图像处理第 1 次作业报告

林子开

2023 年 9 月 30 日

目录

1 Contrast stretching by piecewise linear transformation

1.1 python codes

The python codes of exercise 1 are as follows:

Listing 1: The python codes for piecewise linear transformation

```python
# Exercise1: piecewise linear transformation
from PIL import Image

def piecewise_transf(intensity, r1, r2, s1, s2):
    if intensity < r1:
        newIntensity = s1*(intensity)/r1
    elif intensity <= r2:
        newIntensity = s1 + (s2-s1)*(intensity-r1)/(r2-r1)
    else: # intensity > r2
        newIntensity = s2 + (255-s2)*(intensity-r2)/(255-r2)
    return  int(newIntensity)


def intensity_transformation(pictureName,r1,r2,s1,s2):
    im_raw = Image. open(pictureName)
    im = im_raw.convert('L')
    pixels = im.load()
    for i in  range(im.size[0]): # for every pixel:
        for j in  range(im.size[1]):
            pixels[i,j] = piecewise_transf(pixels[i,j],r1,r2,s1,s2)

    im.save('piecewise linear transformation for'+ str(pictureName))
    im.show()


def main():
    myPicture = 'test1.jpeg'
    intensity_transformation(myPicture,5,40,5,80)


if __name__ == '__main__':
    main()
```

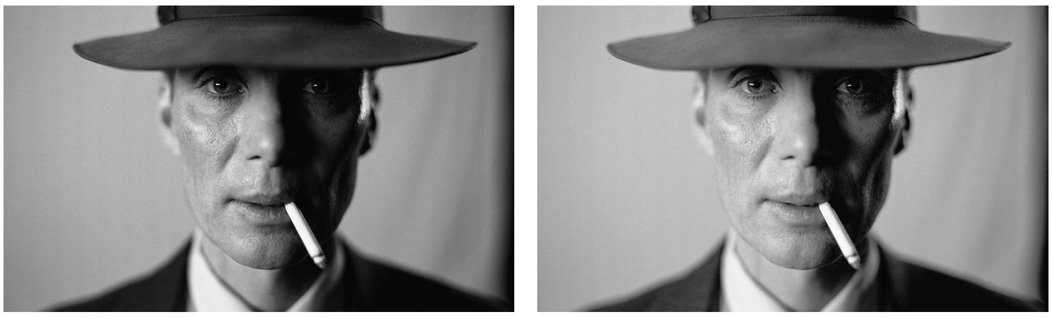
The function piecewise_transf is called to perform piecewise linear transformation on the intensity of the intensity of each pixel.

1.2 test example

Now we shall test the codes above with the following transformation:

$$f(x) = \begin{cases} x & \text{if } x < 5 \\ 5 + \frac{(x-5)(80-5)}{40-5} & \text{if } 5 \le x \le 40 \\ 80 + \frac{(x-40)(255-80)}{255-40} & \text{if } x > 40 \end{cases}$$

Here is the comparison between the original figure and the transformed figure.



(a) the original figure

(b) the transformed figure

图 1: the comparison between the original figure and the transformed figure in exercise 1

It can be observed that the contrast is stretched after the piecewise linear transformation especially for the darker half of the man's face.

2 Computation of joint histogram

2.1 Global n-dimension joint histogram

2.1.1 python codes

The core function is as follows:

Listing 2: The core python codes for n-dimension joint histgoram

```python
def n_dimension_joint_histogram(data,Min,bins,Max):
    """
    input:
    data is a n*k array, where n is the dimension and k is the number of elements.
    Each element is n-dimension.
    bins is a list or a tuple which contains the number of blocks in each direction.
    Min contains the lower bound in each dimension.
    Max contains the upper bound in each dimension.
    output:
    hist: an n-dimension array of frequency
    edges: a list of length n, containing the markers of axises in n directions.
    """
    n, *temp = data.shape
    edges = []
    spacing = []
    for i in  range(n):
        spacing.append(  round((Max[i] - Min[i])/bins[i])  )
        edges.append( list( range(Min[i],Max[i]+spacing[i],spacing[i])))
    hist = np.zeros(bins) # 创建n维数组用于存放histogram
    for i in  range(data.shape[i]): # 遍历所有element
        pos = []  # 用于确认该点的数据应处于联合直方图的哪个位置
        for k in  range(n): # pos记录该点每个维度上的分量
            pos.append( int(data[k,i]//spacing[k]))
        hist[ tuple(pos)] += 1
    return hist, edges
```

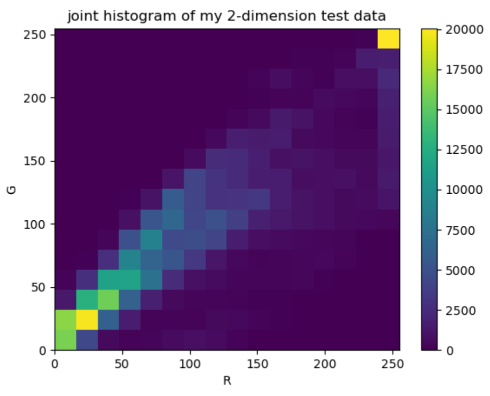
This function visit every element in the data, get the values in each dimension of the element, and add the tuple of values to the n-dimension joint histogram with bins given.

2.1.2 test example: 2-dimension joint histgoram

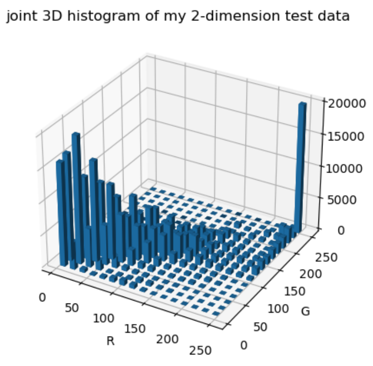
Here I test the codes above by a 2-dimension data. This data consists of red levels and green levels corresponding to every pixel from the following test figure:

图 2: the colored figure for test in exercise 2

I set bins=256 for the two dimensions and the results are as follows:



(a) the 2-D histogram



(b) the 3-D histogram

图 3: the joint histogram of the red levels and the green levels of every pixel from test figure

2.2 Local 2-dimension joint histogram

2.2.1 python code

The core function for efficiently computing the local histogram of an image is as follows:

Listing 3: The core python codes for efficient local joint histgoram computation

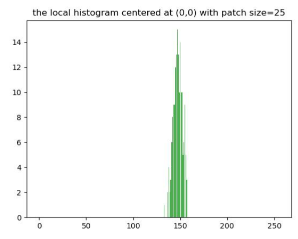
```python
def local_hist(pixels,w):
    """
    input:
        pixels is the intensity of the picture
        w is the size of the local patch, required to be an odd number
        bins is the number of partitions, usually taking 2^p
    output:
        hist_list: the first two elements of each row is the index of the central pixel,
            and the rest elements of each row are the local histogram
    """
    count = 0
    hist = np.zeros((pixels.size, 2 + 256)) # pixels.size返回的是pixels中有多少元素
    for i in  range(pixels.shape[0]):
        up =  max(0,i-w//2)
        down =  min(pixels.shape[0]-1,i+w//2)
        for j in  range(pixels.shape[1]):
            if j==0: # 第一个元素，重新计算patch
                hist[count,0] = i
                hist[count,1] = j
                rt =  min(pixels.shape[1]-1,j+w//2) # 边界检查，防止越界
                for p in  range(up,down+1):
                    for q in  range(rt+1):
                        hist[count,2+pixels[p,q]] += 1
                count += 1
            else: # 其他非首位元素的local histogram只需要做列更新
                hist[count,:] = hist[count-1,:]
                hist[count,0] = i
                hist[count,1] = j
                lf =  max(0,j-w//2)
                rt =  min(pixels.shape[1]-1,j+w//2)
                if lf==0: # 边界检查，防止越界
                    old_col = []
                else:
                    old_col = pixels[up:down+1,lf-1]
                if j+w//2 > rt: # 边界检查，防止越界
                    new_col = []
                else:
                    new_col = pixels[up:down+1,rt]
                for item in old_col:
                    hist[count,2+item] -= 1
                for item in new_col:
                    hist[count,2+item] += 1
                count = count + 1
    return hist
```

This function updating the local histogram with the new column introduced and the old column abandoned in a motion step.

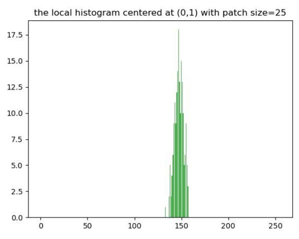
2.2.2 test example

I test the codes on the original figure1.2 from exercise 1. Since there are too many local histogram, I've seleced three groups of neighboured ones. They are the first 3, the middle 3, and the last 3 local histograms.

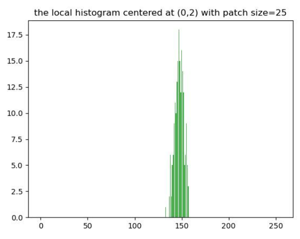
I set the patch size 25*25, and here are the results:



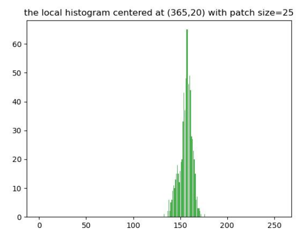
(a) centered at (0,0)



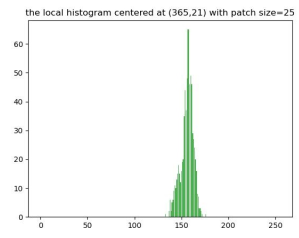
(b) centered at (0,1)



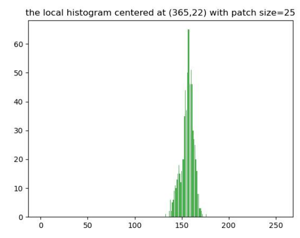
(c) centered at (0,2)



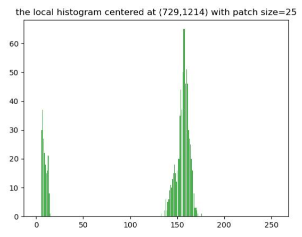
(d) centered at (365,20)



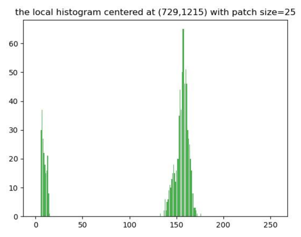
(e) centered at (365,21)



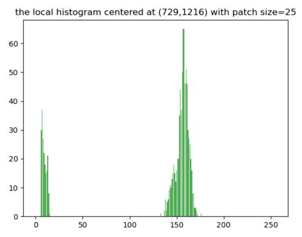
(f) centered at (365,22)



(g) centered at (729,1214)



(h) centered at (729,1215)



(i) centered at (729,1216)

图 4: some of the neighboured local histograms

3 Histgoram equalization

3.1 Global equalization

3.1.1 python codes

The core python codes for global equalization are as follows:

Listing 4: The core python codes for global equalization

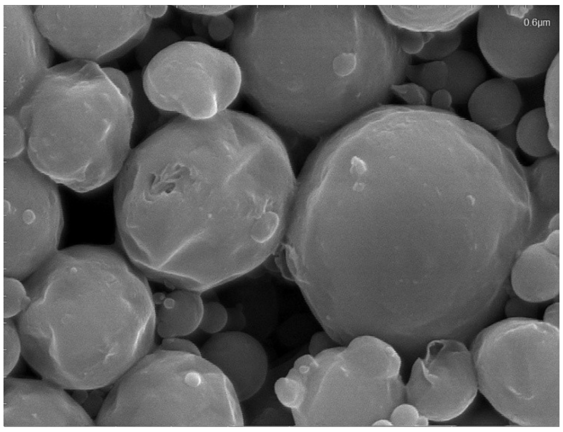
```python
def global_hist(pixels):
    """
    input: a 2-dimension array containg the intensity of each pixel
    output: a global histogram

    """
    hist = [0]*256
    for i in  range(pixels.shape[0]):
        for j in  range(pixels.shape[1]):
            hist[pixels[i,j]] += 1
    return hist


```

```python
def trans_function(hist):
    """
    input: a histogram in [0,255]
    output: a list of length 256, mapping the intensity of the original picture
        to that after equalization
    """
    cum_list = np.empty(256)
    cum_list[0] = hist[0]
    for i in  range(1,256):
        cum_list[i] = cum_list[i-1] + hist[i]
    cum_list = cum_list/cum_list[-1]
    trans = np.zeros(256)
    for i in  range(256):
        trans[i] = ceil(255*cum_list[i])
    return trans


def main():
    myPicture = 'test3.png'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)
    pixels = pixels.transpose()

    global equalization
    g_hist = global_hist(pixels)
    trans = trans_function(g_hist)
    print(trans)
    # plt.bar(list(range(256)),trans)
    # plt.show()
    im2 = Image.new('L', im.size)
    pixels2 = im2.load()
    for i in  range(im.size[0]):
        for j in  range(im.size[1]):
            pixels2[i,j] =  int(trans[pixels[i,j]])
    im2.save('global equalization.jpg')
    im2.show()
```

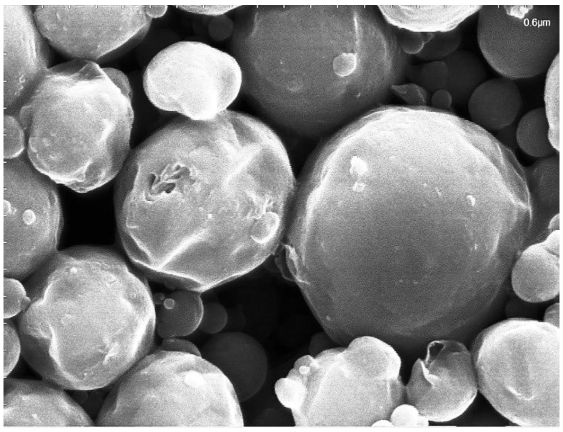
The codes above first generate the global histgoram of the figure. Based on the global histgora, a list is computed to map from the intensity in the original figure to that in the globally equalized figure.

3.1.2 test example

Here I select a gray figure under electronic microscope. To enhance its contrast, first perform global equalization on it. The comparison is as follows:



(a) the original figure



(b) the globally equalized figure

图 5: the orignal figure compared with the globally equalized figure

3.2 Local equalization

3.2.1 python codes

The core python codes are as follows:

Listing 5: The core python codes for local equalization

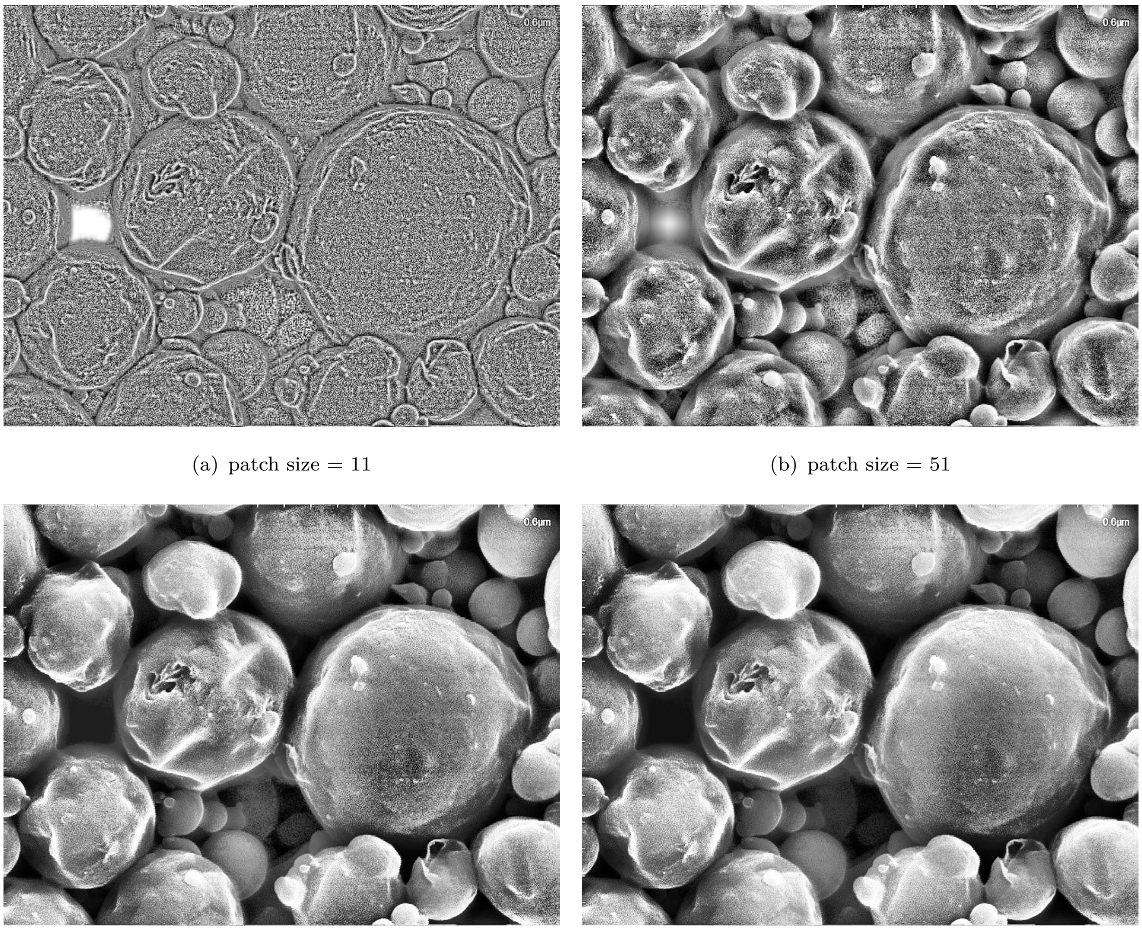
```python
# local equalization
for patch_size in (51,151,201):
    hist_list = local_hist(pixels,patch_size)
    print('local hist done!',patch_size)
    im3 = Image.new('L', im.size)
    pixels3 = im3.load()
    for row_index in  range(hist_list.shape[0]):
        h = hist_list[row_index,:]
        i =  int(h[0])
        j =  int(h[1])
        # print(i,j)
        trans = trans_function(h[2:])
        pixels3[i,j] =  int(trans[pixels[i,j]])
        # print(pixels[i,j],pixels3[i,j])
    im3.save('local equalization with patch size = '+ str(patch_size)+'.jpg')
    # im3.show()
```

The local_hist function is the same as that in exercise 2, so we shall omit the details of it.

For local equalization, we still generate the local histograms corresponding to every pixel. Based on the local histogram, we can perform the equalization transformation on the centered pixel. After that, we move to its neighboured one.

3.2.2 test exaple

I test the local equalization on orgianl figure3.1.2 with different patch sizes of 11, 51, 151, and 201. The results are as follows:



(a) patch size = 11 (b) patch size = 51

(c) patch size = 151 (d) patch size = 201

图 6: local equalization with different patch sizes

It seems that the local equalization with patch size=151 can show the best details among the four patch sizes.