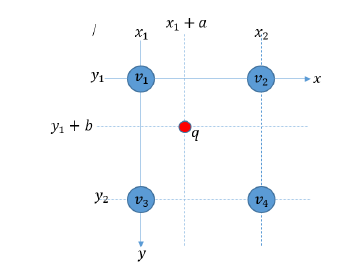
双线性插值

**一 实验原理分析**

双线性插值是在两个方向上分别做线性插值。在作业说明文件中给出了三种情况的公式，显然第一种点重合的情况和第二种同行列的情况是第三种情况的一个特例，所以只需针对前两种特殊情况进行特判，即可使用如下公式对所有情况进行计算。



参照图一进行说明，特判方法如下：

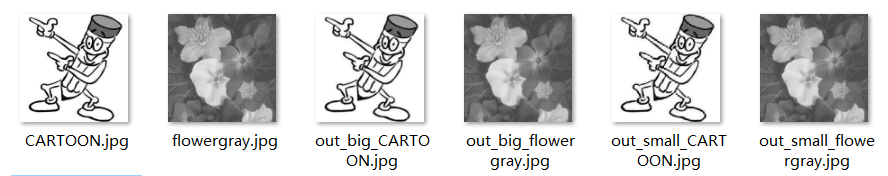


图一：第三种情况的说明图

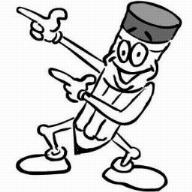
对于第一种情况，此时通过计算可知，点与点重合，直接对该点进行赋值并跳过循环。此时双线性插值变成零阶插值。

对于第二种情况。如果与同行，则此时为零，公式的第三四项为零，双线性插值变成线性插值。只需在边缘进行特判，对的坐标进行改变防止越界。在程序中我将的坐标改变至。如果与同列，那么操作方式与同行时类似。当与同行或同列时，相当于至整体平移，情况与刚才讨论的两种一致。

**二 结果展示**



图二：原始图片与处理后图片展示

图三：cartoon效果展示



图四：flowergray效果图展示

如上，图二是在程序运行结束之后，在文件夹中使用截图获得。图三和图四是两幅图片分别按照原图、放大1.5倍率、缩小0.75倍率的原图。由于在word2017中会自动设置图片压缩，并且只提供了以厘米为单位的图片大小设置。所以这里设置了在330dpi下的图片，其中256、384、192像素分别对应1.5厘米、2厘米、3厘米。

**三 代码与debug**

使用python完成本次作业，代码如下：



import numpy

import imageio #用来做图像读写

def imresize(ori\_im, ori\_size, tar\_size):

tar\_im=numpy.zeros(tar\_size)

ratio\_i=tar\_size[0]/ori\_size[0]

ratio\_j=tar\_size[1]/ori\_size[1]

for i in range(len(tar\_im)):

for j in range(len(tar\_im[i])):

ori\_i=i/ratio\_i

ori\_j=j/ratio\_j

x1=int(ori\_i)

y1=int(ori\_j)

if x1==ori\_i and y1==ori\_j:

tar\_im[i][j]=ori\_im[x1][y1]

continue

a=ori\_i-x1

b=ori\_j-y1

t1=(1-a)\*(1-b)\*ori\_im[x1][y1]

t2=a\*(1-b)\*ori\_im[x1][y1] if y1+1>=ori\_size[1] else a\*(1-b)\*ori\_im[x1][y1+1]

t3=(1-a)\*b\*ori\_im[x1][y1] if x1+1>=ori\_size[0] else (1-a)\*b\*ori\_im[x1+1][y1]

t4=a\*b\*ori\_im[x1][y1] if x1+1>=ori\_size[0] or y1+1>=ori\_size[1] else a\*b\*ori\_im[x1+1][y1+1]

tar\_im[i][j]=t1+t2+t3+t4

return tar\_im.astype('uint8')

def main(path\_work,num\_im,big\_ratio,small\_ratio,\*arg):

for i in range(num\_im):

im = imageio.imread(path\_work+eval("arg["+str(i)+"]"))

ori\_size=im.shape

out\_im=imresize(im,ori\_size,[int(ori\_size[0]\*big\_ratio),int(ori\_size[1]\*big\_ratio)])

imageio.imwrite(path\_work+"out\_big\_"+eval("arg["+str(i)+"]"), out\_im)

out\_im=imresize(im,ori\_size,[int(ori\_size[0]\*small\_ratio),int(ori\_size[1]\*small\_ratio)])

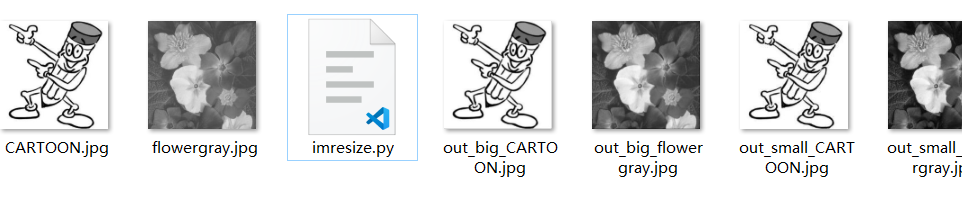
imageio.imwrite(path\_work+"out\_small\_"+eval("arg["+str(i)+"]"), out\_im)

if \_\_name\_\_ == '\_\_main\_\_':

main('C:/Users/19041/Desktop/图像处理/图像处理练习1/',2,1.5,0.75,'CARTOON.jpg','flowergray.jpg')

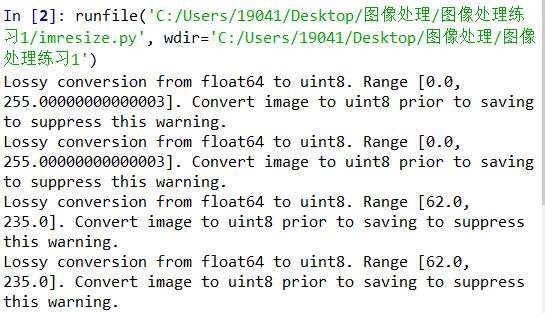
在代码中设置主函数规定运行入口，方便进行程序间调用。其中imresize函数声明并使用ori\_im, ori\_size, tar\_size三个参数并返回tar\_im。在本文第一个部分中所提到的两种特判，第一种通过if x1==ori\_i and y1==ori\_j:来实现，第二种通过为t1-t4赋值时的递推式来实现。在代码中图像的坐标表示比较复杂。作业说明文档中轴从左至右轴从上到下，而在编程中，数组（列表）的表示方法是第一个下标表示行，从上至下排列，第二个下标表示列，从左至右排列。所以实际上对应，对应。同理和、和也是相反的。

在测试过程中遇到了一个非常有趣的bug，经操作后的图片出现了深浅不一的情况，具体效果见图五。



图五：错误的图像处理效果展示

可以看到花的放大和缩小有明显的加深现象，这个现象令我非常的困惑，一开始我的debug思路是检查坐标对应关系和公式的实现，未果。后来我发现在程序执行时会报如图六所示的信息，由于这个信息是一个warning，一开始我并没有对其进行处理，直到我进行调试时输出了图像的矩阵，发现图像并没有以8位深度储存，而是自动被归一化了。经过一定的查阅和交流，我认为问题出在由float64到uint8的强制转换造成的损失上，正确的方法是在返回图象时规定格式return tar\_im.astype('uint8')。

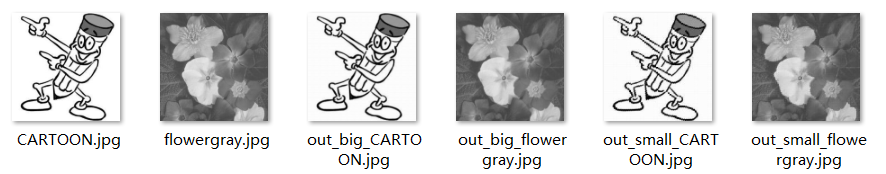


图六：warning照样致命

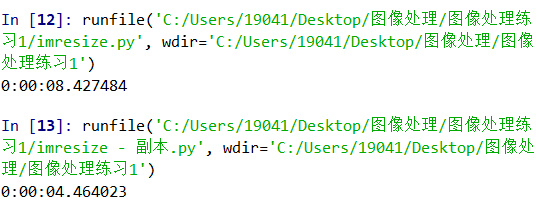
**四 优化与几何中心法**

最后进行一些优化分析。可以看出原公式做了8次乘法，在编程时可对公式进行如下变形以提升性能，此时只有6次乘法。不过我相信python的解释器足够智能，为了提升代码可读性，故没有采用这种方案。

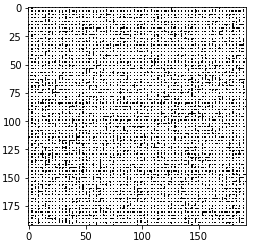


更进一步的优化方案是使用空间换时间的策略。可以使用类似于哈希寻址的打表方式避开所有的乘法开销。完整代码见文末。由于点的灰度值在0-255之间且为整数，所以对小数部分打表。表的结构是二维的，行索引是从0-1的保留两位的小数的值，由于小数无法作为下标进行寻址，所以使用小数的值乘100作为下标，列索引是0-255的灰度值，表中内容是行索引乘以列索引后获得的新灰度值。经计算当保留两位小数时表的大小约为3kb。值得一提的是，当保留三位小数时可以近似为对小数进行10位移位操作，避免了小数乘100获得下标的乘法开销，不过在解释器中位移操作较复杂且没有c语言中提升幅度大，所以放弃了位移方案。该方法与公式中对每项单独取整获得灰度等价，与对公式中对所有项取整获得灰度有期望值不超过1/2的误差。运行的时间开销如图七，上方是使用朴素版本的运行时间，下方是优化后的运行时间。因为python是解释语言，并且python的算数运算经过优化，所以打表法的优势并不明显，如果使用cython进一步优化，可以使程序速度有较大提升，预计可以让加速倍率达到一个数量级。cython对类的支持较为复杂，这也是为什么我没有使用面向对象编程的原因之一。图七是使用优化后的效果图，与先运算后取整版本相类似。图八是优化前后时间对比。图十是缩小花图的优化版本与朴素版本的差值。

图七：时间优化后的效果图

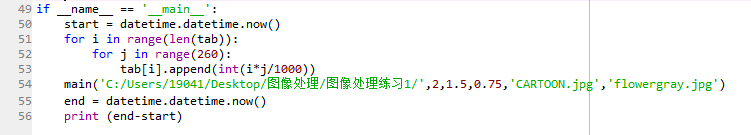


图八：优化前后时间对比

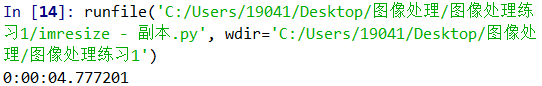


图九：图像差值

为了证明python对算术运算的优化程度，我又运行了计算打表时间后整个程序的运行时间，见图十图十一，令我惊讶的是计算一个25万大小的表只用了0.3秒，可见如果能够使用操作物理层的语言速度确实会有大的提升。



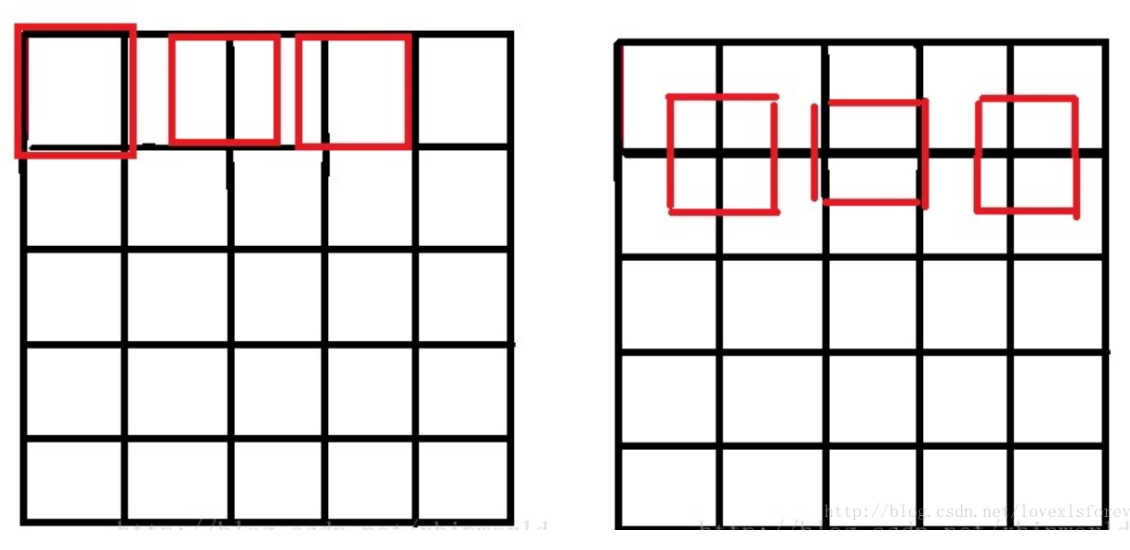
图十：打表时间开销



图十一：计算打表时间后的总时间开销

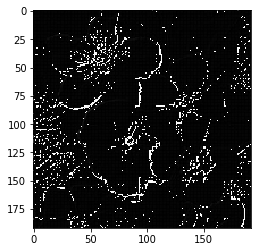
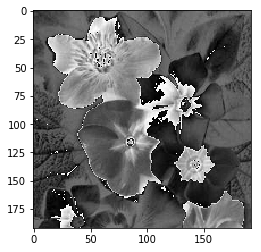
在查阅资料进行优化时，我发现在matlab和opencv中同样采取了与我类似的优化方案，其中opencv使用了整数代替float和移位运算等。值得一提的是在opencv中源图像和目标图像使用了几何中心对齐的方法，而我采用的是四周对齐，opencv的对齐代码如下：  
 SrcX=(dstX+0.5)\* (srcWidth/dstWidth) -0.5  
 SrcY=(dstY+0.5) \* (srcHeight/dstHeight)-0.5

如图十二（来源csdn）所示，左边是四周对齐，右侧是中心对齐，从主观角度来看右侧更为匀称一些。



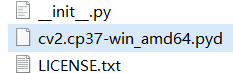
图十二：四周对齐和中心对齐

调用opencv的库函数，与我的未优化速度版本的效果进行了差值对比如图十三左侧图，与优化后版本见图十三右侧图，这里进行了翻转灰度操作。可见有一定差别，发生了整体的偏移。



图十三：与opencv差值

最后我测了一下cv库resize的速度如图十四，cv库居然能接近零秒完成一幅图？？于是我看了一下cv的库，发现cv调用的cv2.cp37-win\_amd64.pyd文件是由其他语言编写的编译文件，又到网上翻了一下源码确实是c++。



图十四：cv的速度与调用

优化版代码，应使imresize文件与该文件处于同一文件夹中

