DIP第一次实验报告

黄志鹏 PB16150288

实验题目：三种图像插值方法

实验目的

- 使用代码实现双三方在内的 三种图像插值方法
- 比较三种插值方式的异同和优劣
- 熟悉matlab或者python 中的一些用于图像处理的包

实验内容

实现三个图像插值函数

- nearest 最近邻算法

```
function [img_2] = nearest(img_1, n)
%% 这里将图像的灰度值大小从255 的int 变成了0到1 的double
img_1 = im2double(img_1);
%% 获取新的图像的尺寸
size_1 = size(img_1);
h_1 = size_1(1);
w_1 = size_1(2);
h_2 = floor(h_1 * n);
w_2 = floor(w_1 * n);
img_2 = zeros(h_2, w_2);
%% 对新的图像的每个像素点的灰度值进行赋值
for x = 1: h_2
    for y = 1: w_2
        %% 获取这个像素点在原图的坐标， 以及坐标的整数部分和小数部分
        i = floor(x / n);
        j = floor(y / n);
        u = x / n - i;
        v = y / n - j;
        %% 寻找原图中离这个坐标最近的点 这里使用min，max来防止边缘处的越界错误
        if u < 0.5 && v < 0.5
            img_2(x, y) = img_1(min(max(i, 1), h_1), min(max(j, 1), w_1));
        elseif u >= 0.5 && v < 0.5
            img_2(x, y) = img_1(min(max(i + 1, 1), h_1), min(max(j, 1), w_1));
        elseif u < 0.5 && v >= 0.5
            img_2(x, y) = img_1(min(max(i, 1), h_1), min(max(j + 1, 1), w_1));
        else
            img_2(x, y) = img_1(min(max(i + 1, 1), h_1), min(max(j + 1, 1), w_1));
        end
    end
end
%% 这里将灰度值从0到1的double 变到了0到255 的int型
```

```
img_2 = im2uint8(img_2);
end
```

- 双线插值算法

```
function [img_2] = bilinear(img_1, n)
%% 这里将图像的灰度值大小从255 的int 变成了0到1 的double
img_1 = im2double(img_1);
%% 获取新的图像的尺寸
size_1 = size(img_1);
h_1 = size_1(1);
w_1 = size_1(2);
h_2 = floor(h_1 * n);
w_2 = floor(w_1 * n);
img_2 = zeros(h_2, w_2);
%% 对新的图像的每个像素点的灰度值进行赋值
for x = 1: w_2
    for y = 1: h_2
        %% 获取这个像素点在原图的坐标， 以及坐标的整数部分和小数部分
        i = floor(x / n);
        j = floor(y / n);
        u = x / n - i;
        v = y / n - j;
        i = max(i, 1);
        i = min(i + 1, h_1) - 1;
        j = max(j, 1);
        j = min(j + 1, w_1) - 1;
        %% 使用周围4个整数像素点进行插值 这里使用min，max来防止边缘处的越界错误
        img_2(x, y) = (1-u) * (1-v) * img_1(i, j)...
            + (1-u) * v * img_1(i, j+1)...
            + u * (1-v) * img_1(i+1, j)...
            + u * v * img_1(i+1, j+1);
    end
end
%% 这里将灰度值从0到1的double 变到了0到255 的int型
img_2 = im2uint8(img_2);
end
```

- 双三方算法

```
function [img_2] = bicubic(img_1, n)
%% 这里将图像的灰度值大小从255 的int 变成了0到1 的double
img_1 = im2double(img_1);
%% 获取新的图像的尺寸
size_1 = size(img_1);
h_1 = size_1(1);
w_1 = size_1(2);
h_2 = floor(h_1 * n);
w_2 = floor(w_1 * n);
```

```
img_2 = zeros(h_2, w_2);
D = zeros(4, 4);
%% 对新的图像的每个像素点的灰度值进行赋值
for x = 1: w_2
    for y = 1: h_2
        %% 获取这个像素点在原图的坐标， 以及坐标的整数部分和小数部分
        i = floor(x / n);
        j = floor(y / n);
        u = x / n - i;
        v = y / n - j;
        %% 这里获得16个相邻像素点的系数，采用bicubic函数， getWeight函数的实现在下面
        W = getWeight(u, v);
        %% 这里获得周围的16个像素点的位置， 这里使用min，max来防止边缘处的越界错误
        for a = 1:4
            for b = 1:4
                D(a, b) = img_1(min(max(i - 2 + a, 1), h_1), min(max(j - 2 + b,
1), w_1));
            end
        end
        %% 这里系数和16个周围像素点依次相乘 然后相加
        img_2(x, y) = sum(sum(W.* D)); %% 注意这里使用的是点乘
    end
end
%% 这里将灰度值从0到1的double 变到了0到255 的int型
img_2 = im2uint8(img_2);
end

function weight = getWeight(u, v)
weight = zeros(4, 4);
x_weight = zeros(4, 1);
y_weight = zeros(1, 4);
%% 对x， 和y 方向分别计算系数， 最后相乘在一起， 调用的Bic函数在下面有定义
for i = 1: 4
    d = abs(u + 2 - i);
    x_weight(i, 1) = Bic(d);
end
for j = 1: 4
    d = abs(v + 2 - j);
    y_weight(1, j) = Bic(d);
end
weight = x_weight* y_weight;
end

function w = Bic(d)
%% bicubic函数的数学定义， 这里参数a 采用 -1
a = - 1;
if d <= 1
    w = 1 - (a + 3) * d ^ 2 + (a + 2) * d ^ 3;
elseif d > 1 && d < 2
    w = -4 * a + 8 * a * d - 5 * a * d ^ 2 + a * d ^ 3;
else
    w = 0;
end
end
```

实验结果

缩小为原来的0.2

- 最近邻

- 双线插值

- 双三方

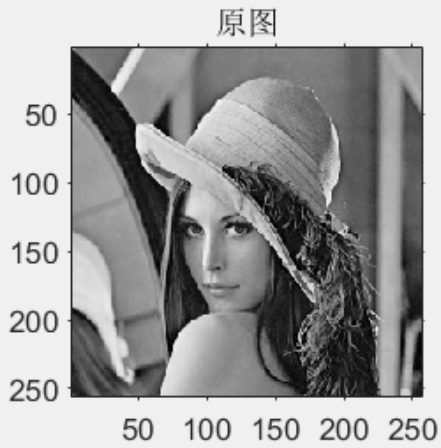
放大为原来的3倍

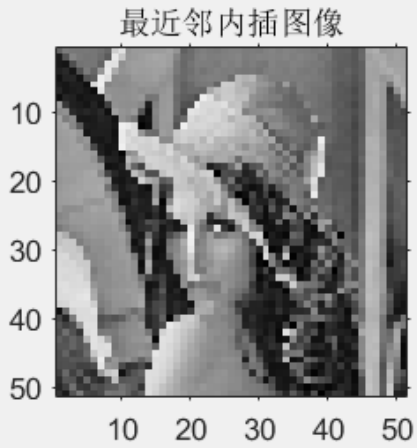
- 最近邻

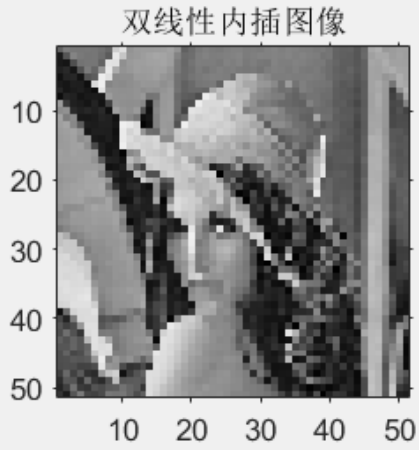
- 双插值

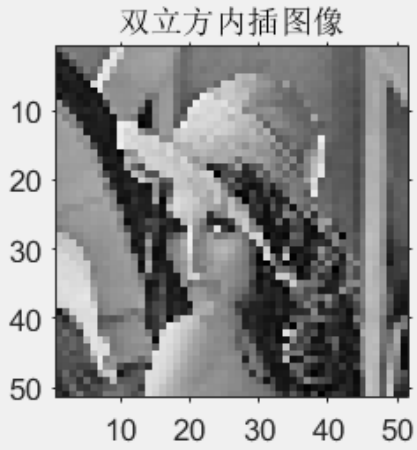
- 双三方

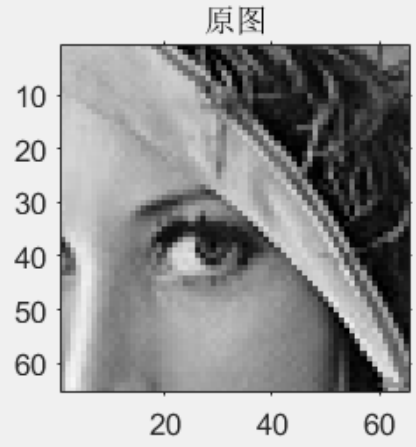
三种方法放在一起进行对比

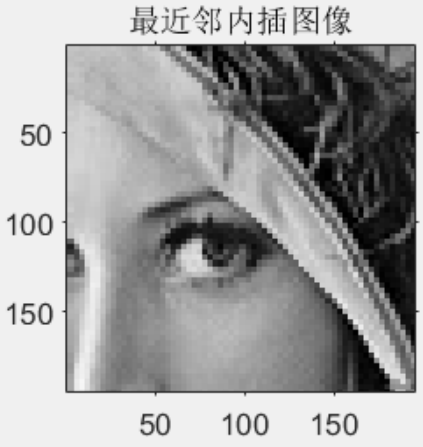

原图
50
100
150
200
250
50 100 150 200 250

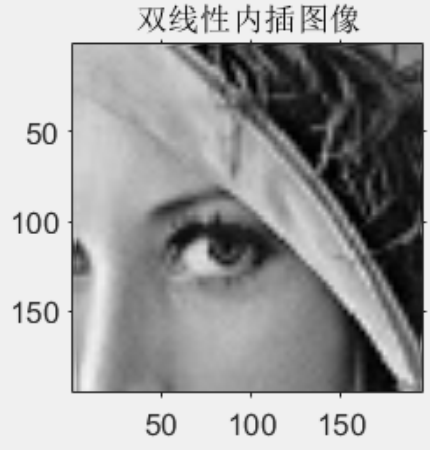

最近邻内插图像
10
20
30
40
50
10 20 30 40 50

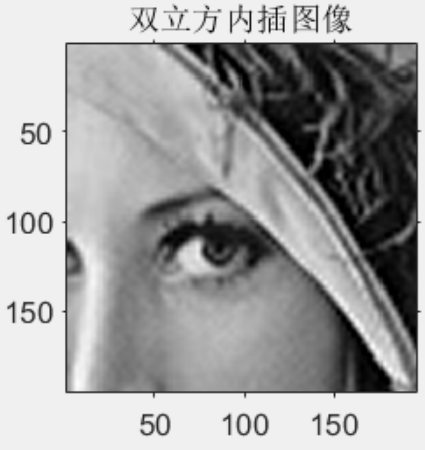

双线性内插图像
10
20
30
40
50
10 20 30 40 50


双立方内插图像
10
20
30
40
50
10 20 30 40 50


原图
10
20
30
40
50
60
20 40 60


最近邻内插图像
50
100
150
50 100 150


双线性内插图像
50
100
150
50 100 150


双立方内插图像
50
100
150
50 100 150

实验分析

1. 对实验结果的分析

从放大3倍的各张图中， 我们可以发现

1. 最近邻法 运算和算法编写最为简单，但是插值的结果最不好， 虽然勉强满足放大和缩小图像的作用，但是图像边缘处有明显的锯齿状，使得图像的整体清晰度不高。
2. 双线内插 运算比最近邻复杂， 但是整体的插值后图像比最近邻法好，没有灰度不连续的缺点，结果基本令人满意。它具有低通滤波性质，使高频分量受损，图像轮廓可能会有一点模糊。
3. 双三次内插 算法最为复杂，插值后的图像效果最好.

2. 编写代码中遇到的困难和解决方法

问题： 在编写代码的过程中，在坐标周围的原图像素点的过程中，总是会发生index error。 即index超过了合法的范围，这是由于有的时候坐标在图像边界时，会周围的像素点超过了合法的范围。 解决方式：

1. 我已开始尝试运算之前使用padding，但是我发现这个题目的运算中，由于int到double的运算，需要padding的尺寸时常是不能事先确定的。
2. 后来我才用异常处理的方式，try...catch 语句，这样发生index 异常， 这个像素点就设置成0, 但是这样的结果就是，产生的图像有黑边。
3. 最后我想到了可以用max 和 min 这个函数来防止index 溢出。如

```
img_2(x, y) = img_1(min(max(i + 1, 1), h_1), min(max(j + 1, 1), w_1));
```

这样的取值指令，就可以做到既能防止出现index error， 又不会影响边界上的图像效果。

3. 实验过程中学习到的知识

1. 知道了三种常用的插值方式的数学原理和主要实现（上面的代码和代码注释已经说明）
2. matlab的一些语法，如matlab 的index从1开始， 而且后面的边界是取得到的， 这个和python 有很大的区别。
3. matlab 有自带的处理图像的一些很好地函数接口，如imread， imwrite，imshow， im2double，im2uint8 等等

4. 缩小图之所以结果不如人意的原因

- 从缩小图中可以看到，采用了三种插值方法的缩小结果相差不多，距离老师给的example 的效果在双线插值和双三次上相差很远， 具体表现在有灰度上的不连续。从过程来看，缩小图返回到原图的坐标之间，本来就相离很远，及时是双三次插值里面的十六个点也无法够到这之间的距离，故产生不了相邻点之间的联系，故产生了灰度上的不连续。

- 想要解决这个问题，可以尝试增加双三次算法中的参考点邻接点的数量， 从16个增加为25个甚至是49个，这样应该可以减少缩小图中灰度不连续的情况。