数字图像处理与模式识别第一次作业

18231213 汪昊霖

在本次作业中，我先使用c++实现了对bmp格式的打开操作。但是由于图片大多数格式并非bmp格式，而是采用了压缩算法，因此为了能够适应其它格式，我使用了java语言对数字图像进行了基础操作。包括：打开数字图片、实现直方图均衡、利用算法进行灰度拉伸这三个方面。下面对各个部分进行具体技术分析。

一、打开图片

首先介绍用c++读取bmp格式的方法（参考wikipedia）。典型的bmp格式图像包括以下三个方面：

- **位图头**：保存位图文件的总体信息。
- **位图信息**：保存位图图像的详细信息。
- **调色板**：保存所用颜色的定义。
- **位图数据**：保存一个又一个像素的实际图像

  这些存储的数据比较复杂，但是事实上对于读取图像而言，最为有用的就是：

  - 10 - 13字节：存储位图数据位置的地址偏移
  - 18 - 21字节：存储位图的宽度
  - 22 - 25字节：存储位图的高度

  在我的实现中，由于未考虑压缩算法，因此总体的实现就较为简单大体思路就是，找到存储位图数据的偏移量然后开始逐字节读取，根据提供的宽度和高度信息，将其存储到二维数组中即可。

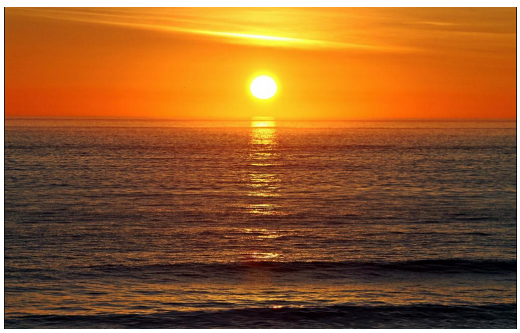
但是因为常见的图像（如jpg格式）不是简单的位图存储，而是压缩编码后的结果，所以直接用文件流的形式读取非常复杂，按照上面的方法往往会导致图片异常。而在Java中已经内置了对图像的输入输出流，这点非常的方便。因此在后面部分中，我直接使用了Java提供的API——ImageIO进行图像的读取。这样可以很方便地打开所有图片，也更利于后续操作的检查。

在Java的API中，提供了getRGB方法。在这个方法中，只要提供像素坐标，就可以返回一个32位整型值表示该处像素的RGB值。这个整型值一共由四个字节组成，分别代表此处像素的alpha，r，g，b值。其中rgb值是容易理解的，而alpha值则表示透明度。在一般的图像中，像素大多是不透明的，因此取值一般是255.

由于后续的处理都用灰度图，因此在此步骤中我直接将其转化为8位灰度图。通过查阅相关资料，有下面转化灰度的经验公式：

$$L = 0.299R + 0.587G + 0.114B$$

之后将得到的所有灰度信息，存储到二维数组中（尺寸和图片大小相同）。这样就完成了图片的打开工作。打开并转化为灰度图的结果测试如下：



打开原图

转化为灰度图

二、直方图均衡

要实现直方图均衡算法，首先要统计各个灰度出现的频率。对于第k级灰度，其出现频率为：

$$P_k = \frac{n_k}{N}$$

然后要进行均匀化处理，通过累加的方法计算得到每个灰度级的s值：

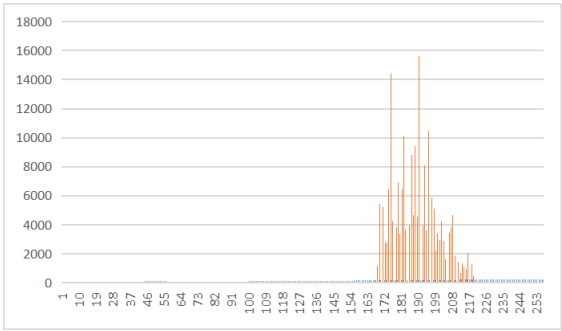
$$S_k = \sum_{i=0}^{k} P_k$$

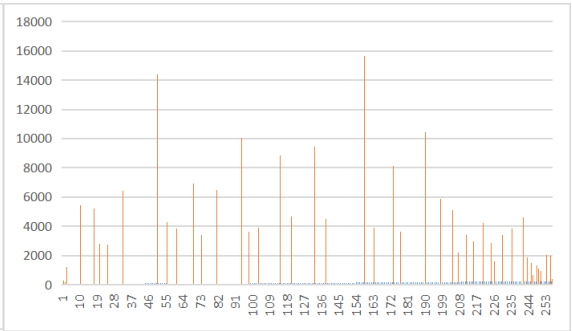
要注意的是为了加速这一阶段的计算过程，应该设置一个累加值，这样就不必每次求和带来大量的重复计算。然后通过S，可以计算出变换后的灰度级，变换如下：(其中round是四舍五入取整函数)

$$k \to round(255 * S_k)$$

之后将每一像素的灰度级都做如上变换，就得到了直方图均衡后的结果。值得一提的是，我在实现了之后，尝试对一幅图多次使用直方图均衡算法，发现从第二次开始就不会有任何变化。这点我随即自己证明了一下，发现是正确的结果。可以做一个这样的理解：直方图均衡算法使得熵达到了最大，那么对此重复做均衡算法自然就是无用的了。我认为或许可以通过这个性质，对算法实现的正确性做一个检查。

根据查阅网上资料，直方图均衡的作用主要在于增强对比度。因此我找了对比度较低的图像进行测试，实验结果如下（包括图片以及对应的颜色直方图）：





原图 **应用直方图均衡算法处理后**

三、灰度拉伸

所谓灰度拉伸，就是对各个像素的灰度级做一个函数变换。具体而言，我实现了一下几种：

线性变换：

$$f(x) = \begin{cases} low & ,x < x_1 \\ \frac{(y_1 - y_2)}{(x_1 - x_2)}(x - x_1) + y_1 & ,x_1 \le x \le x_2 \\ high & ,x > x_2 \end{cases}$$

分段线性变换：

$$f(x)=\begin{cases}\frac{y_1}{x1}(x-x_1)+y_1 & ,x<x_1\\ \frac{y_1-y_2}{x_1-x_2}(x-x_1)+y_1 & ,x_1\le x\le x_2\\ \frac{y_2-255}{x2-255}(x-x_2)+y_2 & ,x>x_2\end{cases}$$

二值化变换：

$$f(x)=\begin{cases}0 & ,x<threshold\\ 255 & ,otherwise\end{cases}$$

在实际代码中，我还实现了对数、指数变换，由于方法相似这里就不赘述了。实验结果如下：

原图　　利用线性变换提亮　　利用分段线性变换变暗　　二值化

附录

具体实现代码请参看https://github.com/kxzxvbk/ImageProcess