可视化作业2

Yanjian Zhang 16300200020

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

   1. Calculate the histogram of the image

   2. Select an initial threshold T0 (e.g. the mean intensity)

   3. Partition the histogram into two parts (R1 and R2) using the T0

   4. Calculate the mean intensity values $\mu 1$ and $\mu 2$ of the parts R1 and R2 with $\sum_{i\in[0,T_0]} i*P(i)$ and $\sum_{i\in[T_0,255]} i*P(i)$

   5. Select a new threshold: $Ti = (\mu 1 + \mu 2)/2$

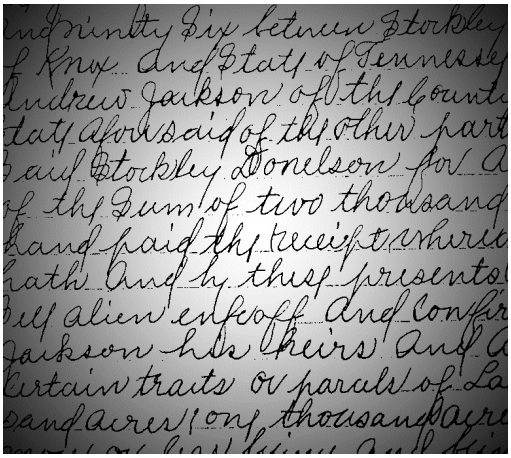
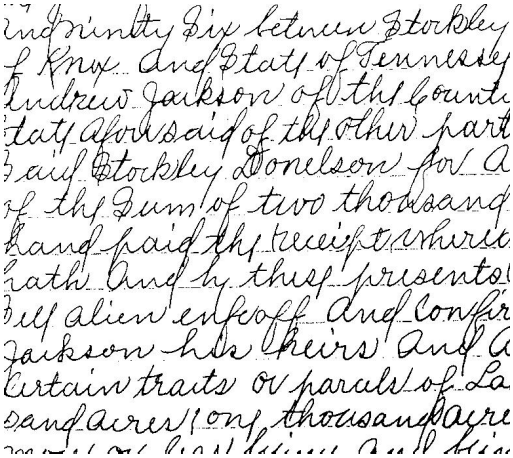
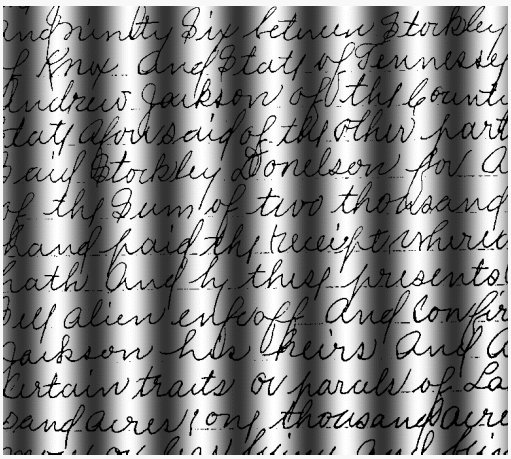
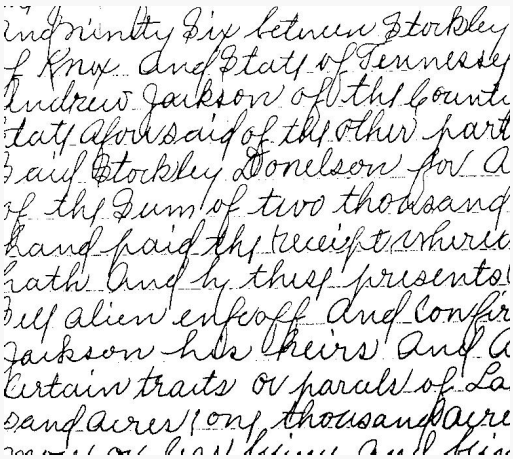
   6. Repeat steps 3-5 until: $T_i = T_{i-1}$

2. Design an algorithm with the function of locally adaptive thresholding (e.g. based on moving average or local OSTU); implement the algorithm and test it on exemplar image(s).

```
from PIL import Image
import numpy as np
import copy

def local_thresh(image, fringe):
    shape_y, shape_x = image.shape
    new_image = copy.deepcopy(image)
    total_sums = {}                          # storage previous computed sum
    for y in range(shape_y):
        for x in range(shape_x):
            if (x-1,y) in total_sums:
                # .... restore the left point sum, see the code
            elif (x,y-1) in total_sums:
                # .... restore the upper point sum, see the code
            else:
                total_sum = np.sum(image[ max(y-fringe,0):
min(y+fringe,shape_y), max(0,x-fringe):min(x+fringe, shape_x)],
dtype='int32')         # compute from None

            total_sums[(x,y)] = total_sum                      # save in
each step
            around_sum = total_sum - image[y][x]            # sum besides
the center
            threshold = around_sum/((min(x+fringe, shape_x)-max(0,x-
fringe))* (min(y+fringe,shape_y)-max(y-fringe,0))-1)
            new_image[y][x] = 255 if image[y][x] >= 0.4* threshold else 0

    return new_image

im = Image.open("./article_round.png")
# im = Image.open("./article_line.png")
im = im.convert('L')
image = np.array(im)
```

```
H ,W = image.shape
new_image = local_thresh(image, 45)
new_im = Image.fromarray(new_image)
if new_im.mode == 'F':
    new_im = new_im.convert('L')
new_im.save("./threshold_article_round.jpg")
# new_im.save("./threshold_article_line.jpg")
```

Result:

Origin Image	Processed Image
	
	

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。

```
import math
import numpy as np

def Interpolation(matrix, point):
    x, y = point
    shape_x, shape_y = matrix.shape
    if x < 1 or x > shape_x:
        return None
    if y < 1 or y > shape_y:
        return None
    x1,x2 = math.floor(x)-1, math.ceil(x)-1 # for index usage in matrix
    y1,y2 = math.floor(y)-1, math.ceil(y)-1
    x, y = x-1, y-1
```

```
    if x2 == x1:
        f1 = matrix[y1][x1]
        f2 = matrix[y2][x1]
    else:
        f1 = (x2 - x)/ (x2 - x1) * matrix[y1][x1] + (x - x1)/(x2 - x1) *
matrix[y1][x2]
        f2 = (x2 - x) / (x2 - x1) * matrix[y2][x1] + (x - x1) / (x2 - x1) *
matrix[y2][x2]
    if y2 == y1:
        fp = f1
    else:
        fp = (y2 - y) / (y2 - y1) * f1 + (y - y1)/(y2 - y1) * f2
    return fp


def scale(array, num):
    shape_y, shape_x = array.shape
    new_array = np.zeros((int(shape_x*num), int(shape_y*num)))
    transformed_step = 1/num
    for j in range(int(shape_y*num)):
        for i in range(int(shape_x*num)):
            new_array[j][i] = Interpolation(array,(transformed_step*i,
transformed_step*j))
    return new_array


from PIL import Image
im = Image.open("./brain_small.jpg")
image = np.array(im)
H ,W = image.shape
new_image = scale(image, 3)
new_im = Image.fromarray(new_image)
if new_im.mode == 'F':
    new_im = new_im.convert('L')
new_im.save("./scale_brain.jpg")
```

Result:

Origin Image	Processed Image