HW2: Histogram and Spatial Filtering

11331049 戴嘉华

1 Exercises

1.1 Histogram Equalization (10 Points)

不相同，举一个反例即可证伪该命题。举书本上的例子：

对于一个 3 位的尺寸为 64 x 64 的图像，像素的信息如下表所示：

r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

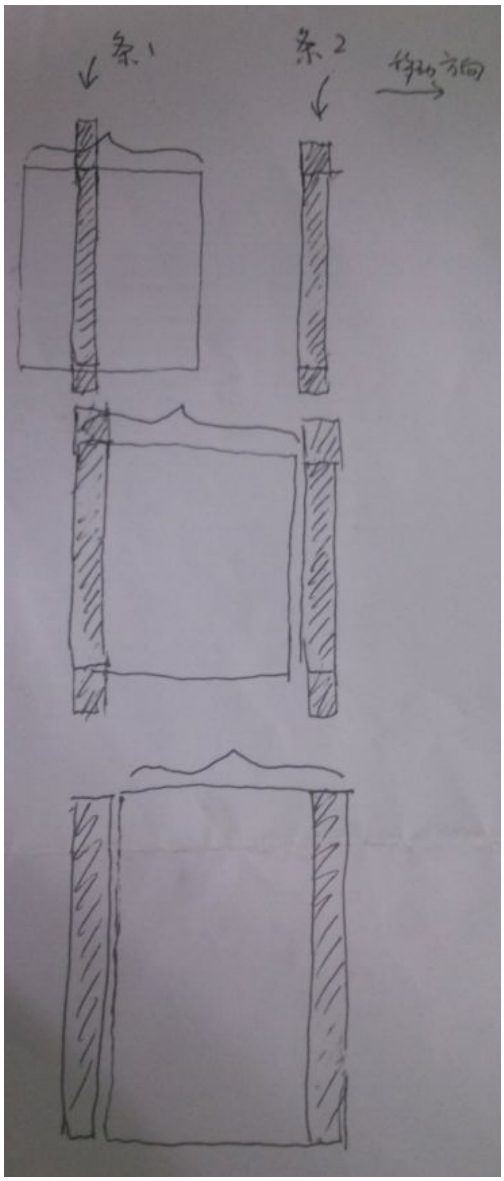
第一次对该图像进行直方图均衡化，得到 s0=1, s1=3, s2=5, s3=6, s4=6, s5=7, s6=7, s7=7.那么可以重新获得一个新的图像，像素的灰度分布为：

r_k	n_k	$pr(r_k)$	$pr(s_k)$
$r_0 = 0$	0	0	0
$r_1 = 1$	790	0.19	0.19
$r_2 = 2$	0	0	0.19
$r_3 = 3$	1023	0.25	
$r_4 = 4$	0	0	
$r_5 = 5$	850	0.21	
$r_6 = 6$	985	0.24	
$r_7 = 7$	448	0.11	

如上图，算到在计算第二遍直方图均衡化处理的时候，计算出来的 $pr(s_2)$!= $pr(r_2)$，可以证伪该命题。

1.2 Spatial Filtering (10 Points)

出现这种情况是因为，对于 25x25 的模板来说，方形均值模板扫过竖条的像素的时候。对于大部分该区域的像素点来说（边缘的忽略），不管是扫到竖条中黑色的像素点的时候还是扫到竖条之间空白的像素点的时候，模板中的黑色像素点的数目是一直的。这是由于模板的尺寸为 25 x 25，而竖条的长度为 5，两条竖条之间的宽度为 20。那么模板离开第一个竖条多少个像素，就会进入第二个竖条多少个像素，所以模板中黑色像素的数目是一样的。那么就会得到相同的均值，就会连起来了。



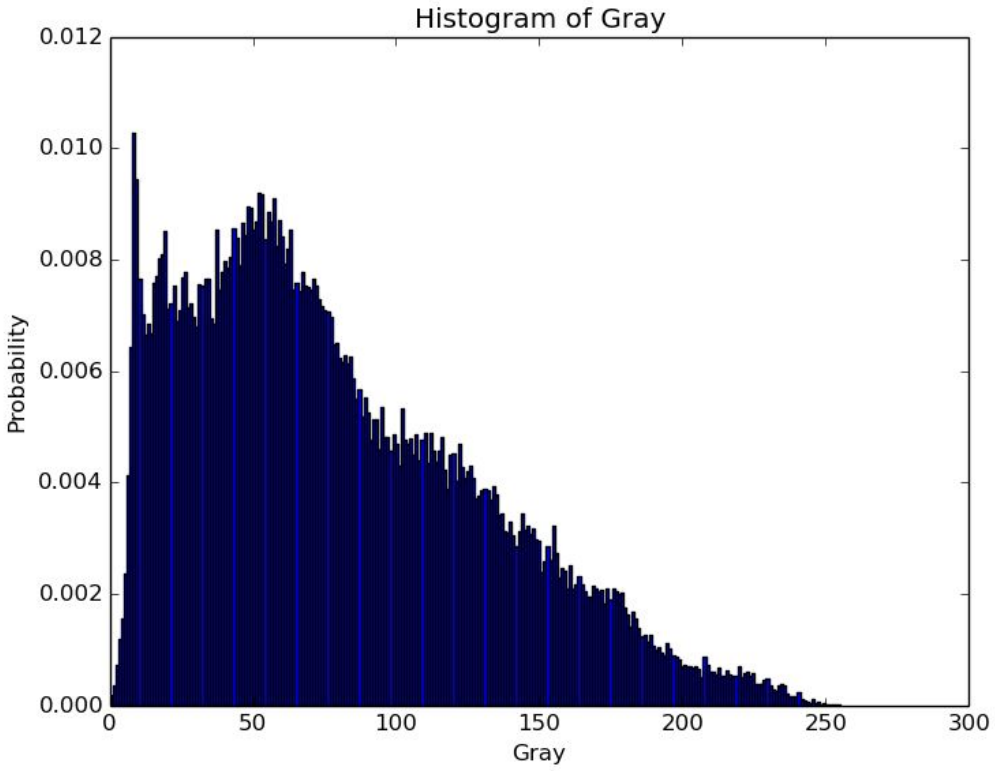
对于 23x23 或者 45x45 的模板来说，就没有模板内总是出现相同数目黑色像素点的情况，所

以一定会出现不均匀的情况，不均匀就会出现 gap 了。

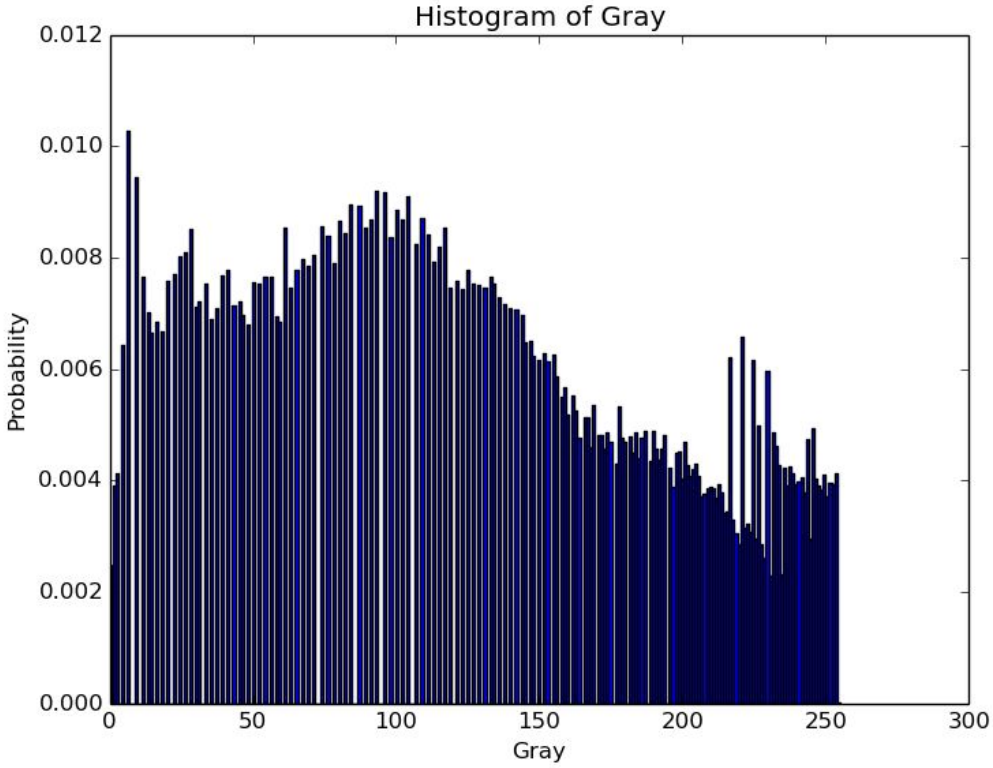
2 Programming Tasks

2.2 Histogram Equalization (35 Points)

1.



2.



3. 在图像进行直方图均衡化之前，整个图像的颜色是偏黑灰色的。在我们的处理之前的概率密度直方图中可以看到，基本上颜色都集中在左边，左边是 0，也就是黑色，所以和图片直观上看到的灰度布局是一致的。在进行了直方图均衡化以后，处理过后的图片整体对比处理前，亮度明显提高。观察处理后的直方图，对比以前颜色分布较为均匀。颜色不会严重偏向于 0 处，处理后的图片对比来说提亮了。

4. 我的算法思路比较直接：遍历图片上所有的像素，计算每个灰度级别的像素出现的频率。

```
def equalize_hist(img):
    width, height = img.size
    total = width * height
    grayData, pixels = getGrayFrequencies(img)
```

然后根据直方图均衡化的公式：

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

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

构建起每个灰度级别对应处理后的灰度级别的映射。接下来再次遍历图片上的元素，把每个像素的灰度级别在映射上找到相应的转换，替代该元素。然后保存该图片即可。详细代码可见附件中的 programming tasks/2.2/equalize_hist.py

2.3 Image Patch Extraction (10 Points)

1. 96x64

2. 50x50

2.4 Spatial Filtering (35 Points)

1. averaging filter

3x3

7x 7

11x11

2. Laplacian filter，使用的是书本上第二个算子：

3. Sobel filter

(d)

(e)

4. 均值滤波器可以用于一些动画、视频效果中的前后景切换效果，例如在做游戏的时候会

用到。还有一些光学显微镜下面观察细胞的时候，可以用 laplacian filter 来突出边缘，观察到细节等。

5. 我把不同的滤波的实现分到了不同的模块，然后采用一个模块组合起所有滤波器模块，通过一个 API（filter2d）暴露功能。对于每一个滤波器的实现，都是采用直接遍历所有的元素，然后根据模板大小和不同的滤波算法对每个像素的值进行计算。写入新的像素值，然后保存。

例如，对于 laplacian filter，会对遍历每个元素周边的像素，统计出和。根据书本第二个算子的算法，周边之和减去该像素乘以 8 得出结果，写入该像素，返回新图像。

```
def makeLaplacian(x, y):
    total = 0
    # 遍历某个元素附近的像素点，统计所有边缘像素的和
    for innerX in xrange(x - h_pixs, x + h_pixs + 1):
        for innerY in xrange(y - v_pixs, y + v_pixs + 1):
            # 本元素不参与计算
            if innerX == x and innerY == y: continue
            try:
                pix = img.getpixel((innerX, innerY))
            except:
                pix = 0
            total += pix
    # 使用书本上第二个Laplacian算子对像素值进行计算和写入
    newImg.putpixel((x, y), total - 8 * img.getpixel((x, y)))

for x in xrange(width):
    for y in xrange(height):
        makeLaplacian(x, y)

return newImg
```