中山大學
SUN YAT-SEN UNIVERSITY

《数字图像处理》项目报告

Digital Image Processing Project Report

项目章节：　　第三章

项目编号：　　03-02，03-05，03-06

姓名：　　刘昕

学号：　　14348085

摘要：

使用 Python3 完成图像直方图的显示，对进行直方图均衡化（03-02），实现了拉普拉斯算子，并且对拉普拉斯算子处理后的图像进行标准化，同时对图像进行锐化处理（03-05），实现了高提升滤波（03-06）

时间：　2017 年 3 月 26 日

目录

PROJECT 03-02 Histogram Equalization

实验要求

(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.

图 1Fig.3.8(a)

技术讨论

(1) 图像本质上是一个二维矩阵，矩阵的每一个像素对应一个像素，，有位置（坐标）和灰度（元素的值）。所以定义x, y 分别表示矩阵的行和列，$f(x, y)$表示坐标为(x, y)处的灰度

(2) 计算机中灰度图一般用 8 位表示 0~255，其中 0 是全黑，255 是全白

(3) 数字图像直方图是离散函数 $h(r_k) = n_k$，其中 r_k 是第k级灰度，n_k是图像中灰度级为k的像素个数。在这里，我们设置灰度级为 256，即$k \in \{0,1,2 \ldots .255\}$

(4) 直方图是图像处理领域中的一个重要工具，也可以用于图像增强。从前面的分析可知，一般情况下若是图像的直方图成均匀分布态势，则图像的对比度较高且有比较丰富的灰度色调。这就为我们提供了一种基于直方图的图像增强处理方法。也就是说为了达到好的图像效果,可寻求一种变换$s = T(r)$，使得变换后的图像具有（尽可能）均匀分布的直方图。这个变换必须满足

   a. $s = T(r)$在区间上单值且单调递增

   b. 若$r \in [a, b]$，则$s \in [a, b]$

(5) 累计分布函数（cdf）可以用来作为直方图均衡化的桥梁。首先计算不同灰度级的概率密度函数$pdf(r_k) = \frac{n_k}{n}$，然后计算累计分布函数$cdf(r_k) = \sum_{j=0}^{k} pdf(r_j) = \sum_{j=0}^{k} \frac{n_j}{n}$，将累计分布函数乘以灰度级变化范围的到变换函数$s = T(r_k) = (L-1)cdf(r_k)$，然后将原图像的灰度值带入$s = T(r_k)$得到均衡化后的灰度值（需要舍入）

结果讨论

(1) 发现 Fig.3.8(a)的灰度值主要集中在 0 附近，有一部分集中在 255 附近，几乎没有其余灰度值。均衡化后，灰度值主要集中在 150 附近

(2) 图像整体变亮，均衡化后的图片黑暗的部分可以看清

结果

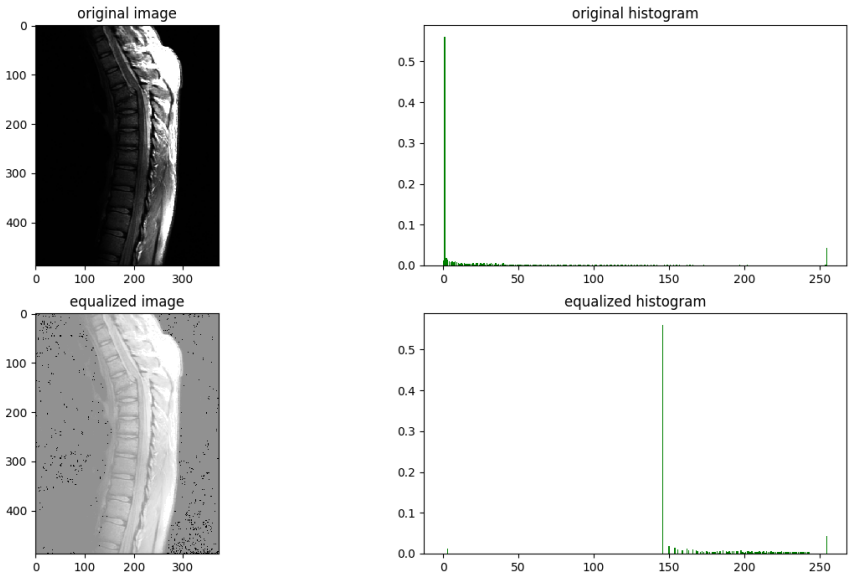

图 2Project03-02 结果

附录

```
#!/usr/bin/env python3
#coding:utf-8
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np


def my_histogram(image):
    height, width = image.shape
    hist = np.zeros((256), dtype=np.float64)
    for x in range(height):
        for y in range(width):
            hist[image[x,y]] += 1
    hist = hist / image.size
```

```
    return hist


def my_equalize_histogram(image):
    hist = my_histogram(image)
    cdf = hist.cumsum()
    T = 255 * cdf
    T = T.astype(np.uint8)
    new_img = np.frompyfunc(lambda x: T[x], 1, 1)(image).astype(np.uint8)
    return new_img


img = np.array(Image.open('../images/images_chapter_03/Fig3.08(a).jpg').convert('L'))
plt.subplot(2,2,1)
plt.title('original image')
plt.imshow(img, cmap='gray')
hist = my_histogram(img)
plt.subplot(2,2,2)
plt.title('original histogram')
plt.bar(range(hist.shape[0]) ,hist, color = 'g')
equ_img = my_equalize_histogram(img)
plt.subplot(2,2,3)
plt.title('equalized image')
plt.imshow(equ_img, cmap='gray')
equ_hist = my_histogram(equ_img)
plt.subplot(2,2,4)
plt.title('equalized histogram')
plt.bar(range(equ_hist.shape[0]) ,equ_hist, color = 'g')
plt.show()
```

PROJECT 03-05 Enhancement Using the Laplacian

实验要求

(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).

图 3Fig.3.40(a)

技术讨论

(1) 滤波的概念来自信号处理中的傅里叶变换，空间滤波指的是直接对图像像素进行处理的操作

(2) 滤波器可以以矩阵的形式给出，所以滤波器又叫做掩模、模板

$$w(x,y)=\begin{bmatrix} w(-1,-1) & w(-1,0) & w(-1,1) \\ w(0,-1) & w(0,0) & w(0,1) \\ w(1,-1) & w(1,0) & w(1,1) \end{bmatrix}$$

那么滤波得到的图像为$g(x,y)=\sum_{s=-1}^{1}\sum_{t=-1}^{1}w(s,t)f(x+s,y+t)$

(3) 当遇到边界时，有多种处理方法，可以重复边沿值、卷绕输入图像、补零、忽略。本项目种选择忽略

(4) 锐化是图像处理中常用的操作，可以用来突出细节或者增强被模糊了的细节。

一般采用微分，使用一阶微分产生较“宽”的边界，二阶微分产生 较“细”的边界。二阶微分处理对细节有较强的响应，如细线和孤立点。一阶微分对阶梯状的灰度变化有较强的响应，二阶微分在处理阶梯状灰度变化时产生双响应，如果灰度的变化相似，二阶微分对线的反应比 对阶梯强，对点的反应比对线强。

(5) 基于二阶微分的图像增强就是使用拉普拉斯算子$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$，如果模板是3×3，那么

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$
$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$
$$\nabla^2 f = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

用滤波器的形式表示就是

0	1	0
1	-4	1
0	1	0

为了增强效果，实际上采用

1	1	1
1	-8	1
1	1	1

锐化后的图像$g(x,y) = f(x,y) - \nabla^2 f$

如果滤波器中心系数为正数

-1	-1	-1
-1	8	-1
-1	-1	-1

那么锐化后的图像为$g(x,y) = f(x,y) + \nabla^2 f$，本项目采用该滤波器和公式

结果讨论

(1) 使用拉普拉斯算子得到的图像基本上灰度值为 0,，少数出现负值，只有在图像变化明显的地方才会出现白色

(2) 为了让得到的图像方便显示，对图像进行标准化，使用的公式是

$$g(x,y)=\frac{f(x,y)-\min\big(f(x,y)\big)*255}{\max\big(f(x,y)\big)-\min\big(f(x,y)\big)}$$

处理后的图片比原先的图片更直观

(3) 最后将原图像和拉普拉斯滤波得到的图像求和，同时对超出 0~255 范围的灰度值进行处理。小于 0 的设置为 0，大于 255 的设置为 255。可以发现，细节得到明显增强，达到预期效果

(4) 小插曲：由于 Python 对图像处理时用的矩阵类型时 uint8 类型，即使转换到 float 类型，最后使用 matplotlib 显示的时候，会出现和预期不同的情况。因为 matplotlib 会自动对图像进行标准化，所以导致部分灰度值的显示效果与真实效果不同。为了显示出预期效果，需要自己对矩阵超出 0~255 范围的灰度值进行处理。在这里感谢王广聪师兄的提醒，打扰师兄很久，真的很感谢师兄

结果

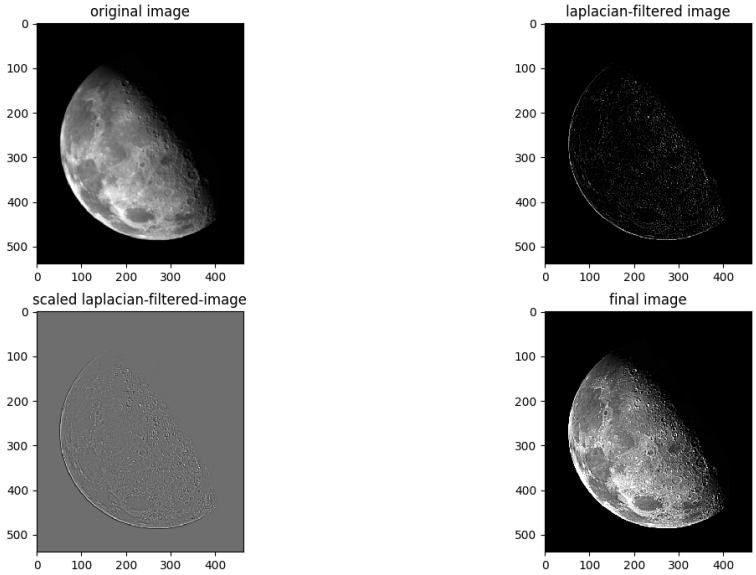

图 4Project03-05 结果

附录

```
#!/usr/bin/env python3
```

```
#coding:utf-8

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_normalize(image):
    scaled_image = np.frompyfunc(lambda x: max(0, min(x, 255)), 1, 1)(image).astype(np.uint8)
    return scaled_image

def my_scale(image):
    scaled_image = image.copy()
    scaled_image = (scaled_image - scaled_image.min()) * 255 / (scaled_image.max() -
scaled_image.min())
    return scaled_image.astype(np.uint8)

def my_spatial_masking(image, mask):
    filtered_image = np.zeros(image.shape, dtype=np.float64)
    height, width = image.shape
    h, w = mask.shape
    mid_w = w // 2
    mid_h = h // 2
    for x in range(height):
        for y in range(width):
            for i in range(h):
                for j in range(w):
                    px = x + i - mid_h
                    py = y + j - mid_w
                    if px >= 0 and px < height and py >= 0 and py < width:
                        filtered_image[x, y] += mask[i,j] * image[px, py]
```

```
    return filtered_image



img = np.array(Image.open('../images/images_chapter_03/Fig3.40(a).jpg').convert('L'))
laplacian_mask = np.array([ [-1, -1, -1],
                            [-1, 8, -1],
                            [-1, -1, -1]])
filtered_img = my_spatial_masking(img, laplacian_mask)
new_img = my_normalize(img + filtered_img)
plt.subplot(2,2,1)
plt.title('original image')
plt.imshow(img, cmap='gray')
plt.subplot(2,2,2)
plt.title('laplacian-filtered image')
plt.imshow(my_normalize(filtered_img), cmap='gray')
plt.subplot(2,2,3)
plt.title('scaled laplacian-filtered-image')
plt.imshow(my_scale(filtered_img), cmap='gray')
plt.subplot(2,2,4)
plt.title('final image')
plt.imshow(my_normalize(new_img), cmap='gray')
plt.show()
```

PROJECT 03-06 Unsharp Masking

实验要求

(a) Use the programs developed in Projects 03-03 and 03-04 to implement high- boost 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).

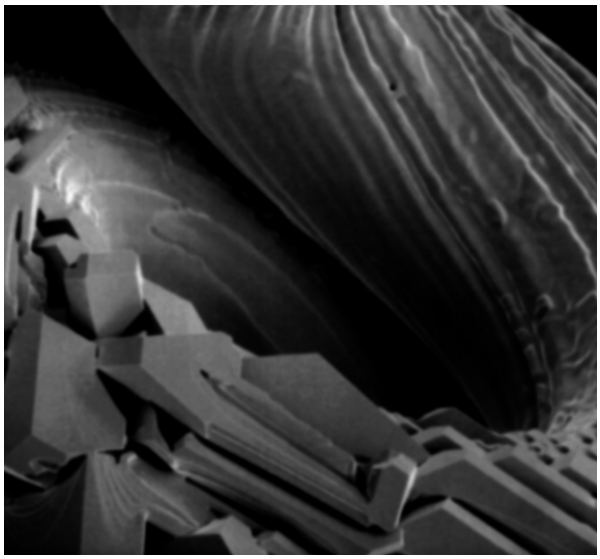

图 5Fig.3.43(a)

技术讨论

(1) 长期以来在出版业种使用的图像锐化处理是将图像模糊形式从原始图像中去除，这种处理称为图像的反锐化掩蔽，可以表示为：

$$f_s(x,y) = f(x,y) - \bar{f}(x,y)$$

其中$f_s(x,y)$表示经过反锐化掩蔽得到的锐化图像，是$f(x,y)$的模糊形式

(2) 反锐化掩蔽进一步的普遍形式称为高提升滤波，定义如下

$$f_{hb}(x,y) = Af(x,y) - \bar{f}(x,y) = (A-1)f(x,y) + f_s(x,y)$$

(3) 3×3均值滤波器掩模是

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

结果讨论

(1) 当用3×3均值滤波器掩模得到$\bar{f}(x,y)$时，显示后发现图像和原图像区别不大

(2) 将参数A设置为 1，得到图像的反锐化掩蔽$f_s(x,y)=f(x,y)-\bar{f}(x,y)$。发现灰度几乎都在 0 附近，只有原图像变化明显的很少一部分灰度值较大

(3) 然后调整参数A的大小，直到和 Fig3.43(d)相近。最后发现，当$A=2.7$时，图片和 Fig3.43(d)基本相似。所以利用图像的反锐化掩蔽和高提升滤波最终能够达到锐化的目的

结果

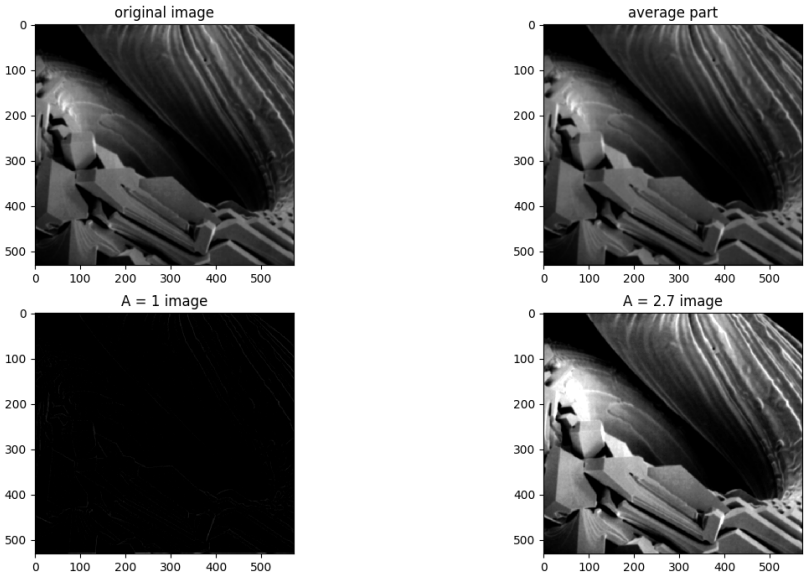

图 6Project03-06 结果

附录

```
#!/usr/bin/env python3
#coding:utf-8

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def my_normalize(image):
    scaled_image = np.frompyfunc(lambda x: max(0, min(x, 255)), 1, 1)(image).astype(np.uint8)
    return scaled_image

def my_spatial_masking(image, mask):
    filtered_image = np.zeros(image.shape, dtype=np.float64)
    height, width = image.shape
    h, w = mask.shape
    mid_w = w // 2
    mid_h = h // 2
    for x in range(height):
        for y in range(width):
            for i in range(h):
                for j in range(w):
                    px = x + i - mid_h
                    py = y + j - mid_w
                    if px >= 0 and px < height and py >= 0 and py < width:
                        filtered_image[x, y] += mask[i,j] * image[px, py]
    return filtered_image
img = np.array(Image.open('../images/images_chapter_03/Fig3.43(a).jpg').convert('L'),
dtype=np.float64)
```

```
laplacian_mask = np.array([ [1, 1, 1],
                            [1, 1, 1],
                            [1, 1, 1]], dtype=np.float64)
laplacian_mask = laplacian_mask / 9
filtered_img = my_spatial_masking(img, laplacian_mask)
new_img_A_1 = img - filtered_img
new_img_A_2_7 = 2.7 * img - filtered_img
plt.subplot(2,2,1)
plt.title('original image')
plt.imshow(img, cmap='gray')
plt.subplot(2,2,2)
plt.title('average part')
plt.imshow(my_normalize(filtered_img), cmap='gray')
plt.subplot(2,2,3)
plt.title('A = 1 image')
plt.imshow(my_normalize(new_img_A_1), cmap='gray')
plt.subplot(2,2,4)
plt.title('A = 2.7 image')
plt.imshow(my_normalize(new_img_A_2_7), cmap='gray')
plt.show()
```