数据可视化 第一次作业

张言健 16300200020

1 (1) implement n-dimensional joint histogram and test the code on two-dimensional data; plot the results. (2) implement computation of local histograms of an image using the efficient update of local histogram method introduced in local histogram processing.

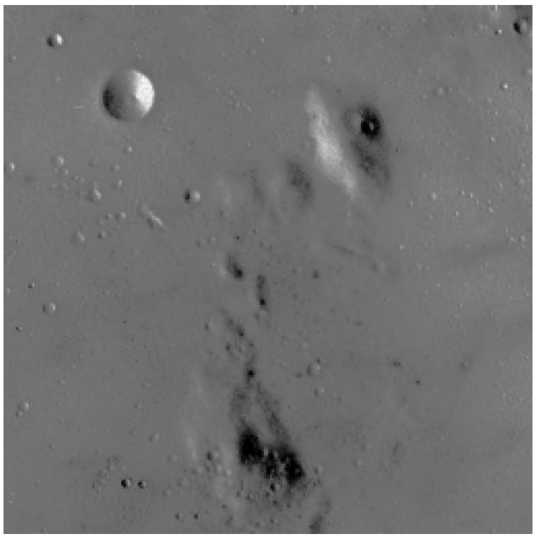
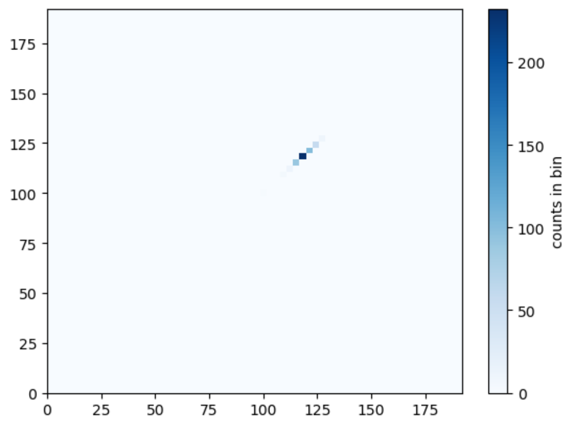
Note that because only one row or column of the neighborhood changes in a one-pixel translation of the neighborhood, updating the histogram obtained in the previous location

with the new data introduced at each motion step is possible and efficient in computation.

```
from skimage import io, data, img_as_float
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

# define bin count, similar to the function of np.bincount
def bincount(my_list, minlength=0):
    length = max(minlength, max(my_list))
    count = Counter(my_list)
    return np.array([count[x] for x in range(length)])


# 1. n-dimensional joint histogram
def eval_hist(my_data: np.ndarray, bins=(64, 64)):

    n_dim = len(bins)
    n_range = [[int(255/bins[i]) * j for j in range(bins[i]+1)]
               for i in range(n_dim)]
    nbin = np.empty(n_dim, int)
    for i in range(n_dim):
        nbin[i] = len(n_range[i])

    Ncount = tuple(
        np.searchsorted(n_range[i], my_data[:, i], side='right')
        for i in range(n_dim)
    )

    xy = np.ravel_multi_index(Ncount, nbin)

    hist = bincount(xy, minlength=nbin.prod())

    hist = hist.reshape(nbin)

    print(hist)
    return hist, n_range
```

```python
def efficien_compute(old_hist, removed_pixels, extend_pixels, bins=256):
    hist_remove = bincount(removed_pixels.reshape(-1), minlength=bins)
    hist_extend = bincount(extend_pixels.reshape(-1), minlength=bins)

    new_list = copy.deepcopy(old_hist)
    new_list = new_list - hist_remove + hist_extend

    return new_list


my_data = data.moon()
hist, edges = eval_hist(my_data, bins=(64, 64))
print(len(edges[0]), hist.shape)
plt.pcolormesh(edges[0], edges[1], hist.T, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
plt.show()
```

原图片与2d直方图

原图	2d直方图
	

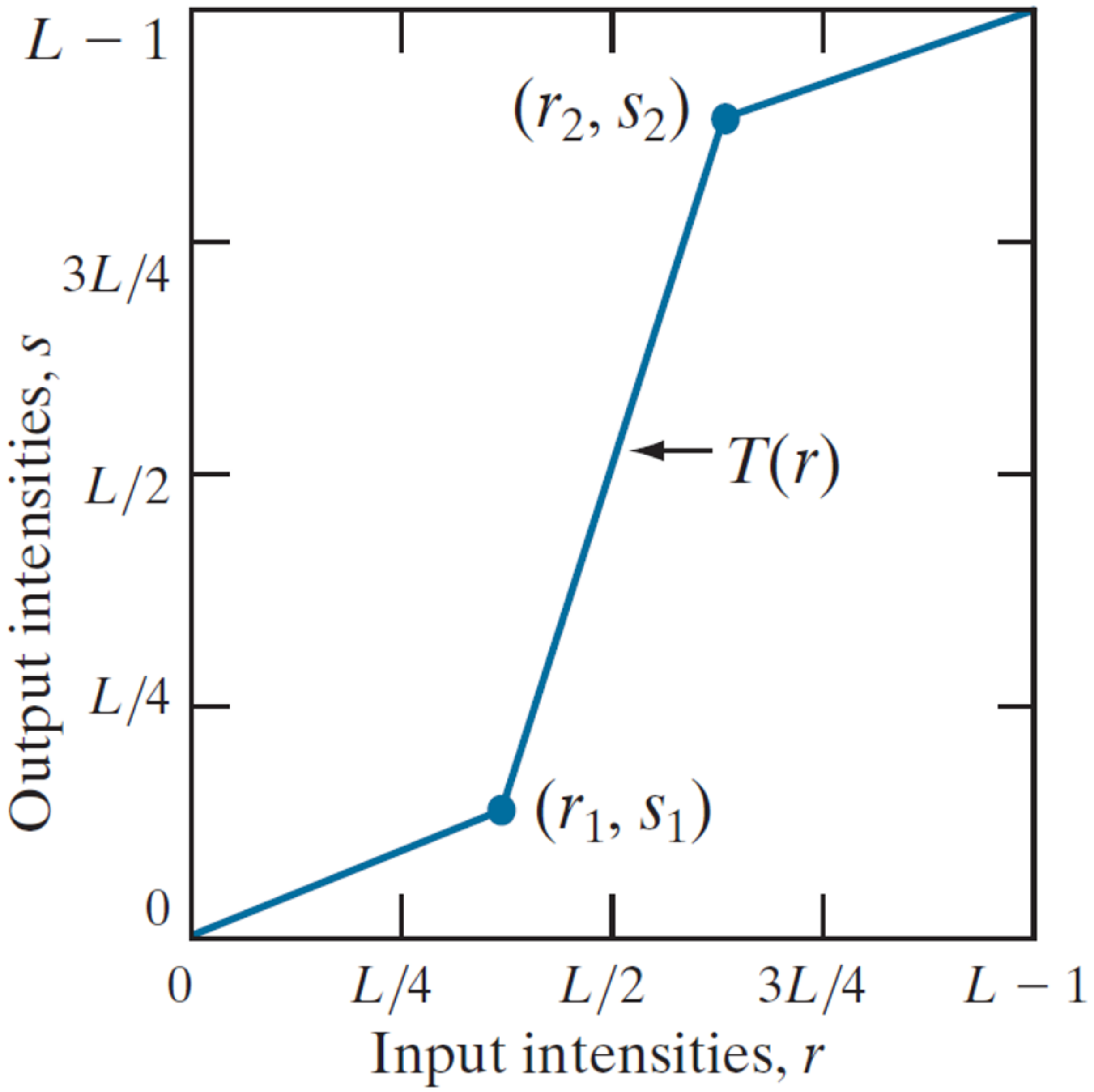
Reference:

[Histograms, Binnings, and Density]

[np.searchsorted 用法剖析]

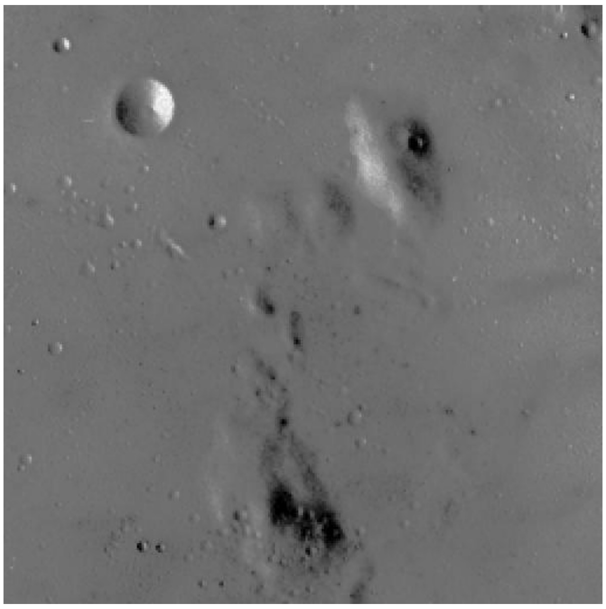
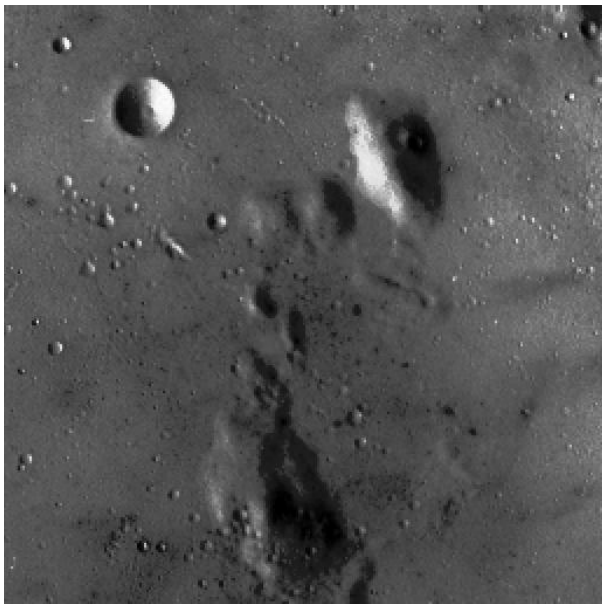
[Numpy histograms.py]

2 Implement a piecewise linear transformation function (below figure) for image contrast stretching. The code should read in an image; for intensity of all pixels, use the function to compute new intensity values; and finally output/ save the image with new intensity values.



```python
def linear(value):
    r1, s1 = (0.375*255, 0.125*255)
    r2, s2 = (0.625*255, 0.875*255)
    # print(value)
    if value <= r1:
        return float(s1)/r1*value
    elif value <= r2:
        return s1 + float(s2-s1)/(r2-r1)*(value-r1)
    else:
        return s2 + float(255.0-s2)/(255.0-r2)*(value-r2)


image = data.moon()
io.imsave("img.jpg", image)
dtype = image.dtype.type
img_rescale = dtype(list(
    [[linear(v) for v in item] for item in image]))
io.imsave("img_rescale.jpg", img_rescale)
```

原图片与变化后图片

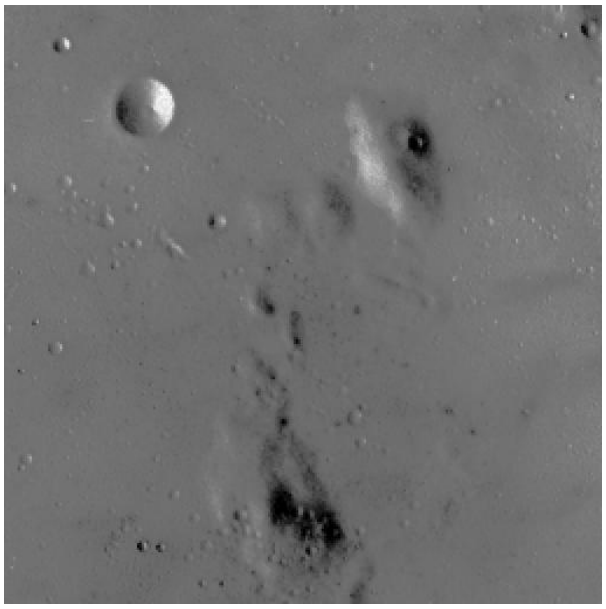
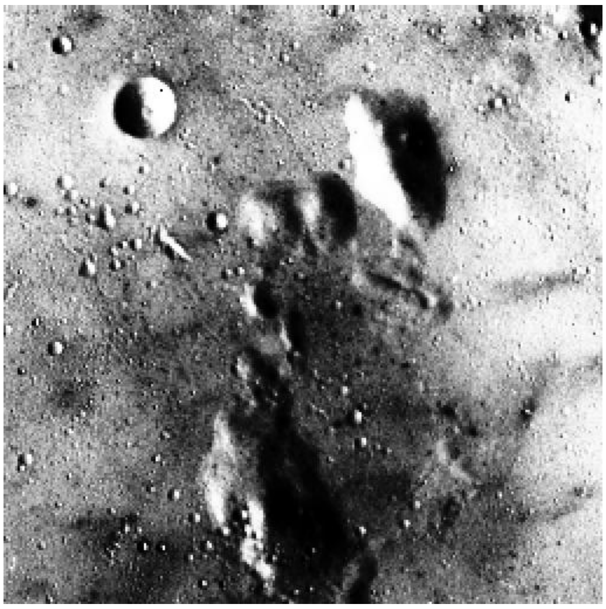
原图	变化后的图片
	

3 Implement the algorithm of local histogram equalization: (1) first implement histogram equalization algorithm, and then (2) implement the local histogram equalization using efficient computation of local histogram. Please test your code on images and show the results in your report.

```
def hist_equal(image, bins=256):
    size = image.shape[0]
    hist = bincount(image.reshape((-1)), minlength=bins)
    img_cdf = hist.cumsum()
    img_cdf = img_cdf / float(img_cdf[-1])
    out = np.interp(
        image.reshape(-1), range(256), img_cdf)
    out = np.array([int(x * 256) for x in out])
    out = out.reshape(size, size)
    new_img = copy.deepcopy(image)
    for i in range(size):
        for j in range(size):
            new_img[i][j] = out[i][j]
    return new_img

def bincount(my_list, minlength=0):
    length = max(minlength, max(my_list))
    count = Counter(my_list)
    return np.array([count[x] for x in range(length)])

def efficien_compute(old_hist, removed_pixels, extend_pixels, bins=256):
    hist_remove = bincount(removed_pixels.reshape(-1), minlength=bins)
    hist_extend = bincount(extend_pixels.reshape(-1), minlength=bins)

    new_list = copy.deepcopy(old_hist)
    new_list = new_list - hist_remove + hist_extend

    return new_list


def local_equal(image, bins=256, filter=64, uni_step = 5):
    origin_image = copy.deepcopy(image) # keep an original image unchanged
    steps = int((image.shape[0] - filter))
    steps2hist = {(i, j): np.array([0])
                  for i in range(steps) for j in range(0, steps, uni_step)}

    hist_init = bincount(
        origin_image[0:filter, 0:filter].reshape(-1), minlength=bins)
    steps2hist[(0, 0)] = hist_init
    local_process = tqdm(range(int(steps*steps/(uni_step*uni_step))))
    for i in range(0, steps, uni_step):
        for j in range(0, steps, uni_step):
            new_hist = np.array([0]) # initial as empty array
            if (i-uni_step, j) in steps2hist and steps2hist[(i-uni_step, j)].any():
                new_hist = efficien_compute(
                    steps2hist[(i-uni_step, j)], removed_pixels = origin_image[i-
uni_step:i, j:j+filter], extend_pixels = origin_image[i+filter-uni_step:i+filter,
j:j+filter])
            elif (i, j - uni_step) in steps2hist and steps2hist[(i, j -
uni_step)].any():
                new_hist = efficien_compute(
                    steps2hist[(i, j - uni_step)], origin_image[i:i + filter, j -
uni_step: j],
                    origin_image[i:i + filter, j + filter - uni_step:j + filter])
            if new_hist.any():
                steps2hist[(i, j)] = new_hist
                sum_count = sum(new_hist)
                mean_hist = np.mean(
                    [i*new_hist[i]/sum_count for i in range(len(new_hist))])
                if mean_hist > 0.4 and mean_hist < 0.5:
                    img_cdf = new_hist.cumsum()
                    img_cdf = img_cdf / float(img_cdf[-1])
                    out = np.interp(
                        origin_image[i:i + filter, j:j + filter].reshape(-1),
range(256), img_cdf)
                    out = np.array([float(x*256) for x in out])
                    out = out.reshape(filter, filter)
                    for m in range(filter):
                        for n in range(filter):
                            image[i+m][j+n]=out[m][n]

            local_process.update(1)

    return image
```

```
image=data.moon()
print("Processing the global histogram equalization")
uniformed_image = hist_equal(image)
io.imsave("img_global.jpg", uniformed_image)
plt.imshow(uniformed_image, plt.cm.gray)
plt.show()

print("Processing the local histogram equalization")
local_uniformed_image=local_equal(image, filter=64, uni_step= 5)
io.imsave("img_local.jpg", local_uniformed_image)
plt.imshow(local_uniformed_image, plt.cm.gray)
plt.show()
```

原图片与变化后图片（全局）

原图	变化后的图片
	

原图片与变化后图片（局部）

原图	变化后的图片