**实验报告**

专业：计算机科学与技术

日期：2022/11/15

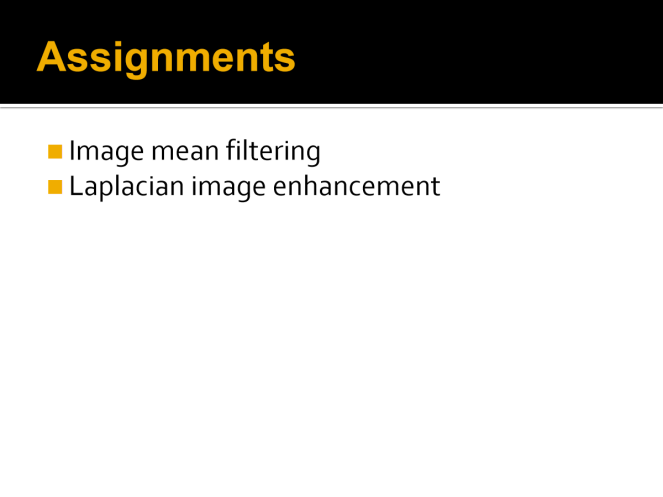
课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： 均值滤波与拉普拉斯增强

**一、实验目的和要求**

完成下述实验任务：

1. 对图像进行均值滤波操作
2. （拓展）对图像进行高斯滤波操作
3. 对图像进行拉普拉斯增强

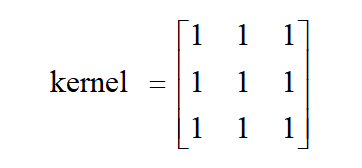


**二、实验内容和原理**

.bmp图像信息的贮存，往往有固定的格式，用二进制的方式打开bmp文件，可以设置相对映的数据结构接受文件头部信息、数据头部信息以及palette信息与bitmap data，从而进行相应的数据操作；本实验选用RGB模式24位色（1080\*1080像素）图像文件进行均值（高斯）滤波及拉普拉斯增强操作，且在实验过程中，以对灰度值进行操作充当对图像像素值数据进行相应的操作。

[均值滤波](https://so.csdn.net/so/search?q=%E5%9D%87%E5%80%BC%E6%BB%A4%E6%B3%A2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_51571728/article/details/_blank)是典型的线性滤波算法，是指用当前[像素](https://so.csdn.net/so/search?q=%E5%83%8F%E7%B4%A0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_51571728/article/details/_blank)点周围 n\*n 个像素值的均值来代替当前像素值。使用该方法遍历处理图像内的每一个像素点，可完成整幅图像的均值滤波，在实际的操作过程中，因为边际的确定性，无法框定正常的 n\*n 矩阵范围，因而不可对边际区域直接使用均值滤波操作，但可以使用类似滤波操作的方法，使替换像素值的位置不再是中心，而是 n\*n矩阵中的指定位置。

利用均值滤波，可以对图像进行去噪操作，均值滤波所使用的卷积核 3\*3 样例如下：

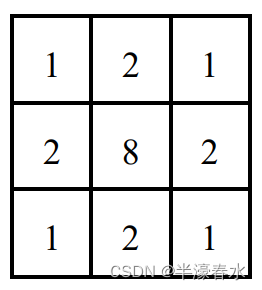


使用均值滤波在去噪的过程中，在有效消去部分噪音的同时，也会对原始图像产生模糊化的效果，是一些细节边缘丧失，因而有使用局限性。

此时可以考虑使用高斯滤波的方法，高斯滤波也是一种线性平滑滤波，适用于消除噪声，广泛应用于图像处理的减噪过程。通俗而言，高斯滤波就是对整幅图像进行[加权平均](https://baike.baidu.com/item/%E5%8A%A0%E6%9D%83%E5%B9%B3%E5%9D%87/9702101?fromModule=lemma_inlink" \o "加权平均)的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。

高斯滤波的具体操作：用一卷积核扫描图像中的每一个像素，将确定的邻域内像素的加权平均灰度值替代模板中心像素点的值。

高斯滤波所使用的卷积核 3\*3 样例如下：



上述两种滤波方法，均用于平滑图像，消除噪声。而拉普拉斯增强，可以锐化图像，使图像边界感增强，更具立体性。

拉普拉斯算子是最简单的各向同性微分算子，它具有旋转不变性。一个二维图像函数的拉普拉斯变换是各向同性的二阶导数，其定义为：

IMG_256

使用拉普拉斯算子，产生的图像将是在暗背景上的一些灰色边线和一些突变点。若将原始图像叠加到拉普拉斯变换后的图像，既可以保护拉普拉斯锐化处理的效果，同时又能复原背景信息，使图像锐化。

1. **实验步骤与分析**
2. 建立存储图像信息的数据结构（头文件）与bmp附带信息基本一一对应。

typedef unsigned char bit;

typedef unsigned short word;

typedef unsigned long dword;

struct tagBITMAPFILEHEADER

{

word bftype; //文件类型

dword bfsize; //文件大小（字节为单位）

word bfreserved1; //保留，必须设为0

word bfreserved2; //保留，必须设为0

dword bfoffbits; //到实际图像实际开始的偏移量

};

struct tagBITMAPINFOHEADER

{

dword bisize; //结构所需字节数

long biwidth; //图像宽度（像素为单位）

long biheight; //图像长度（像素为单位）--正值对映图像倒立

word biplanes; //位面数，常为1

word bibitcounts; //比特/像素比

dword bicompression; //压缩类型

dword bisizeimage; //图像大小

long bixpelspermeter; //水平分辨率

long biypelspermeter; //垂直分辨率

dword biclrused; //颜色索引数

dword biclrimportant; //重要影响颜色索引数

};

1. 声明本实验所需使用函数（具体解析见下）

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, bit\* rgbpic, long int number,

const char name[]);

//YUV转换RGB函数

void change(long int picheight, long int picwidth, float\* newy, float\* yuvpic, bit\* rgbpic1);

//均值滤波函数

void meanchange(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic,

tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, const char name[]);

//拉普拉斯增强函数

void laplacian(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic,

tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information);

//高斯滤波函数

void gauss(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic);

1. 二进制打开.bmp文件，读取文件内数据信息，文件指针偏移至bitmap data数据区域，记录bitmap data区域对映像素索引矩阵。

// 定义图片信息接收变量

tagBITMAPFILEHEADER file;

tagBITMAPINFOHEADER information;

//定义图像长宽

long int picwidth;

long int picheight;

//定义循环指数

long int i;

//提取bmp文件头部信息

FILE\* fp;

fp = fopen("lena.bmp", "rb");

fread(&(file.bftype), 1, sizeof(word), fp);

fread(&(file.bfsize), 1, sizeof(dword), fp);

fread(&(file.bfreserved1), 1, sizeof(word), fp);

fread(&(file.bfreserved2), 1, sizeof(word), fp);

fread(&(file.bfoffbits), 1, sizeof(dword), fp);

fread((&information), sizeof(tagBITMAPINFOHEADER), 1, fp);

//获取bmp大小像素信息

picwidth = information.biwidth;

picheight = information.biheight;

//接收bmp图像数据信息

bit\* rgbpic = (bit\*)malloc(picwidth \* picheight \* 3 \* sizeof(bit));

fseek(fp, file.bfoffbits, SEEK\_SET);

fread(rgbpic, sizeof(bit), picwidth \* picheight \* 3, fp);

1. 将图像从 RGB 格式转化为 YUV格式，便于后续均值（高斯）滤波与拉普拉斯增强的操作。

//利用公式将BGR类型转换为YUV类型

for (i = 0;i < picwidth \* picheight;i++)

{

yuvpic[3 \* i] = 0.11 \* rgbpic[3 \* i] + 0.59 \* rgbpic[3 \* i + 1] + 0.3 \* rgbpic[3 \* i + 2];

yuvpic[3 \* i + 1] = 0.493 \* (rgbpic[3 \* i] - yuvpic[3 \* i]);

yuvpic[3 \* i + 2] = 0.877 \* (rgbpic[3 \* i + 2] - yuvpic[3 \* i]);

}

1. 进行均值滤波操作

//进行均值滤波

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = yuvpic[3 \* i];

//第一次 5\*5 卷积

meanchange(picheight, picwidth, newy, ypic, yuvpic, file, information, "mean fliter first.bmp");

//更新bmp图像当前状态

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = newy[i];

//第二次 5\*5 卷积

meanchange(picheight, picwidth, newy, ypic, yuvpic, file, information, "mean fliter second.bmp");

//更新bmp图像状态

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = newy[i];

//第三次 5\*5 卷积

meanchange(picheight, picwidth, newy, ypic, yuvpic, file, information, "mean fliter third.bmp");

5.1均值滤波函数

输入值：1.图像的总列像素值

2.图像的总行像素值

3.新图像的灰度值

4.图像原有的灰度值

5.图像原有的 YUV 存贮值

6.原图像file 头信息

7.原图像 information 头信息

8.新输出图像的标题

输出值：无

//均值滤波函数

void meanchange(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic,

tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, const char name[])

{

long int i, j, k;

//赋予背景灰度值

for (i = 0;i < picwidth \* picheight;i++)

newy[i] = ypic[i];

long int index;

float sum;

//进行（n-2）\*（n-2）的像素遍历与赋值

//采用5\*5卷积核大小

for (i = 2;i < picheight - 2;i++)

for (j = 2;j < picwidth - 2;j++)

{

//测试程序bug时使用

//printf("%d\n", j);

//计算当前像素位置

index = i \* picwidth + j;

//重置求和基数

sum = 0;

//对中心像素进行5\*5大小的卷积

//系数占比均为1

for (k = -2;k <= 2;k++)

sum = sum + ypic[index + k \* picwidth - 1] + ypic[index + k \* picwidth]

+ypic[index + k \* picwidth + 1]+ ypic[index + k \* picwidth - 2]

+ ypic[index + k \* picwidth + 2];

//新灰度值归一化

newy[index] = round(sum / 25);

//规定新灰度值范围

if (newy[index] > 255)

newy[index] = 255;

else if (newy[index] < 0)

newy[index] = 0;

}

//对边缘像素进行不完全均值滤波操作

//对顶行进行 2\*3 矩阵均值滤波

for (index = 1;index < 2 \* picwidth - 1;index++)

{

//跳过最左最右端像素

if (index == picwidth || index == picwidth - 1);

else

{

sum = 0;

sum = ypic[index - 1] + ypic[index] + ypic[index + 1] + ypic[index + picwidth - 1]

+ ypic[index + picwidth + 1] + ypic[index + picwidth];

newy[index] = sum / 6;

}

}

//对底行进行 2\*3 矩阵均值滤波

for (index = (picheight - 2) \* picwidth + 1;index < picheight \* picwidth - 1;index++)

{

//跳过最左最右端像素

if (index == picwidth \* (picheight - 1) || index == (picheight - 1) \* picwidth - 1);

else

{

sum = 0;

sum = ypic[index - 1] + ypic[index] + ypic[index + 1] + ypic[index - picwidth - 1]

+ ypic[index - picwidth + 1] + ypic[index - picwidth];

newy[index] = sum / 6;

}

}

//对最左列进行 2\*3 矩阵滤波

for (index = 1;index < picheight - 1;index++)

{

//跳过最上最下端像素

for (j = 0;j <= 1;j++)

{

sum = 0;

sum = ypic[index \* picwidth - picwidth + j] + ypic[index \* picwidth + j]

+ ypic[index \* picwidth + 1 + j]+ ypic[index \* picwidth - picwidth + 1 + j]

+ ypic[index \* picwidth + picwidth + 1 + j] + ypic[index \* picwidth + picwidth + j];

newy[index] = sum / 6;

}

}

//对最右列进行 2\*3 矩阵滤波

for (index = 2;index < picheight;index++)

{

//跳过最上最下端像素

for (j = 0;j <= 1;j++)

{

sum = 0;

sum = ypic[index \* picwidth - 1 - picwidth - j] + ypic[index \* picwidth - 1 - j]

+ ypic[index \* picwidth - 1 - 1 - j]+ ypic[index \* picwidth - 1 - picwidth - 1 - j]

+ ypic[index \* picwidth - 1 + picwidth - 1 - j]

+ ypic[index \* picwidth + picwidth - 1 - j];

newy[index] = sum / 6;

}

}

bit\* rgbpic1 = (bit\*)malloc(picheight \* picwidth \* sizeof(bit) \* 3);

change(picheight, picwidth, newy, yuvpic, rgbpic1);

write(file, information, rgbpic1, picheight \* picwidth, name);

free(rgbpic1);

}

1. 进行高斯滤波操作（拓展）

//高斯滤波（拓展部分，采用 3\*3 多次卷积，比较效果）

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = yuvpic[3 \* i];

gauss(picheight, picwidth, newy, ypic, yuvpic);

//以共十次卷积为例

for (j = 0;j <= 8;j++)

{

//更新图像灰度值状态

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = newy[i];

//进行高斯滤波

gauss(picheight, picwidth, newy, ypic, yuvpic);

}

bit\* rgbpic1 = (bit\*)malloc(picheight \* picwidth \* sizeof(bit) \* 3);

//将YUV格式转换为RGB格式

change(picheight, picwidth, newy, yuvpic, rgbpic1);

//写入bmp图像文件

write(file, information, rgbpic1, picheight \* picwidth, " gauss fliter.bmp");

6.1高斯滤波函数

输入值：1.图像的总列像素值

2.图像的总行像素值

3.新图像的灰度值

4.原图像的灰度值

5.原图像的 YUV 存贮值

输出值：无

//高斯滤波函数

void gauss(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic)

{

long int i, j, k;

//赋予背景灰度值

for (i = 0;i < picwidth \* picheight;i++)

newy[i] = ypic[i];

long int index;

float sum;

//进行（n-1）\*（n-1）的像素遍历与赋值

for (i = 1;i < picheight - 1;i++)

for (j = 1;j < picwidth - 1;j++)

{

//计算当前像素位置

index = i \* picwidth + j;

//重置求和基数

sum = 0;

//根据3\*3矩阵不同位置加权求取均值

for (k = -1;k <= 1;k++)

{

if (k == 0)

//若处于矩阵中间行，则加权 2 4 2

sum = sum + 2 \* ypic[index - 1] + 4 \* ypic[index] + 2 \* ypic[index + 1];

else

//若处于矩阵首尾行，则加权 1 2 1

sum = sum + ypic[index + k \* picwidth - 1] + 2 \* ypic[index + k \* picwidth] + ypic[index + k \* picwidth + 1];

}

//新灰度值归一化

newy[index] = round(sum / 16);

//规定新灰度值合法范围

if (newy[index] > 255)

newy[index] = 255;

else if (newy[index] < 0)

newy[index] = 0;

}

}

1. 进行拉普拉斯增强

//拉普拉斯增强

for (i = 0;i < picwidth \* picheight;i++)

ypic[i] = yuvpic[3 \* i];

//进行拉普拉斯增强变换

laplacian(picheight, picwidth, newy, ypic, yuvpic, file, information);// "laplacian add.bmp")

7.1拉普拉斯增强函数

输入值：1.图像的总列像素值

2.图像的总行像素值

3.新图像的灰度值

4.图像原有的灰度值

5.图像原有的 YUV 存贮值

6.原图像file 头信息

7.原图像 information 头信息

输出值：无

//拉普拉斯增强函数

void laplacian(long int picheight, long int picwidth, float\* newy, float\* ypic, float\* yuvpic,

tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information)

{

long int i,j,k;

int index;

//赋予背景灰度值

for (i = 0;i < picheight \* picwidth;i++)

newy[i] = ypic[i];

float sum;

//进行（n-1）\*（n-1）的像素遍历与赋值

for (i = 1;i < picheight - 1;i++)

for (j = 1;j < picwidth - 1;j++)

{

//确定当前像素位置

index = i \* picwidth + j;

//重置求和基数

sum = 0;

//根据3\*3矩阵不同位置加权计算梯度（采用斜角元素包含方式卷积）

for (k = -1;k <= 1;k++)

{

if (k == 0)

//若为中心行，则赋权 1 -8 1

sum = sum + ypic[index - 1] - 8 \* ypic[index] + ypic[index + 1];

else

//若为首尾行，则赋权 1 1 1

sum = sum + ypic[index + k \* picwidth - 1] + ypic[index + k \* picwidth]

+ ypic[index + k \* picwidth + 1];

}

//更新灰度值预期缩减值

newy[index] = sum;

//规定新灰度值合法范围

if (newy[index] > 255)

newy[index] = 255;

else if (newy[index] < 0)

newy[index] = 0;

}

//直接输出拉普拉斯算子图像效果

bit\* rgbpic1 = (bit\*)malloc(picheight \* picwidth \* sizeof(bit) \* 3);

change(picheight, picwidth, newy, yuvpic, rgbpic1);

write(file, information, rgbpic1, picheight \* picwidth, " laplacian operator.bmp");

//更新原图像灰度值，增加/减少相应的拉普拉斯算子

for (i = 0;i < picheight \* picwidth;i++)

{

//根据输出效果判断拉普拉斯算子的运算符号

ypic[i] = ypic[i] - newy[i];

//ypic[i] = ypic[i] + newy[i];

//规定更新灰度值范围

if (ypic[i] < 0)

ypic[i] = 0;

}

change(picheight, picwidth, ypic, yuvpic, rgbpic1);

write(file, information, rgbpic1, picheight \* picwidth, " laplacian enhance.bmp");

//对拉普拉斯算子进行调整，是图像锐化更明显

//增大拉普拉斯算子

for (i = 0;i < picheight \* picwidth;i++)

{

//乘以自定义系数

newy[i] = newy[i] \* 1.2;

ypic[i] = ypic[i] - newy[i];

//规定灰度值输出范围

if (ypic[i] < 0)

ypic[i] = 0;

}

change(picheight, picwidth, ypic, yuvpic, rgbpic1);

write(file, information, rgbpic1, picheight \* picwidth, " laplacian enhance1.bmp");

//减少拉普拉斯算子

for (i = 0;i < picheight \* picwidth;i++)

{

//除以自定义系数

newy[i] = newy[i] / 1.2;

ypic[i] = ypic[i] - newy[i];

//规定灰度值输出范围

if (ypic[i] < 0)

ypic[i] = 0;

}

change(picheight, picwidth, ypic, yuvpic, rgbpic1);

write(file, information, rgbpic1, picheight \* picwidth, " laplacian enhance2.bmp");

free(rgbpic1);

}

1. YUV 转化为 RGB 函数

输入值：1.原图像的总列像素个数

2.原图像的总行像素个数

3.新图像的灰度值

4.原图像的灰度值

5.更新后 RGB 值的贮存区域

输出值：无

//YUV转换为RGB

void change(long int picheight, long int picwidth, float\* newy, float\* yuvpic, bit\* rgbpic1)

{

//将YUV格式转化为RGB格式

//若YUV相应转换值大于255，则保留255

//若YUV相应转换值小于0，则保留0

int i;

for (i = 0; i < picwidth \* picheight; i++)

{

//YUV转化B

if (newy[i] + 2.0284 \* yuvpic[3 \* i + 1] < 0)

rgbpic1[3 \* i] = 0;

else if (newy[i] + 2.0284 \* yuvpic[3 \* i + 1] > 255)

rgbpic1[3 \* i] = 255;

else

rgbpic1[3 \* i] = newy[i] + 2.0284 \* yuvpic[3 \* i + 1];

//YUV转化G

if (newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2] < 0)

rgbpic1[3 \* i + 1] = 0;

else if (newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2] > 255)

rgbpic1[3 \* i + 1] = 255;

else

rgbpic1[3 \* i + 1] = newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2];

//YUV转化R

if (newy[i] + 1.1403 \* yuvpic[3 \* i + 2] < 0)

rgbpic1[3 \* i + 2] = 0;

else if (newy[i] + 1.1403 \* yuvpic[3 \* i + 2] > 255)

rgbpic1[3 \* i + 2] = 255;

else

rgbpic1[3 \* i + 2] = newy[i] + 1.1403 \* yuvpic[3 \* i + 2];

}

}

9.写入函数

输入值：1.bmp文件信息头文件

2.bmp文件数据头文件

3.bmp文件各像素RGB数值记录表

4.bmp文件总像素大小（行像素总数乘以列像素总数）

5.bmp文件名称

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, bit\* rgbpic, long int number, const char name[])

{

FILE\* fp;

//打开/新建文件

fp = fopen(name, "wb");

//写入bmp图像各类数据

fwrite(&(file.bftype), 1, sizeof(file.bftype), fp);

fwrite(&(file.bfsize), 1, sizeof(file.bfsize), fp);

fwrite(&(file.bfreserved1), 1, sizeof(file.bfreserved1), fp);

fwrite(&(file.bfreserved2), 1, sizeof(file.bfreserved2), fp);

fwrite(&(file.bfoffbits), 1, sizeof(file.bfoffbits), fp);

fwrite(&information, 1, sizeof(information), fp);

fwrite(rgbpic, sizeof(bit), number \* 3, fp);

//关闭bmp图像文件

fclose(fp);

}

**四、实验环境及运行方法**

用C语言编写，vs正常编译环境运行，其中需要以下语句忽略vs对fopen函数的安全性中断。

#pragma warning(disable:4996) //忽略vs对fopen的安全性错误

对于实验结果，点击运行即可。

输入图像为 24位色（1080\*1080像素）RGB 格式图像，共输出八张新生成的bmp图片。 其中 gauss fliter.bmp 为高斯滤波处理结果。

lapacian enhance.bmp 为拉普拉斯增强处理（原始）结果，lapacian enhance1.bmp 与 lapacian enhance2.bmp 为拉普拉斯算子优化后（分别对映为乘以一自定义常数与除以一自定义常数）图像锐化处理结果，lapacian operator.bmp 为拉普拉斯算子图像化输出结果。

mean fliter first.bmp 为一次均值滤波处理结果，mean fliter second.bmp 为两次均值滤波处理结果，mean fliter third.bmp 为三次均值滤波处理结果（用于相互间效果比较），可以关注 lena 紫色头发部分的细节变化，发现随着均值滤波次数的增加，细节逐渐被抹去，头发图像部分变得平滑。

**五、实验结果展示**

（可以重点关注 lena 的紫色头发区域，锐化或平滑操作在此区域效果十分明显）



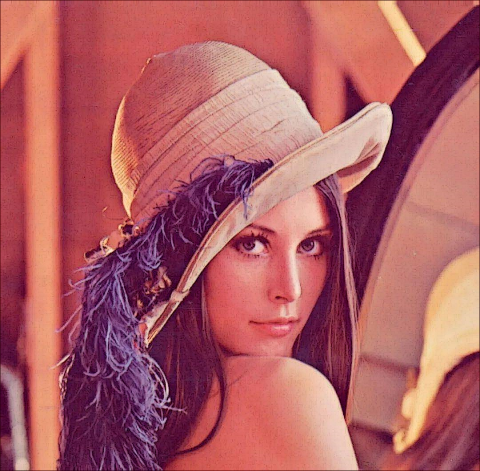
一次（5\*5）均值滤波输出结果 两次（5\*5）均值滤波输出结果



十次（5\*5）均值滤波输出结果

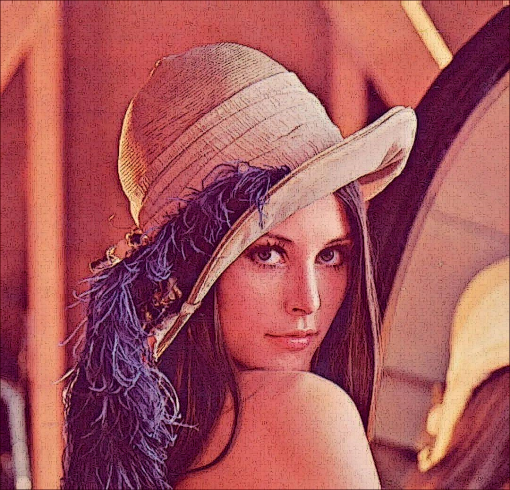


十次（3\*3）高斯滤波输出结果



拉普拉斯算子图像结果 拉普拉斯增强输出结果

拉普拉斯算子优化后增强操作输出结果：



1. **心得体会**

没想到第一次接触卷积核，尽然是在图像信息处理的课程之中，当宋老师刚提及这个概念时，心中只有惊讶于恐惧，卷积核这一概念，仿佛高高在上，常与神经网络关联，像我这样的编程小白，居然可接触使用，不可思议。

当然，在经过宋老师的讲解与PPT的学习后，我渐渐理解了卷积核这一数学过程，其实基础的知识并没有我想象的那么复杂，且在图像处理的使用中，卷积核更多的是被视为一块模板，去覆盖一定区域的数值，取均值，取加权平均，或对区块自定义赋值以达到某种数学等式的关系；从此看来，均值滤波或者高斯滤波，亦或是拉普拉斯增强，其本质都是对一定范围内的像素值进行加权运算（权可以完全相等），从而进行像素处理，得到宏观的结果，因而三个函数在使用方面也可以互相借鉴。

通过本实验的学习，逐渐掌握了图像朦胧感的创造与精细化的实现，又离使用C语言处理 bmp图像的小目标更近一步，真是受益匪浅。