# HW2: Histogram and Spatial Filtering

# 1 Exercises

Please answer the following questions in the report.

## 1.1 Histogram Equalization (10 Points)

Ans：做两次直方图规范化和做一次直方图规范化的效果是一样的。由书上公式所得，第一次规范化结果为：

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} P_r(r_j)$$

$$= (L-1)\sum_{j=0}^{k} \frac{n_k}{n}, \quad k = 0, 1, 2, \cdots, L-1$$

第二次规范化即把 $n_k$ 替换成 $n_{sk}$，而在做直方图规范化的时候，等于 k 的像素值的个数和等于 sk 像素值的个数是一样的，所以在做第二次规范化的 $s^1_k = s_k$

## 1.2 Spatial Filtering (10 Points)

Ans：当我们选择的 filter 的宽度为 25px 的时候，滤波器的宽度正好等于竖条宽度与间隔宽度之和，无论滤波器怎么移动，其中包含的像素点都是一个竖条上的 25*5 个像素点和中间的 25*20 个像素点，求和平均之后所得像素值是相同的，所以出现间隔消失的情况。而滤波器宽度不为 25 时就不存在这种情况。

# 2 Programming Tasks

Write programs to finish the following three tasks, and answer questions

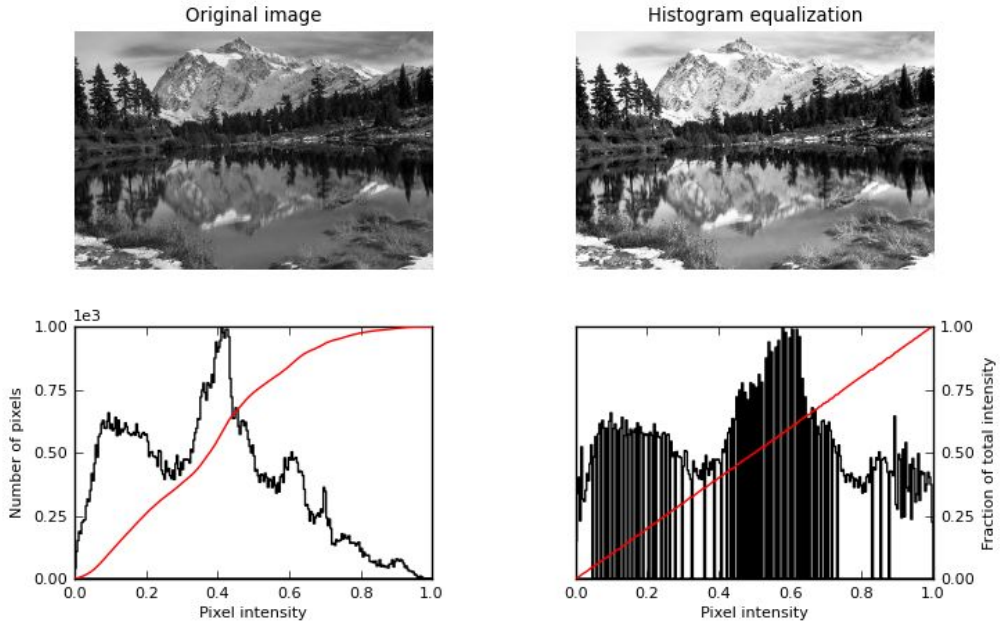
in your rep ort. Don’t forget to submit all relevant codes.

## 2.2 Histogram Equalization (35 Points)

### 2.2.1 结果图：

Original Image

Histogram Equalized




Plot Hist

### 2.2.2 结果分析

直方图均衡化所起到的作用是，把原本分布不均的灰度直方图在整个灰度级别内均匀分布。由直方图规范化的算法可知，新的像素值等于原图像素值的 cdf 乘以 255，在比较暗的区域，也就是低灰度值多的地方，在进行规范化之后，灰度值会变大，也就是会变白，对应原图和所得结果图，效果符合算法。

### 2.2.3 代码分析

```
def equalize(data, total, level=256):
    pdf = map(lambda x: (x[1], float(x[0])/total), data)
    cdf = [sum(map(lambda x: x[1], takewhile(lambda x: x[0] <= i, pdf))) for i in range(level)]
    pixels = [round((level - 1) * i) for i in cdf]
    return pixels
```

```
output_img = Image.new(input_img.mode, input_img.size)
for y in range(input_img.size[1]):
    for x in range(input_img.size[0]):
        output_img.putpixel((x, y), pixels[input_img.getpixel((x, y))])
return output_img
```

我的均衡化算法分为以下几个部分：

① 算出原图灰度值的 pdf，即概率密度值

② 算出原图灰度值的 cdf，即累计分布值

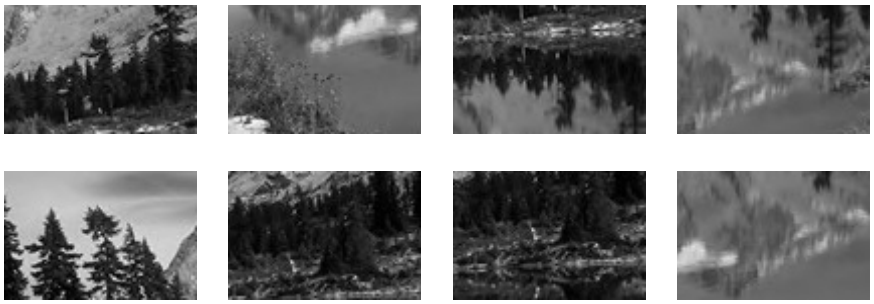
③ 将概率累积分布值乘以 255 得到新的灰度值数组

④ 将原图的灰度值作为新的灰度值数组的下标，获得每个位置的灰度值，组成新的图片

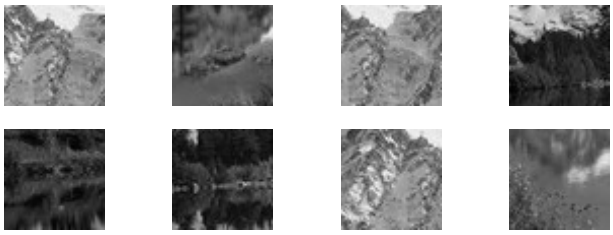
P.S. 我在写 plot_hist 的时候用到了 skimage 包，但是没有用包里面的 histogram_equalize 函数

## 2.3 Image Patch Extraction (10 Points)

### 2.3.1 结果图：



Patches for 96*64



Patches for 50*50

## 2.4 Spatial Filtering (35 Points)

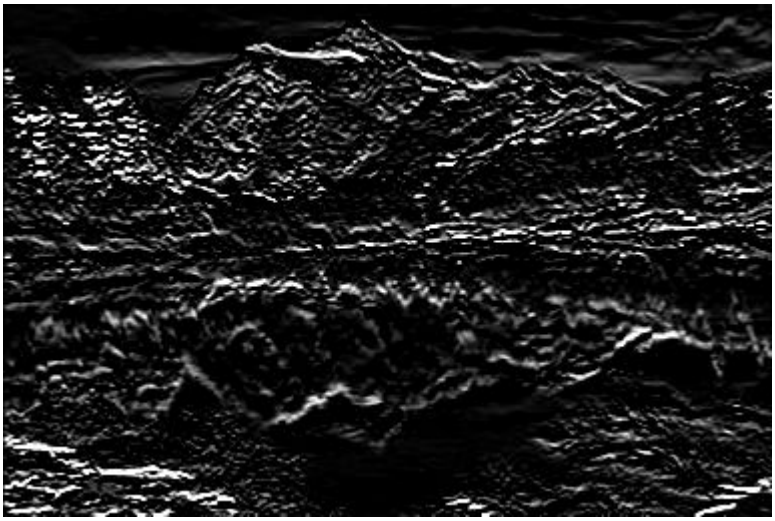
### 2.4.1 结果图：

Average filter 3*3

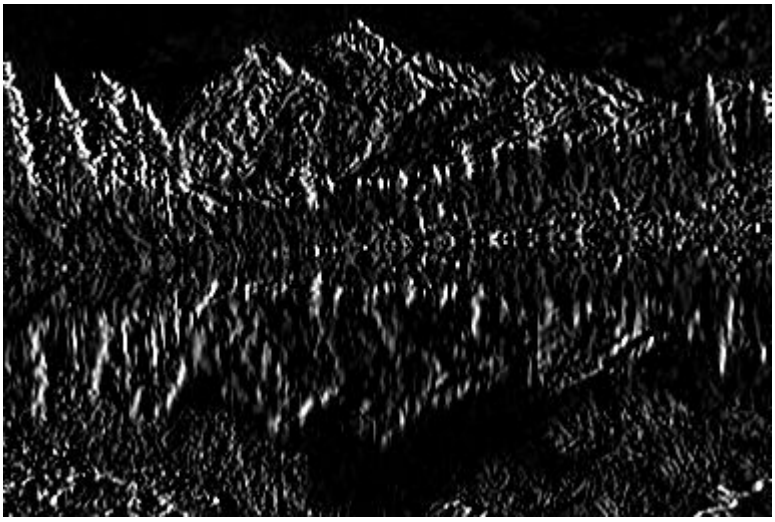
Average filter 7*7

Average filter 11*11

Laplacian filter



Sobel filter for Matrix1



Sobel filter for Matrix2

### 2.4.2 滤波应用：

均值滤波：

1. 使图像变得模糊或者清晰
2. 降低噪声，特别是对高斯噪声有良好的去噪能力

中值滤波：

1. 滤除脉冲噪声
2. 保护图像边缘

拉普拉斯滤波：

1. 图像锐化处理
2. 保留图像的背景色调，并突出细节

Sobel 滤波：

1. 可以做出浮雕效果

### 2.4.3 代码分析：

```
    if filter == 'average':
        weight = np.full(size*size, float(1)/(size*size))
    if filter == 'laplacian':
        laplacian = np.array([[-1, -1, -1],
                              [-1,  8, -1],
                              [-1, -1, -1]])

        weight = laplacian.flatten()
    if filter == 'sobel1':
        sobel1 = np.array([[-1, -2, -1],
                           [0,  0,  0],
                           [1,  2,  1]])
        weight = sobel1.flatten()
    if filter == 'sobel2':
        sobel2 = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
        weight = sobel2.flatten()
    for y in range(input_img.size[1]):
        for x in range(input_img.size[0]):
            z = np.full(size * size, pixels[y][x])
            for j in range(y - a, y + a + 1):
                for i in range(x - b, x + b + 1):
                    if i > 0 and i < imgheight and j > 0 and j < imgwidth:
                        z[(j - y + a) * size + i - x + b] = pixels[j][i]
            newimg.putpixel((x, y), np.dot(weight, z))
    return newimg
```

我的滤波算法分为以下几个部分：

① 根据 filter 判断滤波种类并算出权值

② 对每一个和权值相乘的矩阵，初始化为边界值，即 pixels[y][x]

③ 构建四个循环，外两层是图片像素位置的循环，即 y:0-256，x:0-384；内两层循环式为了构建与权值相乘的 patch 矩阵，当 i:0-imgheight, j:0-imgwidth 时，patch 矩阵的第 i 行 j 列的元素就等于 pixels[j][i]

④ 在内两层循环结束时，往新图像中写入像素点，其灰度值为 weight 矩阵与 z 矩阵(patch)的乘积

⑤ 在外两层循环结束时，返回新的图片 newimg