可视化作业2

`18300290007 加兴华`

Note: Please provide a report file (in PPT/word/pdf) of your homework consisting of everything including description of data, code and result. Also please submit a zip file to elearning including the report, data/image if you use, and source code (with comments).

文件说明：

附件包含 otsu+bgt阈值法.py、local_otsu.m、linear_interpolation.m分别对应作业题1，2，3；

另，所使用照片位于“图片库”文件夹中，将其移至相应环境根目录或者修改代码中路径即可正常运行程序；

以及第三题保存的处理图像也位于“图片库”文件夹中。

1

Restate the Basic Global Thresholding (BGT) algorithm so that it uses the histogram of an image instead of the image itself. (Please refer to the statement of OSTU algorithm)

代码：(python)

```python
import matplotlib.pyplot as plt
from skimage import io

# 定义函数实现寻找otsu阈值
def otsu_threshold(img):
    rows,cols = img.shape
    N = rows*cols
    bin=[0 for x in range(256)]
    # 得到图片的灰度分布列
    for x in range(rows):
        for y in range(cols):
            bin[img[x,y]] = bin[img[x,y]]+1
    s_max = (0, -1)
    # 遍历寻找最优划分阈值
    for threshold in range(0,255):
        n_0 = sum(bin[:threshold])  # 阈值以下像素数
        n_1 = sum(bin[threshold:])   # 阈值以上像素数
        # 两侧频率
        w_0 = n_0/N
        w_1 = n_1/N
        # 阈值下平均灰度
        u_0 = sum([i*bin[i] for i in range(0, threshold)])/n_0 if n_0>0 else 0   #考虑极端划分
        # 阈值上平均灰度
        u_1 = sum([i*bin[i] for i in range(threshold, 256)])/n_1 if n_1>0 else 0
```

```
        # 总平均灰度
        u = w_0*u_0 + w_1*u_1
        # 类间方差
        Dbet2 = w_0*((u_0-u)**2) + w_1*((u_1-u)**2)
        # 跟先前最优比较(取绝对大者意味着右偏，二分时阈值需分配给左半边)
        if Dbet2>s_max[1]:
            s_max=(threshold,Dbet2)
    return s_max[0]

# 定义函数实现寻找bgt阈值
def bgt_threshold(img):
    rows,cols = img.shape
    bin=[0 for x in range(256)]
    # 设定初始阈值
    threshold=123
    t=0
    # 得到图片的灰度分布列
    for x in range(rows):
        for y in range(cols):
            bin[img[x,y]] = bin[img[x,y]]+1
    # 迭代至阈值收敛
    while True:
        t = threshold
        n_0 = sum(bin[:int(threshold)])  # 阈值以下像素数
        n_1 = sum(bin[int(threshold):])   # 阈值以上像素数
        # 阈值下平均灰度
        u_0 = sum([i*bin[i] for i in range(0, int(threshold))])/n_0 if n_0>0 else 0   #考虑极端划分
        # 阈值上平均灰度
        u_1 = sum([i*bin[i] for i in range(int(threshold), 256)])/n_1 if n_1>0 else 0
        # 得到新阈值
        threshold = (u_0+u_1)/2
        if abs(t-threshold)<0.1:
            break
    return threshold

#主函数部分
path='图片库//finger.tif'
img = io.imread(path)
# 原始图像
plt.subplot(1,3,1)
io.imshow(img)
plt.title('origin image')
plt.axis("off")

# otsu法
threshold=otsu_threshold(img)
print("otsu_threshold is",threshold)
# 根据阈值二分
rows,cols = img.shape
for i in range(rows):
    for j in range(cols):
        img[i,j]=0 if img[i,j]<=threshold else 255
plt.subplot(1,3,2)
io.imshow(img)
plt.title('after otsu_threshold')
plt.axis("off")
```

```
# bgt法
img = io.imread(path)
threshold=bgt_threshold(img)
print("bgt_threshold is",threshold)
rows,cols = img.shape
# 根据阈值二分
for i in range(rows):
    for j in range(cols):
        img[i,j]=0 if img[i,j]<=threshold else 255
plt.subplot(1,3,3)
io.imshow(img)
plt.title('after bgt_threshold')
plt.axis("off")
# 展示
plt.show()
```

运行结果：

otsu_threshold is 160
bgt_threshold is 159.40149844112722

分析：bgt法与otsu法得到的阈值接近且处理图片结果都很好，两种方法可以相互印证

2

Design an algorithm of locally adaptive thresholding based on local OTSU or maximum of local entropy; implement the algorithm and test it on exemplar image(s).

代码：（matlab）（基于局部otsu）

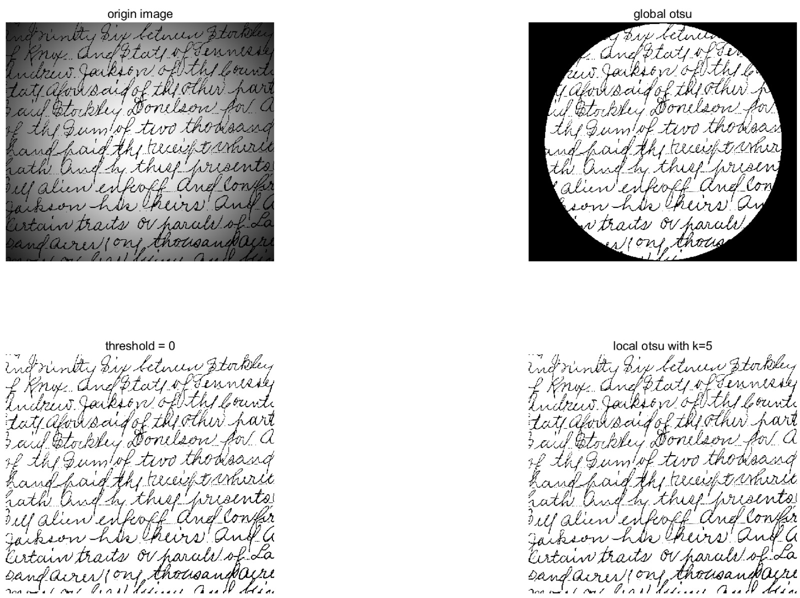
```
clear all;clc;close all;
img=imread("图片库//writting.tif");
```

```
subplot(2,2,1);
imshow(img);
title("origin image");
bin=makebin(img);
t=otsu_find(bin,0);
img2=partition(img,t);
subplot(2,2,2);
imshow(img2);
title("global otsu")
img3=partition(img,0);
subplot(2,2,3);
imshow(img3);
title("threshold = 0")
img4=local_otsu_do(img,5);
subplot(2,2,4);
imshow(img4);
title("local otsu with k=5")

% otsu寻找阈值的函数
function t=otsu_find(bin,k)
s_max =[0,0];
N=sum(bin(:));
i=0:255;    %递增行向量，用于算期望
for threshold=1:256
    u=0;
    n_0 = sum(bin(1:threshold));  % 阈值以下像素数
    n_1 = sum(bin(threshold:256));  % 阈值以上像素数
    % 两侧频率
    w_0 = n_0/N;
    w_1 = n_1/N;
    % 阈值下平均灰度
    if(n_0>0)
        u_0 = i(1:threshold)*bin(1:threshold)/n_0;
    else
        u_0=0;   %考虑极端划分
    end
    % 阈值上平均灰度
    if(n_1>0)
        u_1 = i(threshold:256)*bin(threshold:256)/n_1;
    else
        u_1=0;   %考虑极端划分
    end
    % 总平均灰度
    u = w_0*u_0 + w_1*u_1;
    % 类间方差
    Dbet2 = w_0*((u_0-u)^2) + w_1*((u_1-u)^2);
    % 跟先前最优比较(取绝对大者意味着右偏，二分时阈值需分配给左半边)
    if (Dbet2>s_max(2))
        s_max=[threshold-1,Dbet2];
    end
end
t=s_max(1);
%局部法需开启的修正模式：当选区中存在奇异点，则让更新点跟随大流实现减弱噪音干扰
if(k==1)
    if(w_0>0.95||w_1>0.95)
        if(w_0>0.5)
            t=0;
        else
```

```
            t=255;
        end
    end
end
end

% 生成bin的函数
function bin=makebin(img)
[m,n]=size(img);
bin=zeros(256,1);
for i=1:m
    for j=1:n
        bin(img(i,j)+1)=bin(img(i,j)+1)+1;
    end
end
end

% 根据阈值进行二分的函数
function img_new=partition(img,t)  % t为阈值
[m,n]=size(img);
img_new=zeros(m,n);
for i=1:m
    for j=1:n
        if(img(i,j)<=t)
            img_new(i,j)=0;
        else
            img_new(i,j)=255;
        end
    end
end
end

% 局部otsu的函数
function img_new=local_otsu_do(img,k) % k为滑窗大小，应为奇数
[m,n]=size(img);
img_new=zeros(m,n);
q=(k-1)/2;  % 滑窗半径
b=zeros(m+2*q,n+2*q); %扩张工作空间减少边界讨论
b(1+q:m+q,1+q:n+q)=img; %搬运原数组
% 柔化扩充部分边界
b(1:q,1+q:n+q)=flipud(img(2:q+1,1:n));
b(m+q+1:m+2*q,1+q:n+q)=flipud(img(m-q:m-1,1:n));
b(1+q:m+q,1:q)=fliplr(img(1:m,2:q+1));
b(1+q:m+q,1+n+q:n+2*q)=fliplr(img(1:m,n-q:n-1));
bin=makebin(b(1:1+2*q,1:1+2*q)); %初始化
flag=1; % 采用缝针式滑窗，flag用于判断当前行是左滑还是右滑以及更新bin
j=1+q;
% 直接对上次作业滑窗代码进行改写
for i=1+q:m+q
    t=otsu_find(bin,1);
    img_new(i-q,j-q)=partition(b(i,j),t);
    sym=(1-flag)/2;  %配合flag实现判断当前行是左滑还是右滑
    qqq=1;  %判断是不是一行的开始的标志，如果是则跳过一次更新
    for j=(sym*(2*q+1+n)+flag*(1+q)):flag:(sym*(2*q+1+n)+flag*(n+q))
        if(qqq==1)
            qqq=0;
            continue;
        end
```

```
            for x=i-q:i+q
                bin(b(x,j-flag*(q+1))+1)= bin(b(x,j-flag*(q+1))+1)-1;
                bin(b(x,j+flag*q)+1)= bin(b(x,j+flag*q)+1)+1;
            end
            t=otsu_find(bin,1);
            img_new(i-q,j-q)=partition(b(i,j),t);
        end
        flag=-1*flag;
        if(i~=m+q)
            for x=j-q:j+q
                bin(b(i-q,x)+1)= bin(b(i-q,x)+1)-1;
                bin(b(i+1+q,x)+1)= bin(b(i+q+1,x)+1)+1;
            end
        end
    end
end
```

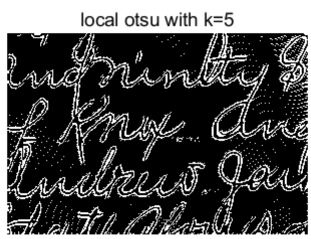
运行结果：



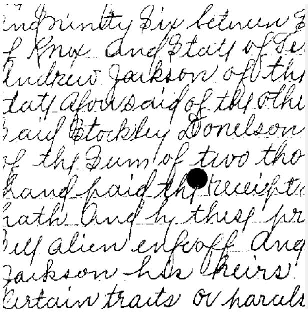
分析：

1.我使用的图片素材为上课使用的，但经过我研究认为这一素材不适合用于局部otsu，原因在于：正如上面左下图（t=0）处理结果与局部otsu基本一样，原因在于字迹点灰度0，其余均严格＞0，因此使用固定阈值就可以处理这样的图像，比局部otsu高效太多了。

2.局部otsu需要配合修正使用，如果不修正，那么背景内部点常被误分为前景点，素材中体现为有背景为黑色（如下图，为运行更快只对图片的一小块应用代码）



2.（续）但是！如果修正的条件不合适，可能会不能完全修正，比如我第一次使用类间方差很小作为条件，结果修正图像正中间亮度最大处有坏点（如下图）。经研究发现使用集中性条件最好，即选区内极多数点都被划分到一侧时进行修正。



3

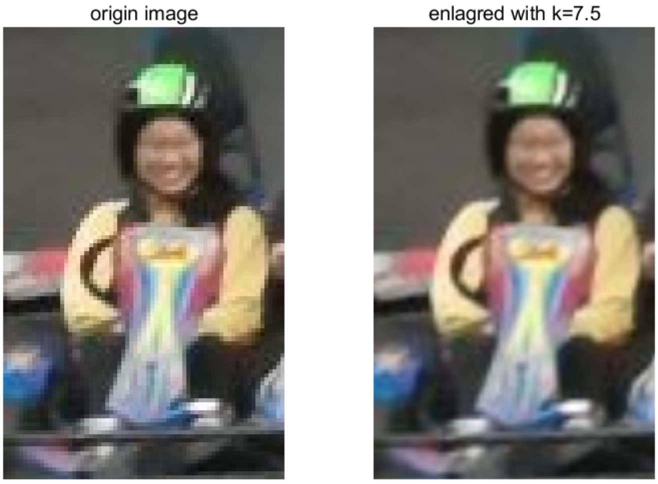
编程实现线性插值算法(不能调用某个算法库里面的插值函数)，并应用：读出一幅图像，利用线性插值把图片空间分辨率放大N倍，然后保存图片。

代码： （matlab）

```
clc;clear all;close all;
img=imread('图片库//yxy.jpg');%读取图像信息
% 可灰度化，下文代码兼容
% img=rgb2gray(img);
subplot(121);
imshow(img);%显示原图
title('origin image');
subplot(122);
b=enlarge(img,7.5);
imshow(b);%显示处理后图
title('enlagred with k=7.5')
imwrite(b,'./图片库/双线性插值放大结果.png'); % 将图片保存到图片库

%进行双线性放大的函数
function b=enlarge(img,k)    % k为放大倍率，可以为非整数，但要大于一
[m,n,c]=size(img);   % 记录原图数组三维
B=zeros(m+1,n+1,c); % 扩张原图数组省去（1）处讨论边界
B(1:m,1:n,:)=img;    % 搬运
% 生成目标图数组的工作空间
M=ceil(k*m); %向上取整
N=ceil(k*n);
b=zeros(M,N,c);
% 对目标图数组每个元素依此双线性插值
for i=1:M
    for j=1:N
        % 将新图坐标仿射变换到原图，(x,y)为插值所用点的左上点坐标，u、v为组合系数
        x=floor((i-1)*(m-1)/(M-1)+1);
        y=floor((j-1)*(n-1)/(N-1)+1);
        u=(i-1)*(m-1)/(M-1)+1-x;
        v=(j-1)*(n-1)/(N-1)+1-y;
        b(i,j,:)=(1-v)*(1-u)*B(x,y,:)+v*(1-u)*B(x,y+1,:)+(1-v)*u*B(x+1,y,:)+u*v*B(x+1,y+1,:); % （1）
    end
end
```

```
b=uint8(b); % 整型化
end
```

运行结果：



分析：经双线性插值放大后明显看到像素块不那么明显了，说明处理确实增加了分辨率。

在写代码时我最初将m*n阵中(i，j)元素先映到(ki，kj)再插值，但完善后左边和上边会存在宽度为k-1的黑边，若直接用邻域像素填充会让图形扭曲，实际只把原图放大到右下角(km-k+1,kn-k+1)一块，放大倍率小于k，且全图不是均匀放大，对此我不能接受，最后采用了放射变换整数化进行插值，实现真正意义上等比放大k倍。