Homework2

April 9, 2022

PROBLEM1 Answer:

BGT算法的步骤为：
1.选择初始阈值T_0（一般为整个图像的平均像素值）
2.根据阈值T_i将图像分成两组$R_1^{(i)}$，$R_2^{(i)}$
3.在两组中分别计算平均像素值$\mu_1^{(i)}$,$\mu_2^{(i)}$
4.计算新的阈值$T_{i+1} = (\mu_1^{(i)} + \mu_2^{(i)})/2$
5.重复步骤2-4直到两次阈值间的距离$T_{i+1} - T_i$小于参数ϵ

BGT算法我们可以基于直方图对其进行改进，记：
N:总像素点的个数 n_i:像素值为i的像素点的个数
p_i:像素值为i的像素点的频率
$\mu_1^{(i)}$:根据T_i划分出的第1组的平均像素值
$\mu_2^{(i)}$:根据T_i划分出的第2组的平均像素值
$m_1^{(i)}$:根据T_i划分出的第1组的像素点频率和
$m_2^{(i)}$:根据T_i划分出的第2组的像素点频率和
则:

$$m_1^{(i)} = \sum_{k=0}^{T_i} p_k$$

$$m_2^{(i)} = \sum_{k=T_i+1}^{L-1} p_k = 1 - m_1^{(i)}$$

$$\mu_1^{(i)} = \sum_{k=0}^{T_i} kp_k/m_1^{(i)}$$

$$\mu_2^{(i)} = \sum_{k=T_i+1}^{L-1} kp_k/(1 - m_1^{(i)})$$

$$T_{i+1} = \frac{1}{2}[\mu_1^{(i)} + \mu_2^{(i)}]$$

以下为代码：

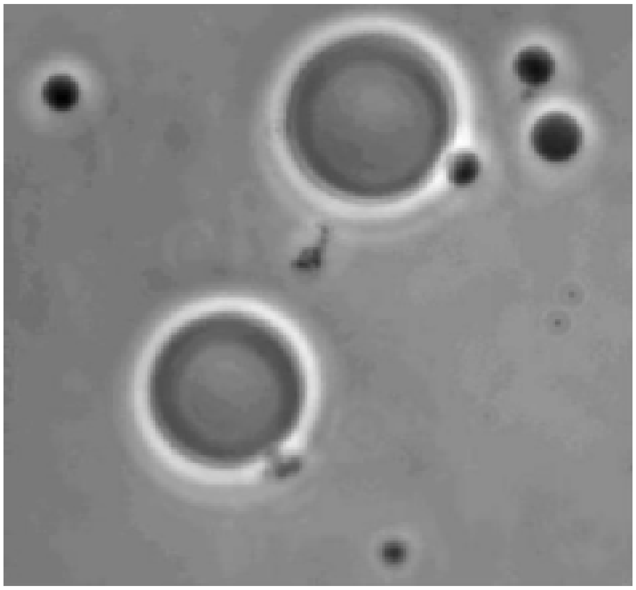
[79]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
np.set_printoptions(threshold=np.inf)
```

[80]:
```
#
def nD_histogram(data,dimension,nbins,pInMin,pInMax):
    pHistogram = []  # store histogram points
    pHsize = 1
    for idim in range(dimension):
        pHsize *= nbins[idim]
    for i in range(pHsize):
        pHistogram.append(0)
    pBinSpacings = []   # store bin width
    pBinPos = []   # store bin position
    for i in range(dimension):
        pBinSpacings.append(0)
        pBinPos.append(0)
    for idim in range(dimension): #store bin width of different dimensions
        pBinSpacings[idim] = (pInMax[idim] - pInMin[idim])/nbins[idim]
    for idata in range(len(data)):
        for idim in range(dimension):
            value = data[idata][idim]
            pBinPos[idim] = int((value - pInMin[idim])/pBinSpacings[idim])
            #
            pBinPos[idim] = max(pBinPos[idim],0)
            pBinPos[idim] = min(pBinPos[idim],nbins[idim] - 1)
        index = pBinPos[0]
        for idim in range(1,dimension):
            vSize = 1
            for i in range(idim):
                vSize *= nbins[i]
            index += pBinPos[idim] * vSize
        pHistogram[index] += 1
    return np.array(pHistogram)
```

[172]:
```
def BGT(imhist, epsilon = 0.1):
    L = imhist.shape[0]
    N = np.sum(imhist)
    mu_g = 0   #
    for i in range(L):
        mu_g += i*imhist[i]/N
    T = int(mu_g)
    while True:
        R_1 = imhist[:T]/N
        R_2 = imhist[T:]/N
```

```
        m_1 = np.sum(R_1)
        m_2 = 1 - m_1
        mu_1 = np.sum(np.arange(T)*R_1)/m_1
        mu_2 = np.sum(np.arange(T,L)*R_2)/m_2
        new_T = int((mu_1+mu_2)/2)
        if abs(new_T - T) < epsilon:
            break
        T = new_T
    return T
```

[173]:
```
im = Image.open('polymersomes.tif').convert('L')
im_ary = np.array(im, dtype='int32')
new_image = plt.imshow(im_ary,cmap='gray')
plt.axis("off")
plt.show()
```



[174]:
```
m,n = im_ary.shape
N = m*n    #
L = 256
imhist = nD_histogram(im_ary.reshape((N, 1)), 1, [L], [0], [L])  #
T = BGT(imhist)
im_ary[im_ary < T] = 0
im_ary[im_ary >= T] = 1
new_imarry = im_ary * (L - 1)
plt.imshow(new_imarry, cmap='gray')
plt.axis("off")
plt.show()
```

如图所示，很好的实现了二值化处理。

PROBLEM2

根据像素点的邻域确定此像素点使用的阈值，进行自适应的二值化处理。代码如下：

[84]:
```
from tqdm import tqdm
```

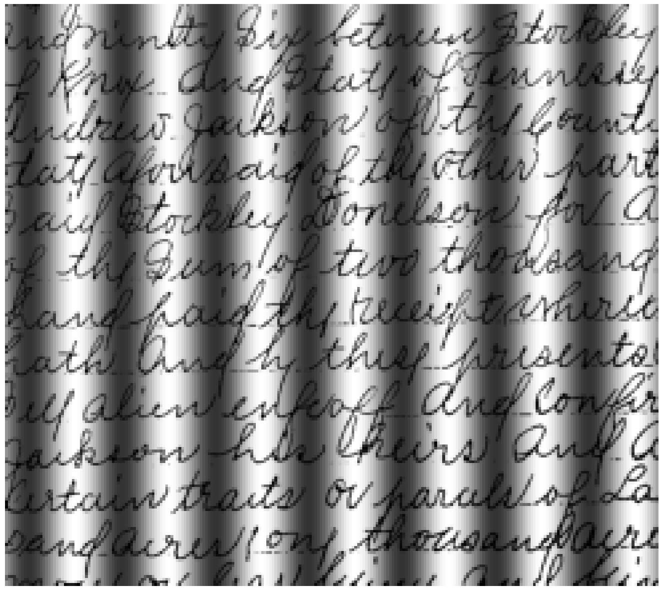
对于下面的图片，我们采用按行移动平均的方式进行对图片进行阈值处理。（使用局部直方图的高效更新算法）。

[139]:
```
def OTSU2(imhist):
    L = imhist.shape[0]
    N = np.sum(imhist)
    standn = np.arange(L)
    mu_g = np.sum(standn*imhist)/N    #
    w_1 = 0
    m_1 = 0
    max_y = float('inf')
    index = 0
    for i in range(L):   #
        w_1 += imhist[i]/N    # Frequency
        m_1 += i*imhist[i]/N
        if w_1 == 0:
            y = 0
        elif w_1 == 1:
            break
        else:
            y = (mu_g * w_1 - m_1) ** 2 / (w_1 * (1 - w_1))
        if y >= max_y:
```

```
            max_y = y
            index = i
    return index
```

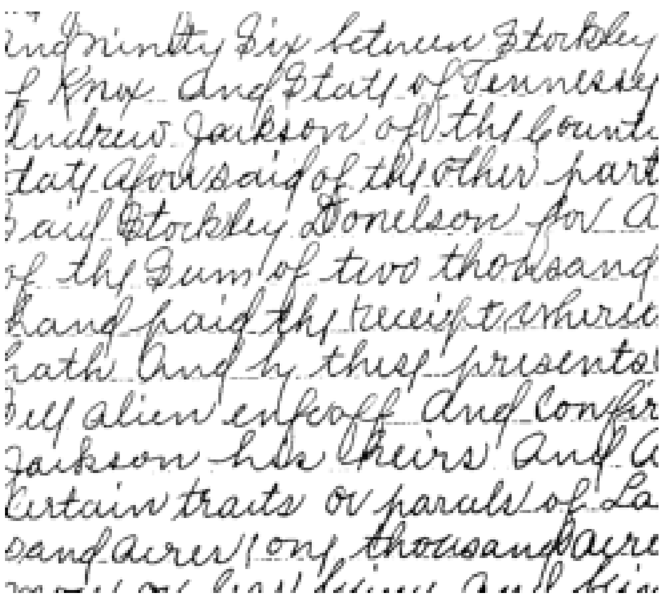
[151]:
```
def local_Thresholding2(imary, local_size, L = 256, s_OTSU = False, s_Entropy =␣
 ↪False):
    k = local_size
    im_o = imary
    row_length,col_length = im_o.shape
    #padding
    col_length += k - 1
    im = np.zeros((row_length,col_length),dtype='int32')
    im[:,k//2:col_length-k//2] = im_o
    # initialize local histgram:
    #store the result
    s_data = np.zeros(im.shape)
    for row in tqdm(range(row_length)):
        hist = nD_histogram(im[0, 0:k].reshape((k, 1)), 1, [L], [0], [L])
        for col in range(col_length-k+1):
            if col != 0:
                hist[im[row,col-1]] -= 1
                hist[im[row, col + k - 1]] += 1
            if s_OTSU:
                T = OTSU2(hist)
                # print(T)
                if im[row, col + k // 2] <= T:
                    s_data[row, col + k // 2] = 0
                else:
                    s_data[row, col + k // 2] = L - 1
    return s_data[:,k//2:col_length-k//2]
```

[163]:
```
im2 = Image.open('sine_shaded.tif').convert('L')
im_ary2 = np.array(im2, dtype='int32')
new_image2 = plt.imshow(im_ary2,cmap='gray')
plt.axis("off")
plt.show()
```



[164]:
```python
new_imarry2 = local_Thresholding2(im_ary2, 15, s_OTSU = True)
plt.imshow(new_imarry2, cmap='gray')
plt.axis("off")
plt.show()
```

```
100%|
   | 686/686 [03:14<00:00,  3.52it/s]
```

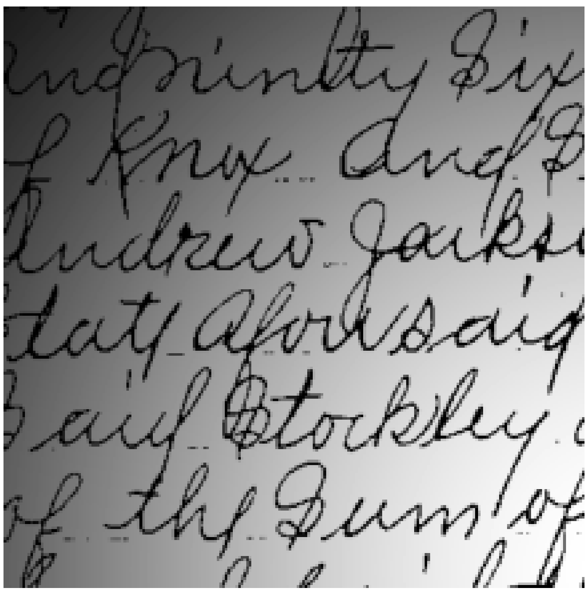


从实验结果来看，效果很好。
此外，我们还可以用小正方块形状的邻域进行滑动平移。

[168]:
```
def local_Thresholding(image, local_size, L = 256, s_OTSU = False):
    k = local_size
    im_o = np.array(image,dtype='int32')
    row_length,col_length = im_o.shape
    #padding
    row_length += k // 2 * 2
    col_length += k // 2 * 2
    im = np.zeros((row_length,col_length),dtype='int32')
    im[k//2:row_length-k//2,k//2:col_length-k//2] = im_o
    # initialize local histgram:
    hist = nD_histogram(im[0:k,0:k].reshape((k*k,1)),1,[L],[0],[L])
    #store the result
    s_data = np.zeros(im.shape)
    for row in tqdm(range(row_length-k+1)):
        if row % 2 == 0:
            for col in range(col_length-k+1):
                if col != 0:
                    sub_hist = nD_histogram(im[row:row+k,col-1].
↪reshape((k,1)),1,[L],[0],[L])
                    add_hist = nD_histogram(im[row:row+k,col+k-1].
↪reshape((k,1)),1,[L],[0],[L])
                    hist = hist + add_hist - sub_hist
                if s_OTSU:
                    T = OTSU2(hist)
                    # print(T)
                    if im[row + k // 2, col + k // 2] <= T:
                        s_data[row + k // 2, col + k // 2] = 0
                    else:
                        s_data[row + k // 2, col + k // 2] = L - 1
            if row != row_length-k:
                sub_hist = nD_histogram(im[row,col_length-k:col_length].
↪reshape((k,1)),1,[L],[0],[L])
                add_hist = nD_histogram(im[row+k,col_length-k:col_length].
↪reshape((k,1)),1,[L],[0],[L])
                hist = hist + add_hist - sub_hist
        else:
            for col in range(col_length-1,k-2,-1):
                if col != col_length - 1:
                    sub_hist = nD_histogram(im[row:row+k,col+1].
↪reshape((k,1)),1,[L],[0],[L])
                    add_hist = nD_histogram(im[row:row+k,col-k+1].reshape((k,␣
↪1)), 1, [L], [0], [L])
                    hist = hist + add_hist - sub_hist
                if s_OTSU:
```

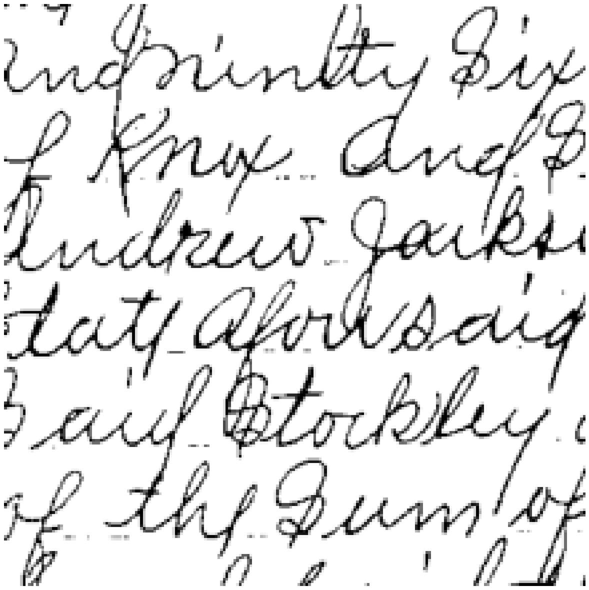
```
                    T = OTSU2(hist)
                    # print(T)
                    if im[row + k // 2, col - k // 2] <= T:
                        s_data[row + k // 2, col - k // 2] = 0
                    else:
                        s_data[row + k // 2, col - k // 2] = L - 1
            if row != row_length-k:
                sub_hist = nD_histogram(im[row,0:k].
↪reshape((k,1)),1,[L],[0],[L])
                add_hist = nD_histogram(im[row+k,0:k].reshape((k,␣
↪1)),1,[L],[0],[L])
                hist = hist + add_hist - sub_hist
    return s_data[k//2:row_length-k//2,k//2:col_length-k//2]
```

[169]:
```
im1 = Image.open('spot_shaded.tif').convert('L')
im_ary1 = np.array(im1, dtype='int32')[:355,:355]
new_image1 = plt.imshow(im_ary1,cmap='gray')
plt.axis("off")
plt.show()
```



[170]:
```
new_imarry1 = local_Thresholding(im_ary1, 15, s_OTSU = True)
plt.imshow(new_imarry1, cmap='gray')
plt.axis("off")
plt.show()
```

```
100%|
  | 355/355 [01:02<00:00,  5.72it/s]
```



可以猜想，对于竖直方向上像素值变化不明显的图，采用按照行的移动平均来进行自适应的阈值处理会有比较好的

PROBLEM3

二维图片线性插值的方式如下图所示:

$$I(p_a) = (1-r)I(p_2) + rI(p_3)I(p_b) = (1-r)I(p_0) + rI(p_1)$$

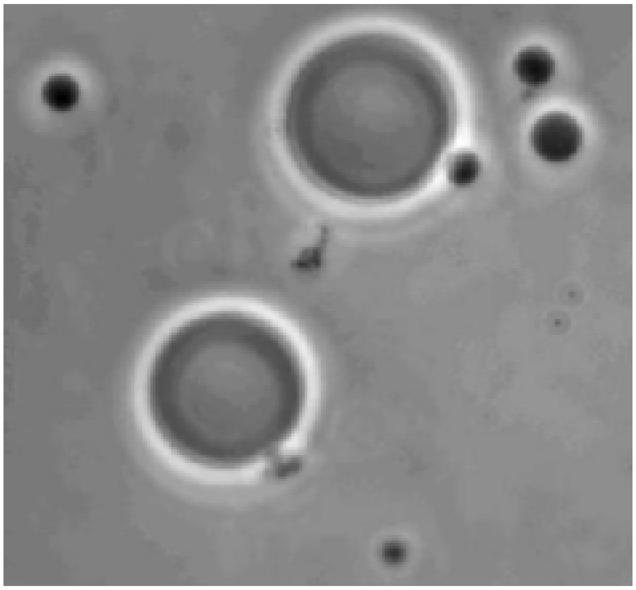
$$\begin{aligned} I(p) &= (1-s)I(p_b) + rI(p_a) \\ &= (1-r)(1-s)I(p_0) + r(1-s)I(p_1) + s(1-r)I(p_2) + srI(p_3) \end{aligned}$$

代码如下:

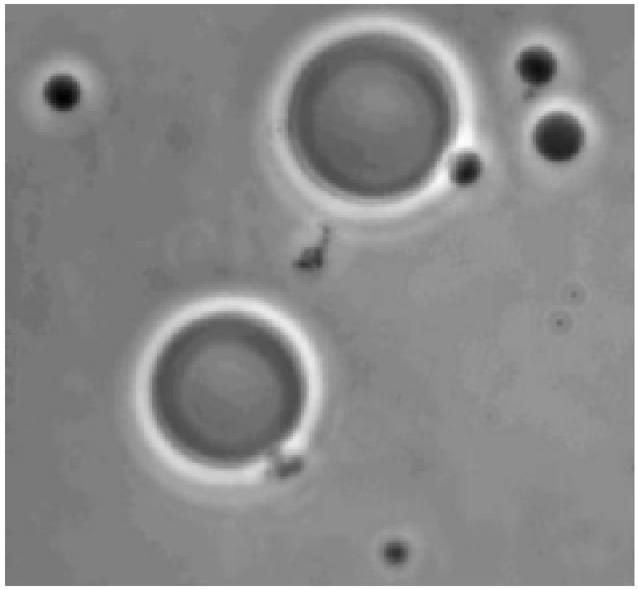
[14]:
```python
def linearinter(imary,N):
    """
    input:
    imary: ndarry
    N: int(>1)      (N=2
    output:
    newary: ndarry
    """
    m,n = imary.shape
    new_m = N * (m-1) + 1
    new_n = N * (n-1) + 1
    newary = np.zeros((new_m,new_n))
```

```
    for p_row in tqdm(range(m-1)):
        for p_col in range(n-1):
            I_p0 = imary[p_row,p_col]
            I_p1 = imary[p_row,p_col+1]
            I_p2 = imary[p_row+1, p_col]
            I_p3 = imary[p_row+1, p_col+1]
            newary[p_row*N,p_col*N] = I_p0
            newary[p_row*N,p_col*N+N] = I_p1
            newary[p_row*N+N,p_col*N] = I_p2
            newary[p_row*N+N,p_col*N+N] = I_p3
            for i in range(0,N+1):
                for j in range(0,N+1):
                    s = i/N
                    r = j/N
                    newary[p_row*N+i,p_col*N+j] = (1-s)*(1-r)*I_p0 +␣
↪r*(1-s)*I_p1+ \
                                                  s*(1-r)*I_p2 + s*r*I_p3
    return newary
```

[15]:
```
im = Image.open('polymersomes.tif').convert('L')
im_ary = np.array(im, dtype='int32')
new_image = plt.imshow(im_ary, cmap='gray')
plt.axis("off")
plt.show()
new_ary = linearinter(im_ary,2)
plt.imshow(new_ary, cmap='gray')
plt.axis("off")
plt.show()
```



```
100%|
   | 647/647 [00:29<00:00, 22.29it/s]
```



查看图片像素分辨率，发现变为两倍。而且视觉上更加清晰。

[146]:
```
im_ary.shape
```

[146]: (648, 702)

[111]:
```
new_ary.shape
```

[111]: (1295, 1403)