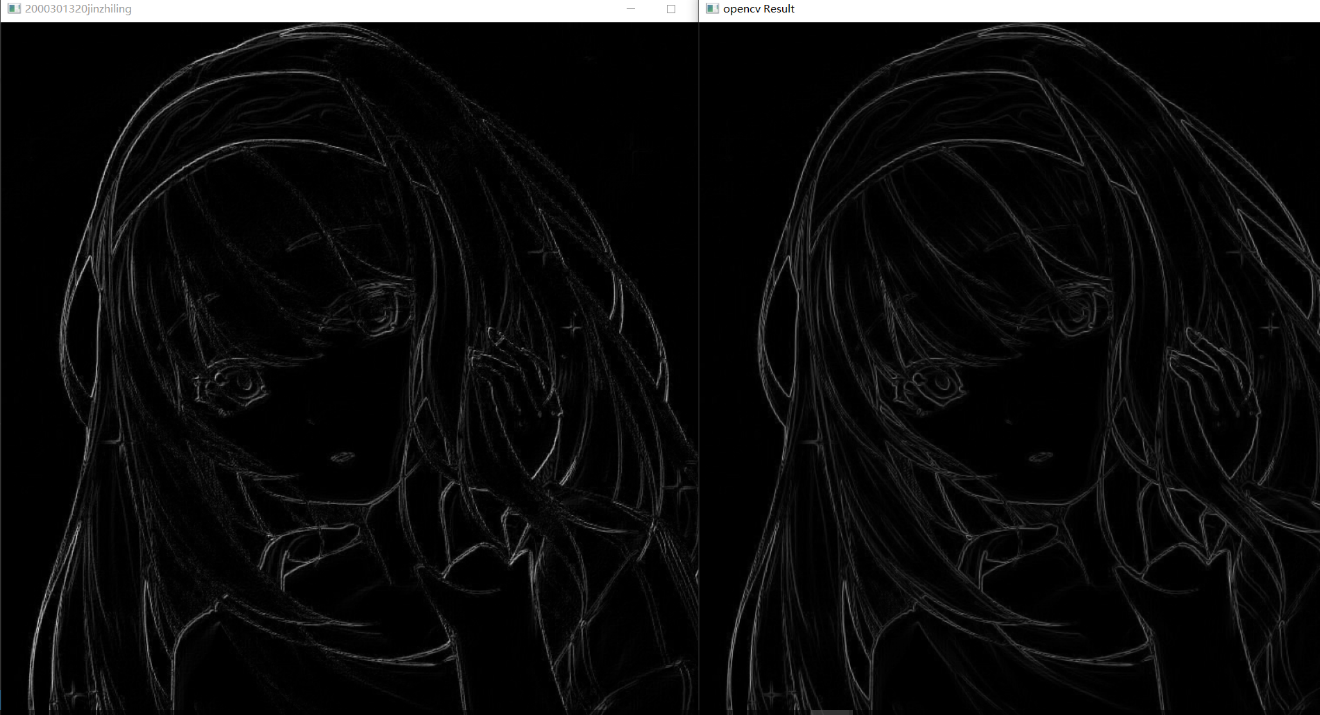
在实验室进行实验，学生可选Matlab、Python等等语言实现。

## 四、实验步骤

1. 描述算法的原理或实现流程

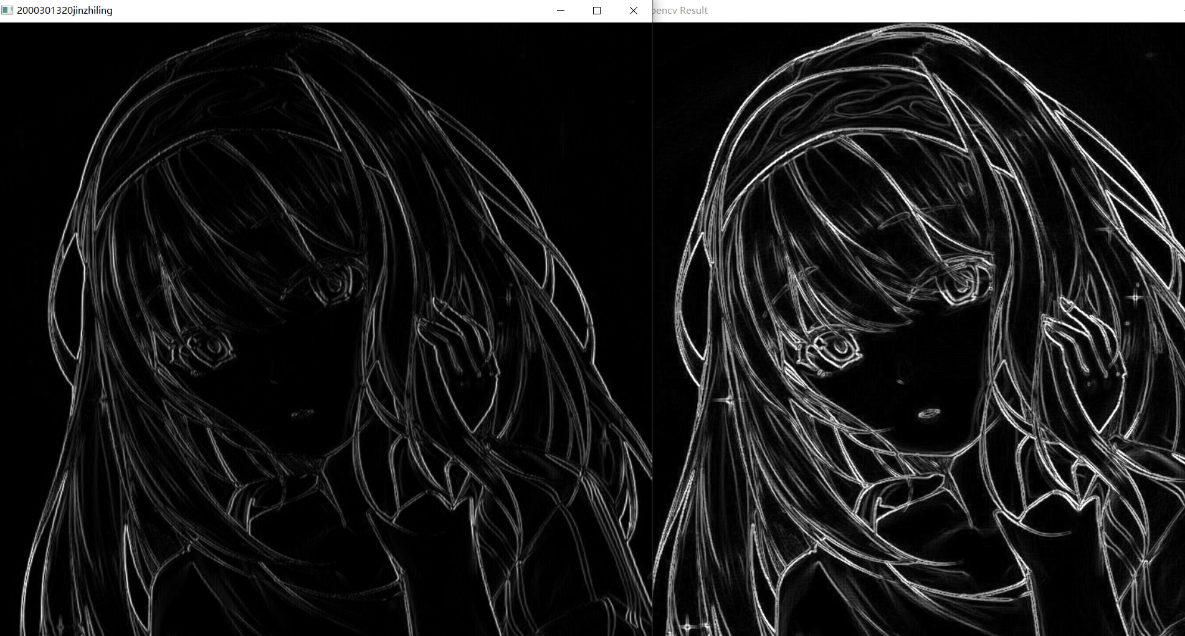
图像锐化目的是为了使模糊的图像变得清晰。既然是与图像平滑相反的目的，那么操作应该也是相反的，平滑使用的是图像邻域的加权求和或者是积分运算，而锐化则是使用求导数(梯度)或有限差分来实现。

之前图像平滑的目的是为了消除噪声而不是突出边缘，所以各种各样的填充方法都可以尝试，影响不大；但是现在是为了突出图像中有意义的边缘，所以当我们在边界是做全0填充时，很有可能填充的部分和图像的边界就形成了变化剧烈的边缘，进而较大的梯度幅值在我们调整灰度范围时，将真正的图像中有意义的边缘的数值范围压缩到很小的范围，导致图像不清晰。所以最好要使用重复填充或者是镜像填充。注意一下，因为我们的模板中含有负数，而图像中的像素点的数值类型是uint8，在相乘时很可能会发生截断。所以首先要确保先将图像中的像素点的数值类型在与模板相乘时转化成整数型。然后计算完的数值，有可能大于0也有可能小于0，我们关心的是绝对值的大小所以可以取绝对值。最后需要将数值的范围调整到0-255之间即可。

调用编写的roberts算子和opencv库中的函数之后得到的效果图如下：

由效果图可以看出，roberts算子产生较宽的边缘。

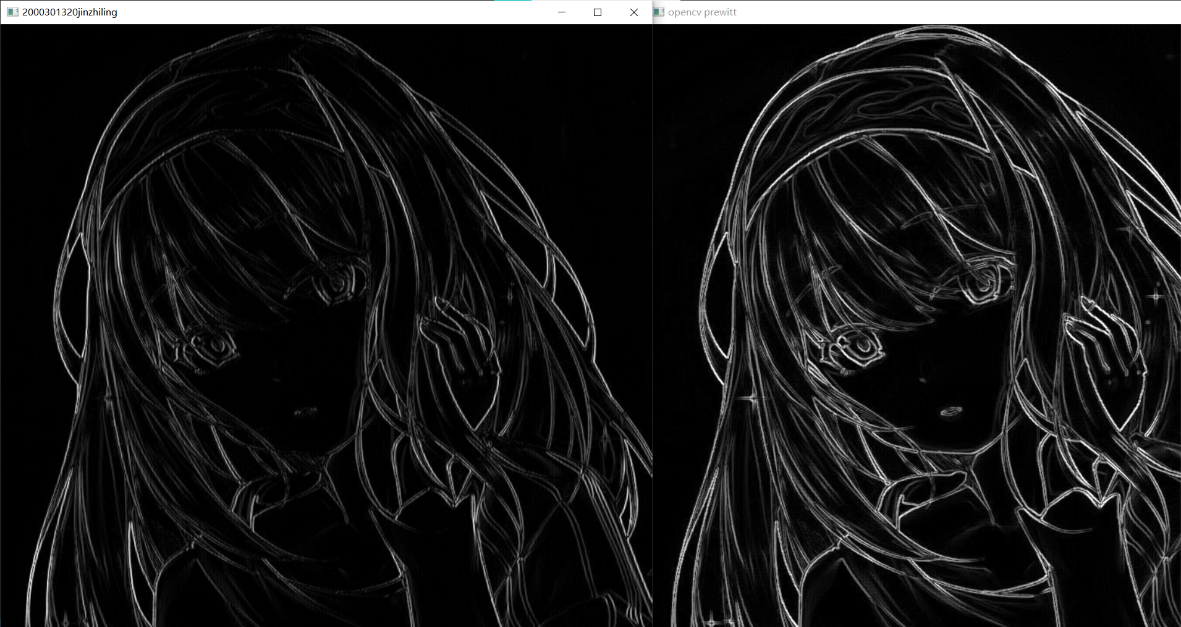
sobel算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导。该算子用于计算图像明暗程度近似值，根据图像边缘旁边明暗程度把该区域内超过某个数的特定点记为边缘。Sobel算子在Prewitt算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓。sobel算子的边缘定位更准确，常用于噪声较多、灰度渐变的图像。

调用sobel算子对侧重于对竖直方向上的边缘进行突出显示，又调用opencv库的sobel函数对图像进行了水平和竖直方向进行了边缘加强处理后的效果如下：

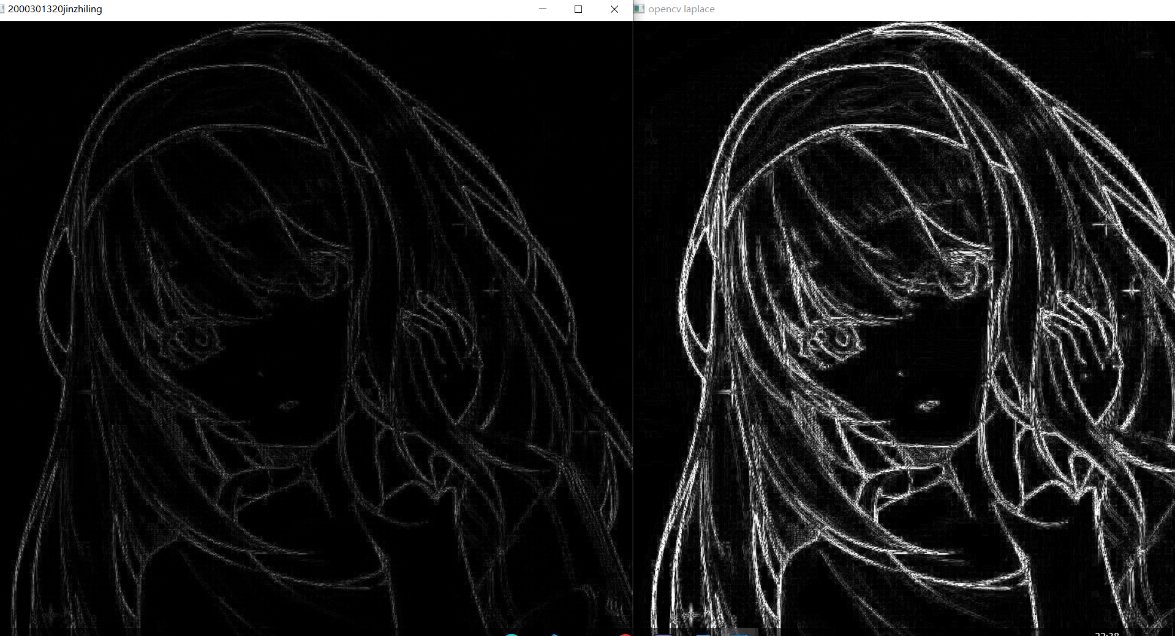
明显看出sobel算子对边缘的强化效果要好于roberts算子。

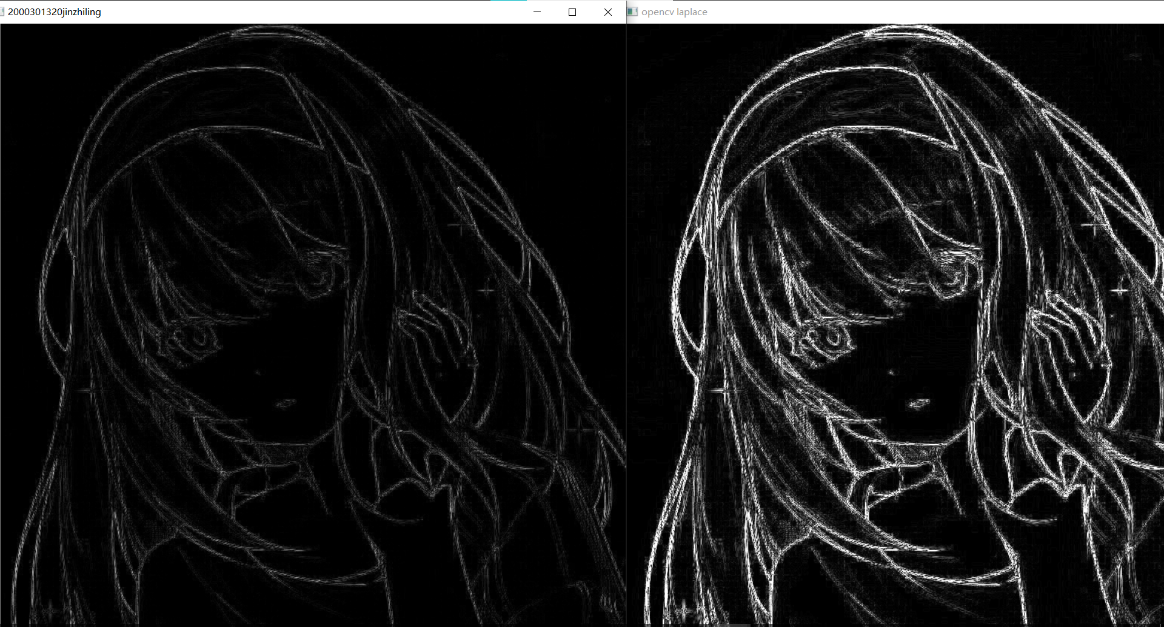
Prewitt算子的实现过程与Roberts算子比较相似。通过Numpy定义模板，再调用OpenCV的filter2D()函数实现对图像的卷积运算，最终通过convertScaleAbs()和addWeighted()函数实现边缘提取。

以下是prewitt算子，以及调用opencv中函数实现的prewitt算法，效果图如下：

拉普拉斯（Laplacian）算子是n维欧几里德空间中的一个二阶微分算子，常用于图像增强领域和边缘提取。它通过灰度差分计算邻域内的像素，基本流程是：判断图像中心像素灰度值与它周围其他像素的灰度值，如果中心像素的灰度更高，则提升中心像素的灰度；反之降低中心像素的灰度，从而实现图像锐化操作。在算法实现过程中，Laplacian算子通过对邻域中心像素的四方向或八方向求梯度，再将梯度相加起来判断中心像素灰度与邻域内其他像素灰度的关系，最后通过梯度运算的结果对像素灰度进行调整。

以下是laplace算子和调用opencv库中的函数生成的效果图：

 这是对四个方向求梯度的laplace算子：

这是对八个方向求梯度的laplace算子：

从效果图看出，这个laplace算子处理后的图像比较糊，效果没有之前算子处理的好。

实验代码：

import cv2

import numpy as np

def Filter(src, dst, filter, padding = "replicate"):

    imarray = cv2.imread(src,0)

    height, width = imarray.shape

    new\_arr = np.zeros((height, width), dtype = "uint16")

    k = filter.shape[0]

    for i in range(height):

        for j in range(width):

            total = 0

            for n in range(pow(k,2)):

                '''

                k = 3, n = 0, 1, 2 ..., 8, a = -1, 0, 1, b = -1, 0, 1

                k = 5, n = 0, 1, 2, 3 ..., 24, a = -2, -1, 0, 1, 2

                '''

                a, b = int(n//k - (k-1)/2), int(n%k - 1)

                #filter\_value

                aa, bb = int(n//k), int(n%k)

                f\_value = filter[aa, bb]

                if i + a <= 0:

                    if j + b <= 0:

                        total += imarray[0, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[0, -1]\*f\_value

                    else:

                        total += imarray[0, j + b]\*f\_value

                elif i + a >= height - 1:

                    if j + b <= 0:

                        total += imarray[-1, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[-1, -1]\*f\_value

                    else:

                        total += imarray[-1, j + b]\*f\_value

                else:

                    if j + b <= 0:

                        total += imarray[i + a, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[i + a, -1]\*f\_value

                    else:

                        total += imarray[i + a, j + b]\*f\_value

            new\_arr[i, j] = abs(total)

    max = np.max(new\_arr)

    min = np.min(new\_arr)

    final\_arr = np.zeros((height, width), dtype = "uint8")

    for i in range(height):

        for j in range(width):

            final\_arr[i, j] = 255\*(new\_arr[i, j] - min)/(max - min)

    cv2.imshow('2000301320jinzhiling',final\_arr)

    cv2.imwrite(dst,final\_arr)

    print("Suceess.")

src = "Images\girl\_gray.jpg"

dst1 = "Images/a.jpg"

roberts1 = np.array([[0,0,0],[0,1,0],[0,0,-1]])

roberts2 = np.array([[0,0,0],[0,0,1],[0,-1,0]])

sobel1 = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])  #突出竖直方向上的边缘

sobel2 = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])  #突出水平方向上的边缘

prewitt1 = np.array([[1,0,-1],[1,0,-1],[1,0,-1]])

prewitt2 = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])

laplace1 = np.array([[0,1,0],[1,-4,1],[0,1,0]])     #四邻域 四个方向求梯度

laplace2 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]) #八邻域 对八个方向求梯度

img = cv2.imread(src, 0)

# roberts算子

""" Filter(src,dst1,roberts1)

x = cv2.filter2D(img,cv2.CV\_16S,roberts1)

y = cv2.filter2D(img,cv2.CV\_16S,roberts2)

absX = cv2.convertScaleAbs(x)

absY = cv2.convertScaleAbs(y)

dst = cv2.addWeighted(absX,0.5,absY,0.5,0)

cv2.imshow('opencv Result',dst) """

# sobel算子

"""

Filter(src, dst1, sobel1)

x = cv2.Sobel(img,cv2.CV\_16S,1,0)

y = cv2.Sobel(img,cv2.CV\_16S,0,1)

absX = cv2.convertScaleAbs(x)   # 转回uint8

absY = cv2.convertScaleAbs(y)

dst = cv2.addWeighted(absX,0.5,absY,0.5,0)

# cv2.imshow("opencv absX", absX)

# cv2.imshow("opencv absY", absY)

cv2.imshow("opencv Result", dst) """

# prewitt算子

""" Filter(src,dst1,prewitt1)

x = cv2.filter2D(img,cv2.CV\_16S,prewitt1)

y = cv2.filter2D(img,cv2.CV\_16S,prewitt2)

absX = cv2.convertScaleAbs(x)

absY = cv2.convertScaleAbs(y)

dst = cv2.addWeighted(absX,0.5,absY,0.5,0)

cv2.imshow('opencv prewitt',dst) """

# Laplace算子

Filter(src,dst1,laplace1)

dst = cv2.Laplacian(img,cv2.CV\_16S,ksize=3)

dst = cv2.convertScaleAbs(dst)

cv2.imshow('opencv laplace',dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

## 五、问题记录和实验总结（必写）

记录实验过程遇到的问题以及解决方法，并对实验收获进行总结。

通过图像锐化实验的学习，我了解到了多种图像锐化的方法，并通过实践进一步了解了如何算法的思想。边缘检测算法主要是基于图像强度的一阶和二阶导数，但导数通常对噪声很敏感，因此需要采用滤波器来过滤噪声，并调用图像增强或阈值化算法进行处理，最后再进行边缘检测。

Sobel算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息。因为Sobel算子结合了高斯平滑和微分求导（分化），因此结果会具有更多的抗噪性，当对精度要求不是很高时，Sobel算子是一种较为常用的边缘检测方法。

 一阶导数和二阶导数都有锐化图像，突出边缘的作用。区别可以归纳成三个：第一，一阶导数通常会产生较宽的边缘；第二，二阶导数对于阶跃性边缘中心产生零交叉，而对屋顶状的边缘，二阶导数取极值；第三，二阶导数对细节有较强的响应，如细线和孤立噪声点。