

# 浙江大学实验报告

专业：计算机科学与技术  
姓名：沈一芄  
学号：322010827  
日期：2023/12/7

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图像增强

## 1. 实验目的和要求

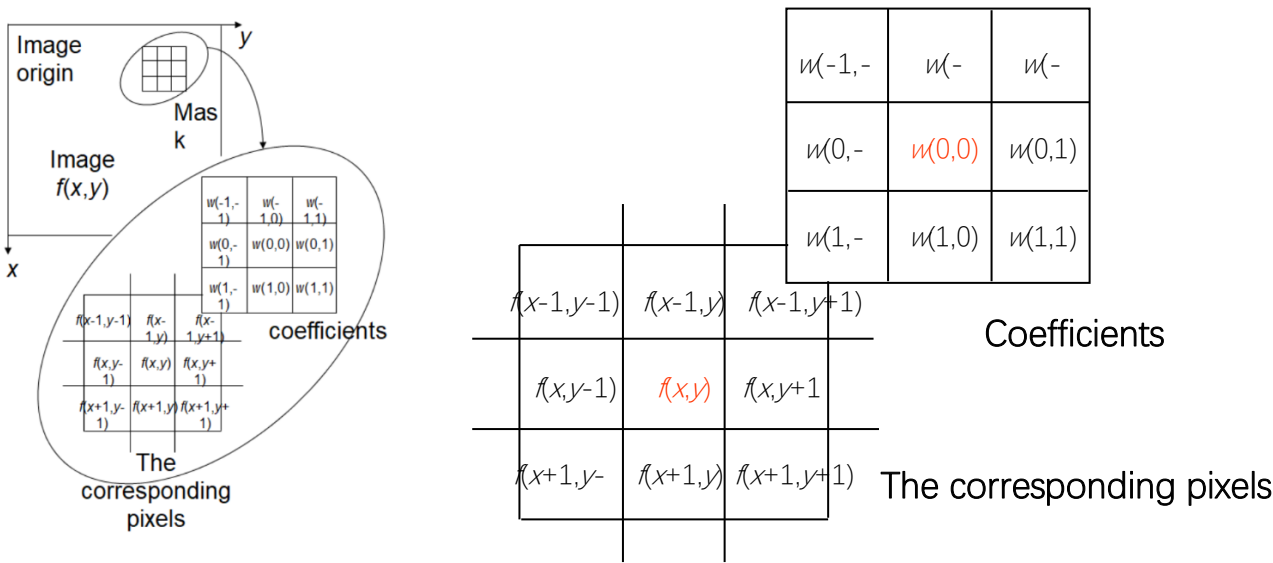
### 1.1. 实验目的

- 实现均值滤波，对图像进行模糊处理。
- 实现拉普拉斯图像增强，对图像进行锐化处理。

## 2. 实验内容和原理

### 2.1. 滤波

滤波器是一个大小为  $M \times N$  的窗口。对于不同的滤波方式，令窗口中的元素与原图像的处于窗口内的像素进行相对应的运算，得到的结果作为新图像的一个像素（一般为滤波窗口中心像素点在原图像中重合位置处的像素）。当窗口滑过原图像并完成上述运算之后，就能够得到一幅新图像。



故可将滤波看作以  $x$  为中心进行原像素点与滤波窗口对应像素点运算后结果求和：

$$R = w(-1, -1)f(x-1, y-1) + w(-1, 0)f(x-1, y) + \cdots \\ + w(0, 0)f(x, y) + \cdots + w(1, 0)f(x+1, y) + w(1, 1)f(x+1, y+1)$$

从而也可将滤波看成一种卷积运算

$$\text{Convolution: } g(x) = f(x) * h(x) = \frac{1}{M} \sum_{t=0}^{M-1} f(t)h(x-t)$$

通常，掩膜（滤波窗口）的长宽均为奇数。设长  $\text{length} = 2a + 1$ ，宽  $\text{width} = 2b + 1$ 。当滤波窗口的中心像素  $(a, b)$  处于像素  $(x, y)$  处时，对掩膜和对应位置像素进行相应运算，得到结果为像素点  $(x, y)$  的新像素值。掩膜滑动处理整张图片，得到新图像  $g$ 。

## 2.2. 平滑滤波

平滑处理可以消除噪点和模糊，同时在重点关注大目标时去除无关紧要的小细节。因此，平滑滤波常被应用于图像预处理阶段。

常见的平滑滤波有线性平滑滤波和统计排序滤波。对于前者，其实现原理即讲滤波掩膜邻域内像素的平均值作为输出。左图展示的是简单平均，即滤波窗口中每一个像素的贡献是一样的。右图则是加权平均，滤波窗口中每一个像素有不同的贡献大小。而在计算时，要将贡献度求和并将其倒数作为系数。

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \quad \frac{1}{16} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

其表达式可以写成如下形式，其中滤波器大小为  $(2a+1) \times (2b+1)$ ， $w$  为滤波器， $f$  为输入图像， $g$  为输出图像：

$$g(x, y) = \frac{\sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x+s, y+t)}{\sum_{s=-a}^a \sum_{t=-b}^b w(s, t)}$$

从性质上讲，图像的平滑效果与滤波掩膜的大小有直接关系。掩膜 size 越大，模糊程度越高。从而可以通过选择掩膜与实际图像的相对大小来得到期望的处理效果，如过滤、筛选一定大小的斑点/物体。

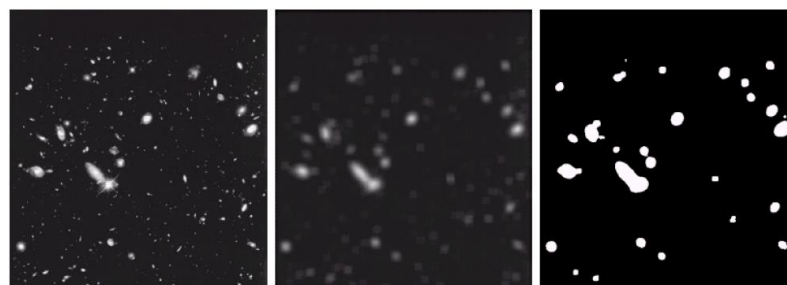
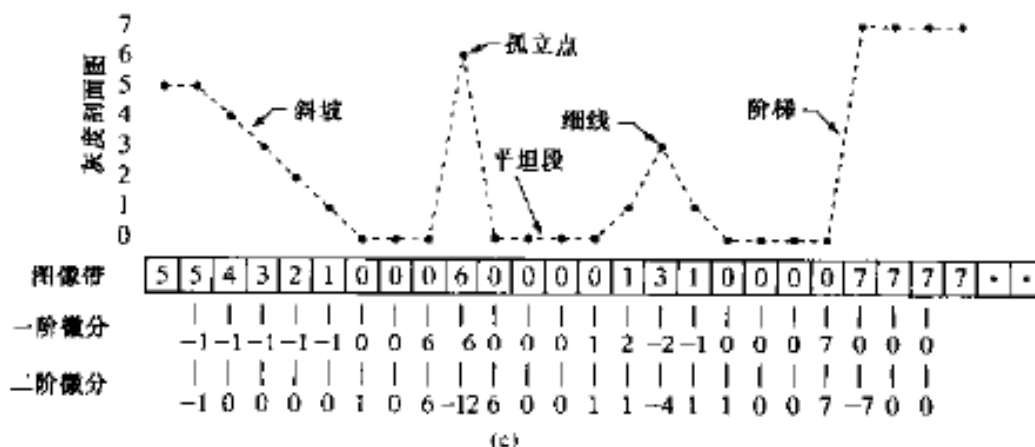


FIGURE 3.36 (a) Image from the Hubble Space Telescope. (b) Image processed by a  $15 \times 15$  averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)

## 2.3. 微分算子

微分算子是实现锐化的工具，其响应程度与图像在该点处的突变程度有关。微分算子增强了边缘和其他突变（如噪声）并削弱了灰度变化缓慢的区域。下图为一例，将某一图片沿一条水平线做灰度剖面图，对像素进行一阶、二阶微分处理后得到更突出的图像。



而在进行二阶微分操作时，对函数  $f(x, y)$ ，定义如下二维列向量。则拉普拉斯算子的定义给出为：

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

直接带入方程可得到各向同性、旋转不变的拉普拉斯算子：

$$\nabla^2 f = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y)$$

若将对角线元素也纳入考量来设计掩膜：

$$\nabla^2 f = \sum_{i=-1}^1 \sum_{j=-1}^1 f(x+i, y+j) - 9f(x, y)$$

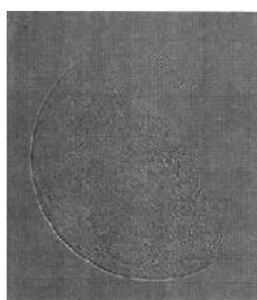
得到的掩膜如下图所示，左侧未考虑对角线元素，右侧考虑对角线元素。

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

通过将原始图像和拉普拉斯图像叠加在一起，可以保护拉普拉斯锐化处理的效果，同时复原背景信息。具体操作与掩膜中心系数的正负有关，如下图所示：

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & \text{如果拉普拉斯掩模中心系数为负} \\ f(x, y) + \nabla^2 f(x, y) & \text{如果拉普拉斯掩模中心系数为正} \end{cases}$$



(a)Original image

(b)Laplacian result

(c)Rearranged Laplacian result

(d) After fusion

### 3. 实验步骤与分析

#### 3.1. 准备工作

- 定义结构体并声明变量如下；每个变量对应一张 BMP 图片：

```
typedef struct{
    BMFH bmfh;
    BMIH bmih;
    RGBQUAD rgbquad[256];
    BYTE *data;
    int LineBytes, WidthBMP, HeightBMP, SizeImageBMP;
}IMAGE;
IMAGE input, gray, spatial_filtering, laplacian_filtering;
```

- 定义平滑中值滤波器如下，可选择不同 size 与加权方式：

```
typedef struct{
    int size=5, coef_sum=0, filter_num;
    vector<int> coef[4]={
        {1,1,1, 1,1,1, 1,1,1}, //only for size=3*3
        {1,5,1, 5,1,5, 1,5,1}, //only for size=3*3
        {1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1}, //only 5*5
        {1,5,1,5,1, 5,1,5,1,5, 1,5,1,5,1, 5,1,5,1,5, 1,5,1,5,1} //only 5*5
    };
    Coord* pos;
}Filter_spatial;
```

- 定义拉普拉斯算子滤波如下，可选择是否考量对角线元素及中心系数正负：

```
typedef struct{
    int size=3, filter_num;
    vector<int> coef[4]={0,1,0, 1,-4,1, 0,1,0},
        {0,-1,0, -1,4,-1, 0,-1,0},
        {1,1,1, 1,-8,1, 1,1,1},
        {-1,-1,-1, -1,8,-1, -1,-1,-1}};
    Coord* pos;
}Filter_laplacian;
```

## 3.2. 均值滤波

### 3.2.1. 基于灰度图的均值滤波

- 若图像为灰度图，则直接进行均值滤波并输出。
- 若图像为彩图，将图像另存为灰度图，对灰度图进行均值滤波，将新的 Y 与原图的 U, V 转换回 R, G, B, 得到新的图像。这种方法只依据亮度上进行平滑模糊处理。

```
• void Do_spatial_filtering1(IMAGE *input, IMAGE *output){
•     IMAGE SFgray;
•     if(input->bmih.biBitCount!=8) color_to_gray(input, &SFgray);
•     else CopyGray(input, &SFgray);
•
•     Filter_spatial SpatialFilter;
•     SpatialFilter.size=5;
•     SpatialFilter.filter_num=2;
•     SpatialFilter.pos = new Coord[SpatialFilter.size*SpatialFilter.size];
•     for(int i=0;i<SpatialFilter.size;i++){
•         for(int j=0;j<SpatialFilter.size;j++){
•             SpatialFilter.pos[i*SpatialFilter.size+j].x = i-SpatialFilter.size/2;
•             SpatialFilter.pos[i*SpatialFilter.size+j].y = j-SpatialFilter.size/2;
•         }
•     }
•     for(int i=0;i<SpatialFilter.coef[SpatialFilter.filter_num].size();i++){
•         SpatialFilter.coef_sum+=SpatialFilter.coef[SpatialFilter.filter_num][i];
•         cout << SpatialFilter.coef_sum << endl;
•     }
•     for(int i=0;i<SFgray.HeightBMP;i++){
•         for(int j=0;j<SFgray.WidthBMP;j++){
•             int ori_pixel, new_pixel=0;
•             Coord temp;
•             for(int k=0;k<SpatialFilter.size*SpatialFilter.size;k++){
•                 temp.x=i+SpatialFilter.pos[k].x;
•                 temp.y=j+SpatialFilter.pos[k].y;
•                 if(temp.x<0||temp.y<0||temp.x>=SFgray.HeightBMP||temp.y>=SFgray.WidthBMP)
•                     ori_pixel=0;
•                 else{
•                     if(input->bmih.biBitCount==8)
•                         ori_pixel=input->data[temp.x*SFgray.LineBytes+temp.y];
•                     else
•                         ori_pixel=input->data[temp.x*input->LineBytes+temp.y*3]*0.299
+input->data[temp.x*input->LineBytes+temp.y*3+1]*0.587+input->data[temp.x*input->LineByte
s+temp.y*3+2]*0.114;
•                 }
•             }
•         }
•     }
• }
```

```

•         new_pixel+=SpatialFilter.coef[SpatialFilter.filter_num][k]*ori_pixel;
•     }
•     new_pixel/=SpatialFilter.coef_sum;
•     SFgray.data[i*SFgray.LineBytes+j]=new_pixel;
• }
• }
•
•     if(input->bmih.biBitCount==8){
•         CopyGray(&SFgray, output);
•     }else{
•         CopyColor(input, output);
•         for(int i=0;i<output->HeightBMP;i++){
•             for(int j=0;j<output->WidthBMP;j++){
•                 double R, G, B, Y, U, V;
•                 B=input->data[input->LineBytes*i+j*3];           //store in BGR or
der read B,G,R
•                 G=input->data[input->LineBytes*i+j*3+1];
•                 R=input->data[input->LineBytes*i+j*3+2];
•                 U=-0.147*R-0.289*G+0.435*B;
•                 V=0.615*R-0.515*G-0.100*B;
•                 Y=SFgray.data[SFgray.LineBytes*i+j];
•
•                 R=Y+1.4075*V;           //calculate R,G,B
•                 G=Y-0.3455*U-0.7169*V;
•                 B=Y+1.779*U;
•
•                 if(R>255)R=255;
•                 if(R<0)R=0;
•                 if(G>255)G=255;
•                 if(G<0)G=0;
•                 if(B>255)B=255;
•                 if(B<0)B=0;
•
•                 output->data[output->LineBytes*i+j*3]=B;       //write in data
•                 output->data[output->LineBytes*i+j*3+1]=G;
•                 output->data[output->LineBytes*i+j*3+2]=R;
•             }
•         }
•     }
• }

```

### 3.2.2. 基于 RGB 的均值滤波

- 若图像为灰度图，则直接进行均值滤波并输出。
- 若图像为彩图，则另存为后对 R, G, B 三个通道分别进行均值滤波，最后将新的 R, G, B 合并组成新的图像。这种方法依据单独的 R, G, B 进行平滑模糊处理后进行叠加。

```
• void Do_spatial_filtering2(IMAGE *input, IMAGE *output){
•
•     if(input->bmih.biBitCount!=8)    CopyColor(input, output);
•     else    CopyGray(input, output);
•
•     Filter_spatial SpatialFilter;
•     SpatialFilter.size=5;
•     SpatialFilter.filter_num=2;
•     SpatialFilter.pos = new Coord[SpatialFilter.size*SpatialFilter.size];
•     for(int i=0;i<SpatialFilter.size;i++){
•         for(int j=0;j<SpatialFilter.size;j++){
•             SpatialFilter.pos[i*SpatialFilter.size+j].x = i-SpatialFilter.size/2;
•             SpatialFilter.pos[i*SpatialFilter.size+j].y = j-SpatialFilter.size/2;
•         }
•     }
•
•     for(int i=0;i<SpatialFilter.coef[SpatialFilter.filter_num].size();i++){
•         SpatialFilter.coef_sum+=SpatialFilter.coef[SpatialFilter.filter_num][i];
•         cout << SpatialFilter.coef_sum << endl;
•     }
•
•     for(int i=0;i<output->HeightBMP;i++){
•         for(int j=0;j<output->WidthBMP;j++){
•             int ori_pixel, tempx, tempy;
•             int B=0, G=0, R=0, Y=0;
•
•             for(int k=0;k<SpatialFilter.size*SpatialFilter.size;k++){
•                 tempx=i+SpatialFilter.pos[k].x;
•                 tempy=j+SpatialFilter.pos[k].y;
•                 if(tempx<0||tempy<0||tempx>=output->HeightBMP||tempy>=output->WidthBMP)
•                     continue;
•                 if(input->bmih.biBitCount==8)
•                     Y+=input->data[tempx*output->LineBytes+tempy]*SpatialFilter.coef[SpatialFilter.filter_num][k];
•                 else{
•                     //cout << SpatialFilter.coef[SpatialFilter.filter_num][k] << endl;
•                     B+=input->data[tempx*output->LineBytes+tempy*3+0]*SpatialFilter.coef[
SpatialFilter.filter_num][k];
```



```

•           G+=input->data[tempx*output->LineBytes+tempy*3+1]*SpatialFilter.coef[
SpatialFilter.filter_num][k];
•           R+=input->data[tempx*output->LineBytes+tempy*3+2]*SpatialFilter.coef[
SpatialFilter.filter_num][k];
•       }
•   }
•
•       Y/=SpatialFilter.coef_sum;
•       B/=SpatialFilter.coef_sum;
•       G/=SpatialFilter.coef_sum;
•       R/=SpatialFilter.coef_sum;
•       if(Y<0) Y=0;   if(B<0) B=0;   if(G<0) G=0;   if(R<0) R=0;
•       if(Y>255) Y=255; if(B>255) B=255; if(G>255) G=255; if(R>255) R=255;
•
•       if(input->bmih.biBitCount==8)
•           output->data[i*input->LineBytes+j]=Y;
•       else{
•           output->data[i*input->LineBytes+j*3+0]=B;
•           output->data[i*input->LineBytes+j*3+1]=G;
•           output->data[i*input->LineBytes+j*3+2]=R;
•       }
•
•   }
• }
•
• }

```

### 3.3. 拉普拉斯算子增强图像

- 原理与分通道实现均值滤波基本相同。
- 更改掩膜为拉普拉斯算子掩膜。
- 对图像进行处理后映射回[0, 255]的范围。
- 对图像叠加后进行处理。

```
• void Do_laplacian_filtering(IMAGE *input, IMAGE *output){  
•  
•     if(input->bmih.biBitCount!=8)    CopyColor(input, output);  
•     else    CopyGray(input, output);  
•  
•  
•  
•     int c;  
•     Filter_laplacian LaplacianFilter;  
•     LaplacianFilter.size=3;  
•     LaplacianFilter.filter_num=2;  
•     LaplacianFilter.pos = new Coord[LaplacianFilter.size*LaplacianFilter.size];  
•     for(int i=0;i<LaplacianFilter.size;i++){  
•         for(int j=0;j<LaplacianFilter.size;j++){  
•             LaplacianFilter.pos[i*LaplacianFilter.size+j].x = i-LaplacianFilter.size/2;  
•             LaplacianFilter.pos[i*LaplacianFilter.size+j].y = j-LaplacianFilter.size/2;  
•         }  
•     if(LaplacianFilter.filter_num==0||LaplacianFilter.filter_num==2)    c=-1;  
•     else    c=1;  
•  
•     for(int i=0;i<output->HeightBMP;i++){  
•         for(int j=0;j<output->WidthBMP;j++){  
•             int ori_pixel, tempx, tempy;  
•             int B=0, G=0, R=0, Y=0;  
•  
•             for(int k=0;k<LaplacianFilter.size*LaplacianFilter.size;k++){  
•                 tempx=i+LaplacianFilter.pos[k].x;  
•                 tempy=j+LaplacianFilter.pos[k].y;  
•                 if(tempx<0||tempy<0||tempx>=output->HeightBMP||tempy>=output->WidthBMP) c  
•                 continue;  
•                 if(input->bmih.biBitCount==8)  
•                     Y+=input->data[tempx*output->LineBytes+tempy]*LaplacianFilter.coef[LaplacianFilter.filter_num][k];  
•                 else{  
•                     //cout << LaplacianFilter.coef[LaplacianFilter.filter_num][k] << endl  
•  
•                 }  
•             }  
•         }  
•     }  
• }
```

```

•         B+=input->data[tempx*output->LineBytes+tempy*3+0]*LaplacianFilter.coe
f[LaplacianFilter.filter_num][k];
•         G+=input->data[tempx*output->LineBytes+tempy*3+1]*LaplacianFilter.coe
f[LaplacianFilter.filter_num][k];
•         R+=input->data[tempx*output->LineBytes+tempy*3+2]*LaplacianFilter.coe
f[LaplacianFilter.filter_num][k];
•     }
• }
•
•     if(input->bmih.biBitCount==8)
•         output->data[i*input->LineBytes+j]=Y;
•     else{
•         output->data[i*input->LineBytes+j*3+0]=B;
•         output->data[i*input->LineBytes+j*3+1]=G;
•         output->data[i*input->LineBytes+j*3+2]=R;
•     }
•
•     Y+=c*input->data[i*input->LineBytes+j];
•     B+=c*input->data[i*input->LineBytes+j*3+0];
•     G+=c*input->data[i*input->LineBytes+j*3+1];
•     R+=c*input->data[i*input->LineBytes+j*3+2];
•
•     //if(Y<0) Y*=-1;  if(B<0) B*=-1;  if(G<0) G*=-1;  if(R<0) R*=-1;
•     if(Y<0)  Y=0; if(B<0) B=0; if(G<0) G=0; if(R<0) R=0;
•     if(Y>255) Y=255; if(B>255) B=255; if(G>255) G=255; if(R>255) R=255;
•
•     if(input->bmih.biBitCount==8)
•         output->data[i*input->LineBytes+j]=Y;
•     else{
•         output->data[i*input->LineBytes+j*3+0]=B;
•         output->data[i*input->LineBytes+j*3+1]=G;
•         output->data[i*input->LineBytes+j*3+2]=R;
•     }
• }
• }
•
• }

```

4. 实验结果展示

4. 1. 平滑均值滤波

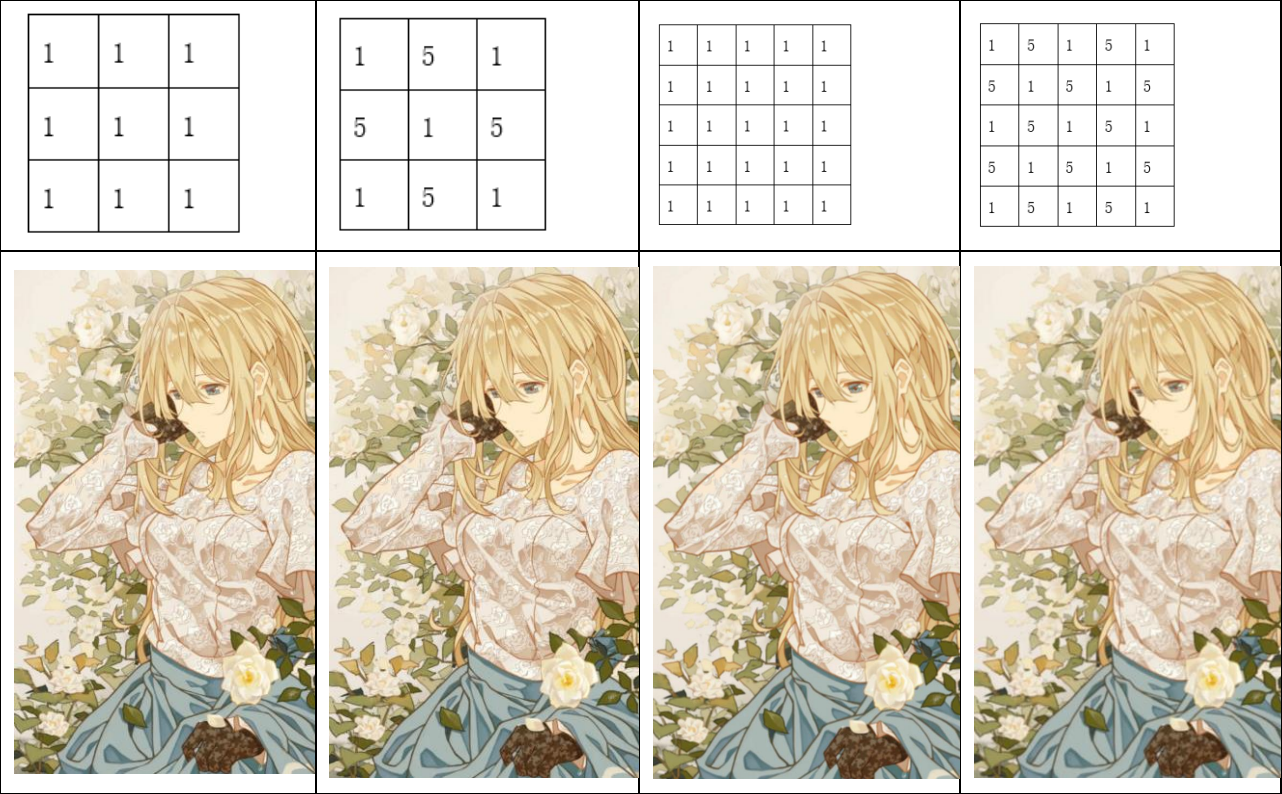
使用两种算法分别进行均值滤波，掩膜采取  $\text{size} = 5 \times 5$ ，得到结果如下图所示。

观察发现，两种算法都能有效模糊图像。但基于灰度图的滤波会对原图的颜色产生影响，整体色调偏粉。故而该方法并不只是对亮度作模糊处理。而将 RGB 三个通道分开进行均值滤波，最后再叠加的做法则在模糊的同时暴露了原图的色彩。

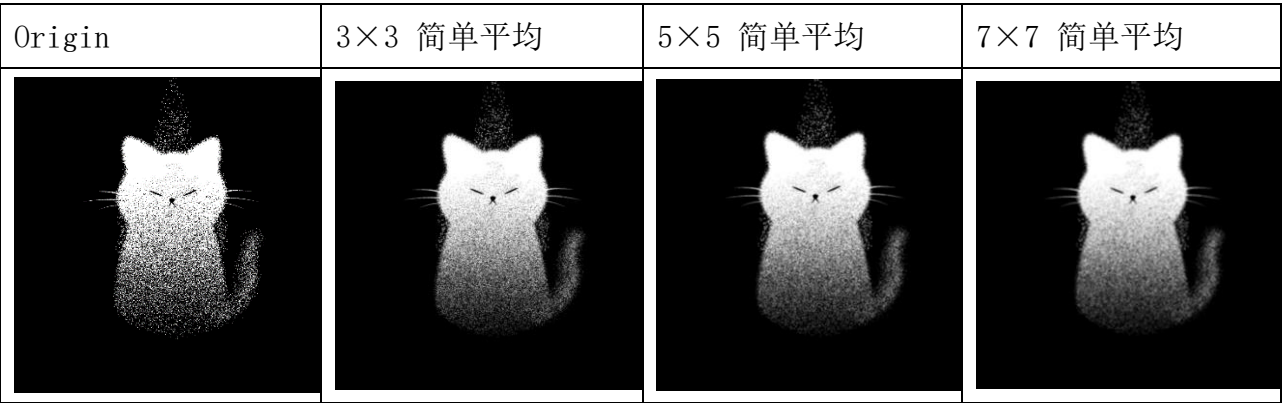
<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table> <div>原图</div>	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	<div>算法 1:</div> <div>基于灰度图均值滤波</div>	<div>算法 2:</div> <div>基于 RGB 均值滤波</div>
1	1	1	1	1																							
1	1	1	1	1																							
1	1	1	1	1																							
1	1	1	1	1																							
1	1	1	1	1																							
																											

更改掩膜 size，并更改加权方式，得到效果如下所示：

观察发现，size = 3 × 3 的模糊效果明显逊于 size = 5 × 5 的效果。而在简单平均和加权平均之间，区别较小。对于此图片，加权平均在脸部处理上相较简单平均亮度稍高。



而对于噪点，则有如下去除表现：




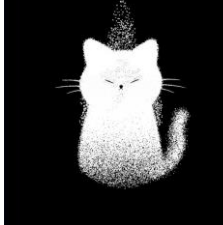
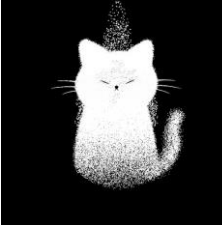


可以明显发现在图像逐渐模糊的同时，主体（脸部）信息被保留，而由离散度较大的点组成的下半身、轮廓和尾巴越来越不明显。




4.2. 拉普拉斯增强

采用四种滤波器对图像进行拉普拉斯增强：

通过观察可以发现，拉普拉斯算子对图像进行了较强锐化。且考虑对角线的掩膜设计会对图像有更强的锐化处理（体现于面部极少黑色噪点的突出）。而中心系数的正负对图像处理的结果无影响。

Origin	<table><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>-4</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr></table>	0	1	0	1	-4	1	0	1	0	<table><tr><td>0</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>4</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>0</td></tr></table>	0	-1	0	-1	4	-1	0	-1	0	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>-8</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	-8	1	1	1	1	<table><tr><td>-1</td><td>-1</td><td>-1</td></tr><tr><td>-1</td><td>8</td><td>-1</td></tr><tr><td>-1</td><td>-1</td><td>-1</td></tr></table>	-1	-1	-1	-1	8	-1	-1	-1	-1
0	1	0																																						
1	-4	1																																						
0	1	0																																						
0	-1	0																																						
-1	4	-1																																						
0	-1	0																																						
1	1	1																																						
1	-8	1																																						
1	1	1																																						
-1	-1	-1																																						
-1	8	-1																																						
-1	-1	-1																																						
																																								

Origin										
由此组则可明显发现，拉普拉斯算子在考虑对角线元素时对图像的锐化增强要强于不考虑对角线元素的掩膜设计。人物轮廓能被很好地勾勒。										
<table><tr><td>0</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>4</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>0</td></tr></table>	0	-1	0	-1	4	-1	0	-1	0	
0	-1	0								
-1	4	-1								
0	-1	0								
<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>-8</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	-8	1	1	1	1	
1	1	1								
1	-8	1								
1	1	1								

## 5. 心得体会

通过这次实验我了解了卷积以及滤波的原理，并实现了均值滤波与拉普拉斯图像增强。

对于均值滤波，最为直观的感受是 size 对滤波效果的影响。此处的模糊处理让人联想到前几次实验中的腐蚀操作。而固定 size 后对均值权重的赋值也很有意思，不同的设计方法会导致图片有较小的不同。

对于拉普拉斯算子，若要输出拉普拉斯算子图像，需要注意图像的数据定义为 BYTE，typedef 前为 unsigned char，无符号且取值  $\in [0, 255]$ 。故当拉普拉斯算子计算出现负值时，需要 +255 来映射回 BYTE 的范围内输出。在掩膜的设计上，中心系数的正负对结果没有影响，而是否将对角线元素纳入掩膜设计则会使图像的锐化程度有不同的体现；纳入考量后图像锐度更高。

本次实验还是有较多售后。希望能够在以后的实验中用到本次所学的滤波和拉普拉斯图像增强来预处理图片，为后续提供便利。

2023.12.7