

Cover Page

- Matlab实现降低图像灰度级和通过双线性插值缩放图像
- 实验一
- 课程名称：数字图像处理
- 姓名：刘洁琳
- 学号：22920192204247
- 截止日期：2021.9.30
- 提交日期：2021.9.26
- 摘要

本次实验，我使用Matlab完成了两个具体任务：降低图像灰度级和通过双线性插值缩放图像。在任务一中，我让用户自定义自己希望输出的灰度图像的k，在完成实验要求的基础上，还输出了灰度级为4、8、16、32、64、128的头骨图像。在任务二中，我让用户自定义输出图像的宽度和高度，程序可以根据用户输入的值来确定输出图像的大小，进行插值计算，可以缩小图像，也可以放大图像。两个任务我都提供了保存输出图像，以及对比展示源图和输出图像的功能。

1. Reducing the Number of Gray Levels in an Image

Technical discussion

首先，要理解灰度级的概念，先要理解图像的存储方式。在计算机中，图像是以离散的像素点的形式存储的，它可以用矩阵来表示。公式1中表示一个由M*N像素组成的图像，矩阵中的元素表示每个像素的灰度，如f(0, 0)的值是坐标为(0, 0)的像素点的灰度值。这样一个矩阵就组成了整个灰度图像。

注意：在matlab中，的矩阵第一行第一列是(1, 1)而不是(0, 0)。

公式 1

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

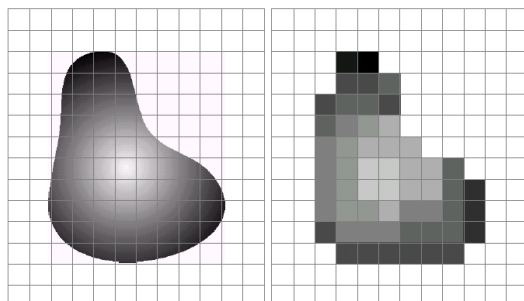
如公式2所示，灰度级的大小取决于k（存储空间的bit数）。比如，当k = 8时，灰度级为256，即像素点的灰度可取自0~255间的任意整数。

公式 2

$$L = 2^k$$

如图1所示，右边是经过数字图像处理后，灰度级为8的数字图像。简单来说，灰度级是多少，灰度图片里**最多**就会有**多少**种不同的颜色。

图 1



因此，当我们希望降低一张图像的灰度级时，其实是减少k的值，即降低2的幂次方。方法是用原始图像像素大小除以降低的灰度级，去除小数部分得到整数，

然后再乘降低的灰度级来量化图像，就得到了降低灰度级后的图像。

Discussion of results

结合上述分析以及实验要求，确定实验步骤如下：


- (1) 读入原图片，并让用户输入预期输出图像灰度级中的k
- (2) 根据输入的k，计算出新图像
- (3) 对比展示原图像和新图像

这里我把（1）放在主程序中，（2）和（3）放在降低灰度级的函数中。

函数的代码如下：

```
function reduce_gray_levels(imgpath,reduce_factor)
    InputImg = imread(imgpath);
    % 完整性约束
    if reduce_factor<0
        reduce_factor=0
    else if reduce_factor>8
        reduce_factor=8
    end
    end
    % 降低灰度级
    dfactor=uint8(2^reduce_factor);
    OutputImg=(InputImg/dfactor)*dfactor;
    %保存输出图像
    output_filename = ['ctskull-output/Ctskull_OutputImg(gray-',num2str(2^( 9 -
reduce_factor)),').tif'];
    imwrite(OutputImg,output_filename);
    % 图像对比
    subplot(1,2,1);
    imshow(InputImg);
    title('灰度级: 256');
    subplot(1,2,2);
    imshow(OutputImg);
    title(['灰度级: ',num2str(2^( 9 - reduce_factor))]);
end
```

在命令行窗口，用户可以输入自己想要的k。

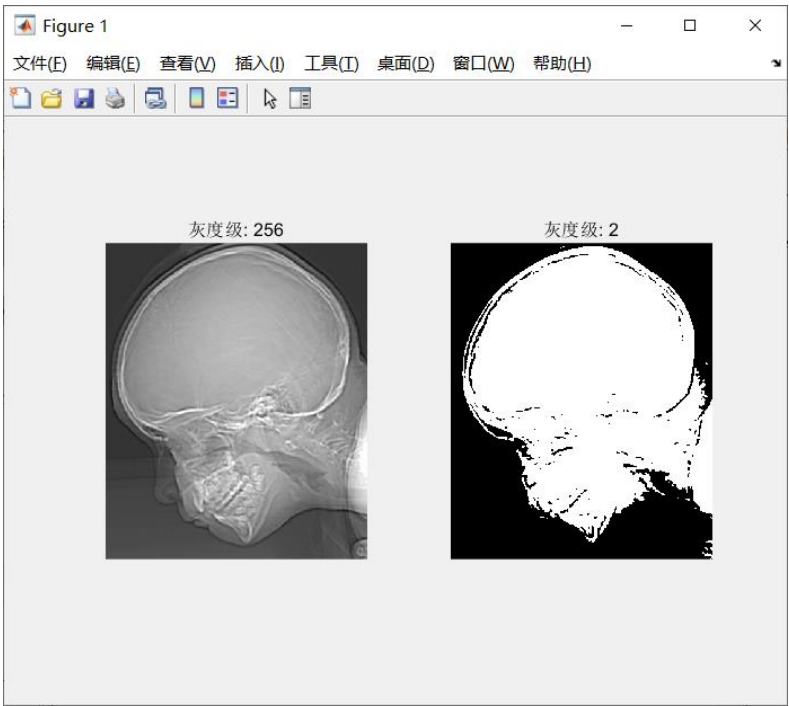
 Please input the desired k :

如果输入的k的值小于0或大于8，程序会自动修正为0或8。

Results

根据实验要求，将原256灰度级的图像降低为灰度级2的图像，如图2所示，符合最终预期的实验结果。

图 2



除此之外，当 $k = 2, 3, 4, 5, 6, 7$ 时，分别对应灰度级4、8、16、32、64、128，如图3、图4、图5、图6、图7、图8所示。

图 3

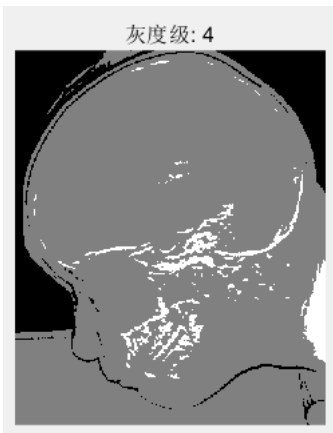


图 4

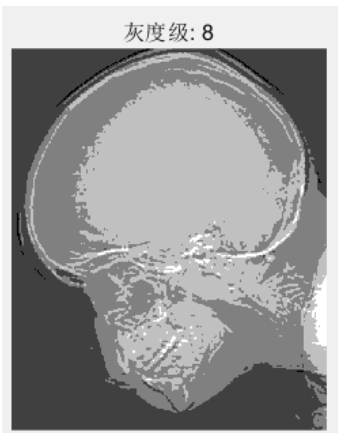


图 5



图 6



图 7

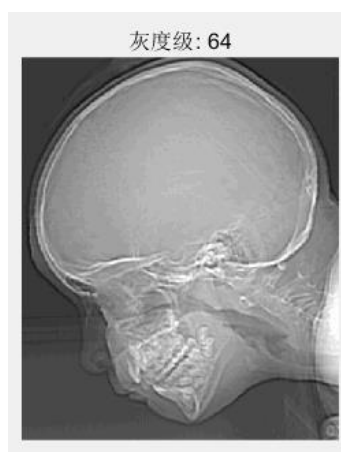


图 8



Appendix

% 主程序代码

```
imgpath = pwd + "\Fig0221(a)(ctskull-256).tif" ;
```

```
k= input(" Please input the desired k : ");
```

```
reduce_factor = 9 - k;
```

```
reduce_gray_levels(imgpath,reduce_factor);
```

2. Zooming and Shrinking Images by Bilinear Interpolation

Technical discussion

首先，必须认识到图像在计算机中是如何存储的。一张彩色图片可以用一个三维矩阵来表示，前两个维度分别表示它的高度和宽度，第三个维度大小为3，分别表示它的r、g、b值。而灰度图像的RGB值相等，只需要两个维度就能表示。说明这点是因为在之后的matlab中我们需要用到三维矩阵，来保留原图像的rgb值（这样程序就可以从灰度图像拓展到彩色图像的缩放了）。

之后，对于图像的缩放，**双线型插值算法的原理是利用了源图中虚拟点四周的四个真实存在的像素值来共同决定目标图中的一个像素值**，因此缩放效果比简单的最邻近插值要好很多。虽然计算量比零阶插值大，但缩放后图像质量高，不会出现严重的图像失真问题。

虚拟点指的是，新图像中的像素点在源图上的映射，可以理解成，新图像根据比例缩放到源图大小后，其像素点在源图中对应的位置。**虚拟位置的坐标是浮点数。**

一个完整的双线性插值算法可描述如下：

- (1) 根据要输出的新图像的大小创建新图像。
- (2) 由新图像的某个像素 (x, y) 映射到原始图像 (x', y') 处。
- (3) 对 x', y' 取整得到 (xx, yy) 并得到 (xx, yy) 、 $(xx+1, yy)$ 、 $(xx, yy+1)$ 和 $(xx+1, yy+1)$ 的值。
- (4) 利用双线性插值得到像素点 (x, y) 的值并写回新图像。
- (5) 重复步骤 (2) 直到新图像的所有像素写完。

双线性插值的公式可表示如下：

$$f(x, y) = (1 - \alpha) * (1 - \beta) * f(u, v) + (1 - \alpha) * \beta * f(u, v + 1) \\ + \alpha * (1 - \beta) * f(u + 1, v) + \alpha * \beta * f(u + 1, v + 1)$$

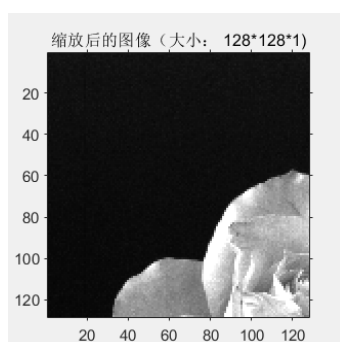
在双线性插值的过程中，有两个关键点：

(1) 计算虚拟点的位置

这里我选择的计算方法是先计算出根据源图和新图像的比例计算出**步长**，利用步长来得到映射关系。

```
% 步长
step = (width-1)/(new_width-1);
```

如果虚拟点的对应关系出错，可能就会出现如下情况：



(2) 边缘点的处理

这里我将源图的四条边都拓展了一排，即矩阵的长度增加2，宽度也增加2，然后将原来四个顶点的值对应赋给拓展后的四个顶点，原来四条边的值对应赋给拓展后四条边（除顶点外）。

Discussion of results

函数的代码如下：

```
function[output_filename] = bilinear_interpolation( imgpath, sizex, sizey)
%% 1. 读入图像,并根据用户输入的size创建新图像。
InputImg = imread(imgpath);
[height,width,rgb] = size(InputImg);
new_width = round(sizex);
new_height = round(sizey);
OutputImg = zeros(new_width,new_height,rgb);
```

```

%% 2. 拓展原图像边缘
InputTemp = zeros(height+2,width+2,rgb);
InputTemp(2:height+1,2:width+1,:) = InputImg;

% 处理原图四条边 (除四个顶点外)
InputTemp(1,2:width+1,:)=InputImg(1,:,:);
InputTemp(height+2,2:width+1,:)=InputImg(height,:,:);
InputTemp(2:height+1,1,:)=InputImg(:,1,:);
InputTemp(2:height+1,width+2,:)=InputImg(:,width,:);

% 处理原图四个顶点
InputTemp(1,1,:) = InputImg(1,1,:);
InputTemp(1,width+2,:) = InputImg(1,width,:);
InputTemp(height+2,1,:) = InputImg(height,1,:);
InputTemp(height+2,width+2,:) = InputImg(height,width,:);

%% 3. 由新图像的某个像素 (i,j) 映射到原始图像(ii, jj)处, 并插值计算。
% 步长
step = (width-1)/(new_width-1);

% 扫描新图像的每个像素
for j = 1:new_height
    for i = 1:new_width
        % (i,j)对应原图中的虚拟位置[从(0, 0)开始的表示法]
        ii = step*(i-1);
        jj = step*(j-1);
        % 向下取整 (决定距离最近的四个像素点位置)
        u = floor(ii);
        v = floor(jj);
        % 小数部分 (决定权值)
        a = ii - u;
        b = jj - v;
        % 图像中最小坐标值设为(1, 1)
        u = u + 1;
        v = v + 1;
        % 双线性插值计算公式
        OutputImg(i,j,:) = (1-a)*(1-b)*InputTemp(u,v,:) + (1-a)*b*InputTemp(u,v+1,:) +
        a*(1-b)*InputTemp(u+1,v,:) + a*b*InputTemp(u+1,v+1,:);
    end
end

%% 4. 输出结果
% 输出图像
OutputImg = uint8(OutputImg);

```



```

output_filename =
['rose-output/Rose_OutputImg(',num2str(new_width),'x',num2str(new_height),').tif'];
imwrite(OutputImg,output_filename);
% 对比展示
subplot(1,2,1);
imshow(InputImg);
title(['原图像 (大小: ',num2str(height),'*',num2str(width),')']);
subplot(1,2,2);
imshow(OutputImg);
title(['插值后的图像 (大小: ',num2str(new_width),'*',num2str(new_height),')']);
end

```

用户可以输入自己希望输出的图像的size x和size y，执行主程序后，就会保存输出的图像，并在窗口中展示对比图像。如图9所示。

图 9

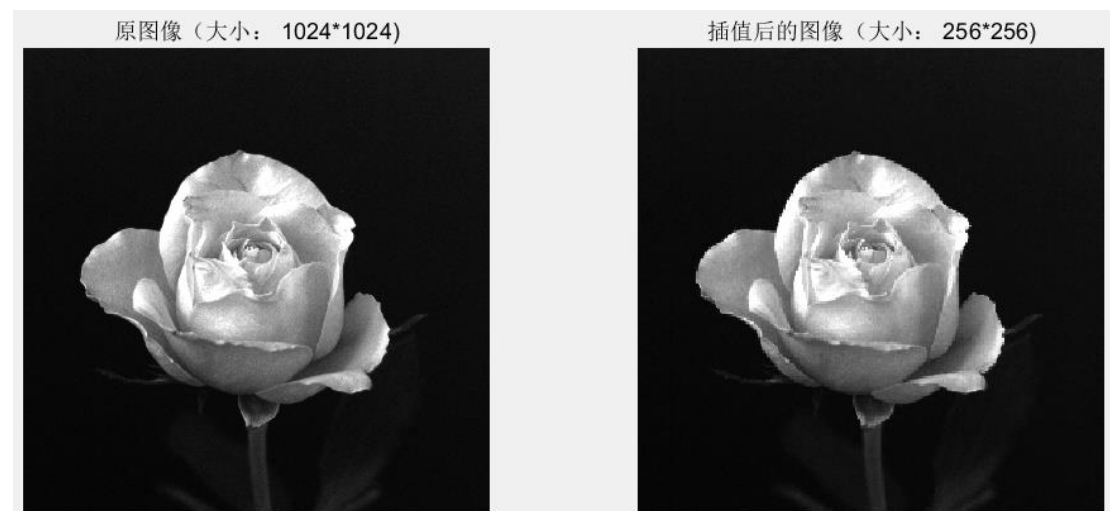
```

output_filename = bilinear_interpolation(imgpath, sizex, sizey);
Please input the desired size x : 256
Please input the desired size y : 256
fx >>

```

将1024x1024的图片缩小为256x256后，效果如图10。

图 10



将256x256的图片重新放大为1024x1024后，效果如图11

图 11



放大后，对比经过缩小再放大的1024x1024图像和原来的1024x1024图像，发现重新放大后的图像质量不如原来的图像质量了，原因是：图片经过插值变为256x256分辨率的图像后，已经丢失了许多像素点的信息，再次放大后，也只能在256x256分辨率的图像已有的信息基础上，对像素插值，原来的已经信息丢失了，所以质量肯定是下降了。

Results

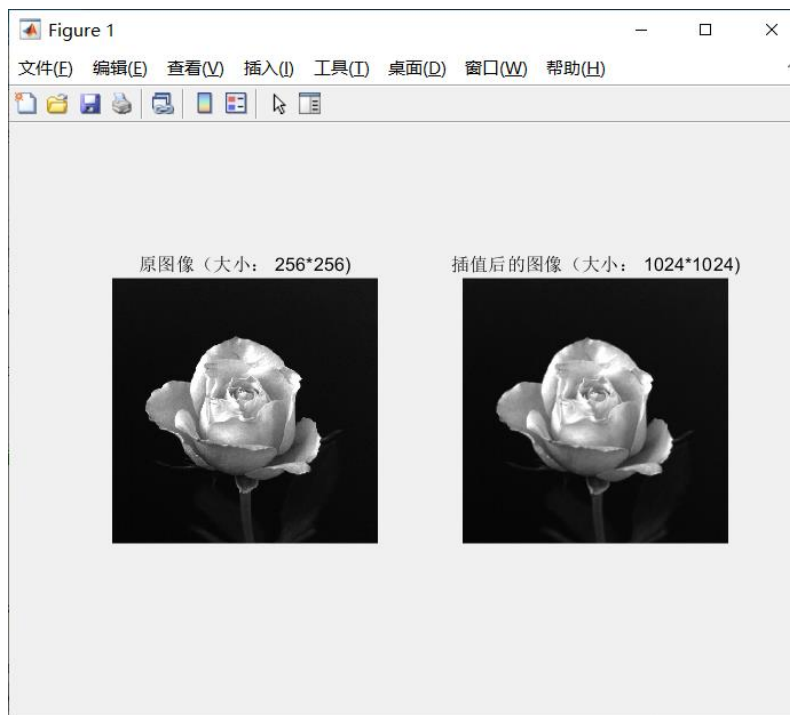
将1024x1024的图片缩小为256x256

图 12



将256x256的图片重新放大为1024x1024

图 13



Appendix

```
%% 双线性插值主程序
% step1: shrink image
imgpath = 'Fig0219(rose1024).tif';
sizex= input(" Please input the desired size x : ");
sizey= input(" Please input the desired size y : ");
output_filename = bilinear_interpolation(imgpath, sizex, sizey);

% step2: zoom image
bilinear_interpolation(output_filename, 1024, 1024);
```