HW2: 直方图和空间滤波

数字媒体 15331229 罗剑杰

1 习题

请完成下列问题，并将你的答案写到报告中。

1.1 直方图均衡化 （15 分）

假设你对一张图已经进行了一次直方图均衡化的操作。如果对这张图进行第二次直方图均衡化，得到的结果跟第一次均衡化的结果一样吗？请给予证明。

解：
两者一样，下面开始证明：

1. 以连续函数的情况来看，对于第一次直方图均衡化，从$r \rightarrow s$, 有以下变换：

$$s = T(r) = (L-1) * \int_0^r p_r(w)dw$$

则转换后的s的概率密度函数$p_s(s)$为：

$$p_s(s) = p_r(r) * | \frac{dr}{ds} | = p_r(r) * | \frac{1}{(L-1) * p_r(r)} | = \frac{1}{L-1}$$

对于第二次直方图均衡化，从$s \rightarrow t$, 同理，有：

$$t = T(s) = (L-1) * \int_0^s p_s(w)dw$$

$$p_t(t) = p_s(s) * | \frac{ds}{dt} | = p_s(s) * | \frac{1}{(L-1) * p_s(s)} | = \frac{1}{L-1}$$

所以$p_t(t) = p_s(s)$, 所以第二次直方图均衡化的结果和第一次的一样。

2. 以离散的情况来看，似乎不太好在数学公式上进行证明，直接从变换公式上看，变换$T(r_k)$和灰度值的大小没有关系，而是和灰度值出现的概率有关系，并且变换$T(r_k)$保证了转换后的灰度的相对大小不变，对应的非零概率的大小也不变，所以第二次转换出来的图像和第一次的一样。

1.2 空间滤波（20 分）

给定一张 4 x 4 的灰度图和一个 3 x 3 的滤波器：

图像：$\begin{bmatrix} 85 & 13 & 20 & 80 \\ 169 & 8 & 243 & 20 \\ 18 & 155 & 163 & 44 \\ 12 & 34 & 50 & 80 \end{bmatrix}$ 滤波器：$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

1. （7 分）用给定的滤波器对这张灰度图（边界补零）进行卷积，写出卷积后的结果（大小应为 4x4）。

$$\begin{bmatrix} 177 & 420 & 271 & 263 \\ 75 & 218 & 249 & 107 \\ -131 & -324 & -107 & -133 \\ -173 & -336 & -362 & -267 \end{bmatrix}$$

2. （8 分）请说出你得到的卷积结果中正数和负数分别表示什么含义。

    - 负数的话应该表示新图上该点的灰度值很小（会显示出黑色），该点应该不是边界检测出来的边界上的点。
    - 正数的话表示新图上该点的灰度值大于0，会显示出来，该点应该是边界检测出来的边界上的点。

3. （5 分）根据你所学到的知识，谈一谈题目中给出的 3 x 3 滤波器可以有哪些应用。
   这个滤波器其实是一个 `Prewitt operator`，它可以用来检测图像横向的边界，当用这个滤波器扫过一遍图像之后，水平方向上的边界就会显示出来。不是边界的那些地方就会呈现出黑色。

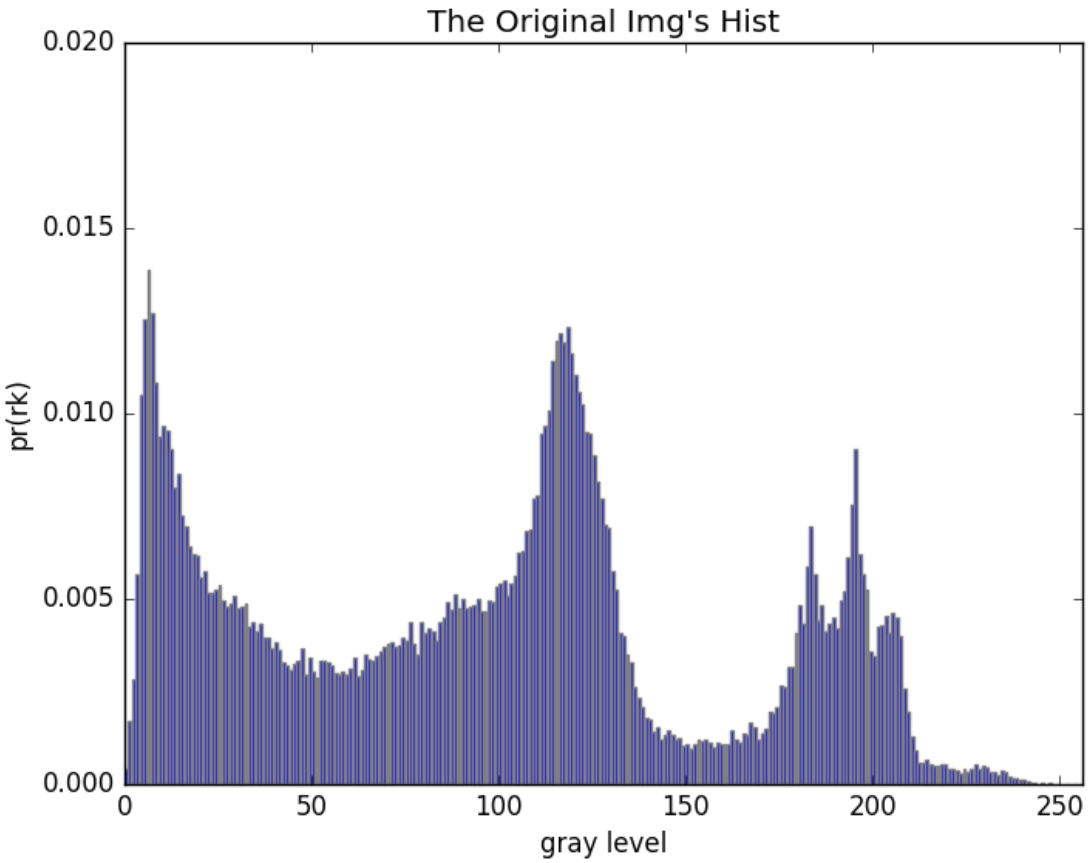
2 编程题

2.2 直方图均衡化(35 分)

实现一个对灰度图做直方图均衡化的函数(不允许直接调用现成的直方图均衡化接口,例如 `Matlab` 的 `histeq` 接口)。函数的格式为 `equalize_hist(input_img) => output_img` 该函数返回一张灰度级分布均匀的灰度图。

如有必要,你可以修改该函数的格式。请载入对应你学号的输入图像,并用你实现的程序来完成以下任务:

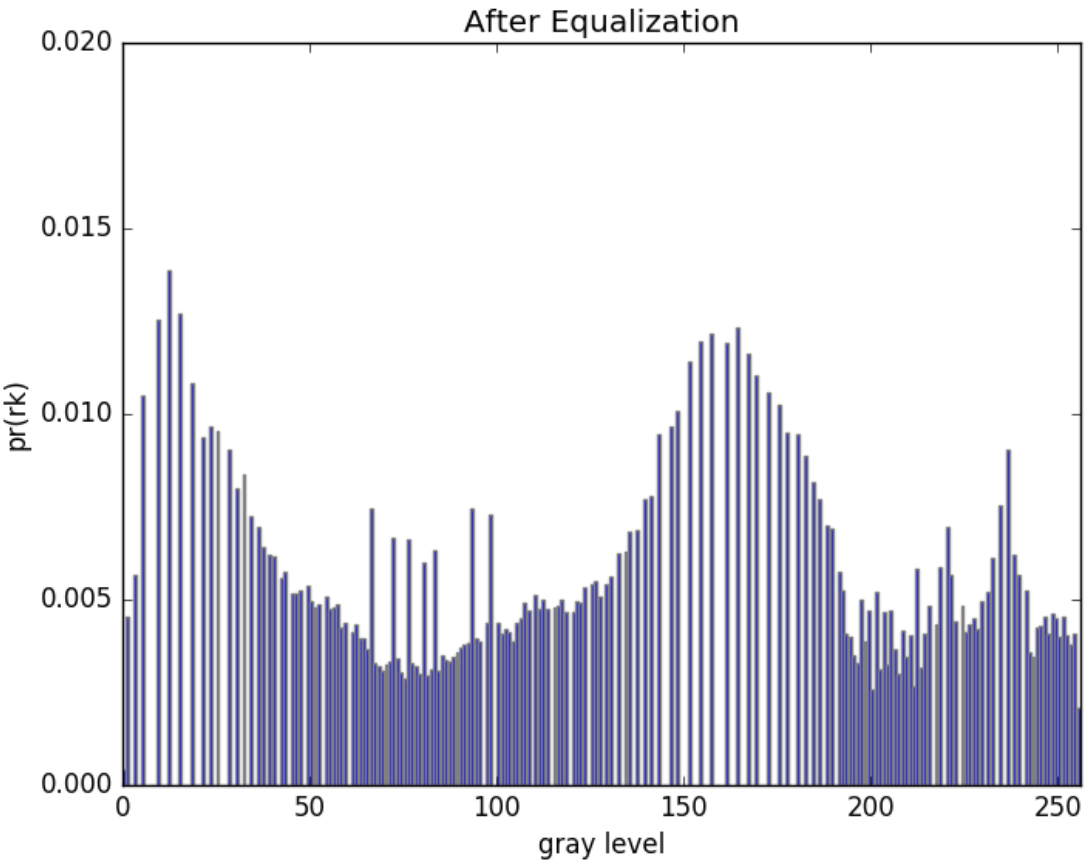
1. **(5** 分)计算并显示图像的直方图,并把结果粘贴到报告里。注意:你必须用你自己实现的函数来计算直方图,但是允许调用现成的 **API** 来显示直方图。(例如,你不能调用 **Matlab** 的“**imhist**”来计算直方图,但是可以调用“**subplot**”, “**hist**”来显示直方图。)



原图直方图

2. **(10** 分)进行直方图均衡化,将均衡化后的结果和相应的直方图粘贴到报告里。

均衡化后的图片



均衡化后的图片对应的直方图

3. **(8** 分**)**分析直方图均衡化后的结果,字数不能超过一页。

    1. 在图片呈现上：
        1. 视觉上的变化还是比较清楚的，整体变得比原图更亮了，特别是远处的山脊的颜色亮度的变化明显。

    2. 在直方图的对比上：

1. 对比两张直方图，可以发现，进行了直方图均衡化之后，直方图依然不算是相当平均，但是可以在宏观上看出灰度的分布相比与原图有了比较明显的均衡的趋势。
2. 主要的变化体现在灰度值为[140，190]， [200，256]的这两个区间里面，它们的灰度频率有了显著的提高。然后在[0，50]这个区间里，灰度值的取值密度稀疏了很多。

4. **(12 分)**详细描述你是如何实现直方图均衡化操作的,也就是说,针对**“equalize_hist”**函数进行算法说明,字数不能超过两页。请集中在算法描述方面,不要过多地复制/粘贴代码到报告上。
依据书本的思路来实现就好，思路比较清晰易懂：

   1. 统计出该图的灰度直方图：这里使用一个一维的列表 `pr_rk` 来存储， `pr_rk` 的长度是256，下标是[0:255]，下标代表对应的灰度值r_i，**`pr_rk`** 的里面的元素的值就是对应下标灰度在图片中出现的概率。这个过程通过遍历一次原图上的每一个像素进行一次统计即可。
   2. 构建变换T, 这里使用一维列表 `T` 来存储这个变换，因为这只是一个一对一或者多对一的映射而已。遍历 `range(256)` （从0到255），使用直方图均衡的公式$s_k = T(r_k) = (L-1) * \sum_{j=0}^{k} p_r(r_j)$对于每一个灰度值r，计算出对应的s值，然后将s值赋值给 `T[r]`，那么 `T` 这个列表在遍历结束后就形成了一个映射表，通过下标（表示原图上的灰度）映射到下标所指的元素（表示对应在新图上的灰度）。
   3. 新建一幅空的图片，遍历原图，对于原图上的每一点的像素，经过 `T` 的映射后将新的灰度值赋值给新图上对应的点，完成直方图均衡。

2.3 空间滤波(30 分)

实现一个对灰度图进行空间滤波的函数。函数的格式是“filter2d(input_img, filter) → output_img”, 这里的“filter”是给定的滤波器。如有必要,可以修改函数的格式。

请载入对应你学号的输入图像,并用你实现的 `filter2d` 函数来完成以下任务:

1. (9 分)分别用 3 x 3,7 x 7 和 11 x 11 的均值滤波器来平滑你输入的图像,将相应的三个输出结果粘贴到报告里。

mean_filter3x3

mean_filter7x7

mean_filter11x11

2. (6 分)用 3 x 3 的拉普拉斯滤波器来锐化你输入的图像(课本上有 4 种拉普拉斯滤波器,参见图3.37,你可以使用其中任意一种),并将输出结果放在报告中。除此之外,请简单介绍一下为什么拉普拉斯滤波器可以用于图像的锐化。

因为通过画图的探讨我们可以知道，数图中的边缘在灰度上常常类似于灰度上的斜坡过渡，然后在画图尝试的基础上我们知道一般情况下，二阶微分在增强细节方面要比一阶微分要好得多，因为二阶微分会在边界处产生一个更窄的正负的变化。通过二阶微分的离散公式我们来构造一个基于该公式的滤波器模板。可以证明，最简单的各项同行微分算子是拉普拉斯算子，所以将这个算子应用在图像上，可以产生锐化的效果。

3. (5 分)将高提升滤波(high-boost filter)用在输入的图像中(也就是说,g(x, y) = f(x, y) + k ∗ g max (x, y),其他细节参见课本式(3.6-9))。过程中涉及的平滑部分应用用课本图 3.32(a)所示的滤波器来完成。请自行选择合适的 k(式(3.6-9)中的权值)。在报告中,你需要说明你选择的 k 的值,并贴上对应的输出结果。
   选择的k值为3，实验输出结果如下：

4. (10 分)详细描述你是如何实现空间滤波操作的,也就是说,针对“filter2d”函数进行算法说明,字数不能超过两页。

这个算法的整体思路不难，主要的难点是：

1. 如何在使用的编程语言上正确地进行矩阵的增减操作。
2. 找好原图与扩展后的矩阵，扩展后的矩阵和滤波器之间的坐标对应关系。

下面是算法的实现思路：

1. 将所有 `PIL.Image` 的图片转换成 `numpy.array` ，方便后面的进一步计算。
2. 计算出辅助变量m, n, a, b, height, width，方便后面的进一步计算。其中滤波器的大小是$m * n$，$m = 2 * a + 1$, $n = 2 * b + 1$, height 和 width 是原图的高和宽。
3. 构建出扩展矩阵 `matrix` ，准备和滤波器进行运算。因为 `np.lib.pad` 方法被禁用，所以只好新建一个大小为(height+2*m-2, width+2*n-2)的零矩阵，然后在中心填充上原图的每个像素的灰度值。其中遍历原图上的每个像素点，和扩展矩阵之间有着这样的对应关系 `matrix[x+m-1, y+n-1] = img[x][y]`
4. 新建一幅空白图片，在 `matrix` 和滤波器进行卷积运算的时候算出的每个点的像素，同时把新的像素值赋值到空白图片对应的点上。卷积运算的过程如下：
    - 因为最后运算完后也是取中心的那一部分，所以直接以原图大小为遍历的范围
    - 通过 `numpy.array` 的indexing下标取值的快捷运算，我们可以很快地得到在这一轮迭代中，对于中心点(x, y)，我们在 `matrix` 中所取的子矩阵是 `submatrix = matrix[x+m-1-a:x+m-1+a+1, y+n-1-b:y+n-1+b+1]` 。
    - 将子矩阵和滤波器进行相关运算，对应位相乘再把所有位相加，即得出新图中(x, y)这个点的像素 `newvalue = round((submatrix * filter_).sum())`
    - 将 `newvalue` 取整后赋值给空白图片的对应的(x, y)的点，然后重复循环至空白图片的所有像素遍历完毕。
5. 得到滤波过后的新图片。

高提升滤波的实现另外实现了一个函数，主要思路是先用“filter2d”函数模糊处理一遍，然后遍历原图所有点，在对应点上计算新图的灰度
$g(x, y) = f(x, y) + k * (f(x, y) - \bar{f}(x, y))$即可。