西安交通大學

数字图像处理报告

数字图像基础

姓名 姜彦凯

班级 自动化少 61

学号 2140506083

提交日期 2019.3.5

数字图像处理-第一次作业

自动化少 61 姜彦凯

西安交通大学电信学院 710049 陕西 西安

摘要：

本次报告首先简单阐述了 BMP 图像格式及其相关数据结构，随后主要完成了作业要求中关于图像处理与计算的各项任务。本次作业以 PyCharm 为平台，利用 Python3.6.3 通过对 lena.bmp,elain1.bmp 图像文件的编程处理,分别得到了 lena.bmp 图像的 8 到 1 级灰度逐级递减显示,lena.bmp 图像的均值和方差;通过近邻、双线性和双三次插值法对 lena.bmp 放大后得到的 2048×2048 尺寸图像，和对 lena.bmp、elain1.bmp 图像分别进行水平 shear 变换和旋转变换后的图像及其插值放大图像。本报告基本方法没有进行调包从原理上对其进行数学编程，以上任务完成后均得到了预期的结果。

1 BMP 格式初探

7.bmp

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
00000000	42	4D	6E	04	00	00	00	00	00	00	36	04	00	00	28	00	00	00	07	00	00	00	07	00
00000018	00	00	01	00	08	00	00	00	00	00	38	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	01	01	01	00	02	02	02	00	03	03	03	00	04	04
00000048	04	00	05	05	05	00	06	06	06	00	07	07	07	00	08	08	08	00	09	09	09	00	0A	0A
00000060	0A	00	0B	0B	0B	00	0C	0C	0C	00	0D	0D	0D	00	0E	0E	0E	00	0F	0F	0F	00	10	10
00000078	10	00	11	11	11	00	12	12	12	00	13	13	13	00	14	14	14	00	15	15	15	00	16	16
00000090	16	00	17	17	17	00	18	18	18	00	19	19	19	00	1A	1A	1A	00	1B	1B	1B	00	1C	1C
000000A8	1C	00	1D	1D	1D	00	1E	1E	1E	00	1F	1F	1F	00	20	20	20	00	21	21	21	00	22	22
000000C0	22	00	23	23	23	00	24	24	24	00	25	25	25	00	26	26	26	00	27	27	27	00	28	28
000000D8	28	00	29	29	29	00	2A	2A	2A	00	2B	2B	2B	00	2C	2C	2C	00	2D	2D	2D	00	2E	2E
000000F0	2E	00	2F	2F	2F	00	30	30	30	00	31	31	31	00	32	32	32	00	33	33	33	00	34	34
00000108	34	00	35	35	35	00	36	36	36	00	37	37	37	00	38	38	38	00	39	39	39	00	3A	3A
00000120	3A	00	3B	3B	3B	00	3C	3C	3C	00	3D	3D	3D	00	3E	3E	3E	00	3F	3F	3F	00	40	40
00000138	40	00	41	41	41	00	42	42	42	00	43	43	43	00	44	44	44	00	45	45	45	00	46	46
00000150	46	00	47	47	47	00	48	48	48	00	49	49	49	00	4A	4A	4A	00	4B	4B	4B	00	4C	4C
00000168	4C	00	4D	4D	4D	00	4E	4E	4E	00	4F	4F	4F	00	50	50	50	00	51	51	51	00	52	52
00000180	52	00	53	53	53	00	54	54	54	00	55	55	55	00	56	56	56	00	57	57	57	00	58	58
00000198	58	00	59	59	59	00	5A	5A	5A	00	5B	5B	5B	00	5C	5C	5C	00	5D	5D	5D	00	5E	5E
000001B0	5E	00	5F	5F	5F	00	60	60	60	00	61	61	61	00	62	62	62	00	63	63	63	00	64	64
000001C8	64	00	65	65	65	00	66	66	66	00	67	67	67	00	68	68	68	00	69	69	69	00	6A	6A
000001E0	6A	00	6B	6B	6B	00	6C	6C	6C	00	6D	6D	6D	00	6E	6E	6E	00	6F	6F	6F	00	70	70

利用 winhex 可以得到 bmp 的文件头，之后对文件头进行分析，之后便可以从更深处了 BMP 格式的文件。一个 bmp 图片最多由 4 大部分组成：BITMAPFILEHEADER 结构体，BITMAPINFOHEADER 结构体，RGBQUAD 结构体(这个结构体可以有，也可以没有)，DIB 数据区。其中 DIB 意思就是 Device-Independent Bitmap(设备无关位图)。

一个 bmp 文件以 BITMAPFILEHEADER 结构体开始，

typedef struct tagBITMAPFILEHEADER {

WORD bfType;//固定为 0x4d42;

DWORD bfSize; //文件大小

WORD bfReserved1; //保留字，不考虑

WORD bfReserved2; //保留字，同上

DWORD bfOffBits; //实际位图数据的偏移字节数，即前三个部分长度之和

} BITMAPFILEHEADER;

BITMAPFILEHEADER 的第 1 个属性是 bfType(2 字节)，这里恒定等于 0x4D42。由于内存中的数据排列高位在左，低位在右，所以内存中从左往右看就显示成(42 4D)，所以在 winhex 中头两个 字节显示为(42 4D)就是这样形成的，以后的数据都是这个特点，不再作重复说明。

BITMAPFILEHEADER 的第 2 个属性是 bfSize(4 字节)，表示整个 bmp 文件的大小。BITMAPFILEHEADER 的第 3 个、第 4 个属性分别是 bfReserved1、bfReserved2(各 2 字节)，这里是 2 个保留属性，都为 0，这里等于&H0000、0x0000。

BITMAPFILEHEADER 的第 5 个属性是 bfOffBits(4 字节)，表示 DIB 数据区在 bmp 文件中的位置偏移量，比如等于 0x00000076=118，表示数据区从文件开始往后数的 118 字节开始。头 424D 一般指示文件的类型，由 16 进制转化为字符串可以得到 BM 指示该文件为图片 BMP 文件。之后的 0040E6 为文件的具体大小。

2 图像灰度降低级数

2.1 实验原理

通过读取文件 lena.bmp 可以发现其是 256 灰度值的图片，是一个 8 位灰度级数的图片于是可以找到灰度值与灰度级数的对应关系：

灰度级数	1	2	3	4	5	6	7	8
灰度值	2	4	8	16	32	64	128	256

表 1　灰度级数与灰度值对应表

然而对应图片各个像素坐标而言降低灰度值意味着要使得原图片的每一个像素点均要降低所对应需要的像素级数对应灰度值，具体转换公式如下：

$$pixel_{m,n\in\Omega}(x,y)=floor(\frac{pixel_{m\in\Omega}(x,y)}{2^{m-n}}) \tag{1}$$

$pixel(x,y)$ 对应像素点原本的灰度值，Ω 集合为灰度级数所可以包含的集合，灰度值为 0-255 对应灰度级数 1-8，而 m 对应为原来图片的灰度级数，n 则对应为索要降低为的灰度级数，$floor()$ 为向下取整函数，公式(1)即可实现灰度值降低的功能。

2.2 实验方法

通过实验原理构造出来灰度降低函数，传入参数分别为输入的图片以及希望其降低至多少的灰度级数，默认 reduce_level 必须在 1-8 之间(尽管也有异常处理)，对 i,j 进行循环赋予新图片矩阵像素灰度，即可实现。

```
def change_gray_level(str,reduce_level):
    lena = Image.open(str)
    lena_array=np.array(lena)
    [x,y]=np.shape(lena_array)
    if reduce_level<0:
        reduce_level=0
    if reduce_level>8:
        reduce_level=8
    else:pass
    real_reduce=pow(2,reduce_level) #实际降低灰度的值
    tran=np.zeros([x,y])
    result=np.zeros([x,y])
    for i in range(0,x-1):
        for j in range(0,y-1):
            tran[i,j]= floor(lena_array[i,j] / real_reduce)#除以实际降低灰度值向下取整
            result[i,j]=tran[i,j]*real_reduce#再乘实际灰度值
    print(result)
    data = Image.fromarray(result)
    data.show()
```

2.3 实验结果

图 1　lena.bmp 实现 1-8 级灰度递减

可以发现 5-8 级灰度级并没有很明显的区别，肉眼无法分辨其与原图片之间的不同。4 级图片有了小部分不连续的地方，有了一点差异，灰度级在 1-3 的图片可以明显发现具体的不同，灰度区分明显，实验结果成功。由此也可以得到结论，灰度级数越高越与真实图片逼近，越真实。

3 图像均值与方差

3.1 实验原理

对于一张分辨率(图片大小)为 $m \times n$ 的图片来说。图片的均值与方差的计算公式如下：

$$m = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} pixel(i, j) \tag{2}$$

$$\sigma^2 = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (pixel(i, j) - m)^2 \tag{3}$$

```
import numpy as np
from PIL import Image
lena = Image.open("C:\\Users\\lenovo\\Desktop\\1\\lena.bmp")
lena_array=np.array(lena)
sum1=lena_array.sum() #sum
array_square=lena_array*lena_array     #square
[X,Y]=np.shape(lena_array)
N=X*Y
mean=sum1/N
lena_array2=(lena_array-mean)*(lena_array-mean)
sum2=lena_array2.sum()
var=sum2/N
print(mean,var)
```

3.2 实验结果

计算图像 lena.bmp 可以得到：

图像均值计算为 99.05121612548828；

图像方差计算为 2796.0318388835876；

由次可以得出：图像的均值可反应图像整体的明暗程度，而方差可以反应图像整体的对比度情况。

4 图像扩展

4.1 实验原理

三种方法进行实现：

1. 临近插值：

插值法的第一次都是相同的，计算新图的坐标点对应原图中哪个坐标点来填充，计算公式为：

$$srcX = dstX \times (srcWidth / dstWidth) \tag{4}$$

$$srcY = dstY \times (srcWidth / dstWidth) \tag{5}$$

相当于横纵坐标乘放缩比，对于临近插值而言，处理浮点数的方法就是四舍五入选取最接近的整数。这样的做法就会导致像素的变化不连续，在图像中的体现就是会有锯齿就是产生棋盘格现象。即临近插值公式改进为：

$$srcX = round(dstX \times (srcWidth / dstWidth)) \tag{6}$$

$$srcY = round(dstY \times (srcWidth / dstWidth)) \tag{7}$$

$round()$ 函数即为取整运算。

2. 双线性插值

双线性就是利用与坐标轴平行的两条直线去把小数坐标分解到相邻的四个整数坐标点的和，权重为距离

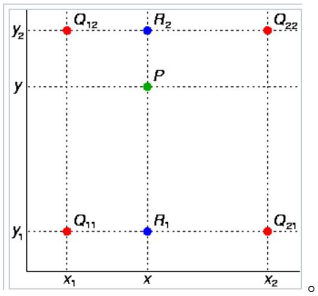


图 2　算法图解

相当于每次转化一个点，通过它周围四个点(包括自身)利用与小数部分的加权组合从而得到图像转化后的坐标。

3. 双三次插值

构造 BiCubic 函数，获得转换像素之间的关系：

$$W(x)=\begin{cases}(a+2)|x|^3-(a+3)|x|^2+1 & \text{for } |x|\le 1\\ a|x|^3-5a|x|^2+8a|x|-4a & \text{for } 1<|x|<2\\ 0 & \text{otherwise}\end{cases}$$

$$f(x,y)=\sum_{i=0}^{3}\sum_{j=0}^{3}f(x_i,y_j)W(x-x_i)W(y-y_j)$$

临近插值　　双线性插值　　双三次插值

图 3 lena.bmp 实现三种不同方式 zoom 至 2048*2048

由图 3 可清楚看到，经过最邻近插值后的图像产生的“棋盘格”效应，而后两种插值方法表现的效果比较平滑，令人满意。由于在编写代码中存在许多循环，算法时间复杂度不低，双三次插值大概运行时间为 3min，双线性插值为 1min，最临近插值最快。可见双三次插值的时间开销最大，双线性其次，最邻近最小。

5 图像错切与图像旋转

5.1 实验原理

根据数字图像处理课本的原理进行编程

5.2 实验结果

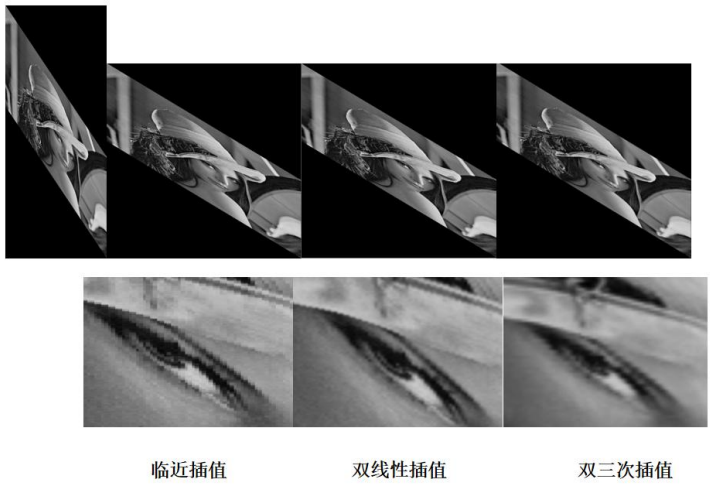


图 4 lena.bmp 实现 shear

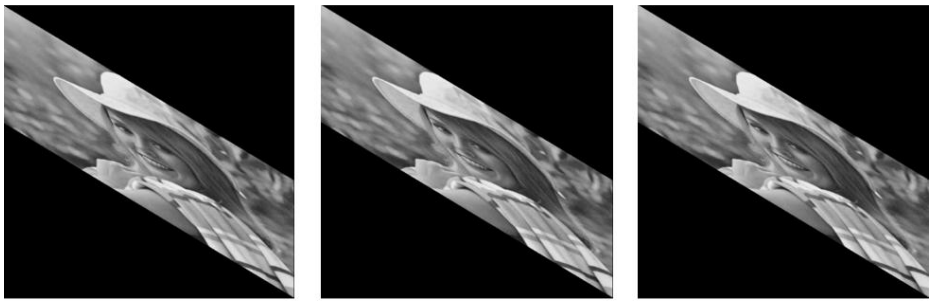
上图也能显示三种不同插值方法的区别。为了方便之后不再进行细节对比。

图 5 lena.bmp 实现 rotate

在看到旋转操作的时候惊奇的发现图片之中会存在一些小孔如下图：

图 6 旋转所产生的“蜂窝”现象

产生蜂窝现象的主要原因是进行坐标变换的时候，通过正弦和余弦函数所计算出来的坐标要进行取整运算，从而会有一些点没有映射完全，由于正弦函数和余弦函数的周期性从而决定了图像旋转后会产生蜂窝现象。



临近插值　　双线性插值　　双三次插值

图 7 elain1.bmp 图片进行错切

临近插值　　双线性插值　　双三次插值

图 8 elain1.bmp 图片进行旋转

可以发现进行旋转的时候也出现了蜂窝现象。

6 参考文献

数字图像处理 [美]冈萨雷斯 电子工业出版社

7 代码见附录 txt