HW2

夏浩 19307130268

P1. 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)

$$P(i) = \frac{n_i}{N},$$

n_i: number of pixels in level i, N : $totlapixelnumberoftheimage$

Step 1. define $T_0 \in (a, b)$,$w_{i,k} = \sum_i^k P(i)$,

Step2.
$u_1 = \sum_a^{T_i} iP(i)/w_{a,T_i}$,
$u_2 = \sum_{T_i}^b iP(i)/w_{T_i,b}$.

Step 3.
Calculate $T_i = (u_1 + u_2)/2$,

Step 4.

Repeat steps 2-4 until: $T_i == T_{i-1}$ or $T_i - T_{i-1} < \varepsilon$

Step 5.

$F(i < T_{end}) = 0, F(i \geq T_{end}) = 1$

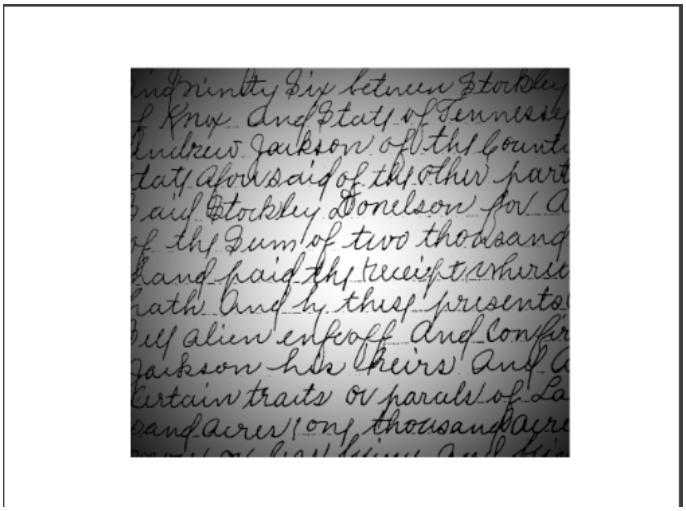
P2.Design an algorithm of locally adaptive thresholding based on local OSTU or maximum of local entropy; implement the algorithm and test it on exemplar image(s).

Take an n-dimensional line vector in the image matrix as the sliding window, scan the whole image line by line in the 'Z' shape (reduce the illumination error), calculate the local threshold T (x, y) with the local Otsu algorithm for each scanning result, and then compare the original image matrix with t to obtain the binary image.

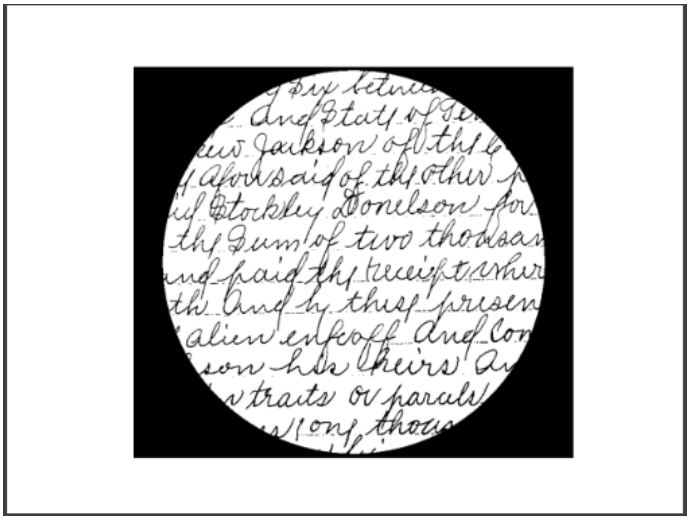
```
import numpy as np
```

```
import cv2 as cv
import matplotlib.pylab as plt

def otsu(im):
    histogram,bins = np.histogram(im.flatten(),256,
[0,256],density=True)
    P1 = np.zeros(256)
    P2 = np.zeros(256)
    m = np.zeros(256) #
    for i,p_i in enumerate(histogram):
        P_k = np.sum(histogram[0:i+1])
        P1[i] = P_k
        P2[i] = 1-P_k
        i_list = np.arange(i+1)
        m[i] = np.sum(i_list * histogram[:i+1])
    m1 = np.zeros(256)
    m2 = np.zeros(256)
    for i,p_i in enumerate(P1):
        i_list = np.arange(i + 1)
        if p_i != 0:
            sip1 = np.sum(i_list * histogram[0:i+1])
            m1[i] = 1/p_i * sip1
        else:
            m1[i] = np.mean(i_list)
    for i, p_i in enumerate(P2):
        i_list = np.arange(i, 256)
        if p_i != 0:
            sip2 = np.sum(i_list * histogram[i:256])
            m2[i] = (1 / p_i * sip2)
        else:
            m2[i] = np.mean(i_list)
    m_G = m[-1]
    i_list = np.arange(256)
    sigma_G2 =  np.sum((i_list-m_G) ** 2 * histogram)
    sigma_B2_np = np.zeros(256)
    sigma_B2_np = P1 *P2 *((m1-m2) ** 2)
    # print(sigma_B2_np)
    # print(m1,m2)
    # print(P1 + P2)
    T = np.mean(np.argwhere(sigma_B2_np == np.amax(sigma_B2_np)))
    return T

def l_walk(im,half_n= 7):
```

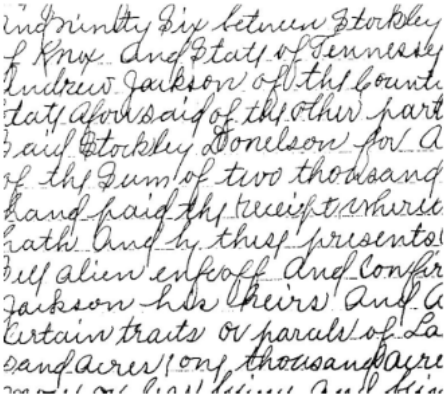
```
    w,h = im.shape
    newimg2 = np.zeros((w,h))
    histogram, bins = np.histogram(im.flatten(), 256, [0, 256],
density=True)
    i_list = np.arange(256)
    m_G = np.sum(i_list * histogram)
    im_border=
cv.copyMakeBorder(im,10,10,10,10,cv.BORDER_CONSTANT,value=int(m_G))
    for x in range(half_n,w + half_n):
        for y in range(half_n,h + half_n):
            T = otsu(im_border[x - half_n:x + half_n,y-half_n:y +
half_n ])
            newimg2[x-half_n,y-half_n] = im[x-half_n,y-half_n] > T
    return newimg2

# read picture
img = cv.imread('img1.tif',0)
newimg = np.zeros(img.shape)
print(img.shape, img.dtype, type(img))

# show image
plt.imshow(img,cmap = 'gray')
plt.axis('off')
plt.show()

# global
T = otsu(img)
newimg =  img > T

plt.imshow(newimg,cmap = 'gray')
plt.axis('off')
plt.show()
# local
newimg2 = l_walk(img,half_n=7)
plt.imshow(newimg2,cmap = 'gray')
plt.axis('off')
plt.show()
```



Original



Global thresholding

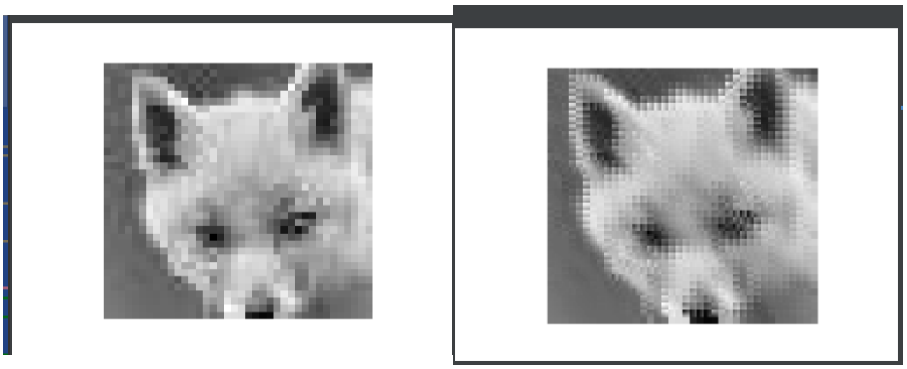


local thresholding

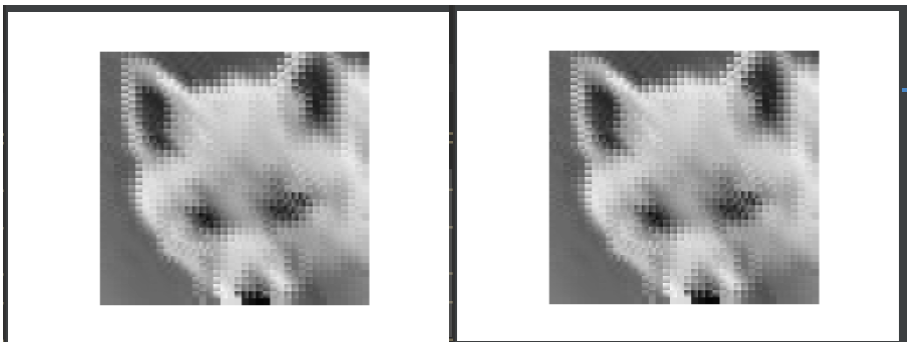
P3.编程实现线性插值算法**(**不能调用某个算法库里面的插值函数**)**，并应用：读出一幅图像，利用线性插值把图片空间分辨率放大**N**倍，然后保存图片。

```
import numpy as np
import cv2 as cv
import matplotlib.pylab as plt


def inter_line(im,n=2):
    w,h = im.shape
    w1,h1 = n*w,n*h
    newimg = np.zeros((w1,h1))
    for x1 in range(w1):
        for y1 in range(h1):
            # original image left top point
            x = x1 / n
            y = y1 / n
            x_int = x1 // n
            y_int = y1 // n
            a = x - x_int
            b = y - y_int

            if x_int + 1 == w or y_int + 1 == h:
                newimg[x1, y1] = im[x_int, y_int]
            else:
                newimg[x1, y1] = int((1. - a) * (1. - b) * im[x_int
+ 1, y_int + 1] + \
                                     (1. - a) * b * im[x_int, y_int +
1] + a * (1. - b) * im[x_int + 1, y_int]\
                                + a * b * im[x_int, y_int])

    return newimg
# read picture
img = cv.imread('img3.jpg',0)
print(img.shape, img.dtype, type(img))

# show image
plt.imshow(img,cmap = 'gray')
plt.axis('off')
plt.show()

# process
newimg = inter_line(img,3)
```

```
print(newimg.shape, newimg.dtype, type(newimg))
# show
plt.imshow(newimg,cmap = 'gray')
plt.axis('off')
plt.show()
# save
loc_gim = cv.imwrite('res3.jpg', img=newimg)
```



n = 3　　n= 5



n= 7　　n= 10

注意到，当临近点level相差不大时，简单提高倍数并不能得到很好的放大效果。