数字图像处理 hw1

15331416 赵寒旭

Exercises

1.1 Storage (3 * 3 = 9 Points)

If we consider an N-bit gray image as being composed of N 1-bit planes, with plane 1 containing the lowest-order bit of all pixels in the image and plane N all the highest-order bits, then given a 1024 × 2048, 128-level gray-scale image:

1. How many bit planes are there for this image?

   Ans: $2^7 = 128$

   7 bit planes

2. Which panel is the most visually significant one?

   Ans: plane 7 contains all the highest-order bits, which is the most visually significant one.

3. How many bytes are required for storing this image? (Don't consider image headers and compression.)

   Ans: 1024*2048 个像素，每个像素 7bits

   1B = 8bits, 1Kb = 1024B, 1MB = 1024KB

$$\frac{1024\times 2048\times 7bits}{8bits/byte} = 0.875\times 2^{21} bytes = 1.75MB$$

1.2 Adjacency (3 * 3 = 9 Points)

Figure 1 is a 5 × 5 image. Let V = {1, 2, 3} be the set of pixels used to define adjacency. Please report the lengths of the shortest 4-, 8-, and m-path between p and q. If a particular path does not exist, explain why.

3	4	1	2	0
0	1	0	4	2(q)
2	2	3	1	4
(p)2	0	4	2	1
1	2	0	3	4

Figure 1: *Adjacency.*

Ans: 4-path 不存在，p, q 之间不存在这样一条路径使得路径上所有点互为 4-neighbors

8-path 最短为 4

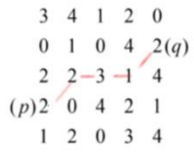


Figure 1: *Adjacency.*

m-path 最短为 5

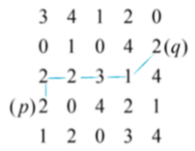


Figure 1: *Adjacency.*

1.3 Logical Operations (3 * 3 = 9 Points)

Figure 2 are three different results of applying logical operations on sets A,B and C. For each of the result, please write down one logical expression for generating the shaded area. That is, you need to write down three expressions in total.

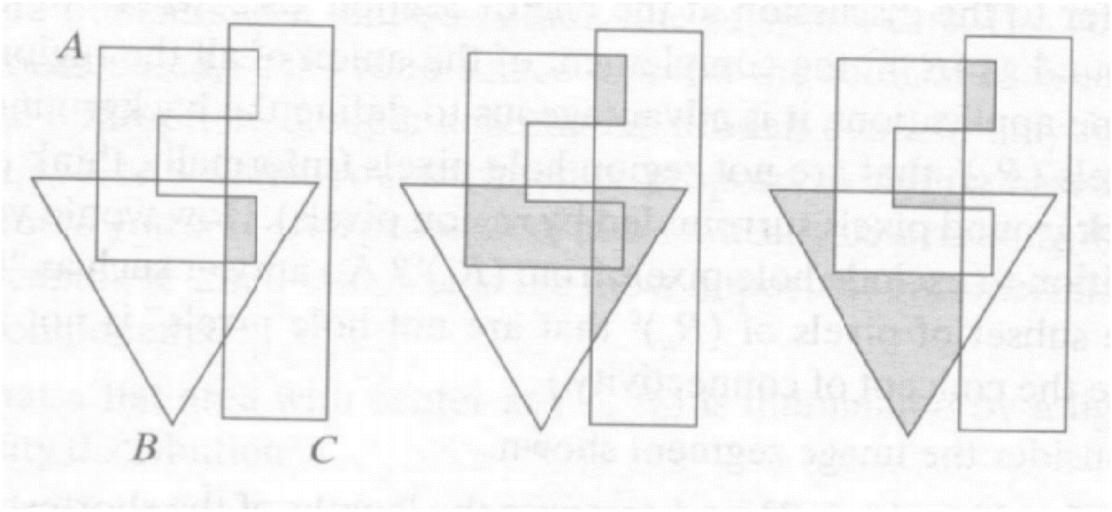


Figure 2: *Logical Operations.*

Ans: 图 1：$A \cap B \cap C$

图 2：$(A \cap B) \cup (A \cap C) \cup (B \cap C)$

图 3：$(A \cap C) \cup (U - (A \cup C)) \cap B$

Programming Tasks

2.2 Scaling (45 Points)

1. Down-scale to 192 × 128 (width: 192, height: 128), 96 × 64, 48 × 32, 24 × 16 and 12 × 8, then manually paste your results on the report. (10 Points)

   192 × 128

   96 × 64

   48 × 32

   24 × 16

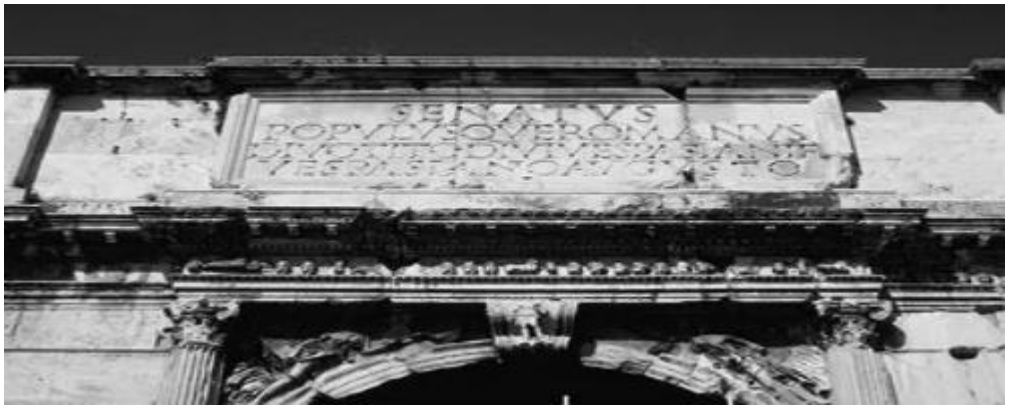
   12 × 8

2. Down-scale to 300 × 200, then paste your result. (5 Points)

3. Up-scale to 450 × 300, then paste your result. (5 Points)

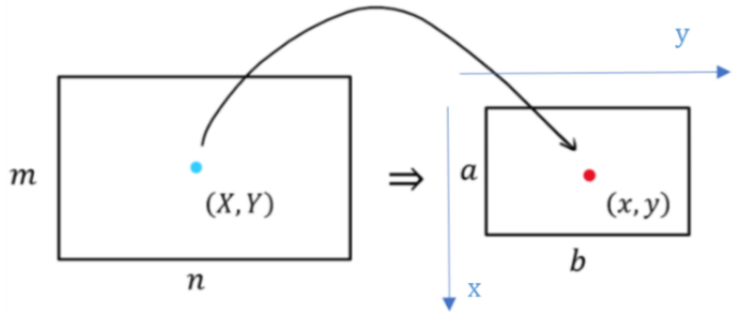
4. Scale to 500 × 200, then paste your result. (5 Points)



5. Detailedly discuss how you implement the scaling operation, i.e., the “scale” function, in less than 2 pages. Please focus on the algorithm part. If you have found interesting phenomenons in your scaling results, analyses and discussions on them are strongly welcomed and may bring you bonuses. But please don’t widely copy/paste your codes in the report, since your codes are also submitted. (20 Points)

Attention: you should not rescale your scaling results in your report, unless they exceed the page height/width.

算法理论：



已知原图像大小为 $m\times n$，缩放后图像大小为 $a\times b$,要通过双线性插值法求出放缩后图像 x 行 y 列像素点 (x,y) 的像素值。

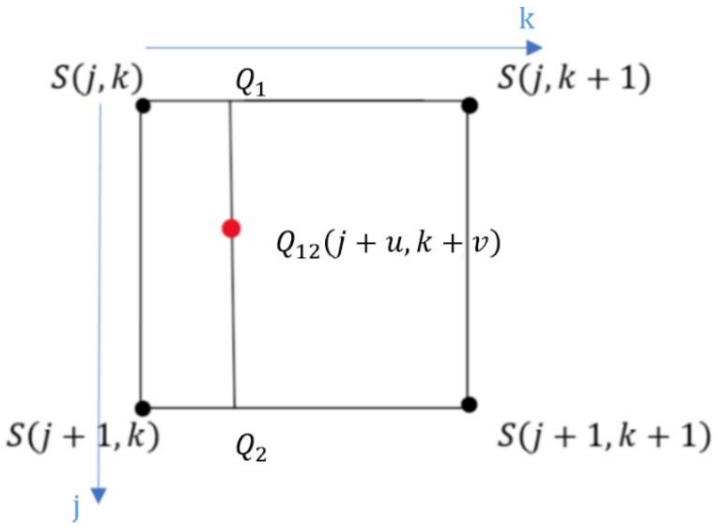
1) 找到原图中对应点 (X,Y)

$$(X,Y)=\left(x\frac{m}{a},y\frac{n}{b}\right)=(j+u,k+v)$$

j, k 分别为 X，Y 的整数部分，u, v 分别为 X，Y 的小数部分。

2) 在原图中找到待插值的四个点

$$S(j,k),S(j,k+1),S(j+1,k),S(j+1,k+1)$$



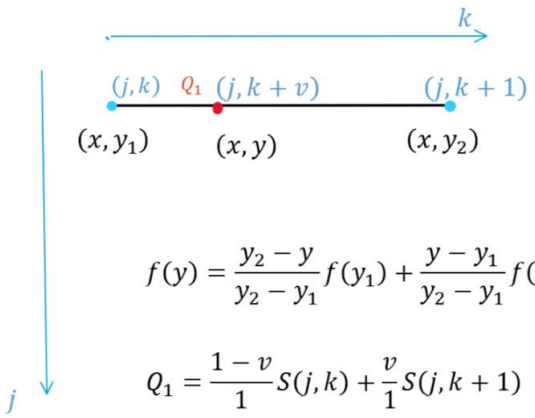
3) y 方向插值分别得到 Q_1，Q_2 的像素值

Q_1 处像素值由 $S(j,k)$ 和 $S(j,k+1)$ 处像素值插值得到（权值大小由 u 决定）。

容易知道,u 值越小则 Q_1 位置越接近 $S(j,k)$，$S(j,k)$ 的像素值在插值中占比例越大，

$u,v\in[0,1)$，故有：

注：考虑一般情况：



$$f(y)=\frac{y_2-y}{y_2-y_1}f(y_1)+\frac{y-y_1}{y_2-y_1}f(y_2)$$

$$Q_1=\frac{1-v}{1}S(j,k)+\frac{v}{1}S(j,k+1)$$

$$Q_1=S(j,k)(1-v)+S(j,k+1)v$$

同理： $Q_2=S(j+1,k)(1-v)+S(j+1,k+1)v$

4) 在 x 方向上对 Q_1， Q_2 插值得到 Q_{12}，同理有：

$$\begin{aligned}Q_{12}&=Q_1(1-u)+Q_2u\\&=(1-u)(1-v)S(j,k)\\&+(1-u)vS(j,k+1)\\&+u(1-v)S(j+1,k)\\&+uvS(j+1,k+1)\end{aligned}$$

Q_{12} 的值即为插值所得对应转换图 (x,y) 的像素值。

实现方法：

我们只需要根据缩放后图像的 width，length，即可遍历输出图像的每一个像素点，通过放缩比例找到对应原图的像素点位置（可能为小数），找到距该位置最近的待插值的四个点，由双线性插值求出输出图像在该点的像素值，遍历完成后就得到缩放后的图像。

2.3 Quantization (28 Points)

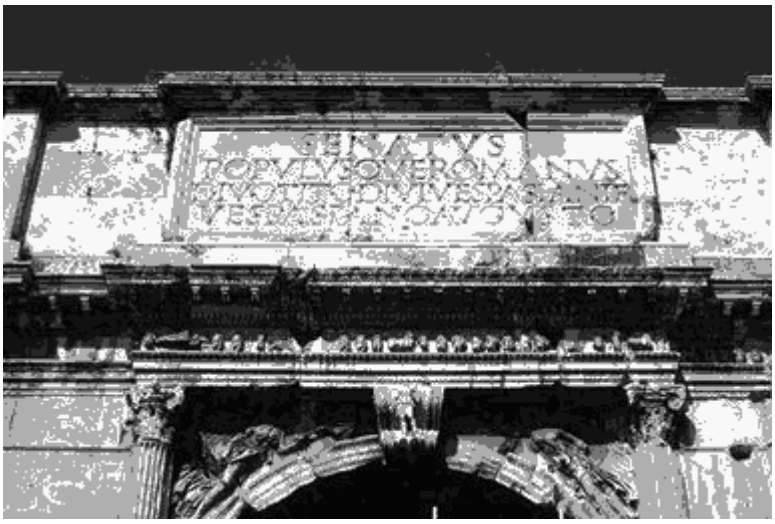
Write a function that takes a gray image and a target number of gray levels as input, and generates the quantized image as output. The function prototype is "quantize(input img, level) → output img", where "level" is an integer in [1, 256] defining the number of gray levels of output. You can modify the prototype if necessary. For the report, please load your input image and use your "quantize" function to:

1. Reduce gray level resolution to 128, 32, 8, 4 and 2 levels, then paste your results respectively. Note that, in practice computers always represent "white" via the pixel value of 255, so you should also follow this rule. For example, when the gray level resolution is reduced to 4 levels, the resulting image should contain pixels of 0, 85, 170, 255, instead of 0, 1, 2, 3. (8 Points)

   128 levels:

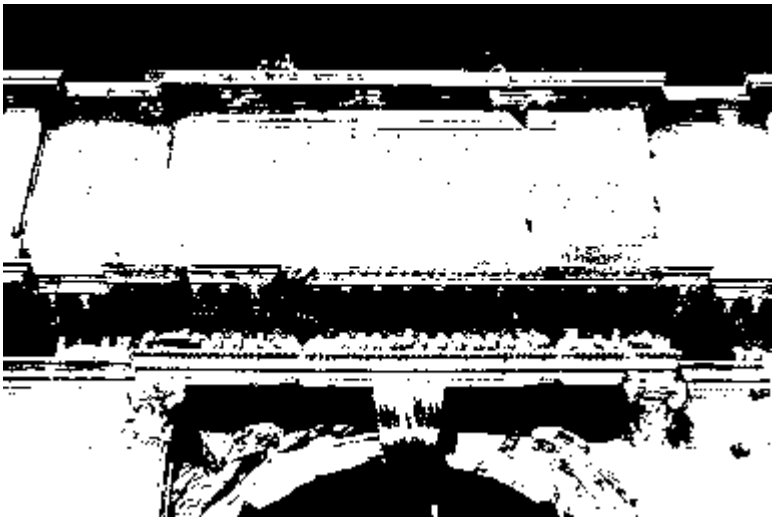
   32 levels:

8 levels:



4 levels:

2 levels:



2. Detailedly discuss how you implement the quantization operation, i.e., the "quantize" function, in less than 2 pages. Again, please focus on the algorithm part. Analyzing and discussing interesting experiment results are also welcomed, but please don't widely copy/paste your codes in the report (20 Points).

算法理论：

对灰度图像进行量化处理，根据量化等级 level 确定原灰度值和量化后灰度值对应关系，为输出图像的各个像素点赋值，从而得到量化后图像。

分级样例：

2-level： 0，255

4-level：0，85，170，255

8-level：0，36，72，108，144，180，216，252

每一级灰度值范围长度 $length$=Math.floor$\left(255/\left(level-1\right)\right)$

1) 确定原图灰度值对应量化后的等级

$$\text{distance} = \text{grayValue}\left(\text{原灰度值}\right) - \text{graylevel}\left(\text{底端灰度级}\right) * \text{length}$$

$$\text{halfofLevellength}\left(\text{新灰度级长度一半}\right) = \text{(double)length/2}$$

量化后等级应取最接近等级：

若 $\text{distance} \geq \text{halfofLevellength}$ ，量化后等级 newLevel = graylevel+1

若 $\text{distance}<\text{halfofLevellength}$，量化后等级 newLevel = graylevel

2) 由量化后等级确定像素值

易得：量化后灰度值 = length * newLevel

实现方法：

输入 BufferedImage 对象（待处理灰度图）和量化等级 level，遍历灰度图像每一个像素点，获取其 ARGB 值，由上述算法量化后把新的 ARGB 值赋给此像素点，最后得到量化后图像。

注：灰度图 RGB 三个分量值均相同，故计算灰度值时可选取一个分量计算，具体实现时选取 R 分量代表此像素灰度值。

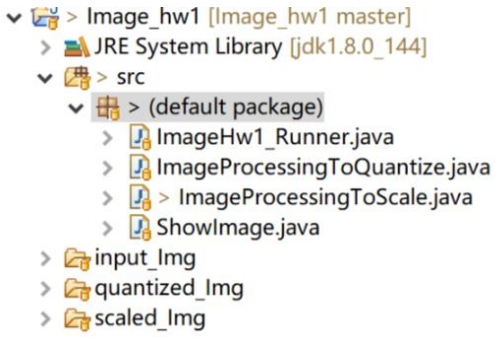
注意：

1. 在本文中坐标系下，若处理目标图像 i 行 j 列的像素点，调用 setRGB(j, i), getRGB(j, i)才能得到正确的处理结果，否则可能会发生越界。
2. 作业提供的 16.png 输入图像是灰度图像，类型：TYPE_BYTE_GRAY，位深度：8
   若处理图像时新声明 BufferedImage 对象使用 TYPE_INT_ARGB 类型，位深度为 32，和原图不符合，会使输出图像效果不佳。

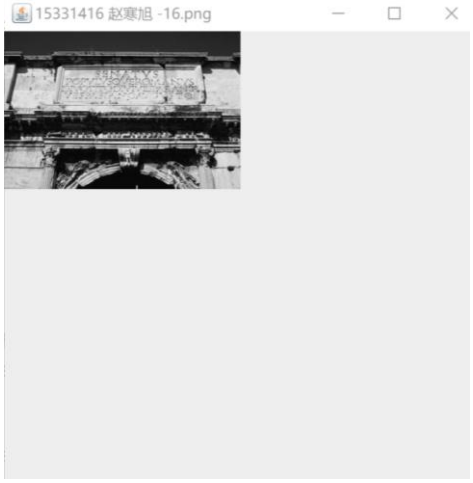
static int **TYPE_INT_ARGB** 表示一个图像，它具有合成整数像素的 8 位 RGBA 颜色分量。	常量值：2
static int **TYPE_BYTE_GRAY** 表示无符号 byte 灰度级图像（无索引）。	常量值：10

程序运行指南：

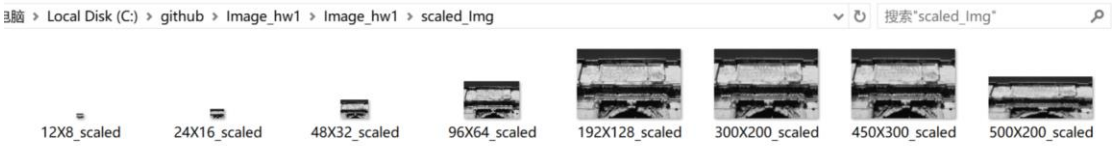
1. 程序结构如图

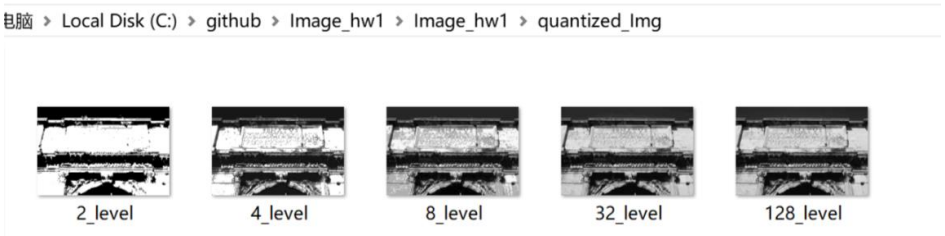


2. 运行 Runner 后，显示输入图像 16.png



并且在项目根目录下生成 scaled_Img 和 quantized_Img 两个文件夹，运行 scale 和 quantize 函数，同时把处理结果保存在相应文件夹中。





可以直接在文件夹中查看结果。

3. 通过 jar 包直接运行，生成两个文件夹，结果可直接查看