**数字图像处理**

**一、编程环境说明**

* Python>=3.7
* scikit-image

一个python的图像处理包，支持使用numpy作为数据结构对图像进行处理。

**二、题目**

1. **打开图像，显示图像，存储图像；对一张图像进行缩放，观察其分辨率，降低灰度分辨率**

**打开灰度图**

原图:



其在内存中是以二维矩阵数组的形式保存，打开后读取灰度值如下：

[[156 157 160 ... 152 152 152]

[156 157 159 ... 152 152 152]

[158 157 156 ... 152 152 152]

...

[121 123 126 ... 121 113 111]

[121 123 126 ... 121 113 111]

[121 123 126 ... 121 113 111]]

**打开彩色图**

原图:



在内存中显示

[[[143 120 104]

[143 120 104]

[141 118 102]

...

[ 45 27 13]

[ 45 27 13]

[ 45 27 13]]

[[146 123 107]

[145 122 106]

[143 120 104]

...

[ 46 29 13]

[ 45 29 13]

[ 47 30 14]]

[[148 126 112]

[147 125 111]

[146 122 109]

...

[ 48 28 17]

[ 49 29 18]

[ 50 30 19]]

...

[[ 92 58 30]

[105 71 43]

[132 98 71]

...

[172 145 138]

[172 145 138]

[172 145 138]]

[[128 92 60]

[139 103 71]

[134 95 64]

...

[166 142 132]

[166 142 132]

[167 143 133]]

[[139 103 71]

[127 88 57]

[125 86 53]

...

[161 137 127]

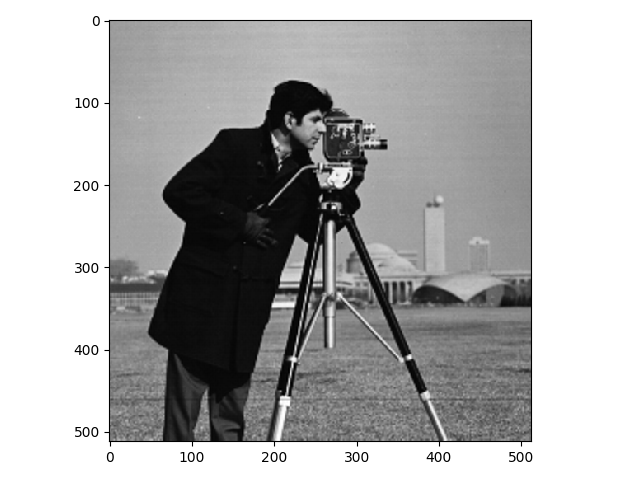
[161 137 127]

[162 138 128]]]

显然，默认rgb模式下彩色图形的每个像素值用一个三元组表示三个通道下的灰度值。

**显示图像**：

以刚才的灰度图为例，将刚才内存中的灰度图显示到屏幕上后为：



最后将其保存到文件中，并重新命名。

**代码如下：**

from skimage import io, util

def imread(fname, as\_gray=False):

"""根据文件名打开图像, 每个像素以8位保存，即灰度范围0-255"""

img = io.imread(fname, as\_gray=as\_gray)

return util.img\_as\_ubyte(img)

def imread\_gray(fname):

"""打开一张灰度图"""

return imread(fname, True)

def imread\_color(fname):

"""打开一张彩色图"""

return imread(fname, False)

def imshow(img, \*\*kwargs):

"""显示图像"""

io.imshow(img, \*\*kwargs)

io.show()

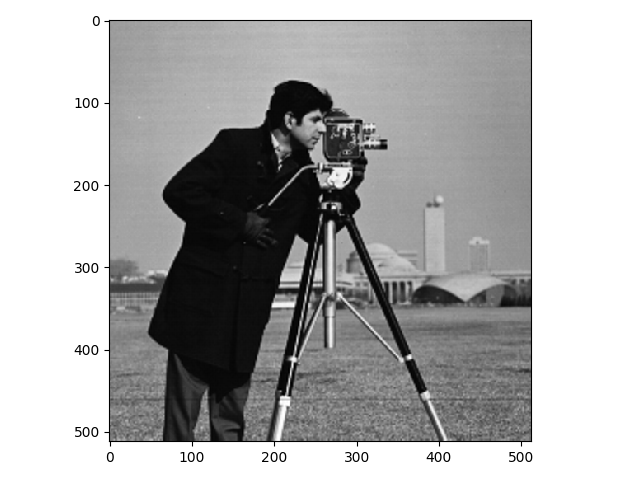
def imsave(fname, img):

"""保存一张图片，保存文件名为fname"""

io.imsave(fname, img)

**缩放：**

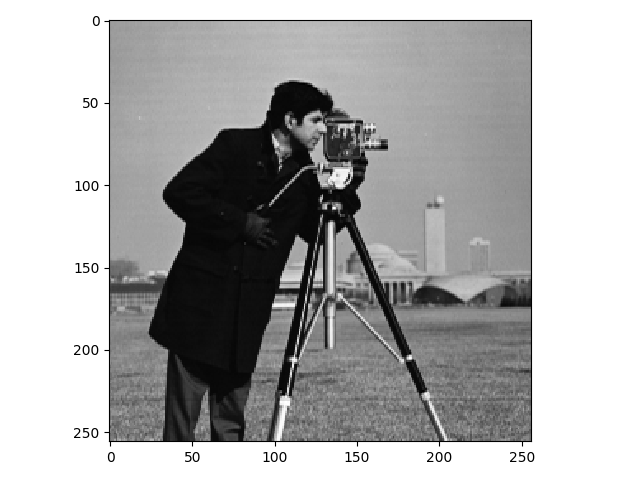
**原图：**



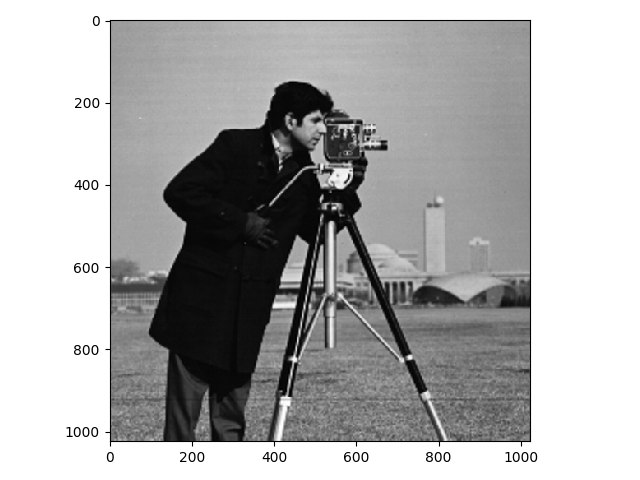
**处理方法：**

策略是放大缩小主要是对采样点进行控制。对于缩小来说，就是减少采样的行和列，对于放大，就采取重复行和列的采样。

**缩小一半后：**



**放大一倍后：**



根据坐标刻度显然可以知道图像分别缩小和放大了一倍

**缩放代码如下：**

import numpy as np

def scale(img, n):

"""

缩放图像，n为放大或缩小的倍数

缩小一半scale(img, 0.5)

放大一倍scale(img, 2)

"""

shape = img.shape

rows, cols = shape

arr = np.array([

[img[int(i/n), int(j/n)] for j in range(0, int(cols\*n))]

for i in range(0, int(rows\*n))

])

return np.array(arr)

**降灰度级：**

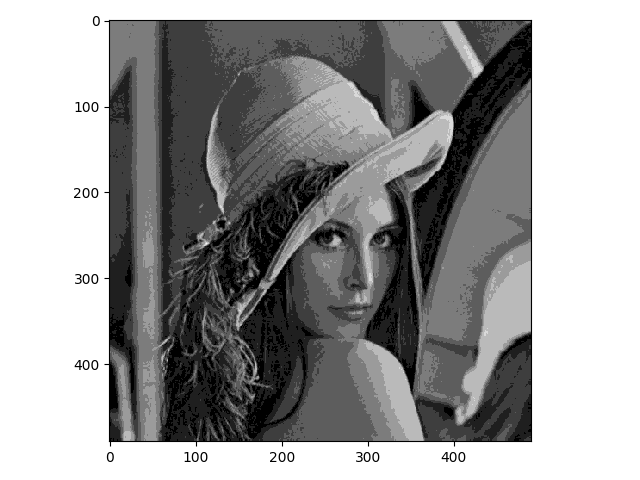
处理方法：

根据灰度级将图片的灰度值区间归并为对应的几个灰度值。

**原图：**



**降级后的图片（降为8个灰度级）：**



可以看到，降级后的图片细节部分变化不大，但层次变差，甚至出现了假轮廓现象。

**代码如下：**

def degray(img, level):

"""

降灰度级

level: 灰度级数，如降为8级则level为8

"""

if level < 1 or level > 255:

return

rows, cols = img.shape

base = int(255 / level)

for i in range(rows):

for j in range(cols):

img[i, j] = int(img[i, j] / base)\*base

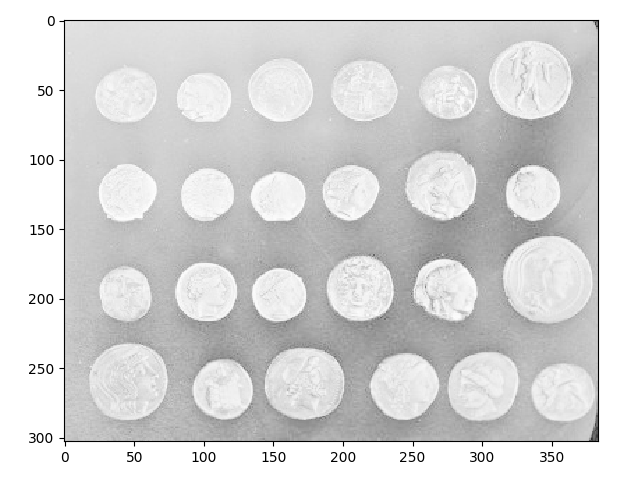
return img

1. **打开一副低对比度图像，拉伸其图像，直方图均衡**

**原图：**



**对数变换后结果：**



**对数变换代码如下：**

import numpy as np

def log\_enhance(img):

"""对数变换"""

max\_val = np.log(1+img.max())

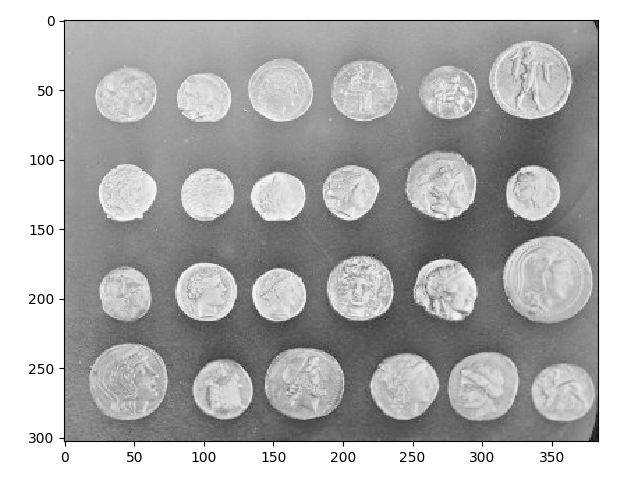
c = 255/max\_val if max\_val else 0

img = c \* np.log(1+img)

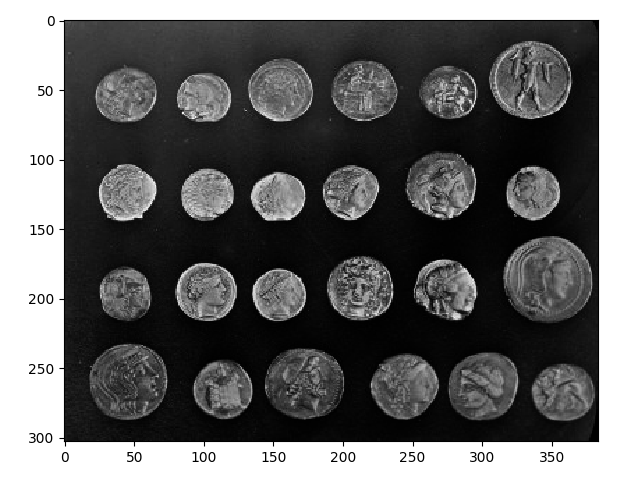
img = np.array(img, dtype=np.uint8)

return img

**幂次变换（指数为0.5）的结果：**



**幂次变换（指数为2）的结果：**

****

**幂次变换代码如下：**

import numpy as np

def exp\_enhance(img, g=0.5):

"""幂次变换, 指数g默认为0.5"""

img = (img/255) \*\* g

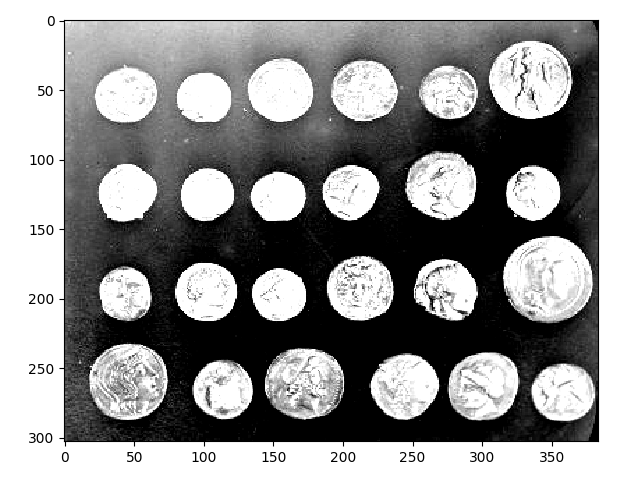
c = 255/img.max()

img \*= c

img = np.array(img, dtype=np.uint8)

return img

**线性拉伸(将50-150区间拉伸到0-255)后的结果：**



**代码如下：**

def linear\_convert(img, origin=(50, 150), ext=(0, 255)):

"""线性变换到整个灰度级"""

rows, cols = img.shape

for i in range(rows):

for j in range(cols):

val = img[i, j]

if val < origin[0]:

img[i, j] = val\*ext[0]/origin[0]

elif val > origin[1]:

img[i, j] = (val-origin[1])\*(255-ext[1]) / \

(255-origin[1]) + ext[1]

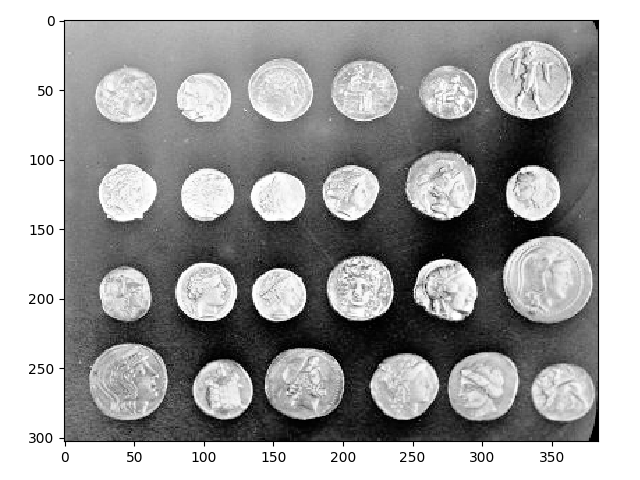
else:

img[i, j] = (val-origin[0])\*(ext[1]-ext[0]) / \

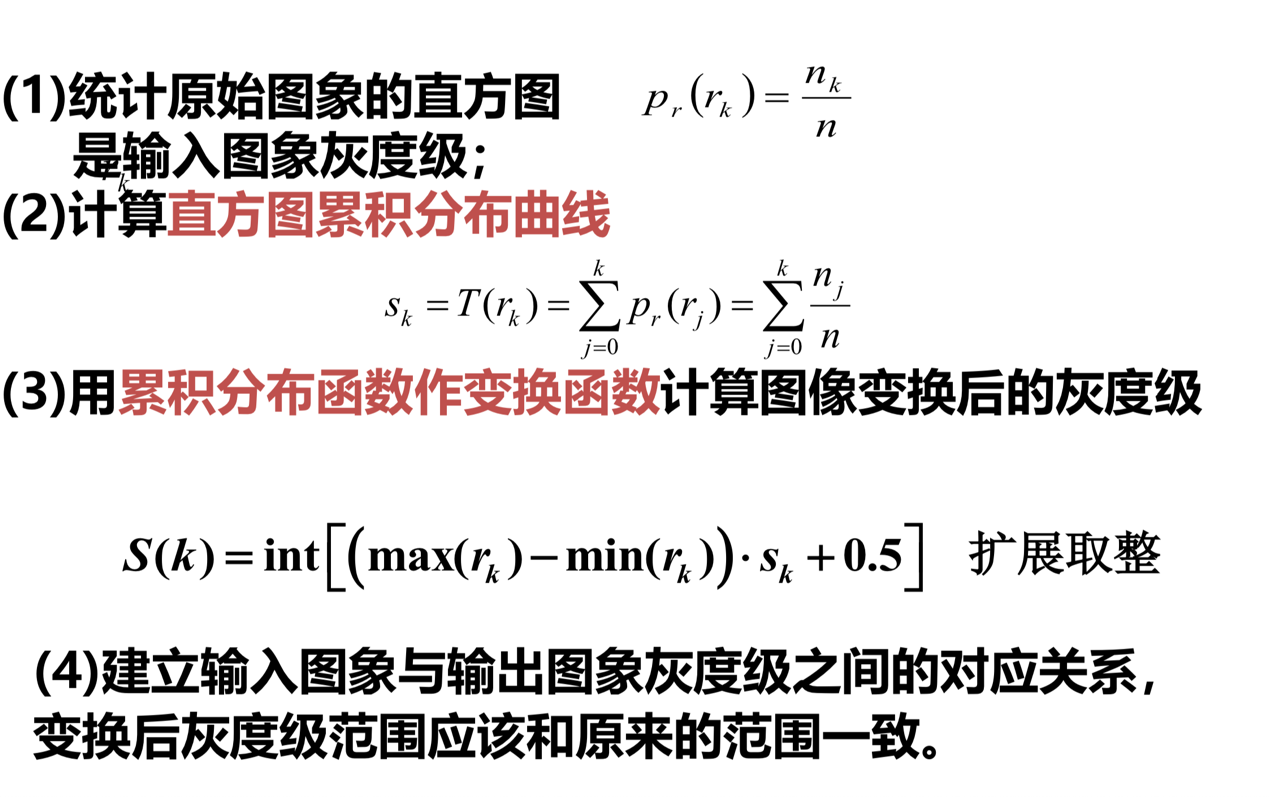
(origin[1]-origin[0]) + ext[0]

return img

**直方图均衡化后的结果：**



**处理方法：**



**代码如下：**

from collections import Counter

def equalize\_hist(img, nbins=256):

"""

直方图均衡

1.统计灰度值

2.计算概率密度

3.计算累计分布

4.扩展求整int[(max-min)\*cumulative\_p+0.5]

5.映射关系

"""

arr = img.flatten()

r = Counter()

# 统计原始图像灰度级个数

for v in arr:

r[v] += 1

# 计算概率密度，并直接求出累计分布函数

rows, cols = img.shape

total = rows \* cols

reduced\_val = 0

cumulative\_dist = [0] \* nbins

for k in range(nbins):

reduced\_val += r[k]

cumulative\_dist[k] = reduced\_val / total

# 扩展取整

sk\_map = {}

for k in range(nbins):

sk\_map[k] = int((nbins-1)\*cumulative\_dist[k] + 0.5)

# 根据灰度映射关系修改灰度值

for i in range(rows):

for j in range(cols):

img[i, j] = sk\_map[img[i, j]]

return img

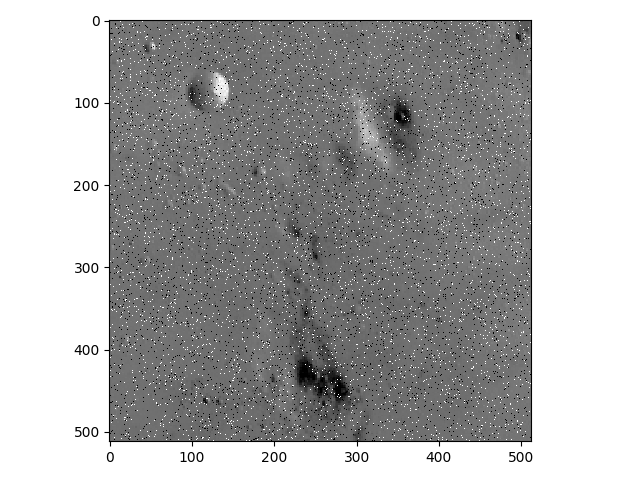
1. **对一副图像加噪声，进行平滑，锐化作用**

**图像加噪**

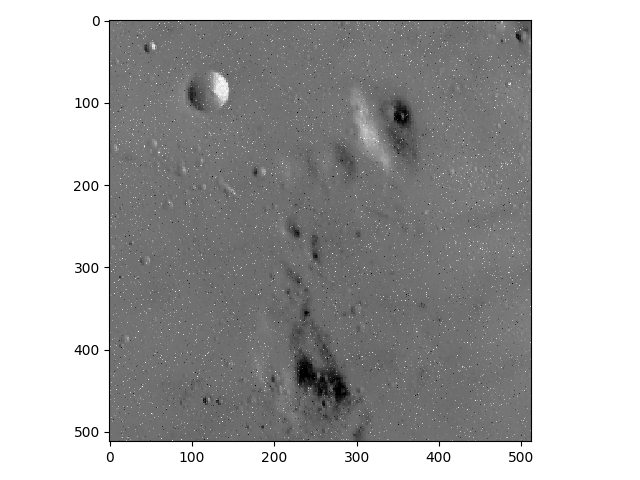
**原图：**

****

**添加椒盐噪声（5%密度）后：**



**添加高斯噪声（5%密度）后：**



**代码如下：**

import numpy as np

import random

def \_rand\_pos(shape):

x = np.random.randint(0, shape[0])

y = np.random.randint(0, shape[1])

return x, y

def add\_salt\_noise(img, percent=0.05):

"""

添加密度为percent的椒盐噪声

"""

shape = img.shape

rows, cols = shape

n = int(percent \* rows \* cols)

for i in range(n):

x, y = \_rand\_pos(shape)

val = np.random.randint(0, 2) \* 255

if img.ndim == 2:

img[x, y] = val

elif img.dim == 3:

for z in range(0, 3):

img[x, y, z] = val

return

def add\_gaussian\_noise(img, percent=0.05):

"""

添加密度为percent的高斯噪声

"""

shape = img.shape

rows, cols = shape

n = int(percent \* rows \* cols)

for i in range(n):

x, y = \_rand\_pos(shape)

val = img[x, y] + random.gauss(20, 40)

if val < 0:

val = 0

elif val > 255:

val = 255

if img.ndim == 2:

img[x, y] = val

elif img.dim == 3:

for z in range(0, 3):

img[x, y, z] = val

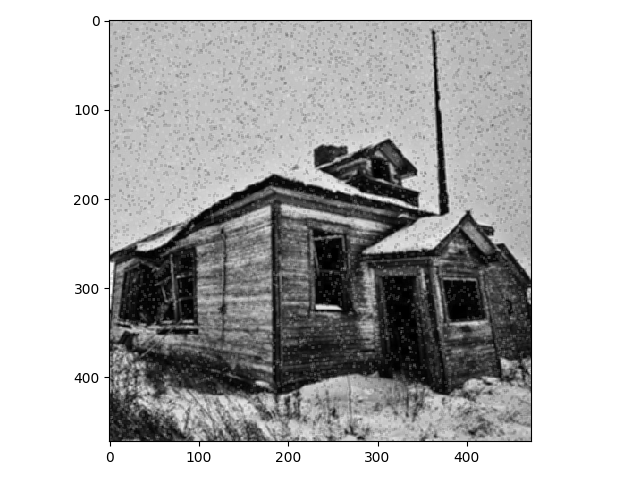
return

**平滑处理**

**原图：**



**3\*3九点均值滤波平滑处理后结果：**



**代码如下：**

def linear\_filter(img):

"""3\*3九点均值滤波"""

operator = [

[1, 1, 1],

[1, 1, 1],

[1, 1, 1]

]

size = 9

dim = len(operator)

shape = img.shape

new\_img = immake(shape) # 构造新像素矩阵

rows, cols = shape

for i in range(rows):

for j in range(cols):

r = 0

for m in range(dim):

# 超出边界行的元素按边界值处理

x = max(i+m-1, 0)

x = min(x, rows-1)

for n in range(dim):

# 超出边界列的元素按边界值处理

y = max(j+n-1, 0)

y = min(y, cols-1)

# 与相应模板值做卷积

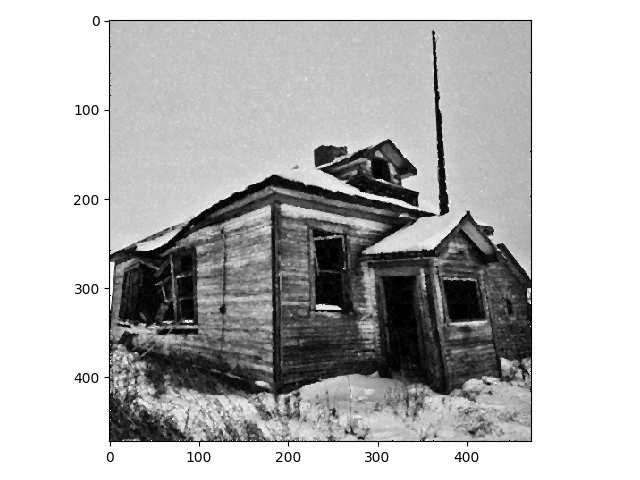
w = operator[m][n]

r += w \* img[x, y]

new\_img[i, j] = r/size

return new\_img

**九点中值滤波平滑处理后结果：**



**代码如下：**

def median\_filter(img):

"""九点中值滤波"""

dim = 3

n = dim \* dim

mid\_pos = int((n+1)/2)

margin = int((dim-1)/2)

shape = img.shape

new\_img = immake(shape) # 构造新像素矩阵

rows, cols = shape

for i in range(0, rows):

for j in range(0, cols):

# 边缘像素不处理

if i <= margin or i >= rows-margin or j <= margin or j >= cols-margin:

new\_img[i, j] = img[i, j]

continue

# 邻域像素排序后取中值

arr = [

img[x, y]

for x in range(i-margin, i+margin+1)

for y in range(j-margin, j+margin+1)

]

arr.sort()

new\_img[i, j] = arr[mid\_pos]

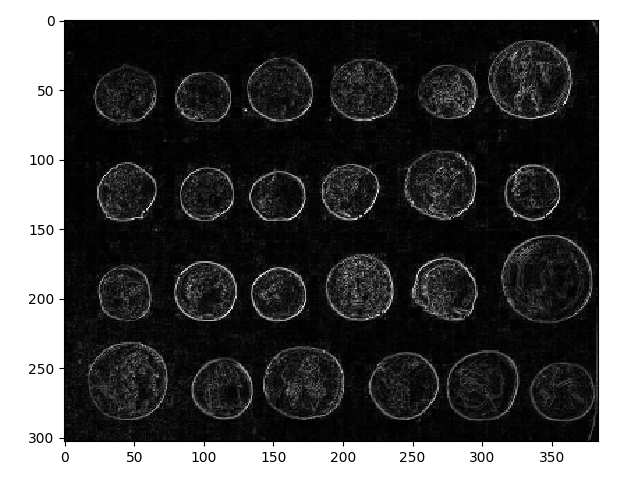
return new\_img

**锐化处理**

**原图：**



**梯度锐化效果如下：**

****

**代码如下：**

def grad\_sharpen(img):

"""梯度法锐化滤波"""

shape = img.shape

new\_img = immake(shape)

rows, cols = shape

for i in range(0, rows):

for j in range(0, cols):

grad = 0

if i < rows - 1:

grad += abs(int(img[i, j]) - int(img[i+1, j]))

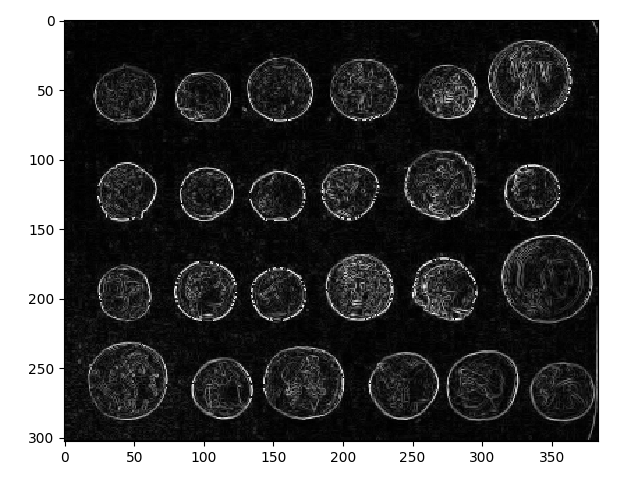
if j < cols - 1:

grad += abs(int(img[i, j]) - int(img[i, j+1]))

new\_img[i, j] = grad

return new\_img

**Roberts算子锐化效果如下：**



**代码如下：**

def roberts\_sharpen(img):

"""Roberts交叉差分锐化滤波"""

shape = img.shape

new\_img = immake(shape)

rows, cols = shape

for i in range(0, rows-1):

for j in range(0, cols-1):

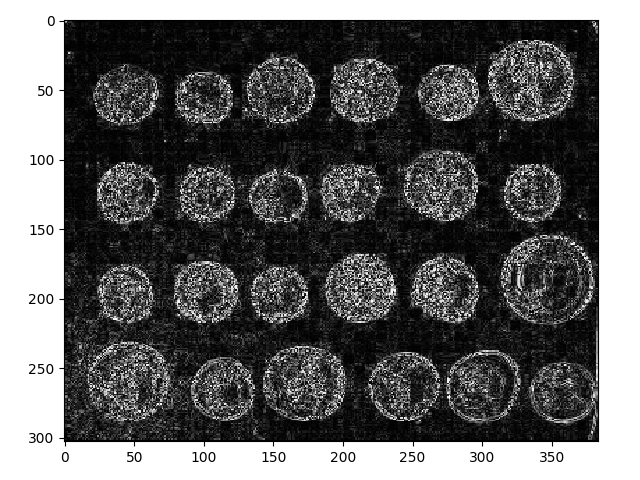
grad = abs(int(img[i, j]-int(img[i+1, j+1]))) + \

abs(int(img[i+1, j])-int(img[i, j+1]))

new\_img[i, j] = grad

return new\_img

**拉普拉斯锐化效果如下：**



**代码如下：**

def laplacian\_sharpen(img):

"""3\*3中心点为-8的掩模拉普拉斯锐化滤波"""

operator = [

[1, 1, 1],

[1, -8, 1],

[1, 1, 1],

]

shape = img.shape

new\_img = immake(shape)

rows, cols = shape

for i in range(1, rows-1):

for j in range(1, cols-1):

s = 0

for m in range(0, 3):

for n in range(0, 3):

w = operator[m][n]

x = i-1+m

y = j-1+n

s += w \* img[x][y]

new\_img[i, j] = abs(int(s))

return new\_img

1. **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**

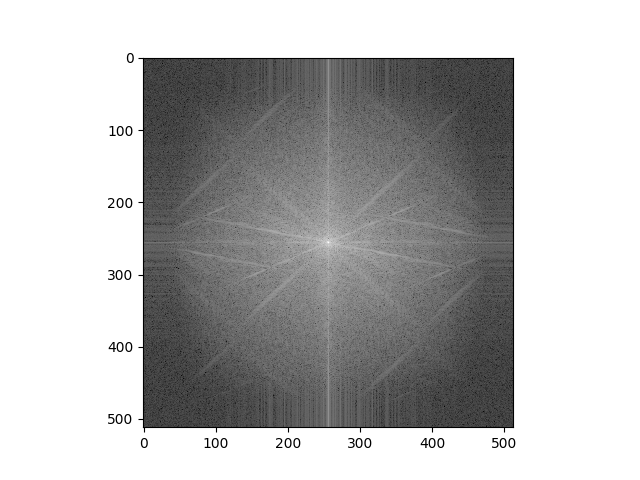
**原图：**



**处理方法：**

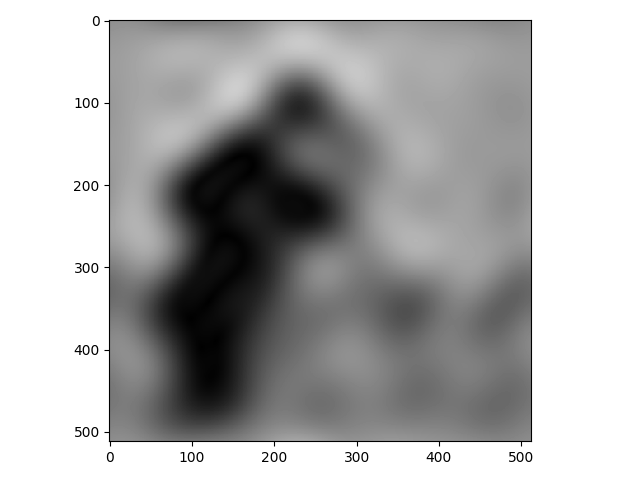
用傅里叶变换把空域像素转成频率域频谱，并进行平移变换，使中心到外围逐渐由低通分量（低频）变化到高通分量（高频），并通过设置阈值频率达到平滑或锐化的效果。

**变换后的频谱图：**



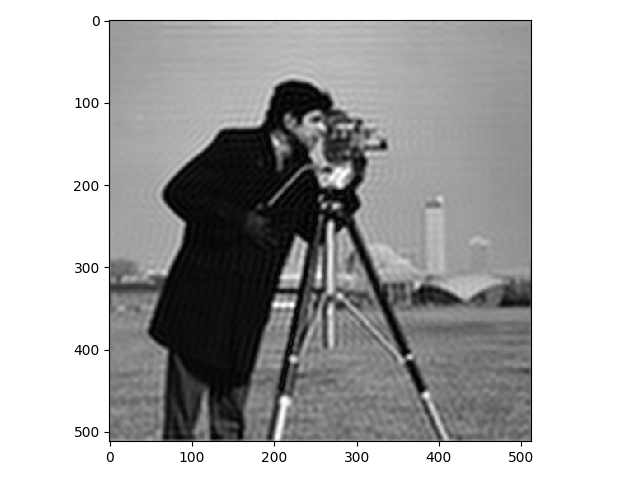
**平滑效果：**

**取其5为截至频率：**

****

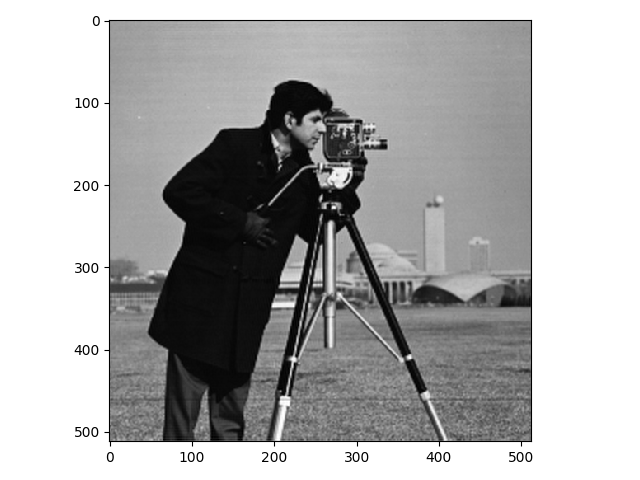
可以看到，阈值过低显然会过滤掉很多细节部分

**取其50为截至频率：**



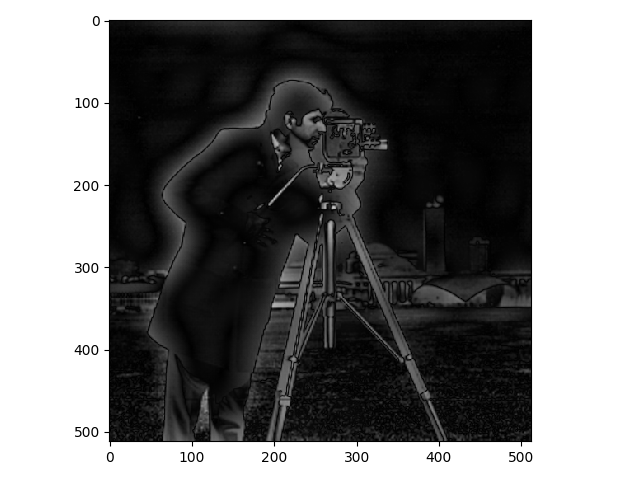
明显有振铃现象产生

**取其150为截至频率：**

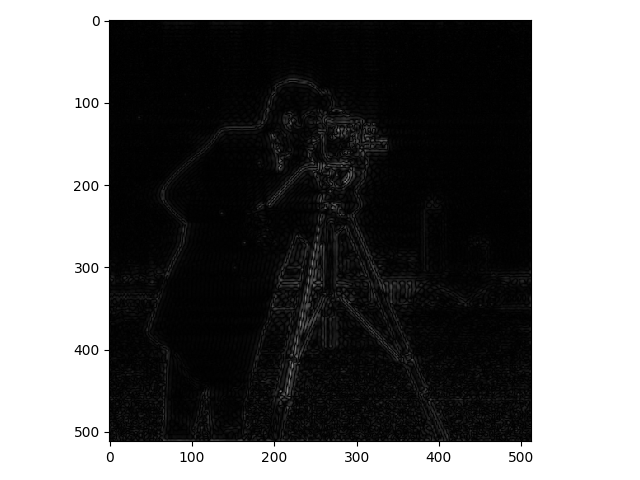


**锐化效果：**

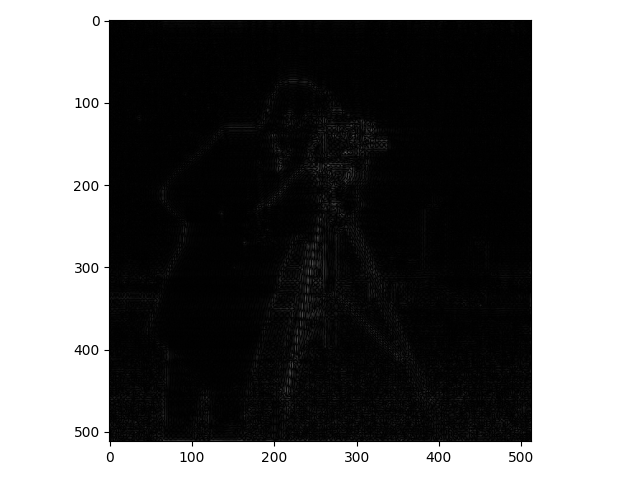
**取其5为截至频率：**

****

**取其50为截至频率：**

****

**取其150为截至频率：**



**代码如下：**

from skimage import img\_as\_float, img\_as\_ubyte

import numpy as np

import matplotlib.pyplot as plt

def imfft(img):

"""傅里叶变换"""

img = img\_as\_float(img)

f = np.fft.fft2(img)

fshift = np.fft.fftshift(f)

return fshift

def imifft(img):

"""傅里叶反变换"""

return np.fft.ifft2(np.fft.ifftshift(img))

def fftshow(img):

"""显示傅里叶变换后频谱图"""

img = np.log(np.abs(img))

plt.subplot(111)

plt.imshow(img, 'gray')

plt.show()

return img

def fft\_smooth(img, val):

"""

频率域低通平滑

1.傅里叶变换

2.根据阈值val(eg. 5, 50, 150)过滤，保留低通分量

3.傅里叶反变换

4.超出数值部分像素处理

5.转换回255灰度级

"""

img = imfft(img)

rows, cols = img.shape

crow, ccol = int(rows/2), int(cols/2)

mask = np.zeros(img.shape)

mask[crow-val:crow+val, ccol-val:ccol+val] = 1

img = np.abs(imifft(img\*mask))

rows, cols = img.shape

for row in range(rows):

for col in range(cols):

pix = img[row, col]

if pix > 1:

img[row, col] = 1

elif pix < -1:

img[row, col] = -1

img = img\_as\_ubyte(img)

return img

def fft\_sharpen(img, val):

"""

频率域高通锐化

1.傅里叶变换

2.根据阈值val(eg. 5, 50, 150)过滤，保留高通分量

3.傅里叶反变换

4.超出数值部分像素处理

5.转换回255灰度级

"""

img = imfft(img)

rows, cols = img.shape

crow, ccol = int(rows/2), int(cols/2)

mask = np.zeros(img.shape)

mask[crow-val:crow+val, ccol-val:ccol+val] = 1

mask = 1-mask

img = np.abs(imifft(img\*mask))

rows, cols = img.shape

for row in range(rows):

for col in range(cols):

pix = img[row, col]

if pix > 1:

img[row, col] = 1

elif pix < -1:

img[row, col] = -1

img = img\_as\_ubyte(img)

return img

1. 对一副图像加噪，进行几何均值，算术均值，谐波，逆谐波处理

**加噪声参考练习3**

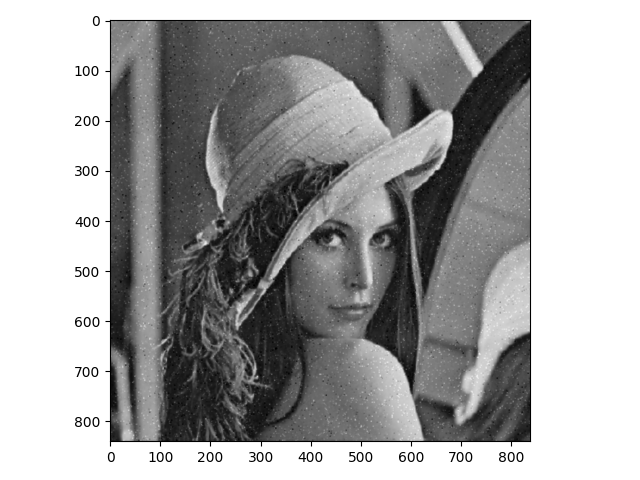
**先进行去燥处理**

**原图：**



此为10%密度生成的高斯噪点图

**几何均值处理后如下：**



**代码如下：**

# 几何均值滤波

def geometricMeanOperator(roi):

roi = roi.astype(np.float64)

p = np.prod(roi)

return p\*\*(1/(roi.shape[0]\*roi.shape[1]))

def geometricMeanAlogrithm(image):

new\_image = np.zeros(image.shape)

image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER\_DEFAULT)

for i in range(1, image.shape[0]-1):

for j in range(1, image.shape[1]-1):

new\_image[i-1, j -

1] = geometricMeanOperator(image[i-1:i+2, j-1:j+2])

new\_image = (new\_image-np.min(image))\*(255/np.max(image))

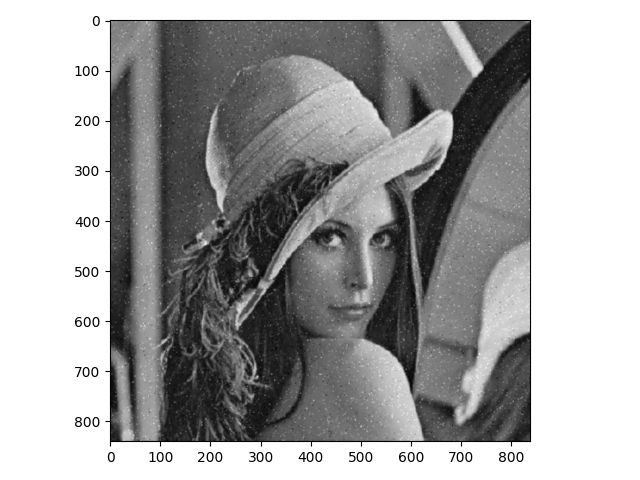
return new\_image.astype(np.uint8)

def gemotricMean(img):

img = geometricMeanAlogrithm(img)

return img

**算术均值处理后如下：**



**代码如下：**

# 算数均值滤波

def arithmeticMeanOperator(roi):

return np.mean(roi)

def arithmeticMeanAlogrithm(image):

new\_image = np.zeros(image.shape)

image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER\_DEFAULT)

for i in range(1, image.shape[0]-1):

for j in range(1, image.shape[1]-1):

new\_image[i-1, j -

1] = arithmeticMeanOperator(image[i-1:i+2, j-1:j+2])

new\_image = (new\_image-np.min(image))\*(255/np.max(image))

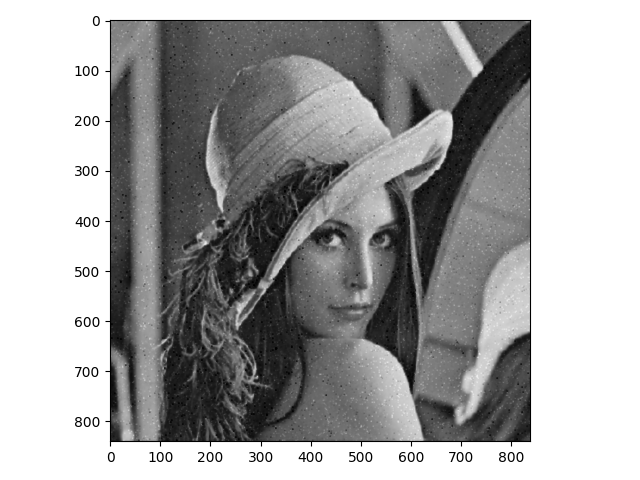
return new\_image.astype(np.uint8)

def arithmeticMean(image):

img = arithmeticMeanAlogrithm(image)

return img

**谐波处理后如下：**



**代码如下：**

# 谐波均值

def HMeanOperator(roi):

roi = roi.astype(np.float64)

if 0 in roi:

roi = 0

else:

roi = scipy.stats.hmean(roi.reshape(-1))

return roi

def HMeanAlogrithm(image):

new\_image = np.zeros(image.shape)

image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER\_DEFAULT)

for i in range(1, image.shape[0]-1):

for j in range(1, image.shape[1]-1):

new\_image[i-1, j-1] = HMeanOperator(image[i-1:i+2, j-1:j+2])

new\_image = (new\_image-np.min(image))\*(255/np.max(image))

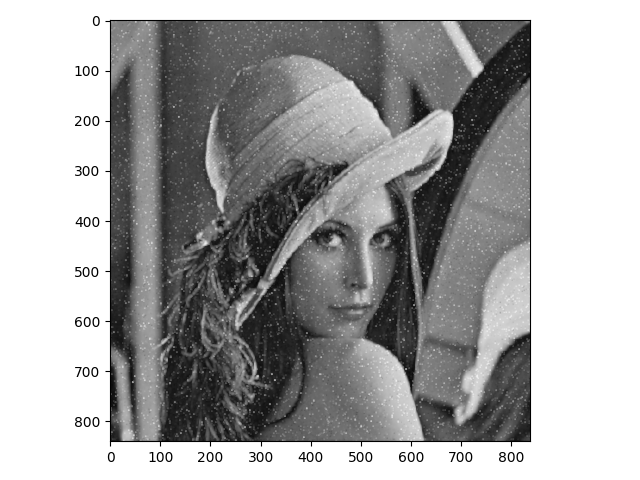
return new\_image.astype(np.uint8)

def HMean(image):

img = HMeanAlogrithm(image)

return img

**逆谐波处理后如下：**



**代码如下：**

# 逆谐波均值

def IHMeanOperator(roi, q):

roi = roi.astype(np.float64)

return np.mean((roi)\*\*(q+1))/np.mean((roi)\*\*(q))

def IHMeanAlogrithm(image, q):

new\_image = np.zeros(image.shape)

image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER\_DEFAULT)

for i in range(1, image.shape[0]-1):

for j in range(1, image.shape[1]-1):

new\_image[i-1, j-1] = IHMeanOperator(image[i-1:i+2, j-1:j+2], q)

new\_image = (new\_image-np.min(image))\*(255/np.max(image))

return new\_image.astype(np.uint8)

def IHMean(image, q):

img = IHMeanAlogrithm(image, q)

return img

1. **提取一副彩色图像中红色，用HSI模型处理**

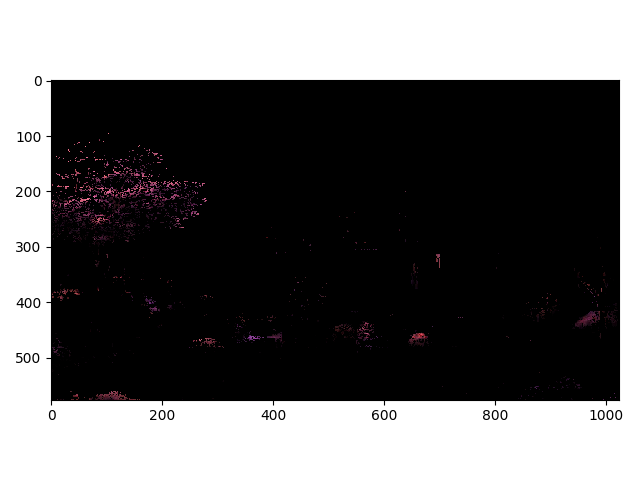
**处理方法：**

将彩色图从rgb模式转为HIS模型，并根据红色的色调范围将范围内的像素置0，还可以适当配合饱和度，如规定50%以上的红色才为要求内的红色，进一步精细化提取范围。

**原图：**



**处理后的图像：**



**代码如下：**

import numpy as np

from skimage import util

def rgb2hsi(rgb):

"""rgb转hsi"""

arr = util.img\_as\_float(rgb)

out = np.empty\_like(arr)

# -- V channel

out\_v = arr.max(-1)

# -- S channel

delta = arr.ptp(-1)

# Ignore warning for zero divided by zero

old\_settings = np.seterr(invalid='ignore')

out\_s = delta / out\_v

out\_s[delta == 0.] = 0.

# -- H channel

# red is max

idx = (arr[:, :, 0] == out\_v)

out[idx, 0] = (arr[idx, 1] - arr[idx, 2]) / delta[idx]

# green is max

idx = (arr[:, :, 1] == out\_v)

out[idx, 0] = 2. + (arr[idx, 2] - arr[idx, 0]) / delta[idx]

# blue is max

idx = (arr[:, :, 2] == out\_v)

out[idx, 0] = 4. + (arr[idx, 0] - arr[idx, 1]) / delta[idx]

out\_h = (out[:, :, 0] / 6.) % 1.

out\_h[delta == 0.] = 0.

np.seterr(\*\*old\_settings)

# -- output

out[:, :, 0] = out\_h

out[:, :, 1] = out\_s

out[:, :, 2] = out\_v

# remove NaN

out[np.isnan(out)] = 0

return out

def hsi2rgb(hsv):

"""hsi转rgb"""

arr = util.img\_as\_float(hsv)

hi = np.floor(arr[:, :, 0] \* 6)

f = arr[:, :, 0] \* 6 - hi

p = arr[:, :, 2] \* (1 - arr[:, :, 1])

q = arr[:, :, 2] \* (1 - f \* arr[:, :, 1])

t = arr[:, :, 2] \* (1 - (1 - f) \* arr[:, :, 1])

v = arr[:, :, 2]

hi = np.dstack([hi, hi, hi]).astype(np.uint8) % 6

out = np.choose(hi, [np.dstack((v, t, p)),

np.dstack((q, v, p)),

np.dstack((p, v, t)),

np.dstack((p, q, v)),

np.dstack((t, p, v)),

np.dstack((v, p, q))])

return out

def pick\_red(img, s=0.5):

"""

提取饱和度在s=50%以上的红色色系图像像素

"""

img\_hsv = rgb2hsi(img)

mark\_h = img\_hsv[:, :, 0] < 0.8 # 筛选出非红色色调的像素

mark\_s = img\_hsv[:, :, 1] < 0.5 # 筛选出饱和度过低的像素

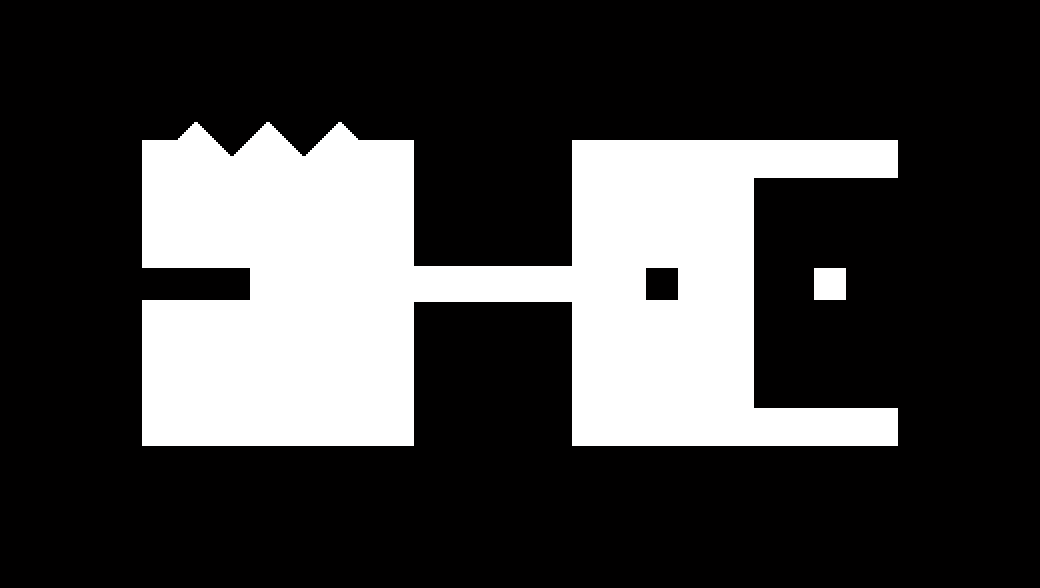
img\_hsv[mark\_h] = [0, 0, 0] # 将筛选出的不合条件的像素归0

img\_hsv[mark\_s] = [0, 0, 0]

return hsi2rgb(img\_hsv)

1. 对一副二值图像进行膨胀、腐蚀、开、闭操作

原图：



**处理方法：**

构造一个35\*35的矩形结构元， 重合点在中心。

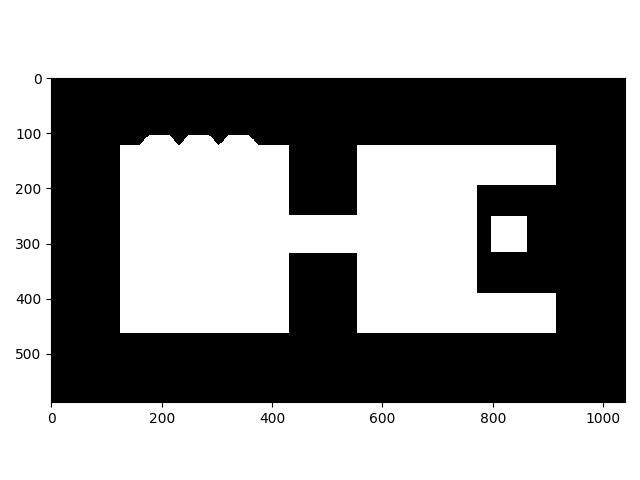
膨胀：移动结构元中心到图像每一个像素上，若结构元邻域像素在图像中相同位置处的灰度不为0，则结构元中心点处该点确认为膨胀点。

腐蚀: 移动结构元中心到图像每一个像素上, 若结构元领域像素有一个在图像中相同位置处的灰度为0，则结构元中心点处该点像素灰度值置0。

开操作：先调用腐蚀操作，再调用膨胀操作。

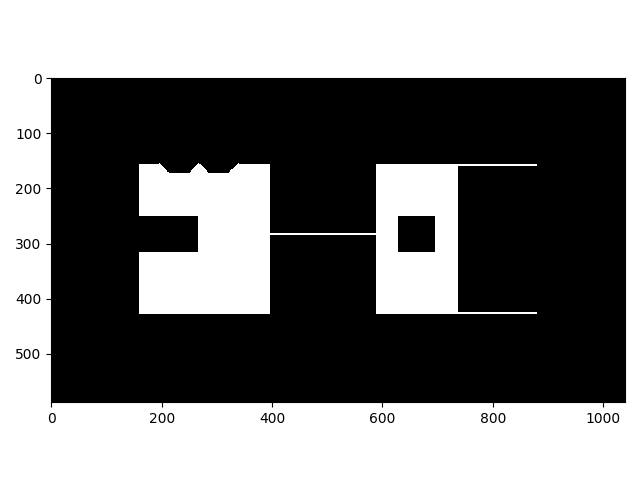
闭操作：先调用膨胀操作，再调用腐蚀操作。

**膨胀结果图：**



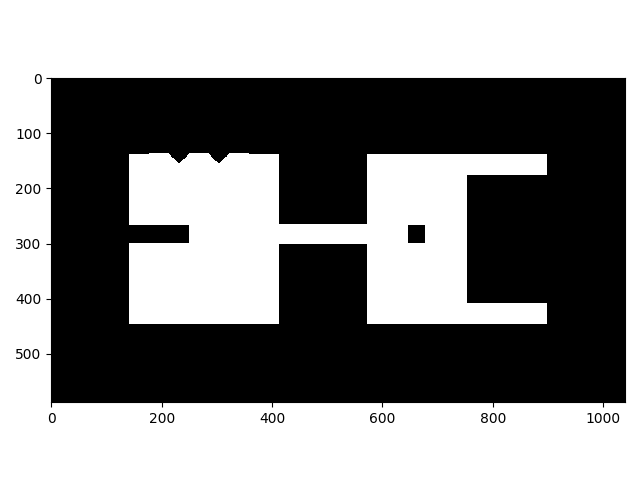
显然，白色部分增多，且尖锐部分开始矩形化

**腐蚀结果图：**

****

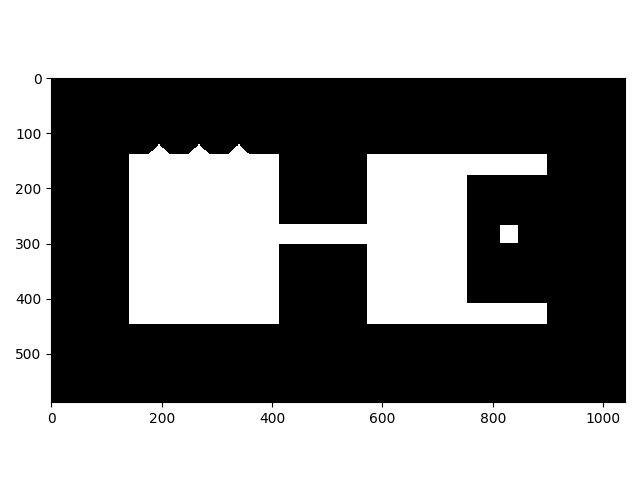
显然，白色部分减少，黑色部分增多。

**开操作结果：**

****

显然，毛刺部分发生去除现象。

**闭操作结果：**

****

显然，缺口部分有被填补，但毛刺部分变化不大。

代码如下：

from exercise1 import imread, imshow

from scipy import ndimage as ndi

import numpy as np

def square\_se(width, dtype=np.uint8):

"""创建矩形结构元"""

return np.ones((width, width), dtype=dtype)

def erode(img, selem):

"""腐蚀操作"""

selem = np.array(selem)

out = np.empty\_like(img)

ndi.grey\_erosion(img, footprint=selem, output=out)

return out

def dilate(img, selem):

"""膨胀操作"""

selem = np.array(selem)

out = np.empty\_like(img)

ndi.grey\_dilation(img, footprint=selem, output=out)

return out

def open\_operate(img, selem):

"""开操作"""

img = erode(img, selem)

img = dilate(img, selem)

return img

def close\_operate(img, selem):

"""闭操作"""

img = dilate(img, selem)

img = erode(img, selem)

return img

if \_\_name\_\_ == "\_\_main\_\_":

filename = input('请输入要打开的图片路径：')

img = imread(filename, as\_gray=True)

opt = input('1.膨胀 2.腐蚀 3.开 4.闭：')

se = square\_se(35) # 创建结构元

if opt == '1':

img = dilate(img, se)

elif opt == '2':

img = erode(img, se)

elif opt == '3':

img = open\_operate(img, se)

elif opt == '4':

img = close\_operate(img, se)

imshow(img)

**附源码地址：**

<https://github.com/mobishift2011/dip-practice>

**联系邮箱：**

[sorrowkid@163.com](mailto:sorrowkid@163.com)