**北京邮电大学软件学院**

**2022-2023学年第二学期实验报告**

**课程名称：** 多媒体技术与应用

**项目名称： 实验三 直方图规定化**

**项目完成人：**

**姓名： 王宇涵 学号： 2020211730**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

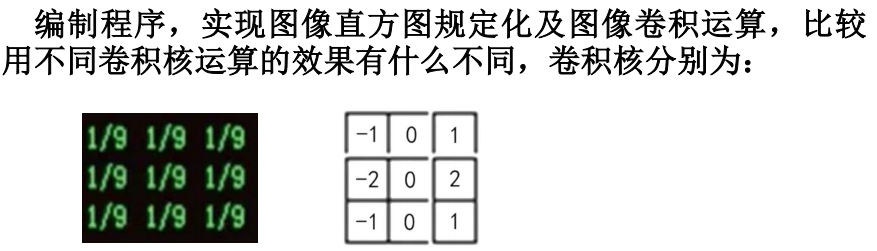
**指导教师：**\_\_\_\_\_\_\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2023年 4 月 20 日**

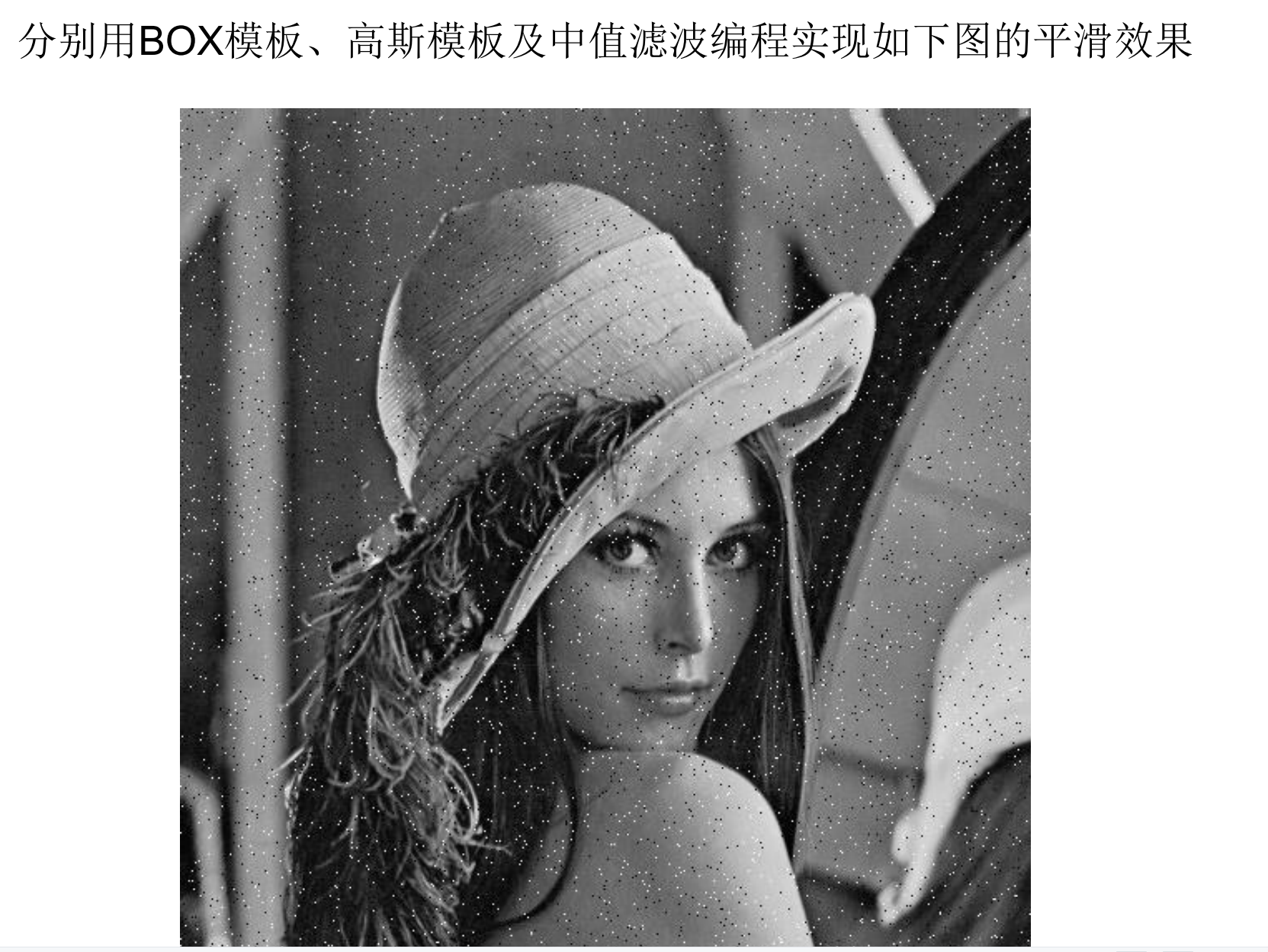
1. **实验目的**

掌握图像图像卷积运算与图像平滑技术

1. **实验内容**
2. **图象卷积技术**



1. **图像平滑**



1. **实验环境**

Windows10操作系统

Pycharm(编译器)

Python3.8

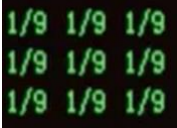
1. **实验结果**

## Part1 图像卷积技术

原图:



## 卷积1



卷积处理后:



## 卷积2

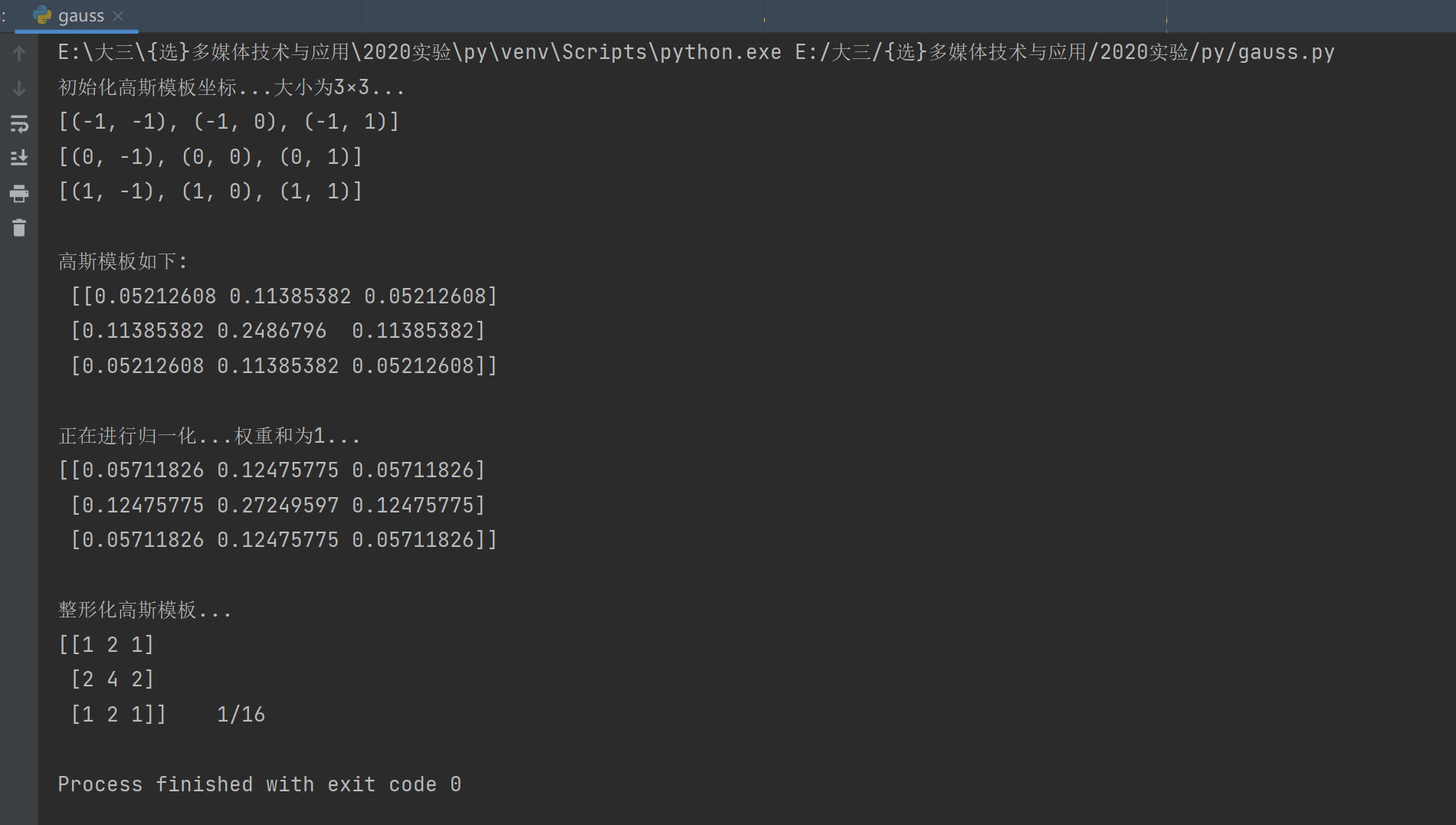


卷积处理后:

## convolution_img2

## Part2 图像平滑

Guass模板生成：



图像平滑处理结果：

|  |  |  |
| --- | --- | --- |
| 所用方法 | 处理后图像 | 原图 |
| BOX | tmpre4pow7z |  |
| GAUSS(3x3) | tmpfswqacmy |  |
| 中值滤波法(3x3) | tmphk6ii449 |  |
| 中值滤波法(5x5) | tmpj11xs65g |  |

1. **附录**
2. 问题分析

**卷积**

在计算机视觉领域中，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。

使用行列嵌套循环遍历原图像，对每个像素进行卷积计算得到新值再赋给新 图像的相应位置的像素。由于图像边界的像素无法完成卷积运算，在嵌套循 环中应从第二行/列开始循环，到倒数第二行/列结束循环。并将新图像的边 界像素统一设为0。需要特别注意的是，对于多通道图像，需要将每一个通 道的像素分别与卷积核对应位置相乘相加，再赋给新图像对应位置像素的对 应通道。

对图像进行卷积运算可以获取某个函数的局部信息。卷积是两个变量在某范围内相乘后求和的结果。对图像进行卷积操作就是卷积核在图像上滑动，对像素值与卷积核逐一相乘再求和，得到新的卷积值。

**平滑**

分别采用均值滤波、高斯滤波、中值滤波三中方式进行图像平滑。

**BOX：**直接采用卷积函数即可，卷积核为1/9(3\*3)

**高斯滤波：**高斯滤波的难点在于高斯模板的生成。生成的过程，首先根据模板的大小，找到模板的中心位置。 然后就是遍历，根据高斯分布的函数，计算模板中每个系数的值。需要注意的是，最后归一化的过程，使用模板左上角的系数的倒数作为归一化的系数（左上角的系数值被归一化为1），模板中的每个系数都乘以该值（左上角系数的倒数），然后将得到的值取整，就得到了整数型的高斯滤波器模板。模板生成后，再利用自定义卷积函数实现即可。

**中值滤波：**中值滤波不需要计算卷积，找到相邻像素的中值即可。

1. 设计方案
2. 首先要根据卷积核的大小对原图像进行补零，然后对图像上的像素逐一进行卷积运算，求和小于0的赋值0，大于255的赋值255，其余的把求和赋值到像素点。输出图像。
3. **中值滤波：**对图像中每个像素点的周围区域内的像素值进行排序，然后选择排序后中间位置的像素值作为该像素点的新值。由于中值对异常值不敏感，因此这种滤波方式可以有效地去除噪声。
4. BOX：直接采用卷积函数即可，卷积核为1/9(3\*3)
5. Gauss模板生成：

* 在make\_template()函数中，首先判断输入的模板大小是否为奇数，如果不是则输出提示信息并退出函数，否则继续执行。
* 生成坐标范围，即以0为中心的坐标轴范围，生成横纵坐标。
* 利用高斯函数公式，生成高斯模板的每个点的值，其中传入的参数为每个点对应的横纵坐标、高斯函数的均值和标准差。
* 将得到的高斯模板进行归一化，使其权重和为1。
* 将归一化后的高斯模板进行整形化，即将所有的数值乘以一个系数，使得其中最小值为1。这样就可以将模板中的权重转换为整数，方便后续的计算。

然后直接采用卷积函数即可

1. 算法

1.卷积

* 首先，传入两个参数：一个是待卷积的二维图像（image），另一个是卷积核（kernel）。
* 函数中使用 image.shape 和 kernel.shape 获取二维图像和卷积核的高度和宽度。
* 计算新图像的尺寸，即去掉卷积核大小后的图像大小。new\_h = height - h + 1，new\_w = width - w + 1。
* 创建一个新的二维数组 new\_image，大小为新图像的尺寸，用于保存卷积后的结果。
* 通过两个循环（i 和 j），遍历整个新图像，计算每个位置的卷积结果。对于每个位置，取出对应的子图像和卷积核，使用矩阵乘法计算卷积结果，并将结果保存到 new\_image 数组中。
* 最后，使用 new\_image.clip(0, 255) 将所有小于0和大于255的像素值重置为0和255，使用 np.rint(new\_image).astype('uint8') 对所有像素进行四舍五入并将结果转换为无符号8位整数类型，然后返回新的图像 new\_image。

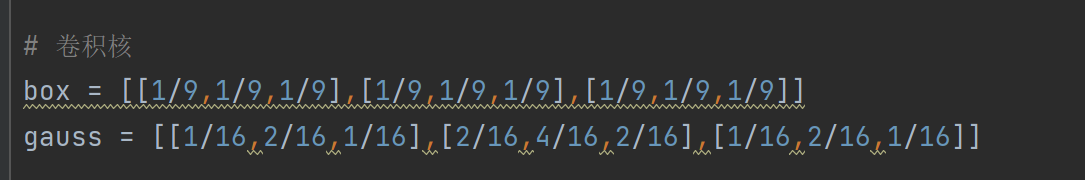
# 定义卷积操作的函数  
def convolution(image, kernel):  
 height, width = image.shape  
 h, w = kernel.shape  
  
 # 经滑动卷积操作后得到的新的图像的尺寸  
 new\_h = height - h + 1  
 new\_w = width - w + 1  
 new\_image = np.zeros((new\_h, new\_w), dtype=float)  
  
 # 进行卷积操作,实则是对应的窗口覆盖下的矩阵对应元素值相乘,卷积操作  
 for i in range(new\_w):  
 for j in range(new\_h):  
 new\_image[i, j] = np.sum(image[i:i+h, j:j+w] \* kernel)  
  
 # 去掉矩阵乘法后的小于0的和大于255的原值,重置为0和255  
 new\_image = new\_image.clip(0, 255)  
 new\_image = np.rint(new\_image).astype('uint8')  
 return new\_image

1. 中值滤波

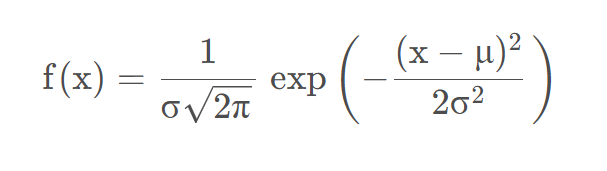
代码首先调用 image.load() 方法将图像转化为像素矩阵，然 后在像素矩阵上进行遍历，对每个像素点的周围3x3区域内的像素值进行排 序，得到中位数作为新的像素值，并将其保存到一个新的像素矩阵 newMap 中。最后，函数返回新的图像 new。

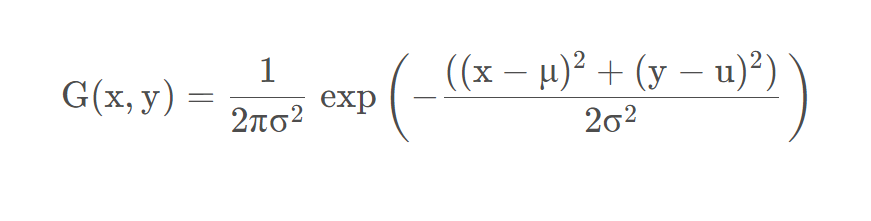
# 中值滤波法 3x3  
def medianFilter\_3(image):  
 map = image.load()  
 width,height = image.size  
 new = image.copy()  
 newMap = new.load()  
  
 for x in range(1,height-1):  
 for y in range(1,width-1):  
 list = []  
 for i in range(-1,2):  
 for j in range(-1,2):  
 list.append(map[x+i,y+j])  
 list.sort()  
 pixel = list[4]  
 newMap[x,y] = pixel  
 return new  
  
# 中值滤波法 5x5  
def medianFilter\_5(image):  
 map = image.load()  
 width,height = image.size  
 new = image.copy()  
 newMap = new.load()  
  
 for x in range(2,height-2):  
 for y in range(2,width-2):  
 list = []  
 for i in range(-2,3):  
 for j in range(-2,3):  
 list.append(map[x+i,y+j])  
 list.sort()  
 pixel = list[12]  
 newMap[x,y] = pixel  
 return new

1. BOX&&gauss

直接调用：  


1. Gauss模板生成

一维模板：

二维：

import numpy as np  
  
def gaussian(x, y, mean=0, std=0.8):  
 g = (1 / (2 \* np.pi \* std \*\* 2)) \* np.exp(-(((x - mean) \*\* 2 + (y - mean) \*\* 2) / (2 \* std \*\* 2)))  
 return g  
  
  
def make\_gaussian\_kernel(kernel\_size=3):  
 print("初始化高斯模板坐标...大小为{}×{}...".format(kernel\_size, kernel\_size))  
 # 找到行与列的关系，用于生成横纵坐标  
 if kernel\_size % 2 == 1:  
 t = (kernel\_size - 1) // 2  
 # 坐标的范围  
 m = np.arange(-t, t + 1)  
 # 重复得到x坐标  
 x = np.repeat(m, kernel\_size)  
 # 重复得到y坐标  
 y = np.repeat(m.reshape(1, -1), kernel\_size, axis=0).flatten()  
 # 利用zip得到坐标数组  
 point = list(zip(x, y))  
 # 循环输出坐标, 调整成行和列的形式  
 for i in range(kernel\_size):  
 print(point[i \* kernel\_size:i \* kernel\_size + kernel\_size])  
 return x, y  
 else:  
 print("请正确输入模板大小...")  
  
  
def normalize\_kernel(kernel):  
 print("\n正在进行归一化...权重和为1...")  
 kernel = kernel / np.sum(kernel)  
 print(kernel)  
 return kernel  
  
  
def integerize\_kernel(kernel):  
 print("\n整形化高斯模板...")  
 # 取第一个值，然后将左上角第一个值变成1，其它的值对应改变，并转换成整形  
 v = kernel[0][0]  
 kernel = np.int32(kernel / v)  
 s = np.sum(kernel)  
 print(kernel, ' 1/' + str(np.sum(kernel)))  
 return kernel  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 设置高斯模板大小，模板请输入奇数  
 kernel\_size = 3  
 # 初始化高斯模板  
 x, y = make\_gaussian\_kernel(kernel\_size=kernel\_size)  
 # 设置高斯函数的均值和标准差  
 mean = 0  
 std = 0.8  
 # 得到结果  
 result = gaussian(x, y, mean=mean, std=std)  
 # reshape  
 gaussian\_kernel = np.reshape(result, (kernel\_size, kernel\_size))  
 print("\n高斯模板如下:\n", gaussian\_kernel)  
 # 归一化  
 normalized\_kernel = normalize\_kernel(gaussian\_kernel)  
 # 整数化  
 integerized\_kernel = integerize\_kernel(normalized\_kernel)

1. **调试心得**

在卷积运算的程序中，一开始补零的时候我直接将卷积核矩阵的维数的一半作为补零的函数，但是这样，3\*3的矩阵的一半就是1.5，程序报错。所以我选择矩阵维数的一半的向下取整。 在处理超出0-255灰度值的值时，我一开始先是取绝对值，再将大于255的数取255，输出结果如下：



可以看见，图像的噪点很多，不符合要求。于是我又将其改成小于0的取0，大于255的取255，这时，处理后的图像的噪点就没有这么大了。



此外，源代码得到的图像不理想：



错误代码如下：

# 图像卷积技术函数  
def convolution(image,ck):  
 pixel\_map = image.load()  
 width, height = image.size  
  
 for x in range(1, width-1):  
 for y in range(1, height-1):  
 ans = 0  
  
 ans = ans + (pixel\_map[x-1,y-1])\*ck[0][0]  
 ans = ans + pixel\_map[x,y-1]\*ck[0][1]  
 ans = ans + pixel\_map[x+1,y-1]\*ck[0][2]  
  
 ans = ans + pixel\_map[x-1,y]\*ck[1][0]  
 ans = ans + pixel\_map[x,y]\*ck[1][1]  
 ans = ans + pixel\_map[x+1,y]\*ck[1][2]  
  
 ans = ans + pixel\_map[x-1,y+1]\*ck[2][0]  
 ans = ans + pixel\_map[x,y+1]\*ck[2][1]  
 ans = ans + pixel\_map[x+1,y+1]\*ck[2][2]  
  
 # 重置 ans 变量  
 ans = min(255, max(0, ans))  
 pixel\_map[x,y] = int(ans)  
  
 return image  
  
ck1 = [[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]]  
ck2 = [[-1,0,1],[-2,0,2],[-1,0,1]]

在循环中，只遍历了 x 和 y 的值，没有遍历通道（channel）的值。如果输入图像具有多个通道，则需要遍历每个通道的像素，否则会导致卷积错误。

因此更改算法后得到相对正确结果



1. 程序

1.卷积：

# -\*- coding: utf-8 -\*-  
from PIL import Image  
import numpy as np  
  
# 定义卷积操作的函数  
def convolution(image, kernel):  
 height, width = image.shape  
 h, w = kernel.shape  
  
 # 经滑动卷积操作后得到的新的图像的尺寸  
 new\_h = height - h + 1  
 new\_w = width - w + 1  
 new\_image = np.zeros((new\_h, new\_w), dtype=float)  
  
 # 进行卷积操作,实则是对应的窗口覆盖下的矩阵对应元素值相乘,卷积操作  
 for i in range(new\_w):  
 for j in range(new\_h):  
 new\_image[i, j] = np.sum(image[i:i+h, j:j+w] \* kernel)  
  
 # 去掉矩阵乘法后的小于0的和大于255的原值,重置为0和255  
 new\_image = new\_image.clip(0, 255)  
 new\_image = np.rint(new\_image).astype('uint8')  
 return new\_image  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 读取图像数据并且转换为对应的numpy下的数组  
 A = Image.open(r"E:\大三\{选}多媒体技术与应用\2020实验\实验4\lena\_noise.bmp").convert("L")  
 a = np.array(A)  
  
 # 卷积核  
 kernel\_1 = np.array(([1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]))  
 kernel\_2 = np.array(([-1, 0, 1], [-2, 0, 2], [-1, 0, 1]))  
 img1 = convolution(a, kernel\_1)  
 img1 = Image.fromarray(img1)  
 img1.show()  
 img1.save(r"E:\大三\{选}多媒体技术与应用\2020实验\实验4\convolution\_img1.jpg")  
 img2 = convolution(a, kernel\_2)  
 img2 = Image.fromarray(img2)  
 img2.show()  
 img2.save(r"E:\大三\{选}多媒体技术与应用\2020实验\实验4\convolution\_img2.jpg")

1. 平滑

from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
from numpy.lib.function\_base import diff  
from numpy.lib.type\_check import imag  
  
def drawHistogram(image):  
 pic = np.array(image)  
 plt.hist(pic[:, :].ravel(), bins=256,facecolor='#328121', edgecolor='#898312')  
 plt.show()  
  
# 卷积  
def convolution(image,ck):  
 map = image.load()  
 new = image.copy()  
 newMap = new.load()  
 width,height = image.size  
   
 for x in range(0,height-2):  
 for y in range(0,width-2):  
 pixel = 0  
 for i in range(0,3):  
 for j in range(0,3):  
 cur = map[x+i,y+j]\*ck[i][j]  
 pixel = pixel + (int)(cur)  
 newMap[x+1,y+1] = pixel  
 return new  
  
# 中值滤波法 3x3  
def medianFilter\_3(image):  
 map = image.load()  
 width,height = image.size  
 new = image.copy()  
 newMap = new.load()  
  
 for x in range(1,height-1):  
 for y in range(1,width-1):  
 list = []  
 for i in range(-1,2):  
 for j in range(-1,2):  
 list.append(map[x+i,y+j])  
 list.sort()  
 pixel = list[4]  
 newMap[x,y] = pixel  
 return new  
  
# 中值滤波法 5x5  
def medianFilter\_5(image):  
 map = image.load()  
 width,height = image.size  
 new = image.copy()  
 newMap = new.load()  
  
 for x in range(2,height-2):  
 for y in range(2,width-2):  
 list = []  
 for i in range(-2,3):  
 for j in range(-2,3):  
 list.append(map[x+i,y+j])  
 list.sort()  
 pixel = list[12]  
 newMap[x,y] = pixel  
 return new  
  
# 卷积核  
box = [[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]]  
gauss = [[1/16,2/16,1/16],[2/16,4/16,2/16],[1/16,2/16,1/16]]  
  
image = Image.open(r'E:\大三\{选}多媒体技术与应用\2020实验\实验4\lena\_noise.bmp')  
image.show()  
  
# 去除噪声  
image1 = convolution(image,box)  
image1.show()  
image2 = convolution(image,gauss)  
image2.show()  
image3 = medianFilter\_3(image)  
image3.show()  
image4 = medianFilter\_3(image)  
image4.show()

1. Gauss模板

import numpy as np  
  
def gaussian(x, y, mean=0, std=0.8):  
 g = (1 / (2 \* np.pi \* std \*\* 2)) \* np.exp(-(((x - mean) \*\* 2 + (y - mean) \*\* 2) / (2 \* std \*\* 2)))  
 return g  
  
  
def make\_gaussian\_kernel(kernel\_size=3):  
 print("初始化高斯模板坐标...大小为{}×{}...".format(kernel\_size, kernel\_size))  
 # 找到行与列的关系，用于生成横纵坐标  
 if kernel\_size % 2 == 1:  
 t = (kernel\_size - 1) // 2  
 # 坐标的范围  
 m = np.arange(-t, t + 1)  
 # 重复得到x坐标  
 x = np.repeat(m, kernel\_size)  
 # 重复得到y坐标  
 y = np.repeat(m.reshape(1, -1), kernel\_size, axis=0).flatten()  
 # 利用zip得到坐标数组  
 point = list(zip(x, y))  
 # 循环输出坐标, 调整成行和列的形式  
 for i in range(kernel\_size):  
 print(point[i \* kernel\_size:i \* kernel\_size + kernel\_size])  
 return x, y  
 else:  
 print("请正确输入模板大小...")  
  
  
def normalize\_kernel(kernel):  
 print("\n正在进行归一化...权重和为1...")  
 kernel = kernel / np.sum(kernel)  
 print(kernel)  
 return kernel  
  
  
def integerize\_kernel(kernel):  
 print("\n整形化高斯模板...")  
 # 取第一个值，然后将左上角第一个值变成1，其它的值对应改变，并转换成整形  
 v = kernel[0][0]  
 kernel = np.int32(kernel / v)  
 s = np.sum(kernel)  
 print(kernel, ' 1/' + str(np.sum(kernel)))  
 return kernel  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 设置高斯模板大小，模板请输入奇数  
 kernel\_size = 3  
 # 初始化高斯模板  
 x, y = make\_gaussian\_kernel(kernel\_size=kernel\_size)  
 # 设置高斯函数的均值和标准差  
 mean = 0  
 std = 0.8  
 # 得到结果  
 result = gaussian(x, y, mean=mean, std=std)  
 # reshape  
 gaussian\_kernel = np.reshape(result, (kernel\_size, kernel\_size))  
 print("\n高斯模板如下:\n", gaussian\_kernel)  
 # 归一化  
 normalized\_kernel = normalize\_kernel(gaussian\_kernel)  
 # 整数化  
 integerized\_kernel = integerize\_kernel(normalized\_kernel)

**实验心得**

基本掌握了图像卷积技术的思路和方法，可以通过编制程序实现图像卷积运算，并比较用不同卷积核运算的不同效果。

BOX简单,但是主要缺点是在降低噪声的同时使图像产生模糊,特别在边缘和细节处.而且邻域越大,在去噪能力增强的同时模糊程度越严重.

Gauss法难点在于模板生成，需要运用公式进行处理并转化为整数。

超限像素平滑法对抑制椒盐噪声比较有效,对保护仅有微小灰度差的细节及纹理也有效.随着邻域增大,去噪能力增强,但模糊程度也大.

中值滤波法能有效小若椒盐噪声,且比BOX和GAUSS法更有效.