数字图像处理实验报告

曾绍豪
14348164
14 级计算机科学

2017/3/28

概要：本实验报告分为 03-02、03-05、03-06 和 02-01 四道题的实验报告。本实验使用 Python 3 以及一些相关的库（如 Numpy、Matplotlib、PIL），我尽量不使用库里直接提供的函数，除了类似于画图、构造基础数据结构（如 Numpy Array）这种与课程无关的较低层的功能。下面代码的截图来自于 Jupyter Notebook。

03-02

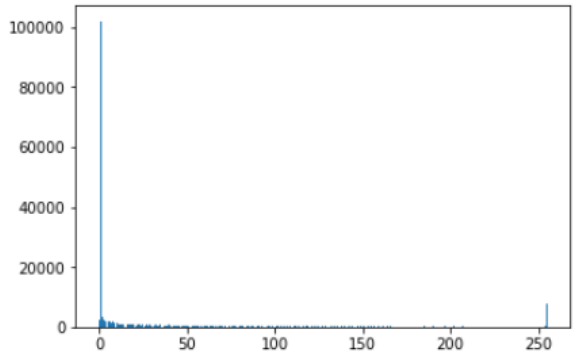
Histogram Equalization
(a) Write a computer program for computing the histogram of an image.
(b) Implement the histogram equalization technique discussed in Section 3.3.1.
(c) Download Fig. 3.8(a) and perform histogram equalization on it.
As a minimum, your report should include the original image, a plot of its histogram, a plot of the histogram-equalization transformation function, the enhanced image, and a plot of its histogram. Use this information to explain why the resulting image was enhanced as it was.

技术讨论

这一题(a)要求画出直方图，(b)要求实现直方图均衡化，(c)要求拿特定的图做演示。

画直方图很容易：只需要维护一个 256 大小的数组，然后对图像每个像素遍历，就可以获得每个灰度的数量统计。再利用 Matplotlib 的一些函数（下面 plt.bar 是画直方图的）画出来就可以了。

```
# Build histogram
count = np.zeros(256, dtype=np.int32)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        count[img[i, j]] += 1
plt.figure(2)
plt.bar(left=np.arange(256), height=count)
```



直方图均衡化，具体理论推导见课件。在实现的时候，只需要求灰度的概率密度分布对应的累积分布，也就是 cumu[i]=cumu[i-1]+count[i]，其中 cumu 为累计（cumulative），count 为记录灰度密度分布的数组（大小 256）。然后再对这个 cumu 数组进行归一化，用图像像素数量去除数组的每一项，并取整。得到的 cumu 数组就是均衡化的映射。

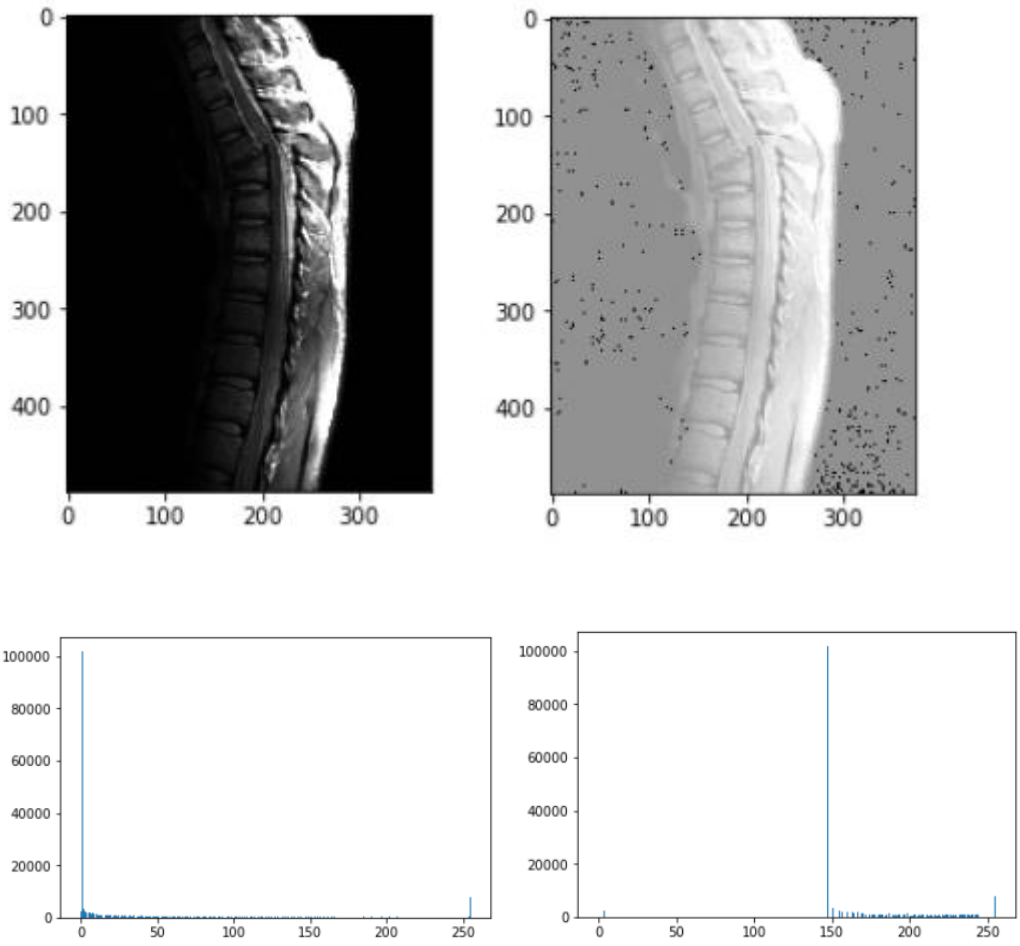
```
# Histogram equalization
cumu = np.zeros(256, dtype=np.float64)
cumu[0] = count[0]
for i in range(1, 256):
    cumu[i] += cumu[i - 1] + count[i]
print(cumu[0], cumu[255])
for i in range(0, 256):
    cumu[i] /= cumu[255]
    cumu[i] *= 255
cumu = np.ceil(cumu)
```

对原图进行变换，这个只需要对原图每个像素都采用上面的映射，替换成新的灰度大小。

```
# Perform transformation and show the image
trans = cumu
trans_img = [[trans[pix] for pix in row] for row in img]
plt.figure(3)
plt.imshow(trans_img, cmap='gray')
```

结果讨论

可以比较一下变换前后的图以及直方图，可以看到在连续情况下的推导的均衡化算法，简单地搬到离散情况下并不一定能做到想象中那么均衡。而且，即使灰度均匀了，图像显示效果也不一定好。这个方法不是银弹。



代码附录

见附件 03-02.py。

03-05

Enhancement Using the Laplacian

(a) Use the programs developed in Projects 03-03 and 03-04 to implement the Laplacian enhancement technique described in connection with Eq. (3.7-5). Use the mask shown in Fig. 3.39(d).

(b) Duplicate the results in Fig. 3.40. You will need to download Fig. 3.40(a).

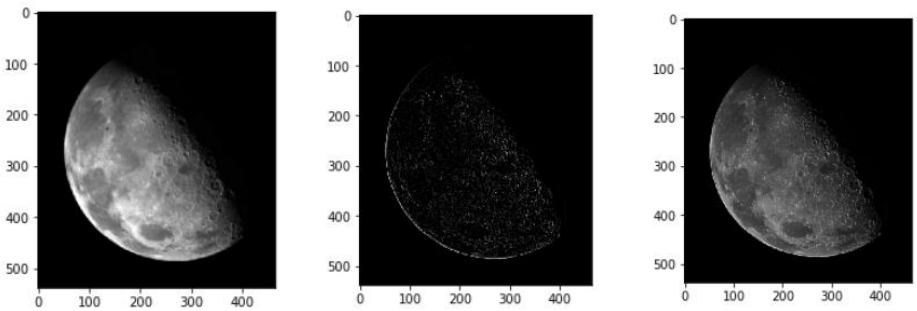
技术讨论

本题(a)要求实现拉普拉斯滤波，(b)要求在一张图是跑出结果。

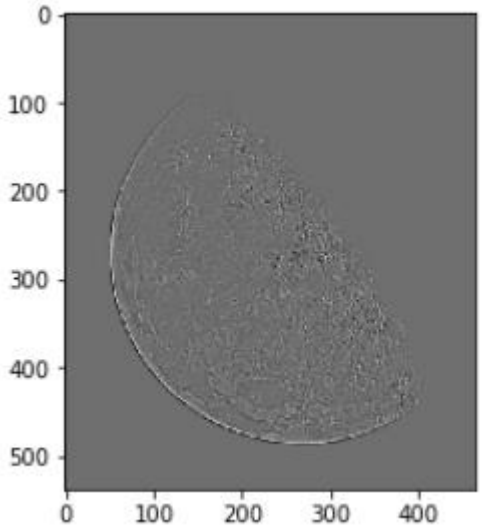
拉普拉斯滤波的原理来自对二阶导数的特性，推导细节来自课件。实现的时候，即对图像每个 3*3 的块都用一个 3*3 的 mask 来乘并且对 9 个积累加并赋值到新的图像的对应位置。对于边缘以外，我是看作为 0，下图代码中 np.lib.pad 函数就是将原图边缘“围上“一圈 0，这样的话就不需要后面进行冗长的边缘判断了。

```
def Laplace(img):
    mask = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
    trans_img = np.zeros(img.shape)
    pad_img = np.lib.pad(img,(1,1),'constant',constant_values=0)
    for i in range(1,img.shape[0]-1):
        for j in range(1,img.shape[1]-1):
            sum1=np.sum(img[i-1:i+2,j-1:j+2]*mask)
            if sum1>255:
                trans_img[i,j]=255
            elif sum1<0:
                trans_img[i,j]=0
            else:
                trans_img[i,j]=sum1
    return trans_img
```

以下三图为原图、滤波出来的结果、滤波结果叠加回原图的效果。



不知道读者是否看到上面附着的代码里面还有对累加结果大小的判断？这个地方曾经困惑了我很久。因为我之前并没有考虑到两个矩阵相乘累加会有负数的结果，所以一度在困惑为什么滤波结果长成下面的图。

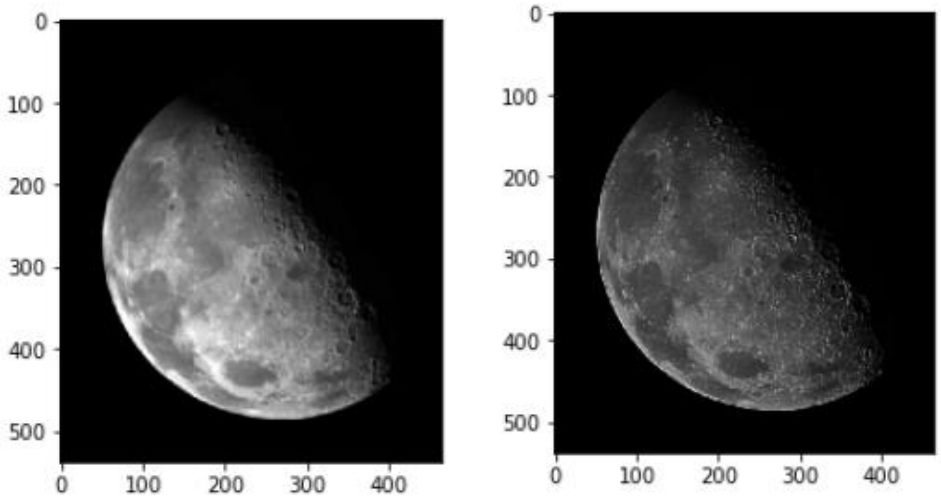


为什么背景是**灰**的？

我一度以为这是 colormap（即由一个数值到 RGB 的映射）的选择问题，但查询许久也没有找到问题。而我又怀疑是计算问题，但其实并没有（我没有意识到负数的问题）。最为诡异的是，上面这张图的[0:100,0:100]在矩阵里面是全 0 的（可以通过 print 来看），但在这张图里是灰色的，如果我单独画这个[0:100,0:100]的部分时，显示的图像是全黑的！后来我直接比较同学的代码，他的显示是正常的。我发现它读图存图都是用 PIL，而我是直接调用 Numpy 的接口，而两者默认的数据类型是不同的。PIL 作为一个专门的图像库，默认的底层数据类型是 uint8，而且**禁止下溢和上溢**！也就是说，给这种变量赋值一个负数，它实际存的是 0，赋值一个大于 255 的值，它实际存的是 255。而 Numpy 默认提供的是 float64。这么对比来看，原因就昭昭然了：滤波结果里面负数应当按 0 处理。

现在回到上面的画图问题，为什么数值为 0 的地方显示的是灰色？通过查询文档，我发现 plt.imshow 这个函数对于输入的矩阵，需要先做归一化（normalization），那么可以猜测：如果输入矩阵里面有负数，那么归一化的矩阵里面原来为 0 的地方应当会被映射到比 0 更大的灰度，因此就是灰色了。不过，具体归一化的算法我没有查清楚，这都是我的猜想而已。

结果讨论



可以看到，拉普拉斯滤波确实有很不错的锐化效果。

代码附录

见附件 03-05.py。

03-06

Unsharp Masking

(a) Use the programs developed in Projects 03-03 and 03-04 to implement highboost filtering, as given in Eq. (3.7-8). The averaging part of the process should be done using the mask in Fig. 3.34(a).

(b) Download Fig. 3.43(a) and enhance it using the program you developed in (a). Your objective is to choose constant *A* so that your result visually approximates Fig. 3.43(d).

技术讨论

这一题(a)要求实现高提升滤波(b)要求找到上述这个滤波适当的 A 值使得效果逼近于给定的一张图像。

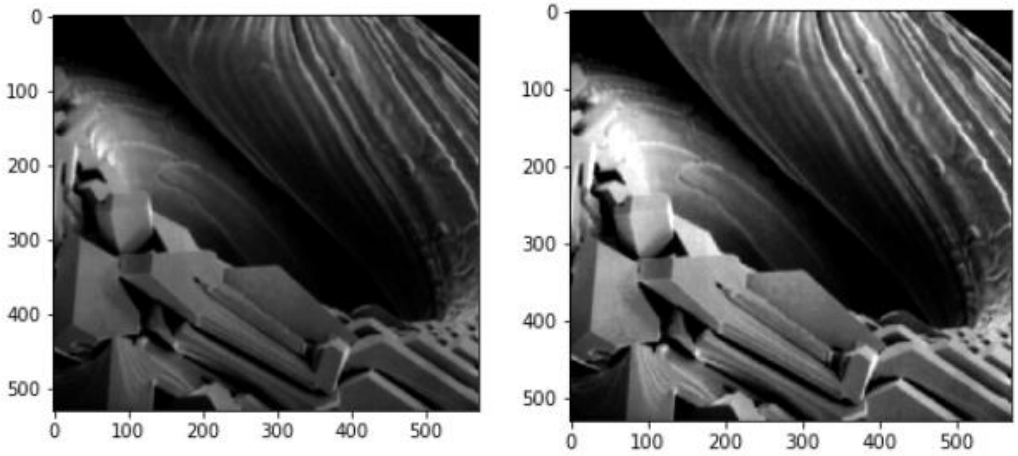
这道题和上一道题没有太大差别，只是 mask 改了一下，并且还需要测试一下适当的 A。

```python
A = 2.6
def Unsharp(img):
    mask = np.array([[1,1,1],[1,1,1],[1,1,1]])
    trans_img = np.zeros(img.shape)
    pad_img = np.lib.pad(img,(1,1),'constant',constant_values=0)
    for i in range(1,pad_img.shape[0]-1):
        for j in range(1,pad_img.shape[1]-1):
            acc = A*img[i-1,j-1]-np.sum(pad_img[i-1:i+2,j-1:j+2]*mask)/9
            if acc < 0:
                trans_img[i-1,j-1]=0
            elif acc > 255:
                trans_img[i-1,j-1]=255
            else:
                trans_img[i-1,j-1]=acc
    return trans_img
```

注意这里同样需要注意，计算时下溢和上溢的问题。我查文档看到，可以通过 np.clip 在这个函数来产生一个禁止上溢下溢的数字，不过我这里并没有使用。

结果讨论

我测试的结果是 A=2.6 左右时，锐化效果是还算可以的，虽然不算特别明显。



左图为原图，右图为 A=2.6 的效果。

代码附录

见附件 03-06.py。

02-01

Image Printing Program Based on Halftoning

The following figure shows ten shades of gray approximated by dot patterns. Each gray level is represented by a 3 x 3 pattern of black and white dots. A 3 x 3 area full of black

dots is the approximation to gray-level *black*, or 0. Similarly, a 3 x 3 area of white dots represents gray level 9, or *white*. The other dot patterns are approximations to gray levels in between these two extremes. A gray-level printing scheme based on dots patterns such as these is called "halftoning." Note that each pixel in an input image will correspond to 3 x 3 pixels on the printed image, so spatial resolution will be reduced to 33% of the original in both the vertical and horizontal direction. Size scaling as required in (a) may further reduce resolution, depending on the size of the input image.
(a) Write a halftoning computer program for printing gray-scale images based on the dot patterns just discussed. Your program must be able to scale the size of an input image so that it does not exceed the area available in a sheet of size 8.5 x 11 inches (21.6 x 27.9 cm). Your program must also scale the gray levels of the input image to span the full halftoning range.
(b) Write a program to generate a test pattern image consisting of a gray scale wedge of size 256 x 256, whose first column is all 0's, the next column is all 1's, and so on, with the last column being 255's. Print this image using your gray-scale printing program.
(c) Print book Figs. 2.22(a) through (c) using your gray-scale printing program. Do your results agree with the conclusions arrived at in the text in pgs. 61-62 and Fig. 2.23? Explain. You will need to download Figs. 2.22(a) through (c).

技术讨论

这一道题要求(a)实现给定要求的这个画图函数(b)用灰度渐变图测试一下(c)用另外一张图测试一下。

这道题并没有太大的难度。具体实现就是将输入图每个像素遍历，并输出一个最“接近“的 3*3 矩阵。显然，输出图的长宽都是原图的三倍。

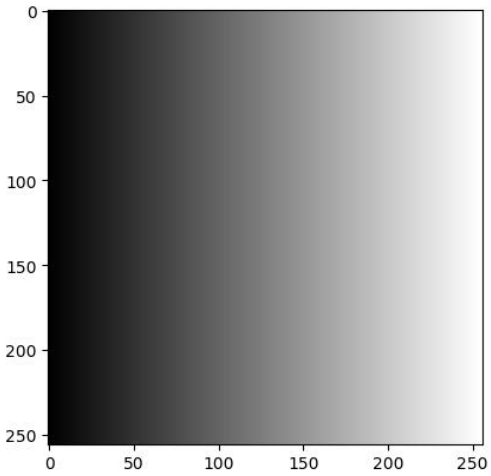
```
def myprint(img):

    pixel2mat3 = np.array([
        [[255, 255, 255],
         [255, 255, 255],
         [255, 255, 255]],

        [[255, 0, 255],
         [255, 255, 255],
         [255, 255, 255]],

        [[255, 0, 255],
         [255, 255, 255],
         [255, 255, 0]],

        [[0, 0, 255],
         [255, 255, 255],
         [255, 255, 0]],

        [[0, 0, 255],
         [255, 255, 255],
```

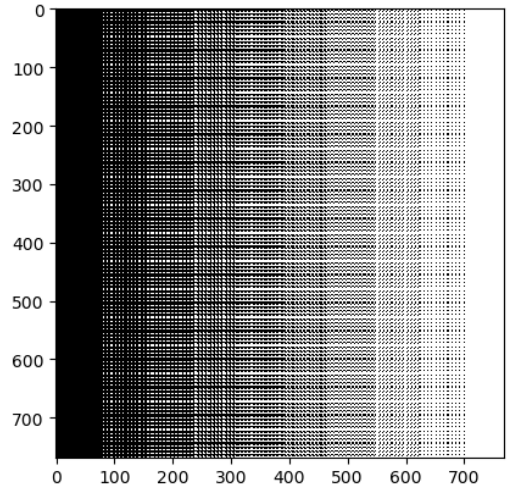
```
# 之前黑色和白色的值写反了。。。
    pixel2mat3 = pixel2mat3[::-1]

    new_map = np.zeros((img.shape[0] * 3, img.shape[1] * 3))

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            new_map[i * 3:i * 3 + 3, j * 3:j * 3 +
                    3] = pixel2mat3[np.int(img[i, j]) // 26]

    return new_map


greyscale = np.zeros((256, 256)) + range(256)
plt.imshow(greyscale, cmap='gray')
trans = myprint(greyscale)
plt.figure(2)
plt.imshow(trans, cmap='gray')

plt.show()
```

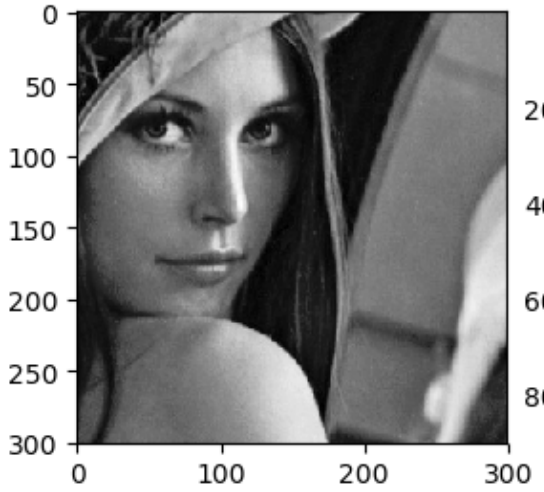
首先先列出那 10 个等级灰度代表的 3*3 矩阵，这里我不小心把黑色和白色对应的灰度弄反了，又懒得逐个改回去，就后面加一条命令颠倒一下。接下来的工作也就很简单了，诸葛遍历，然后再把对应的矩阵“贴到“输出图像对应位置上去。

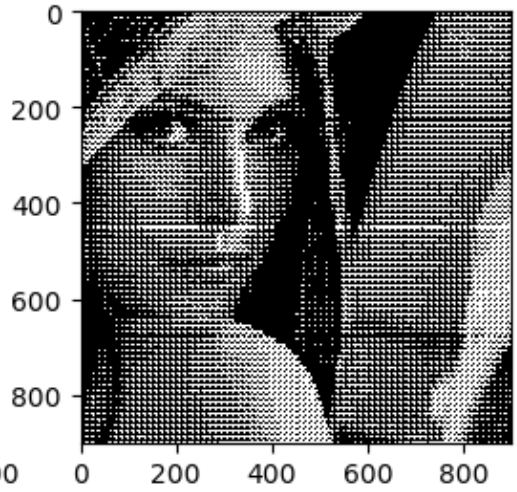
结果讨论

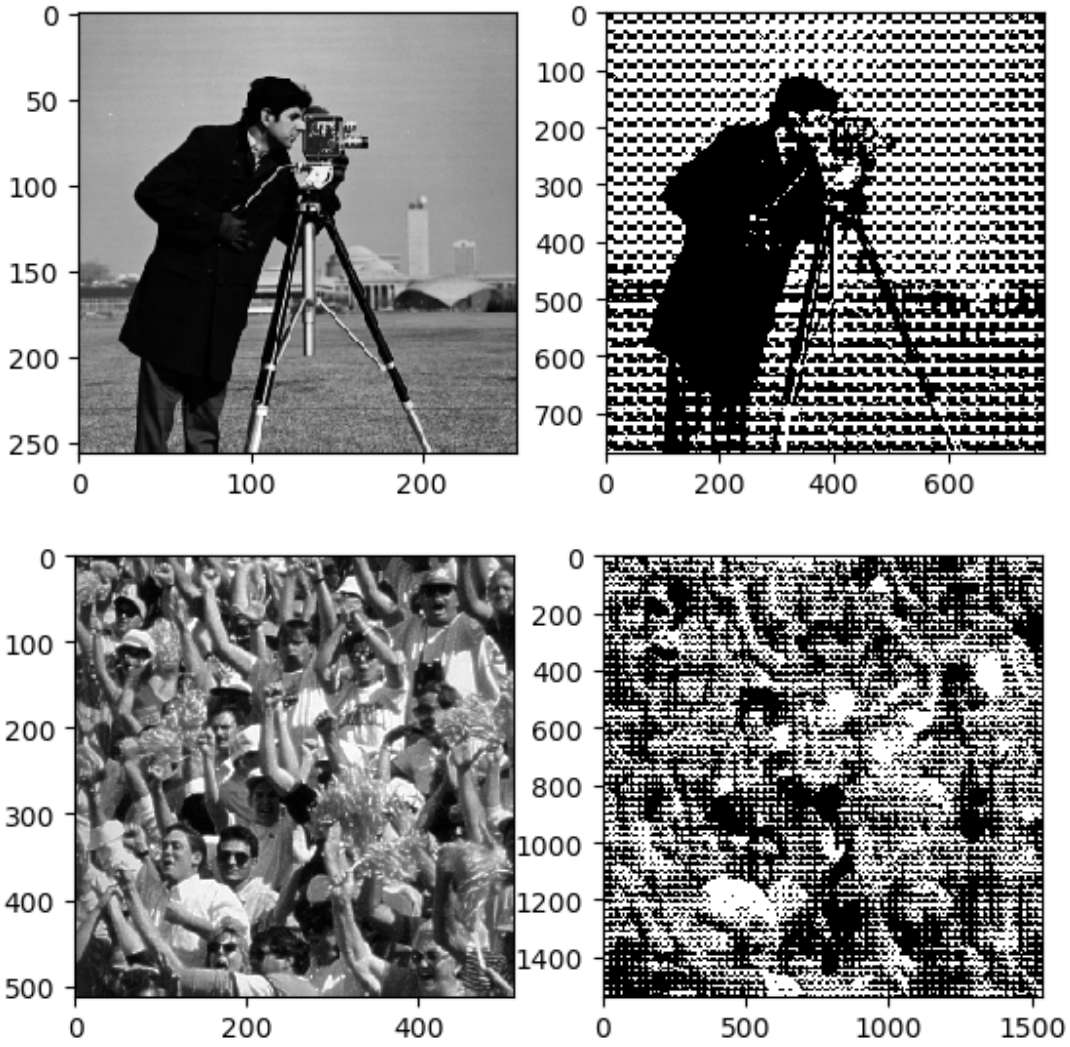
下面分别展示的是灰阶渐变图以及另外三张给定的图，它们的原图以及变换后的图。











代码附录

见附件 02-01.py。