数字图像处理第三次实验图像几何变换报告（文末附代码）

1 问题

实现一幅图像的放大功能

2 问题分析

本次实验涉及到图像的放大功能，根据课本和查阅的资料，该实验的流程具体为：

1 根据放大的倍数，得到放大后的图像 output_img(每个像素值为 0)

2 遍历 output_img 上的像素，将像素映射回原来的图片上

3 由于 ouput_img 的像素密度比输入图像的像素密度大，所以无法做到输出图像的像素和输入图像的像素一一对应。映射后可能有两种情况。A： 输出像素刚好和输入像素对应 B ：输出像素点映射回原图片之后其坐标(i,j)不是整数。对与 B 我们需要根据原图像在（i,j）附近的整数点来对这个映射的像素插值，输出像素点的取值就为插值的结果。

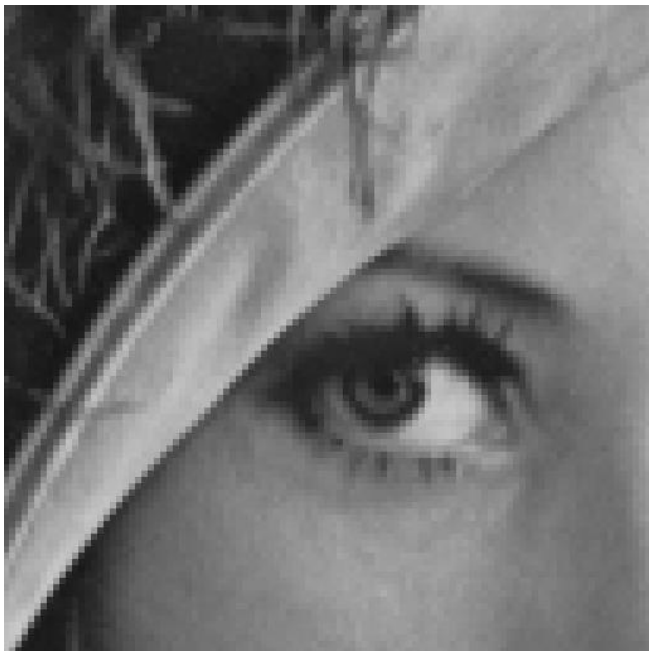
4 根据插值的方法，有最近领插值（即寻找里(i,j)最近的整数点，用该点的像素作为（I,j）的像素，还有双线性插值（用 (i,j)周围 4 个整数点来共同决定该像素的值）

本实验主要自己实现了最近领插值和双线性插值，分析不同插值方式得到的输出图像的不同，尝试总结不同插值方式的优缺点。

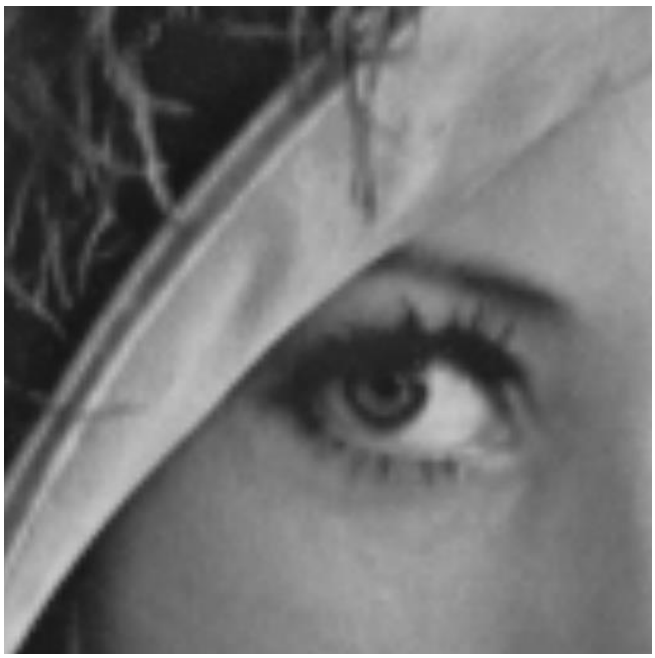
3 实验结果：

原图像

最近领插值（放大 5 倍,取局部效果，大小与原图相同，在 result 文件夹中有完整图片）：



双线性插值（放大 5 倍,取局部效果，大小与原图相同，在 result 文件夹中有完整图片）:



4 实验结果

分析最近领插值和双线性插值我们可以发现，最近领插值存在严重的边缘锯齿效应，产生的原因是插值的时候暴力的选择最近的点的像素作为插值的结果，那么在输出图像的一块区域很有可能选择原图中同一个像素点的像素作为插值结果，这就导致输出图像的一大块区域的像素值相同。同时在图片经过放大之后，直接用原图的像素点进行插值也会造成灰度变化不够自然，不够平滑。从而导致锯齿状的结果。

而双线性插值用原图的 4 个点结合映射后像素点的位置进行插值，使得在输出图像中，像素点之间的变化更加平滑，放大的结果明显要好于用最近领插值得到的结果。

当然双线性插值的时间代价也比最近领插值的时间代价高不少。在实际场景中我们应该根据具体需求来选择插值算法。

5 代码

```
import cv2 as cv
import os
import numpy as np

def resize_img(input,scale,mode,output_mode='same'):
    '''
    放大图像的函数,直接对整个图像进行放大
    :param input:  输入图像,注意输入图像是灰度图
    :param scale: 放大的尺度，是一个数组 [scale_h,scale_w] 或者说是
[scale_i,scale_j]
    :param mode:  插值方式有最近领和双线性
    :param output_mode  输出方式
    :return:
    '''
    assert (mode in ['nearest','bilinear'])
    assert (output_mode in ['same','scaled'])
    input = np.array(input,dtype=np.uint8)
    input_shape = input.shape[0:2]
    output_shape =
[int(input_shape[0]*scale[0]),int(input_shape[1]*scale[1])]  #计算输出尺寸
    output = np.zeros(output_shape,dtype = np.uint8)
    input =  np.pad(  input,((0,1),(0,1)),'constant')  # 填充防止越界
    for i in range(output_shape[0]):
        for j in range(output_shape[1]):
            #遍历输出图像的每一个像素
            if i%scale[0]==0 and j%scale[1]==0:
                output[i][j] = input[int(i/scale[0])][int(j/scale[1])]   #如
果正好对应的话
            else:
                #求其所在方格左上角的点
                i_f = i/scale[0]
                j_f = j/scale[1]
                i_0 = int(np.floor(i_f))
                j_0 = int(np.floor(j_f))
                if mode=='nearest' :
                    #如果是最近邻插值,如果在方格中心取左上角的点
                    if i_f - i_0 <=0.5:
                        offset_i=0
                    else:
                        offset_i = 1
                    if j_f - j_0 <=0.5:
                        offset_j=0
                    else:
```

```
                    offset_j = 1
                output[i][j] =
input[int(i_0+offset_i)][int(j_0+offset_j)]
            elif mode=='bilinear':
                #双线性插值,根据双线性插值的原理 其  y = di*dj(c+a-d-b) +
di*(d-a) + dj*(b-a) + a 其中 [a,b,c,d]分别是4个点的灰度值，a是左上角
                #其余点按顺时针顺序
                di = (i_f- i_0)
                dj = (j_f - j_0)
                a = int(input[i_0][j_0])
                b = int(input[i_0][j_0+1])
                c = int(input[i_0+1][j_0+1])
                d = int(input[i_0+1][j_0])
                output[i][j] = di*dj*(c+a-d-b) + di*(d-a) + dj*(b-a) + a
    # 输出
    if output_mode =='scaled':
        return np.array(output,dtype = np.uint8)
    elif output_mode =='same':
        #取中心点
        output_center = [s/2 for s in output_shape]
        i_begin = int(output_center[0]-0.5* input_shape[0] )
        j_begin =    int(output_center[1]-0.5* input_shape[1] )
        output =
output[i_begin:i_begin+input_shape[0],j_begin:j_begin+input_shape[1]]
        return np.array(output, dtype=np.uint8)

if __name__ =='__main__':
    #主函数调用
    BASE_PATH = os.path.dirname(os.path.abspath(__file__))
    RESULT_PATH = os.path.join(BASE_PATH, "result")
    if not os.path.exists(RESULT_PATH):
        os.mkdir(RESULT_PATH)
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('--img', help="the path of the input img")
    parser.add_argument('--scale', help='the scale(>=1) ')
    parser.add_argument('--mode', help='insert mode')
    parser.add_argument('--output', help='output_mode')
    args = parser.parse_args()
    # 获取图片名称
    img_name = os.path.split(args.img)[-1]
    img_name, ext = os.path.splitext(img_name)
    ext = '.bmp'   #用 .bmp 格式输出
```

```
    input = cv.imread(args.img, flags=cv.IMREAD_GRAYSCALE)
    if args.output:
        img_name = img_name + "_" + str(args.scale) + "_" + args.mode
+'_'+args.output+ ext
        output = resize_img(input,
[float(args.scale),float(args.scale)],args.mode,args.output )
    else:
        img_name = img_name + "_" + str(args.scale) + "_" + args.mode
+'_same'+ ext
        output = resize_img(input, [float(args.scale), float(args.scale)],
args.mode)
    img_name = os.path.join(RESULT_PATH, img_name)
    cv.imwrite(img_name, output)
```