图像处理作业-刘继东

# 第一周作业：打开图像，显示图像，存储图像，对一张图片进行缩放，观察其分辨率，降低灰度处理

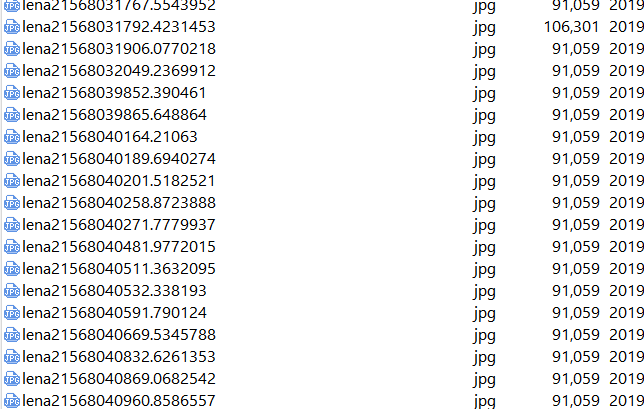
处理步骤：

1. 打开一副jpg文件，读取head部分，获得hight和width，然后读取data中的数据，将data数据放置到三维数组中，每一个点有r，g，b三个数值标识该点的颜色。
2. 使用opencv的imshow显示图片，imsave保存图片，
3. 使用opencv的resize缩放图片，放大或缩小，显示图片
4. 降低灰度级：对灰度图像的每个点的像素值/4,从8bit位降低到6bit位（即从256级降低到64灰度级）

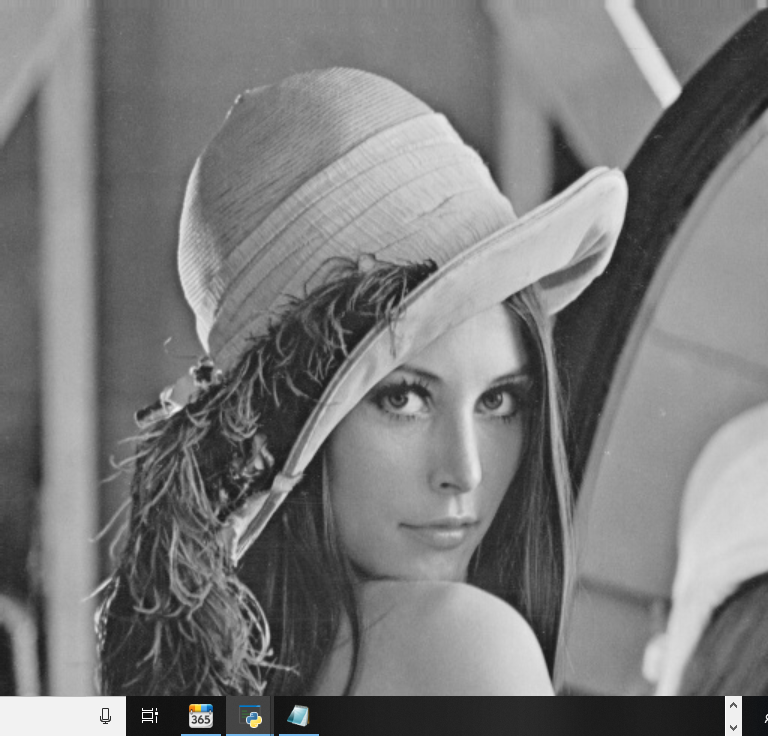
显示图片：



存储：增加时间戳作文文件名，保存如下：



放大1.5倍：



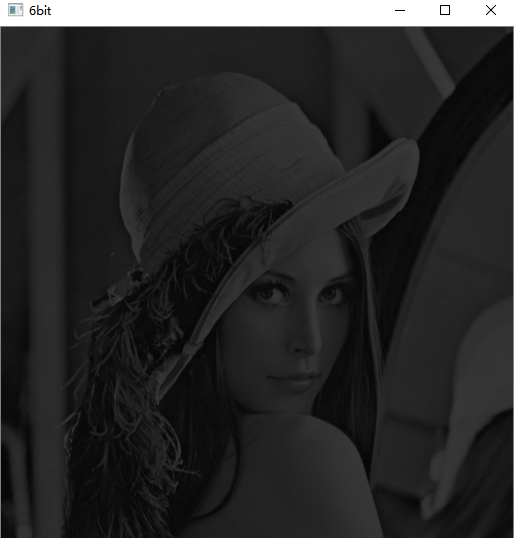
缩小0.5倍：



降低灰度值（每个像素灰度值-100）：



降低灰度级，6bit



源码：

###2019年9月9日 第一周作业：打开图像，显示图像，存储图像，对一张图片进行缩放，观察其分辨率，降低灰度处理  
  
img = cv.imread("./source/lena.jpg",cv.IMREAD\_GRAYSCALE) #读入做灰色处理  
print(img.shape) #打印大小  
(height,width) = img.shape  
  
cv.imshow("lena",img) #显示  
rename = str(time.time())  
print(rename,height,width)  
cv.imwrite("./dest/lena2"+rename+".jpg",img) #存错  
  
#放大一倍  
largedim = (int(height\*1.5),int(width\*1.5))  
img\_enlarge=cv.resize(img,largedim) #1data 2 (高，宽)  
cv.imshow("large",img\_enlarge);  
print(img\_enlarge.shape)  
  
###缩小一半,方法一  
img\_enMin=cv.resize(img,None,0.5,0.5,cv.INTER\_AREA) #1data 2 (高，宽)  
cv.imshow("minor1",img\_enMin);  
print(img\_enMin.shape)  
  
###缩小一半,方法二  
mindim = (int(height\*0.5),int(width\*0.5))  
img\_enMindim=cv.resize(img,mindim,cv.INTER\_AREA) #1data 2 (高，宽)  
cv.imshow("minor2",img\_enMindim);  
print(img\_enMindim.shape)  
  
  
##@改变每个像素的灰度值，每个像素降低100, 生成同大小的二维数组  
subval = np.full(img.shape,100,dtype=np.uint8)  
subimg = cv.subtract(img,subval)  
cv.imshow("subimg",subimg)

###降低灰度级数8bit图像降为6bit。  
destImg = img//4  
cv.imshow("6bit",destImg )  
print(destImg)  
  
cv.waitKey(0)

# 第二周作业：对低对比度的图像进行拉伸和用直方图均值化进行优化

处理步骤：

1. 使用伽马变换进行拉伸：

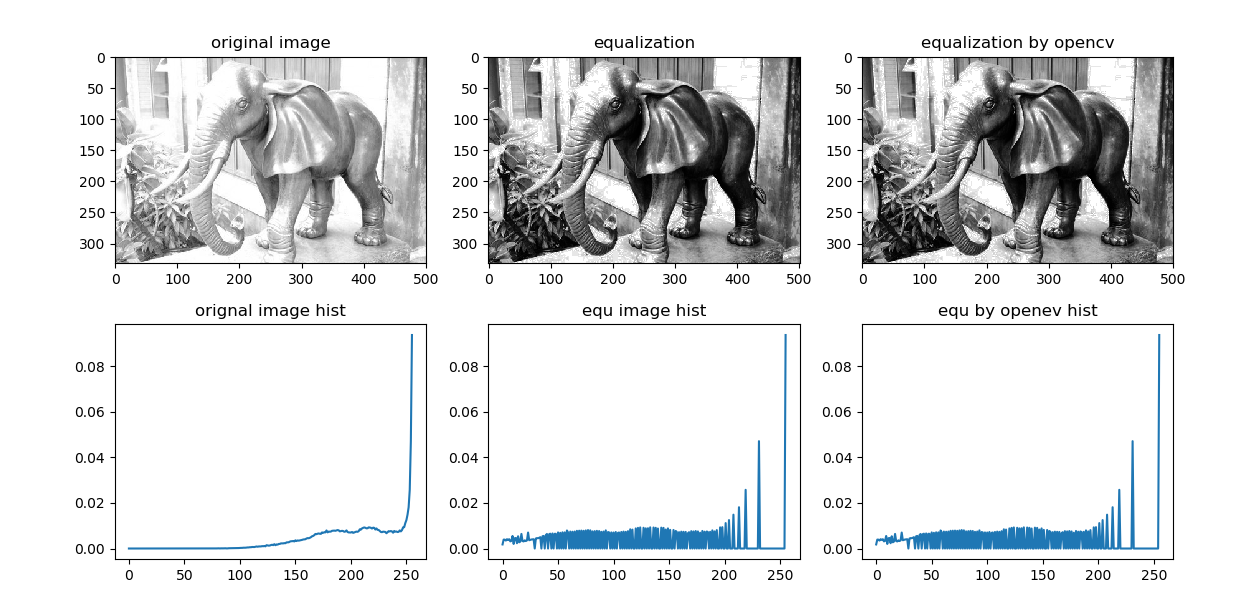
选取一幅对比度图像偏亮（曝光过度）的照片，将其变暗，γ>1，分别取2，3，4，5进行拉伸变换，效果如下。结果完成后，与skimage的伽马变换函数进行效果比对，基本相同。根据显示效果，当γ=4时效果最佳，实验结束。

1. 对图像进行直方图均衡优化

首先计算p(rk),然后求出累计直方图sk，并进行取整操作skInt。建立转换函数s=skInt[rk]然后使用map函数，对原像素值进行转换。效果如下。使用opencv的直方图均衡方法，比较两者差异，效果基本相同，实验结束。

对比度拉伸

对图像进行直方图均衡优化



代码如下：

###2019年9月14日 打开一幅低对比度图像，拉伸其图像，直方图均衡。  
#使用skimage开发包 和 根据原理自行开发，并对比显示其效果。  
  
img = cv.imread("./source/02.lowcontrast.jpg",cv.IMREAD\_GRAYSCALE) #读入做灰色处理  
print(img.shape)

#使用skimage的伽马变换  
gamma\_img = exposure.adjust\_gamma(img, 5)  
#根据原理开发的伽马变换：对图像拉伸，先需做归一化(/255去除量纲），然后进行变换，最后复原回图像空间（需做无符号转换）  
#曝光过度，调黑。尝试γ取2，3，4，5，检查效果  
stretch2 = np.uint8(np.power(img / 255.0, 2) \* 255.0)  
stretch3 = np.uint8(np.power(img/255.0,3)\*255.0)  
stretch4 = np.uint8(np.power(img/255.0,4)\*255.0)  
stretch5 = np.uint8(np.power(img/255.0,5)\*255.0)  
#skimage的直方图均衡函数  
equ = cv.equalizeHist(img)  
#cv.imshow('equalization by opencv', np.hstack((img, equ))) # 并排显示  
  
#按照公式计算：  
#原始图像灰度级统计  
rk = np.bincount(img.ravel(), minlength=256) #图像二维数组先转换为一维，然后使用bincount求出不同像素级数的个数  
prk=rk/img.size #各级像素的概率  
sk=np.zeros(256)  
pos = 0  
sum = 0.0  
for val in prk:  
 sum = sum + val #累计  
 sk[pos] = sum  
 pos = pos+1  
  
#展示处理后的直方图  
#sk取整扩展  
skInt = np.uint8(255\*sk+0.5)  
  
def fn\_change(x):  
 return skInt[x]  
  
equImg2=np.array(list(map(fn\_change,img)),dtype=np.uint8)  
  
#展示图像直方图  
# plt.hist(img.ravel(), 256, [0, 256])#等价于 bincount先求出每个级数的个数，然后画直方图。  
hist = np.bincount(img.ravel(), minlength=256)/img.size  
equAxs[1,0].set\_title("orignal image hist")  
equAxs[1,0].plot(hist) #原图  
  
hist2 = np.bincount(equImg2.ravel(), minlength=256)/img.size  
equAxs[1,1].set\_title("equ image hist")  
equAxs[1,1].plot(hist2)

# 第三周作业：对一副图像进行加噪声，平滑，锐化处理。

处理步骤：

1.噪声处理：

1）.高斯噪声：生成正态分布的随机数，期望为0，方差=0.04（方差太大，图像噪声太多，太小，噪声不明显），矩阵大小与图像大小相同；随机矩阵+原图（归一化）后范围限制在0~1之间，然后进行还原为0~255的灰度值。

2）.添加椒盐噪声：产生x%的椒盐噪声：设总的像素点m，则x%的椒盐量，总共需要点数量为n= m\*x; 随机生成n个点坐标，随机设置该处的值为0或255.

2.平滑处理：

对椒盐噪声图像使用均值平滑与中值平滑

3.锐化处理：

梯度法为：newPixel = abs(fn[i, j+1] - fn[i, j]) + abs( fn[i+1, j] - fn[i, j])

[-1, 1],

[1, 0]

roberts: newPixel = abs(fn[i+1,j+1] - fn[i,j]) + abs(fn[i+1,j] - fn[i, j+1])

[-1, -1]

[1, 1]

sobel算法：检测边缘较平滑，光洁

newPixel = max(HorizontalDifference , VerticalDifference)

HorizontalDifference = abs( fn[

结构：horizontalDifference：

[-1, 0, 1]

[-2, 0, 2]

[-1, 0, 1]

verticalDifference： 将HorizontalDifference 顺时针旋转90度

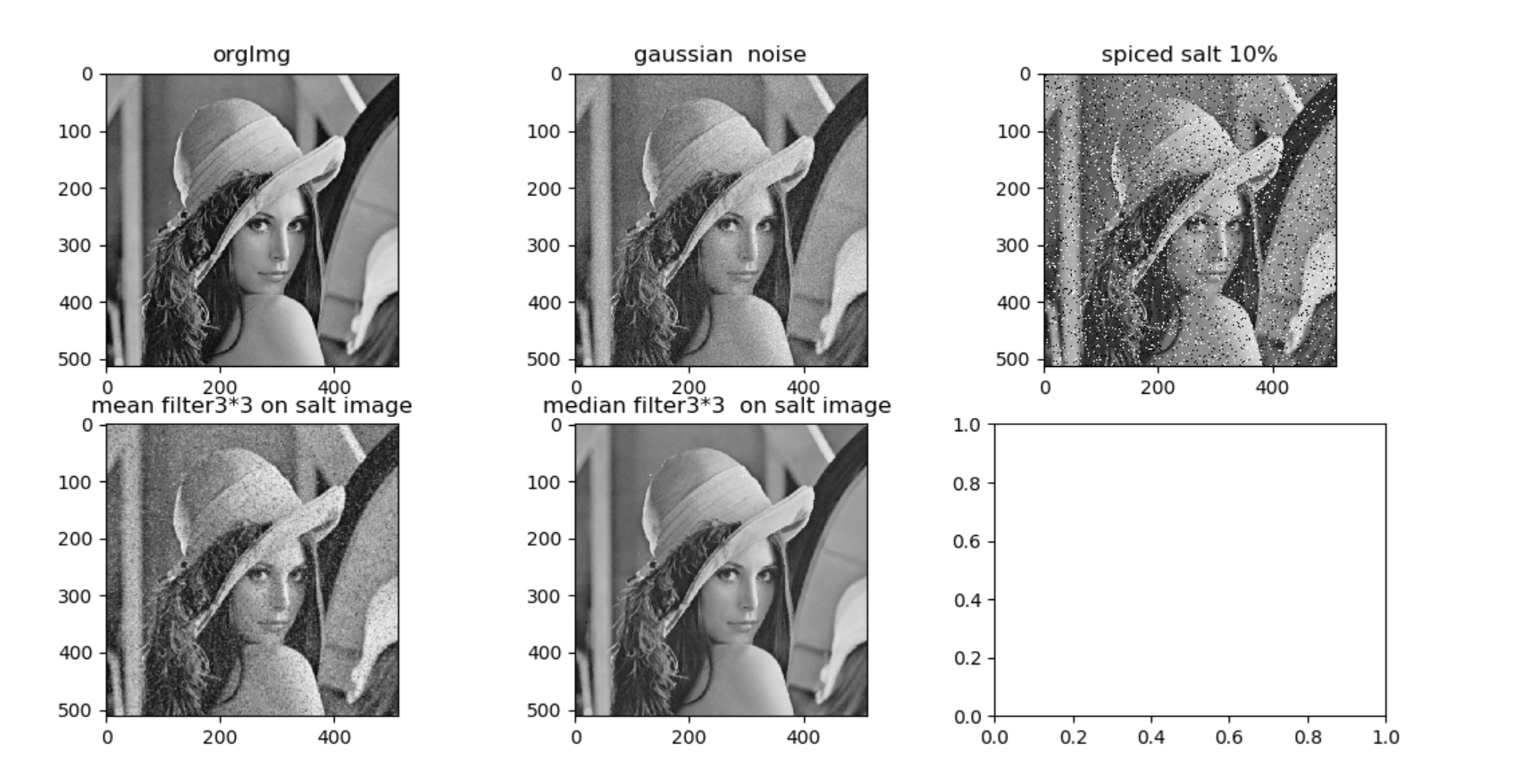
Prewitt算子：

结构：

[-1, 0, 1]

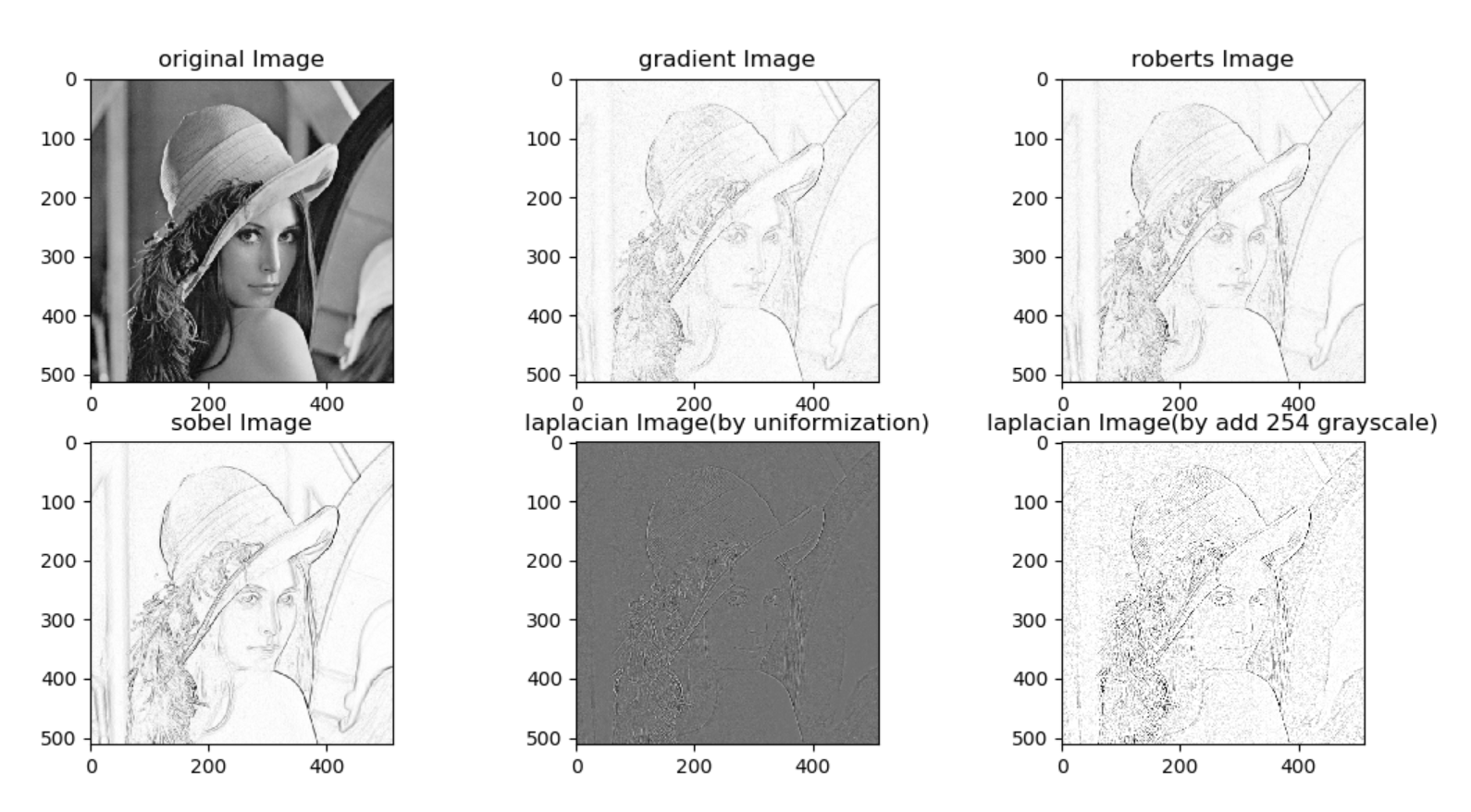
[-1, 0, 1]

[-1, 0, 1]



锐化：

梯度锐化，roberts锐化，sobel，Laplace。



代码如下：

#2019年9月17日 对一副图像加噪声，进行平滑，锐化作用，处理步骤：  
#1.高斯噪声：生成正态分布的随机数，期望为0，方差=0.04（方差太大，图像噪声太多，太小，噪声不明显），矩阵大小与图像大小相同；随机矩阵+原图（归一化）后范围限制在0~1之间，然后进行还原为0~255的灰度值。  
#2.添加椒盐噪声：产生x%的椒盐噪声：设总的像素点m，则x%的椒盐量，总共需要点数量为n= m\*x; 随机生成n个点坐标，随机设置该处的值为0或255.  
#3.平滑处理：高斯噪声的平滑使用，去除椒盐噪声。  
#4.锐化处理：梯度锐化，roberts，sobel，Laplace算子锐化  
  
img = cv.imread("./source/lena.jpg",cv.IMREAD\_GRAYSCALE)  
  
#高斯噪声  
noise=np.random.normal(0,0.04,size=img.shape) #产生高斯（正态）分布噪声，期望为0，方差=0.04 （方差过大1，图像噪声太多，方差过小0，噪声太少）  
gaussImg = np.uint8(np.clip((noise + img / 255), 0, 1) \* 255) #范围在0，1之间，之外的  
#cv.imshow("gauss",np.hstack((img,gaiImg)))  
  
##2019年9月28日 优化椒盐噪声的生成  
# 产生x%的椒盐噪声：设总的像素点m，则x%的椒盐量，总共需要点数量为n= m\*x; 随机生成n个点坐标，随机设置该处的值为0或255.  
def addSalt(orgImg, x):  
 m = orgImg.size  
 n = np.int(m\*x)  
 pos = zip(np.random.randint(0, orgImg.shape[0], size=n), np.random.randint(0, orgImg.shape[1], size= n))  
 newImg = np.copy(orgImg)  
 for x, y in pos:  
 if(np.random.randint(2)):  
 newImg[x, y] = 0  
 else:  
 newImg[x, y] = 255  
 return newImg  
  
saltImg = addSalt(img, 0.1)  
  
##滤波卷积操作：  
# 定义函数指定一个原图，使用规定的3\*3模板进行滤过操作。  
# 采用复制边方式进行处理。  
def fn\_filter33 (orgImg, filterImg, method="sum"):  
 rows,cols = orgImg.shape  
 #为原图扩展边，以便于计算，扩展方式为边复制.  
 extentImg = np.zeros((rows+2, cols+2), dtype=np.uint8)  
 extentImg[1:rows+1, 1:cols+1] = orgImg #中间部分用原图填充  
 extentImg[0, 1:cols+1]=orgImg[0, :] #第一行复制img的第一行  
 extentImg[rows+1, 1:cols+1]=orgImg[rows-1, :]#最后一行复制img的最后一行  
 extentImg[1:rows+1, 0]=orgImg[:, 0] #第一列复制过来  
 extentImg[1:rows+1, cols+1]=orgImg[:, cols-1] #最后一列  
 #补齐四个角  
 extentImg[0, 0]=orgImg[0, 0]  
 extentImg[0, cols+1] = orgImg[0, cols-1]  
 extentImg[rows+1, 0] = orgImg[rows-1, 0]  
 extentImg[rows+1, cols+1] = orgImg[rows-1, cols-1]  
  
 ##均值滤波图像,中值滤波图像结果  
 meanImg = np.zeros(orgImg.shape)  
 medianImg = np.zeros(orgImg.shape, dtype=np.uint8)  
  
 #print(orgImg.shape, "开始卷积滤波", "method = %s"%method)  
  
 if method == "sum" :  
 for i in np.arange(1, rows+1):  
 for j in np.arange(1, cols+1):  
 rect = filterImg\*extentImg[i-1:i+2, j-1:j+2] #i+2的原因是切片时截至数不包括在里面  
 meanImg[i-1, j-1] = np.sum(rect) #np.uint8(abs(np.sum(rect))) #均值： 如果是十字星型模板，则不能使用mean方法，需要计算sum，除以十字星模板的sum  
 return meanImg  
 else:  
 for i in np.arange(1, rows + 1):  
 for j in np.arange(1, cols + 1):  
 rect = filterImg \* extentImg[i - 1:i + 2, j - 1:j + 2] # i+2的原因是切片时截至数不包括在里面  
 medianImg[i - 1, j - 1] = np.uint8(np.median(rect)) # 中值  
  
 return medianImg  
####2. 平滑滤波处理。  
#3\*3的均值滤波器  
templateMean = np.array([[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]])  
  
#中值滤波  
print("start sharpen image by median algorithm")  
medianImg = fn\_filter33(saltImg, templateMean, method="median")  
#均值滤波  
print("start sharpen image by mean algorithm")  
meanImg = fn\_filter33(saltImg, templateMean\*1/9, method="sum")  
  
###3. 锐化处理：包括：1阶的：梯度法，Roberts算法（交叉差分），sobel算法，prewitt算子（对噪声敏感）：突出小缺陷，去除慢变化背景  
# 2阶的拉普拉斯算子：增强灰度突变处的比对度。  
  
#3.1 梯度法，roberts  
#梯度法为：newPixel = abs(fn[i, j+1] - fn[i, j]) + abs( fn[i+1, j] - fn[i, j])  
# [-1, 1],  
# [1, 0]  
#roberts: newPixel = abs(fn[i+1,j+1] - fn[i,j]) + abs(fn[i+1,j] - fn[i, j+1])  
# [-1, -1]  
# [1, 1]  
print(" start sharpen image by gradient and roberts algorithm")  
gradientImg = np.zeros(img.shape, dtype=np.uint8)  
robertsImg = np.zeros(img.shape, dtype=np.uint8)  
for i in np.arange(0,img.shape[0]-1):  
 for j in np.arange(0, img.shape[1]-1):  
 grandient = abs(int(img[i, j+1]) - int(img[i, j])) + abs(int(img[i+1, j]) - int(img[i, j])) #超过255怎么处理？ #要先转换类型为有符号，否则计算有偏差，按无符号处理。  
 gradientImg[i, j] = 255 if grandient > 255 else grandient  
  
 roberts = abs(int(img[i+1, j+1]) - int(img[i, j])) + abs(int(img[i+1, j]) - int(img[i, j+1]))  
 robertsImg[i, j] = 255 if roberts > 255 else roberts  
  
#图像取反  
def fn\_convert(orgImg):  
 convImg = np.array([255]\*orgImg.size).reshape(orgImg.shape)  
 convImg = convImg - orgImg  
 return convImg  
  
###sobel算法：检测边缘较平滑，光洁  
## newPixel = max(HorizontalDifference , VerticalDifference)  
## HorizontalDifference = abs( fn[  
# 结构：horizontalDifference：  
# [-1, 0, 1]  
# [-2, 0, 2]  
# [-1, 0, 1]  
# verticalDifference： 将HorizontalDifference 顺时针旋转90度  
  
#Prewitt算子：  
#结构：  
# [-1, 0, 1]  
# [-1, 0, 1]  
# [-1, 0, 1]  
  
print(" start sharpen image by prewitt and sobel algorithm")  
templateSobelHor = np.array([[-1, 0, 1],  
 [-2, 0, 2],  
 [-1, 0, 1]])  
templateSobelVer = np.array([[-1, -2, -1],  
 [0, 0, 0],  
 [1, 2, 1]])  
print("compute sobel horizontalDifference ")  
sobelHorImg = abs(fn\_filter33(img, templateSobelHor))  
sobelHorImg = np.uint8((sobelHorImg/np.max(sobelHorImg))\* 255) #归一化处理，范围到【0，255】之间  
print("compute sobel verticalDifference ")  
sobelVerImg = abs(fn\_filter33(img, templateSobelVer))  
sobelVerImg = np.uint8((sobelVerImg/np.max(sobelVerImg))\* 255) #归一化处理，范围到【0，255】之间  
  
#取两者中的较大值作为新的值。  
sobelImg = np.zeros(sobelHorImg.shape, dtype=np.uint8)  
for i in np.arange(0, sobelHorImg.shape[0]):  
 for j in np.arange(0, sobelHorImg.shape[1]):  
 sobelImg[i, j] = sobelHorImg[i, j] if sobelHorImg[i, j] > sobelVerImg[i, j] else sobelVerImg[i, j]  
  
###拉普拉斯算子：  
#拉普拉斯边缘检测  
print("start sharpen image by laplace algorithm")  
templateLaplace = np.array([[-1, -1, -1],  
 [-1, 8, -1],  
 [-1, -1, -1]])  
templaplaceImg = fn\_filter33(img, templateLaplace)  
#处理后的图+254，并限制范围在0，255之间，效果并不好。改为将templaplaceImg的值范围重新度量，归一化处理，压缩至【0，255】之间。效果偏暗。  
print("templaplaceImg min, max = ",templaplaceImg.min(), templaplaceImg.max())  
#方法一：灰度级偏移254.  
laplaceImgshift = np.clip(templaplaceImg+ np.array([254]\*templaplaceImg.size).reshape(templaplaceImg.shape), 0, 255).astype(np.uint8)  
#方法二：归一化到【0，255】之间  
scale = 0 #templaplaceImg的值域范围。  
if(templaplaceImg.min()<0):  
 scale = templaplaceImg.max() - templaplaceImg.min()  
 templaplaceImg = templaplaceImg + np.array([-1\*templaplaceImg.min()]\* templaplaceImg.size).reshape(templaplaceImg.shape) #使最小值为0  
else:  
 scale = templaplaceImg.max()  
  
if(scale>255):  
 laplaceImg = np.uint8(templaplaceImg/scale \* 255)

# 第四周作业：对一副图像进行傅立叶变换，显示频谱，取其5，50，150为截至频率，进行频率域平滑，锐化，显示图像

处理步骤：

1）傅里叶变换：直接调用numpy的fft的2维傅里叶变换fft2,并作居中处理

2）显示频谱：取幅度谱，求对数，并通过线性变换至返回【0，255】，显示该图像

（求对数的原因是幅度谱的值比较大，超出显示范围，线性变换的原因是取完对数，范围比较小【0，20】，图像显示不易观察）

3）取D0={5，50，150}，设置滤波器H（理想，butterworth ，gauss），显示图像。步骤：

HUV = 1/（1+(DUV/D0)^2n) butterworth 低通滤波器公式，

HUV = exp(-DUV^2/2D0^2) guass 低通滤波公式

高通 HP = 1- HL

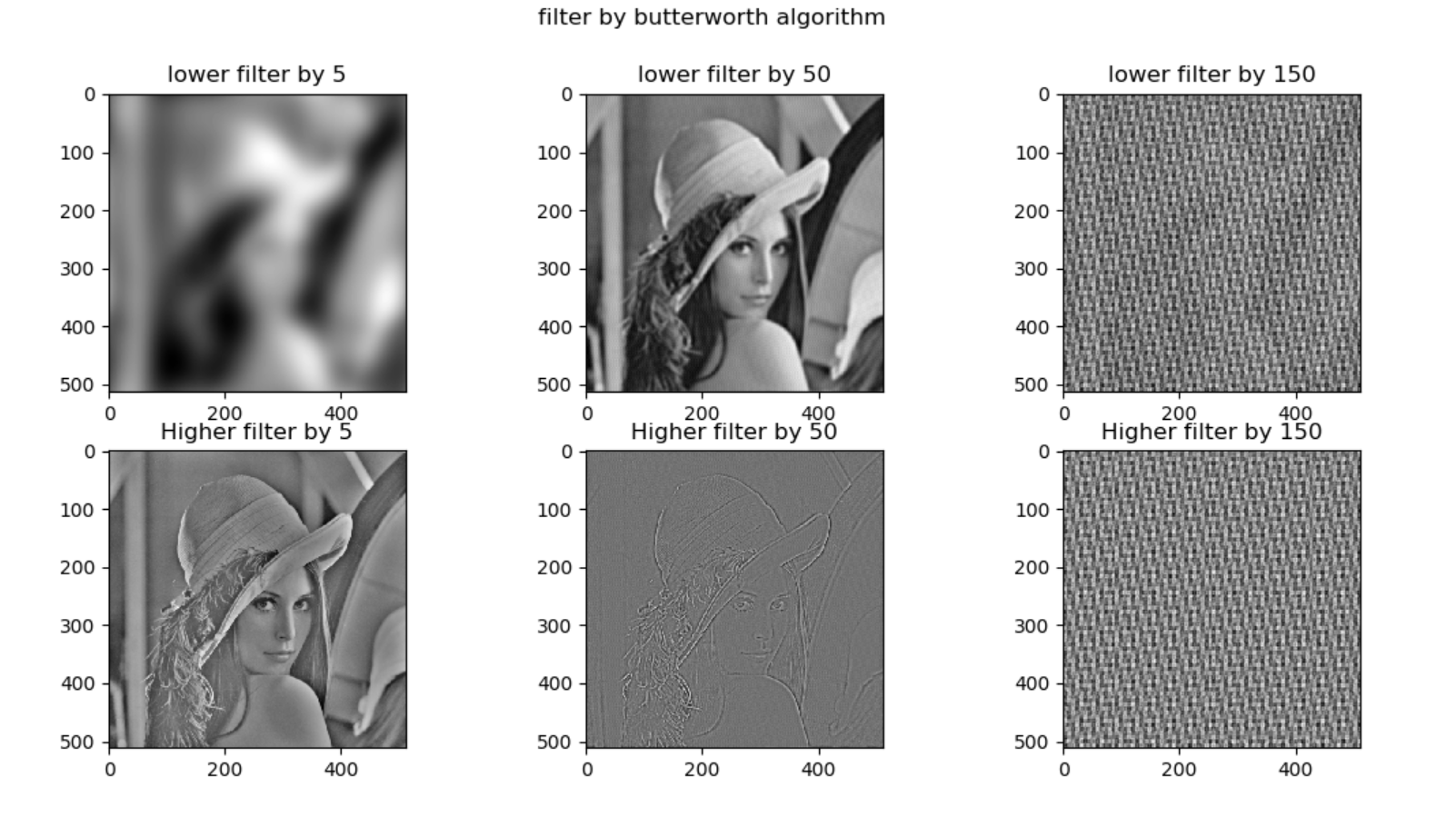
滤波操作 GUV = HUV\*FUV

傅里叶反变换（先将中心点切换回左端点），取出实部

对实部进行范围变换，线性到【0，255】范围，显示图片。

参考：https://blog.csdn.net/cjsh\_123456/article/details/79342300





代码如下：

#线性变换到【0-255】之间  
def line2Scale(originalImg):  
 scale = 0  
 convertedImg = np.zeros(originalImg.shape)  
 if(originalImg.min()<0):  
 scale = originalImg.max() - originalImg.min()  
 convertedImg = originalImg + np.array([-1\*originalImg.min()]\* originalImg.size).reshape(originalImg.shape) #使最小值为0  
 else:  
 scale = originalImg.max()  
 convertedImg = originalImg  
  
 convertedImg = np.uint8(convertedImg/scale \* 255)  
 return convertedImg  
  
#butterworth 低通滤波器：中心在(m + 1, n + 1)  
def func\_LowerFilter(FUV, D0, ng, model="butterworth", isLower = True):  
 HFilter = np.zeros(FUV.shape)  
 m = FUV.shape[0] // 2  
 n = FUV.shape[0] // 2  
 print("m, n ", m, n)  
 if isLower == True :  
 if model == "butterworth" :  
 for u in np.arange(0, FUV.shape[0]):  
 for v in np.arange(0, FUV.shape[1]):  
 duv = (u-m)\*\*2 +(v-n)\*\*2  
 HFilter[u, v] = 1/(1+(duv\*\*ng)/(D0\*\*(2\*ng)))  
 else:  
 if model == "butterworth" :  
 for u in np.arange(0, FUV.shape[0]):  
 for v in np.arange(0, FUV.shape[1]):  
 duv = (u-m)\*\*2 +(v-n)\*\*2  
 HFilter[u, v] =1- 1/(1+(duv\*\*ng)/(D0\*\*(2\*ng)))  
 GUV = HFilter \* FUV  
  
 #反傅里叶变换,先将中心返回到左上角  
 gxy = np.fft.ifft2(np.fft.ifftshift(GUV))  
 gReal = np.real(gxy)  
 print("greal: ", gReal.min(), gReal.max())  
 #获取图片：做归一化，转成【0，255】  
  
 img\_out = line2Scale(gReal)  
 return img\_out  
  
#读取图片，并进行傅里叶变换，显示频谱。  
img = cv.imread("./source/lena.jpg",cv.IMREAD\_GRAYSCALE)  
#快速傅里叶变换算法,并进行居中处理  
FFTImg = np.fft.fft2(img)  
fshift = np.fft.fftshift(FFTImg)  
#展示频谱fft结果是复数, 求幅度谱，并对其求对数  
fimg = np.log(np.abs(fshift))  
  
print("fimg的最小、最大值: ", fimg.min(),fimg.max(),fimg.size,fimg.shape)  
  
fimgFix = line2Scale(fimg)  
# #fimgFix = np.uint8(fimg)  
fimgFix2 = np.uint8(np.ones\_like(fimg) \* 255 - fimg)  
#  
# cv.imshow("fix1", fimgFix)  
# cv.imshow("fix2", fimgFix2)  
  
figure, axes = plt.subplots(2, 2)  
axes[0, 0].set\_title("orgImg")  
axes[0, 0].imshow(img, cmap="gray")  
axes[0, 1].set\_title("Fourier frequency image by linal exchange to [0,255]")  
axes[0, 1].imshow(fimgFix, cmap="gray")  
  
axes[1, 0].set\_title("Fourier frequency original image hist")  
axes[1, 0].hist(fimgFix)  
  
axes[1, 1].set\_title("Fourier frequency original image convert")  
axes[1, 1].imshow(fimgFix2, cmap="gray")  
  
fliterFigure, fliterAxes = plt.subplots(2, 3)  
ipos = 0  
print("star lower filter ")  
#5,50,150 低通巴特沃斯滤波  
for D0 in np.array([5,50,150]):  
 lowerImg = func\_LowerFilter(fshift, D0, 2)  
 fliterAxes[0, ipos].set\_title("lower filter by %d" %D0)  
 fliterAxes[0, ipos].imshow(lowerImg, cmap="gray")  
 ipos += 1  
  
ipos = 0  
print("start higher filter ")  
#5,50,150 高通巴特沃斯滤波  
for D0 in np.array([5,50,150]):  
 lowerImg = func\_LowerFilter(fshift, D0, 2, isLower=False)  
 fliterAxes[1, ipos].set\_title("Higher filter by %d" %D0)  
 fliterAxes[1, ipos].imshow(lowerImg, cmap="gray")  
 ipos += 1

# 第五周作业：对一副图像加噪，进行几何均值，算术均值，谐波，逆谐波处理

处理步骤：

1）给图像增加10%的椒盐噪声；添加方差为0.4的高斯噪声的叠加图片。（此部分同第三章，为做区别，本次做高斯与椒盐的叠加噪声）

2）使用几何平均，f(x,y) = （prod(g(s,t))^(1/mn)

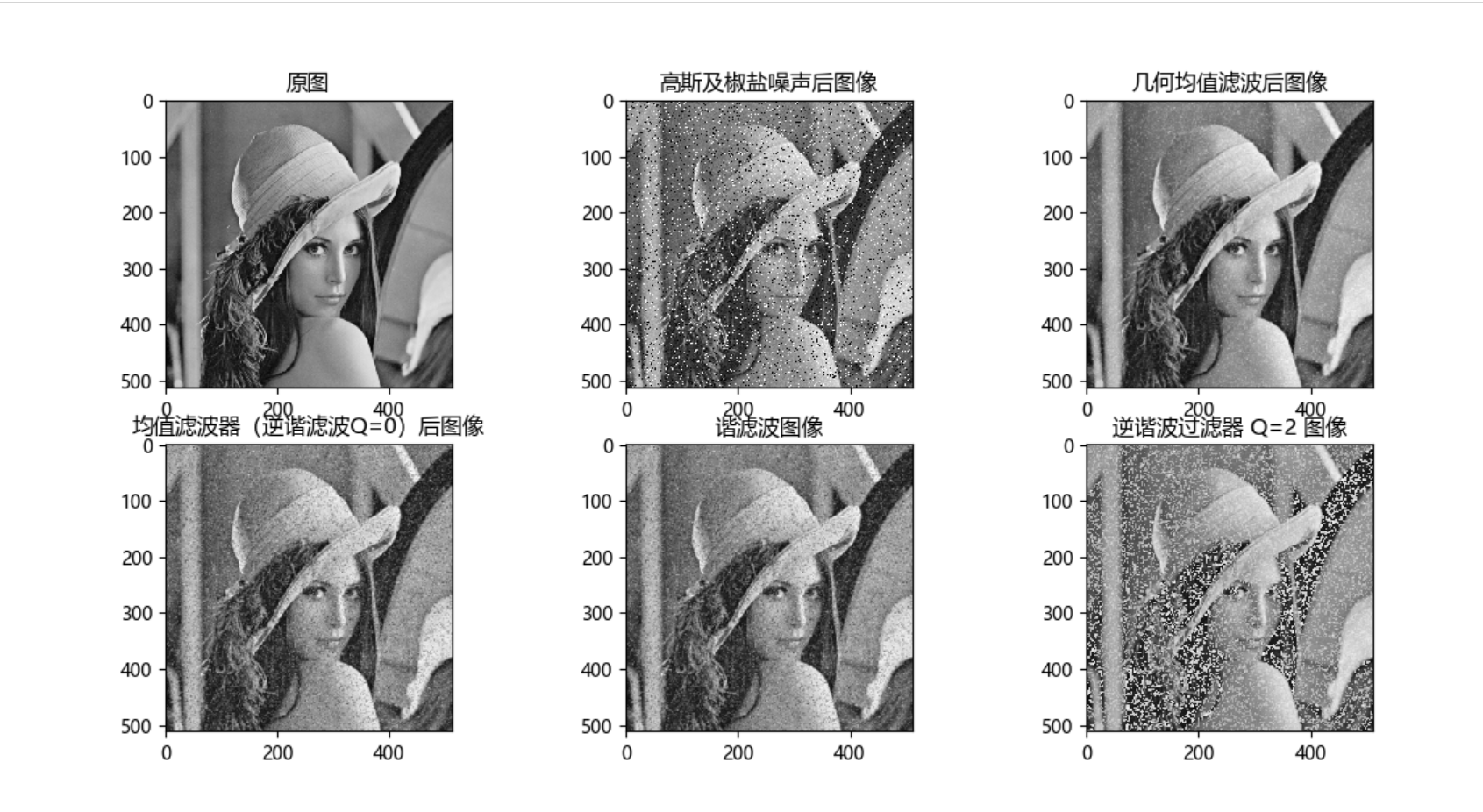
3）编制逆谐波函数，当Q = 0 退化为算术均值，当Q=-1时，退化为谐波均值滤波器

(s,t) ∈ Sxy f(x,y)= sum g(s,t)^(Q+1) / sum g(s,t)^Q

取Q=0进行算术均值计算

取Q=-1 进行谐波计算

取Q=2进行逆谐波计算



代码如下：

img = cv.imread("./source/lena.jpg",cv.IMREAD\_GRAYSCALE)  
  
#高斯噪声  
noise=np.random.normal(0,0.04,size=img.shape) #产生高斯（正态）分布噪声，期望为0，方差=0.04 （方差过大1，图像噪声太多，方差过小0，噪声太少）  
gaussImg = np.uint8(np.clip((noise + img / 255), 0, 1) \* 255) #范围在0，1之间，之外的  
#cv.imshow("gauss",np.hstack((img,gaiImg)))  
  
saltImg = addSalt(gaussImg, 0.1)  
  
##滤波卷积操作：  
# 定义函数指定一个原图，使用规定的3\*3模板进行滤过操作。  
# 采用复制边方式进行处理。  
# method三种方法：1） sum:求和平均数，2）prod：求算术平均数，3）median：求中值。4)antiharm: 逆谐滤波器  
def fn\_filter33 (orgImg, filterImg, method="sum", Q = 0):  
 rows,cols = orgImg.shape  
 #为原图扩展边，以便于计算，扩展方式为边复制.  
 extentImg = np.zeros((rows+2, cols+2), dtype=np.uint8)  
 extentImg[1:rows+1, 1:cols+1] = orgImg #中间部分用原图填充  
 extentImg[0, 1:cols+1]=orgImg[0, :] #第一行复制img的第一行  
 extentImg[rows+1, 1:cols+1]=orgImg[rows-1, :]#最后一行复制img的最后一行  
 extentImg[1:rows+1, 0]=orgImg[:, 0] #第一列复制过来  
 extentImg[1:rows+1, cols+1]=orgImg[:, cols-1] #最后一列  
 #补齐四个角  
 extentImg[0, 0]=orgImg[0, 0]  
 extentImg[0, cols+1] = orgImg[0, cols-1]  
 extentImg[rows+1, 0] = orgImg[rows-1, 0]  
 extentImg[rows+1, cols+1] = orgImg[rows-1, cols-1]  
  
 ##均值滤波图像,中值滤波图像结果  
 out\_img = np.zeros(orgImg.shape)  
  
 #print(orgImg.shape, "开始卷积滤波", "method = %s"%method)  
  
 if method == "sum" :  
 for i in np.arange(1, rows+1):  
 for j in np.arange(1, cols+1):  
 rect = filterImg\*extentImg[i-1:i+2, j-1:j+2] #i+2的原因是切片时截至数不包括在里面  
 out\_img[i-1, j-1] = np.sum(rect) #np.uint8(abs(np.sum(rect))) #均值： 如果是十字星型模板，则不能使用mean方法，需要计算sum，除以十字星模板的sum  
  
 elif method == "prod": #几何平均数  
 mn = np.count\_nonzero(filterImg)  
 print("prod method, mn is :", mn)  
 for i in np.arange(1, rows + 1):  
 for j in np.arange(1, cols + 1):  
 rect = filterImg \* extentImg[i - 1:i + 2, j - 1:j + 2] # i+2的原因是切片时截至数不包括在里面  
 ##剔除0点，否则图片会被黑色污染。  
 tempLine = np.ravel(rect)  
 tempLine = tempLine[tempLine > 0] #剔除0值  
 prod = np.power(tempLine, 1/(tempLine.size) ).cumprod()[-1] #np.cumprod(tempRect)[-1]  
 #print(i, j, prod)  
 out\_img[i - 1, j - 1] = np.uint8( prod ) # 几何平均数  
 elif method == "median":  
 for i in np.arange(1, rows + 1):  
 for j in np.arange(1, cols + 1):  
 rect = filterImg \* extentImg[i - 1:i + 2, j - 1:j + 2] # i+2的原因是切片时截至数不包括在里面  
 out\_img[i - 1, j - 1] = np.uint8(np.median(rect)) # 中值  
 elif method == "antiharm": #逆谐滤波器  
 if Q >=0 :  
 for i in np.arange(1, rows + 1):  
 for j in np.arange(1, cols + 1):  
 rect = filterImg \* extentImg[i - 1:i + 2, j - 1:j + 2] # i+2的原因是切片时截至数不包括在里面  
 val = np.power(rect,Q+1).sum()/np.power(rect,Q).sum()  
 out\_img[i - 1, j - 1] = np.uint8(val) #  
 elif Q <= -1:  
 pind = Q\*-1  
 for i in np.arange(1, rows + 1):  
 for j in np.arange(1, cols + 1):  
 rect = filterImg \* extentImg[i - 1:i + 2, j - 1:j + 2] # i+2的原因是切片时截至数不包括在里面  
 val = np.power(rect,pind).sum()/np.power(rect,pind-1).sum()  
 out\_img[i - 1, j - 1] = np.uint8(val) # 中值  
 return out\_img  
####2. 平滑滤波处理。  
#3\*3的均值滤波器  
templateMean = np.array([[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]])  
  
#中值滤波  
print("start sharpen image by median algorithm")  
medianImg = fn\_filter33(saltImg, templateMean, method="median")  
print("start sharpen image by mean algorithm")  
meanImg = fn\_filter33(saltImg, templateMean, method="antiharm", Q=0)  
prodMeanImg = fn\_filter33(saltImg,templateMean, method="prod")  
print(prodMeanImg)  
#谐滤波  
harmImg = fn\_filter33(saltImg, templateMean, method="antiharm",Q=-1)  
#逆谐滤波 Q为正，清除胡椒噪声。  
antiharmImg = fn\_filter33(saltImg,templateMean,method="antiharm", Q=2)

# 第六周作业：提取一副图像中的红色，用HSI方式。

步骤：

1）使用opencv的cvtColor方法将RGB图像转换成HSL图像，并提取其中的H，S分量，从H中获取红色（范围在【0，10】，【156，180】设置为255，其他为0。

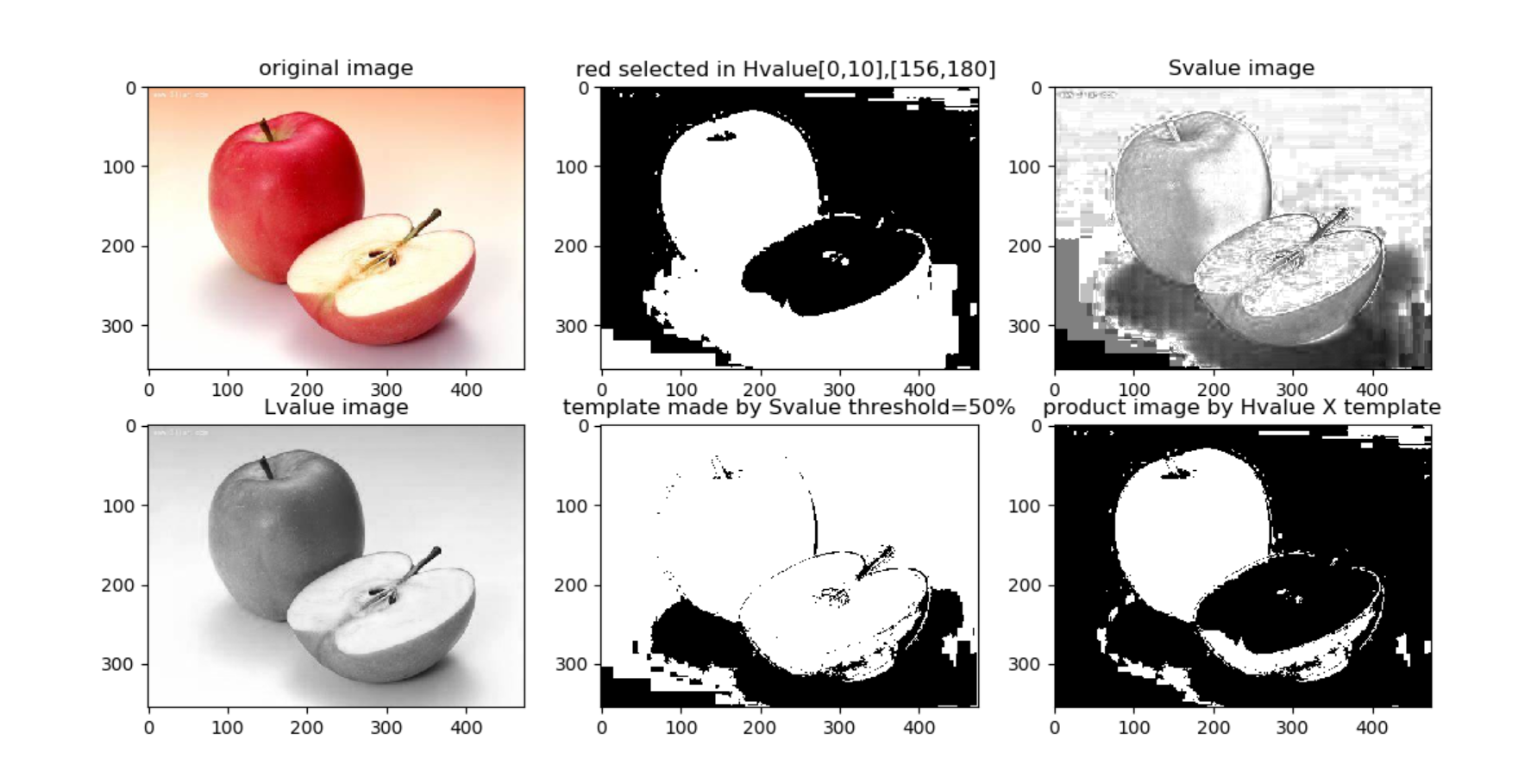
色相范围参考：

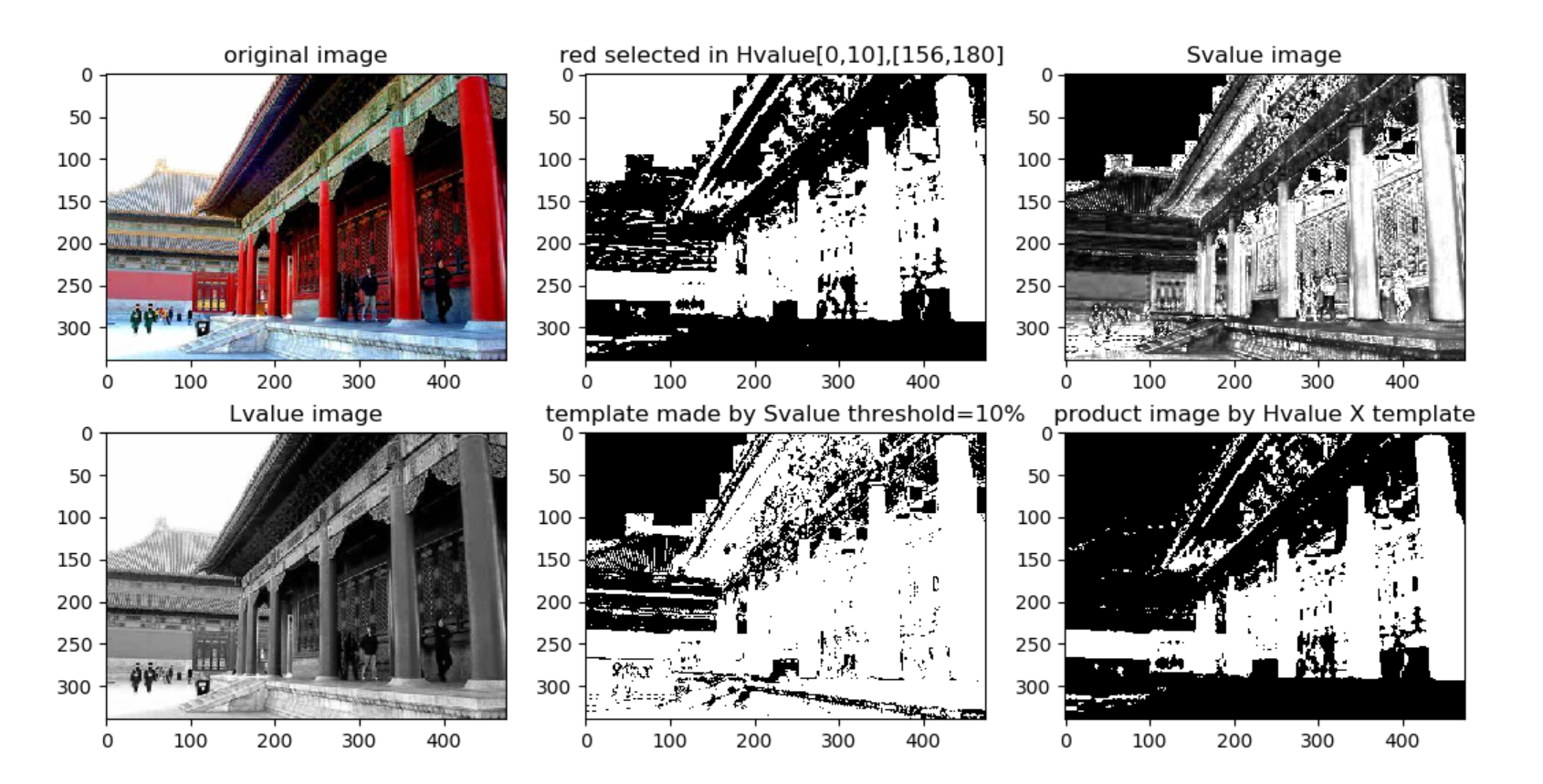
https://zh.wikipedia.org/wiki/%E8%89%B2%E7%9B%B8#%E8%89%B2%E7%9B%B8%E7%8E%AF, https://www.cnblogs.com/wangyblzu/p/5710715.html）。

注：H的正常范围是【0，360】，opencv为了适配到【0，255】范围，H的范围减半。详见：https://docs.opencv.org/master/de/d25/imgproc\_color\_conversions.html#color\_convert\_rgb\_hls

2) 使用饱和度分量制作掩模template（二值化，阈值取最大饱和度的50%，饱和度低于50%的点舍去）

3) H分量与template进行卷积，得到提取的图片。





代码如下：

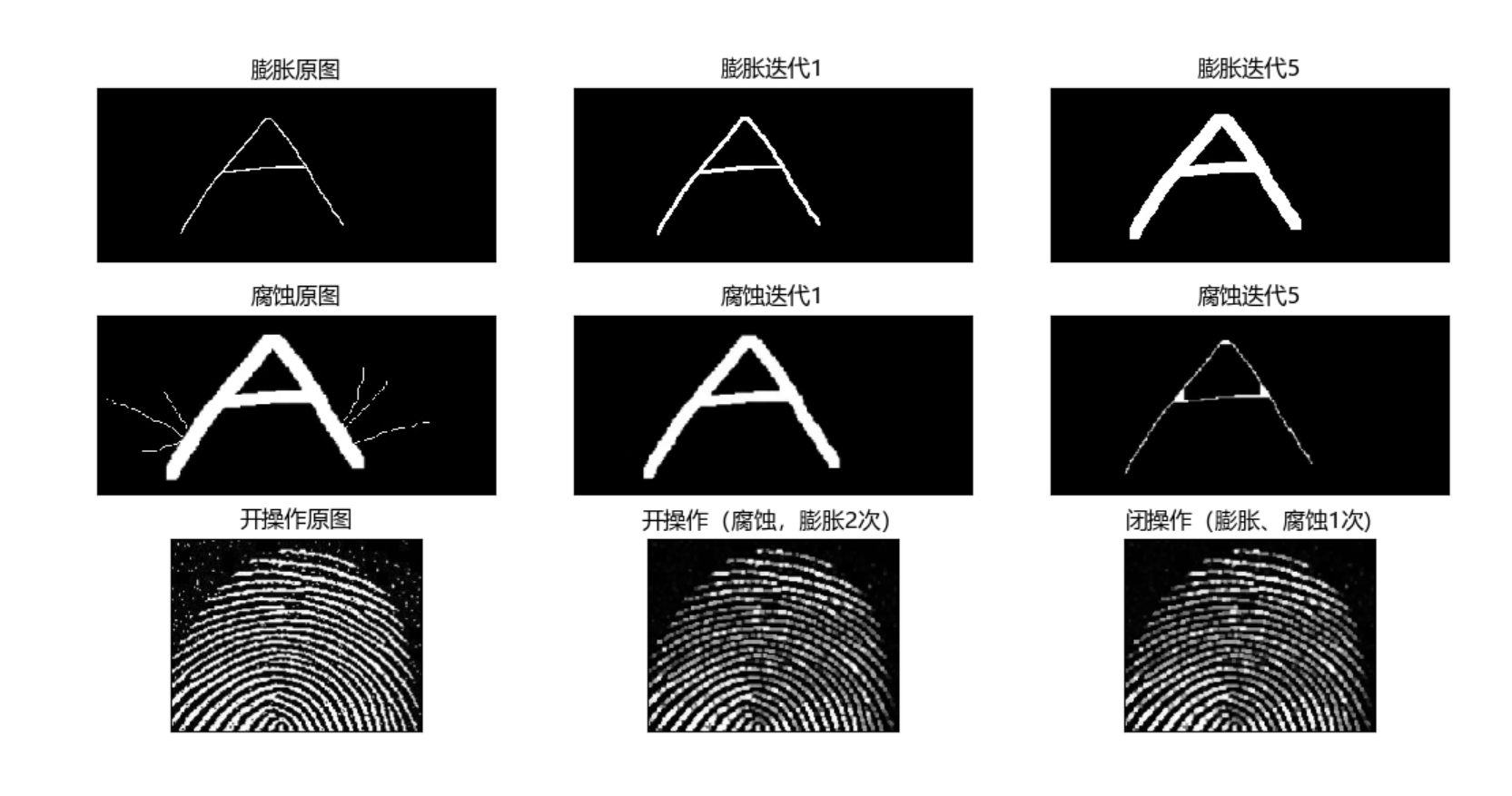
img\_BGR = cv.imread("./source/redApple.jpg") #读取格式默认是BGR格式  
img\_HLS = cv.cvtColor(img\_BGR, cv.COLOR\_BGR2HLS)  
HvalueTemp, Lvalue, Svalue = cv.split(img\_HLS) #获得三个分量  
#从H分量中提取红色  
cond = np.logical\_or(np.logical\_and(HvalueTemp>= 0, HvalueTemp<= 10), np.logical\_and(HvalueTemp>= 156, HvalueTemp<= 180))  
Hvalue = np.where(cond, 255, 0)  
  
  
threshold = Svalue.max()\*0.5 #舍去饱和度低于最大饱和度50%的点。  
template = np.where(Svalue > threshold, 1, 0)  
product =np.uint8(Hvalue\*template)

# 第七周：对一副二值图像进行膨胀、腐蚀、开、闭操作

步骤：

1. 读取二值化图像。
2. 生成5X5的核，对图像进行膨胀操作，外围变大，与操作，全为0的为0，否则为1
3. 生成3X3的核，对图像进行腐蚀操作，剔除毛刺，与操作，存在0的为0，否则为1
4. 开操作：先腐蚀后膨胀 (A-B)+B 剔除毛边后复原
5. 闭操作：先膨胀后腐蚀 (A+B)-B

参考：https://blog.csdn.net/Eastmount/article/details/83581277



代码如下：

#膨胀操作  
img = cv.imread("./source/dilate0.jpg", cv.IMREAD\_GRAYSCALE)  
  
dilateKernal = np.ones((5,5), dtype=np.uint8)  
dilateImg1 = cv.dilate(img, dilateKernal)  
dilateImg5 = cv.dilate(img, dilateKernal, iterations=5)  
  
#腐蚀操作  
img2 = cv.imread("./source/erode0.jpg", cv.IMREAD\_GRAYSCALE)  
erodeKernal = np.ones((3,3), dtype=np.uint8)  
erodeImg1 = cv.erode(img2, erodeKernal)  
erodeImg2 = cv.erode(img2, erodeKernal, iterations=5)  
##开闭操作。  
#开操作：  
fingerImg = cv.imread("./source/fingerprint.jpg", cv.IMREAD\_GRAYSCALE)  
openErodeFingerImg = cv.erode(fingerImg,erodeKernal, iterations=2)  
openFingerImg = cv.dilate(openErodeFingerImg, erodeKernal, iterations=2)  
axes[2, 0].imshow(fingerImg, cmap="gray"), axes[2, 0].set\_title("开操作原图")  
axes[2, 1].imshow(openFingerImg, cmap="gray"), axes[2, 1].set\_title("开操作（腐蚀，膨胀2次）")  
#对开操作图像进行闭操作。  
axes[2, 2].imshow( cv.erode( cv.dilate(openFingerImg, erodeKernal, iterations=1), erodeKernal), cmap="gray"), axes[2,2].set\_title("闭操作（膨胀、腐蚀1次)")