山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

学号：201900130151	姓名： 莫甫龙	
实验题目：图像滤波		

实验过程中遇到和解决的问题：

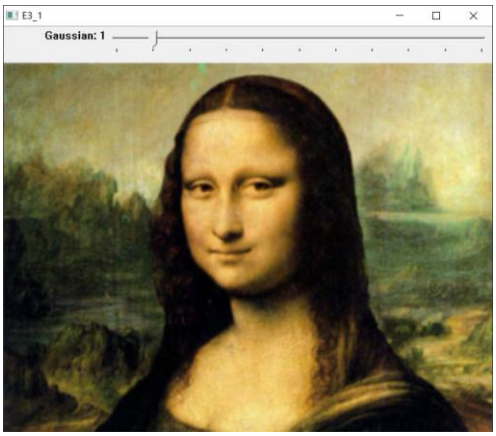
（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

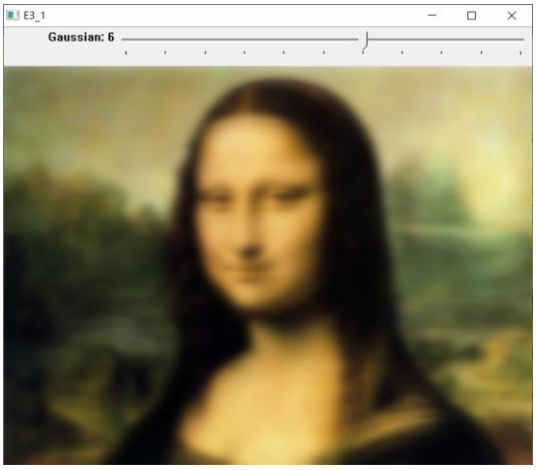
3_1 高斯滤波：

因为要用两次一维的高斯滤波来求，所以就要先求出高斯滤波的第一行向量和第一列向量，而刚开始要先求行的滤波，所以就要拿行高斯滤波来一点点地位移来求，这一块我最开始遇到的困难是不知道怎么去让原图的像素和高斯滤波所对应的位置一一对应，后面找到了规律，就是高斯滤波上 p 这个位置所对应的是原图上的 p-center+y 这个位置。

而这一块遇到的第二个问题就是最外面的轮廓问题，刚开始我是想直接将没有的地方拿 0 来补全，但是效果并不是很好，因为边缘的位置发黑，后面我尝试了使用拉伸的方式，比如对一张像素为 50*50 的图，对（-1，0）来说，就把它当成（0，0）来处理，对于（2，50）来说，就当成（2，49）来处理。

效果如下：






```
for (int i = 0; i < d; ++i) {//求卷积核
    kernel[i] = exp( _X: (-1 * (i - center) * (i - center)) / (2 * sigma * sigma)) / (2 * 3.14 * sigma * sigma);
    sum += kernel[i];
}
for (int i = 0; i < d; ++i) kernel[i] /= sum;

output = Mat::zeros(input.size(), input.type());
int H = output.rows, W = output.cols;
```

```cpp
for (int x = 0; x < H; ++x) {    //对行进行
    for (int y = 0; y < W; ++y) {
        for (int c = 0; c < 3; ++c) {
            sum = 0;
            for (int p = 0; p < d; ++p) {
                int q = p  - center+y;
                if (q < 0) q = 0;
                if (q >= W) q = W - 1;
                sum += (double) input.at<Vec3b>(x, q)[c] * kernel[p];
            }
            output.at<Vec3b>(x, y)[c] = saturate_cast<uchar>(sum);
        }
    }
}

for (int x = 0; x < H; ++x) {   //对列进行
    for (int y = 0; y < W; ++y) {
        for (int c = 0; c < 3; ++c) {
            sum = 0;
            for (int p = 0; p < d; ++p) {
                int q = p  - center+x;
                if (q < 0) q = 0;
                if (q >= H) q = H - 1;
                sum += (double) output.at<Vec3b>(q, y)[c] * kernel[p];
            }
            output.at<Vec3b>(x, y)[c] = saturate_cast<uchar>(sum);
        }
```

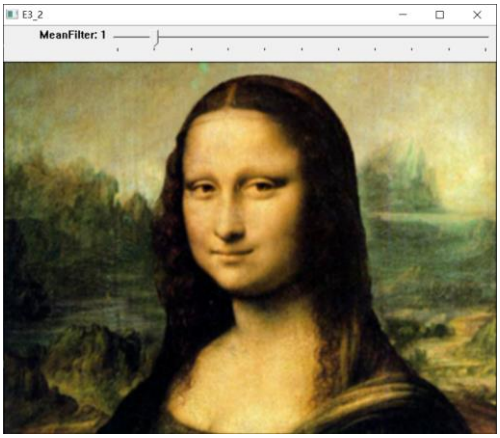
3_2：快速均值滤波：

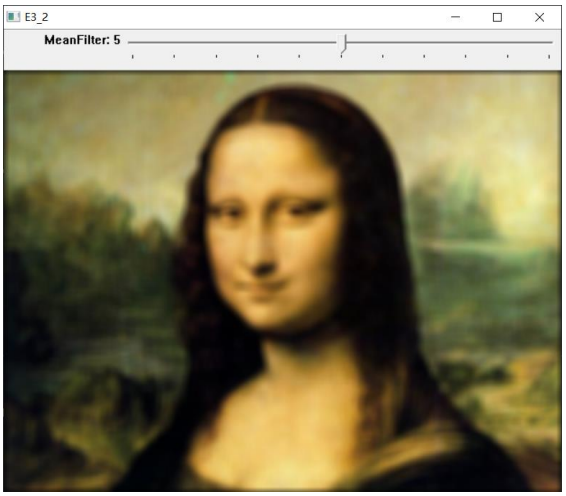
这块只要按照公式套就行

```cpp
for (int c = 0; c < 3; c++) {    //求取积分图
    for (int x = 0; x < H; x++) {
        int sum1 = 0;
        for (int y = 0; y < W; y++) {
            sum1 += input.at<Vec3b>(x, y)[c];
            if (x == 0)
                sum[x][y] = sum1;
            else
                sum[x][y] = sum[x - 1][y] + sum1;
        }
    }
```

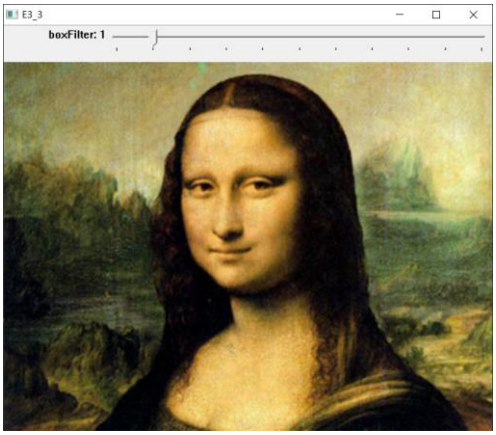
```cpp
int l = 2 * window_size + 1;    //套公式计算
for (int x = 0; x < H; x++) {
    for (int y = 0; y < W; y++) {
        int xt, xb, yl, yr;
        xt = x - window_size - 1;
        xb = x + window_size;
        yl = y - window_size - 1;
        yr = y + window_size;
        if (xt < 0)
            xt = 0;
        if (xb >= H)
            xb = H - 1;
        if (yl < 0)
            yl = 0;
        if (yr >= W)
            yr = W - 1;
        output.at<Vec3b>(x, y)[c] = saturate_cast<uchar>( v: (sum[xb][yr] + sum[xt][yl] - sum[xb][yl] - sum[xt][yr]) / (l * l));
    }
}
```

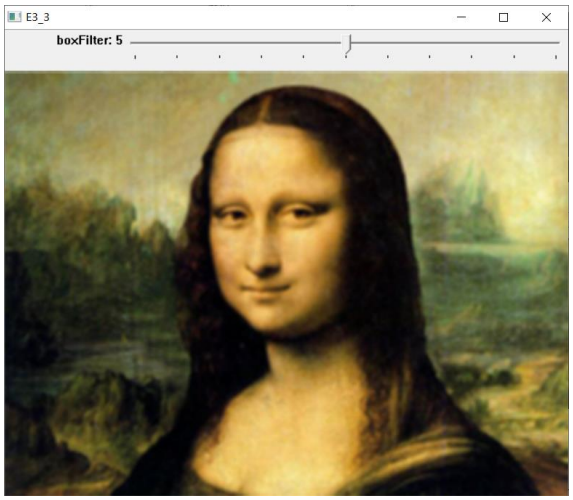
效果图：





BoxFiler 的效果图：

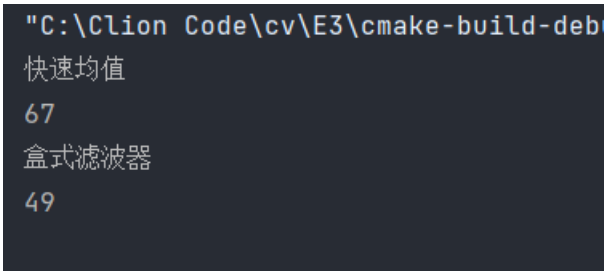




由这两者我们可以看出，对于同一个窗口大小，盒式滤波器没有快速均值滤波明显

结果分析与体会：

对于实验中所提到的问题，两者的运行时间如下：



可以看出，盒式滤波器比快速均值滤波快了很多，这是因为 bosfilter 直接将每个像素的邻域的像素和都存了起来，到时候要用的时候直接取出就行，复杂度为 O(1)。