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：201800130086	姓名： 徐鹏博	
实验题目：图像滤波		

实验过程中遇到和解决的问题：

1.高斯滤波

利用二维高斯函数的行列可分离性

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}=\frac{1}{2\pi\sigma^2}(e^{-\frac{y^2}{2\sigma^2}}+e^{-\frac{y^2}{2\sigma^2}})$$

$$G(x)=\frac{1}{2\pi\sigma^2}e^{-\frac{y^2}{2\sigma^2}}$$

$\frac{1}{2\pi\sigma^2}$ 可以略去,因为最终需要的是窗口内比例和为 1。

所以对一维的滤波窗口而言,只需要计算出每格系数,将所有的 G(x)相加得到 sum,再用 G(x)/sum 归一化即可。另外为了保证窗口大小 D 为奇数，可以进行(6sigma-1)/2*2+1 处理。

```
void Get_Kernel1D(double sigma){
    int d=int(6*sigma-1); D=d/2*2+1;K=D/2; double sum=0;
    for(int i=0;i<D;++i){
        Kernel1D[i] = exp((-1*(i-K)*(i-K))/(2*sigma*sigma));
        sum += Kernel1D[i];
    }
    for(int i=0;i<D;++i)      Kernel1D[i]/=sum;
}
```

可以先进行在行方向上的滤波，对于非边界时的一般数据，计算对应的格和权重值的乘积之和。

边界上，采取对称处理，如果格子在 0 格左边，就取以 0 列为对称轴的对应格的像素值，在 W-1 格右边，就以 W-1 列为对称轴取对应格像素值。

```
for(int c=0;c<outChannels;++c) {
    for (int x = 0; x < H; ++x) {
        for (int y = 0; y < W; ++y) {
            sum = 0;
            for (int p = 0; p < D; ++p) {
                int q = p+y-K;
                if (q < 0) q = -q;
```

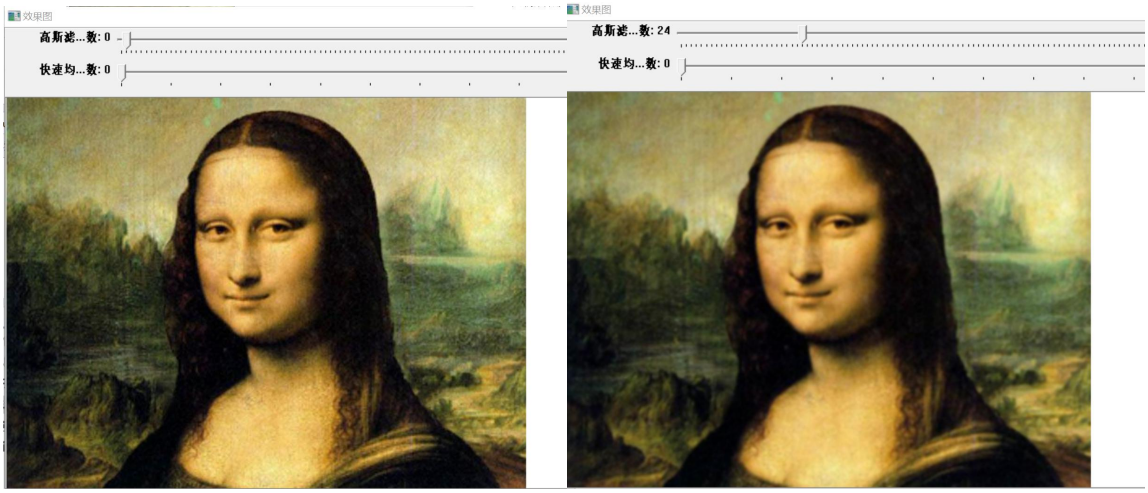
```
                    if (q >= W) q = 2 * W - 1 - q;
                    sum += (double) input.ptr(x, q)[c] * Kernel1D[p];
                }
                tmp.ptr(x, y)[c] = (int) sum;
            }
        }
    }
```

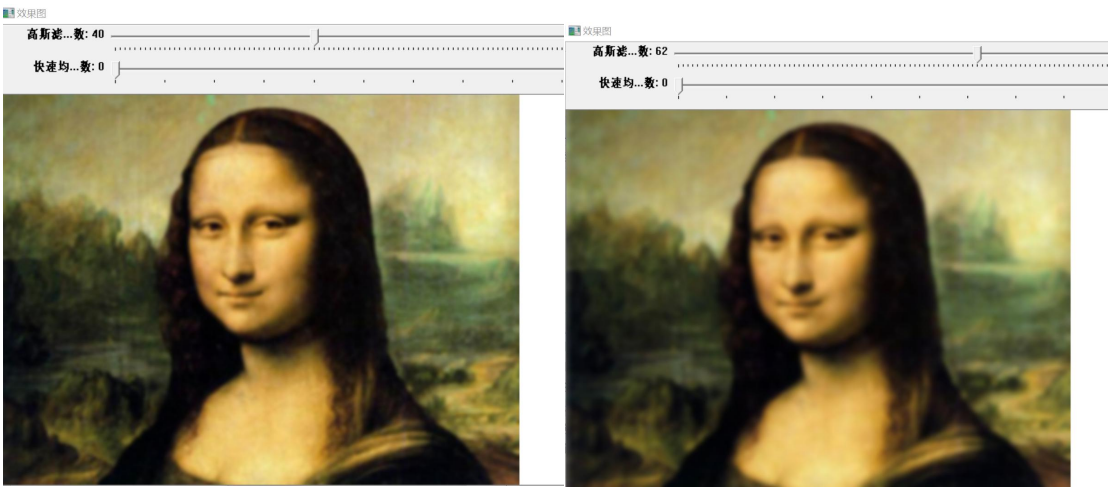
列方向上的滤波同理

```
for(int c=0;c<outChannels;++c) {
    for (int x = 0; x < H; ++x) {
        for (int y = 0; y < W; ++y) {
            sum = 0;
            for (int p = 0; p < D; ++p) {
                int q = p + x - K;
                if (q < 0) q = -q;
                if (q >= H) q = 2 * H - 1 - q;
                sum += (double) tmp.ptr(q, y)[c] * Kernel1D[p];
            }
            output.ptr(x, y)[c] = (int) sum;
        }
    }
}
```

最后的 output 就是输出的效果 Mat。

在滑条中，我对参数进行 value/20.0,这样使得间隔变化能够缓慢一点。下面是效果：





2.快速均值滤波

快速均值滤波采用积分图存储像素值信息，此时取出一个方形区域只需要考虑四个点的积分值即可。如果滤波窗口为 2w+1，Z 为窗口内像素总数，则有：

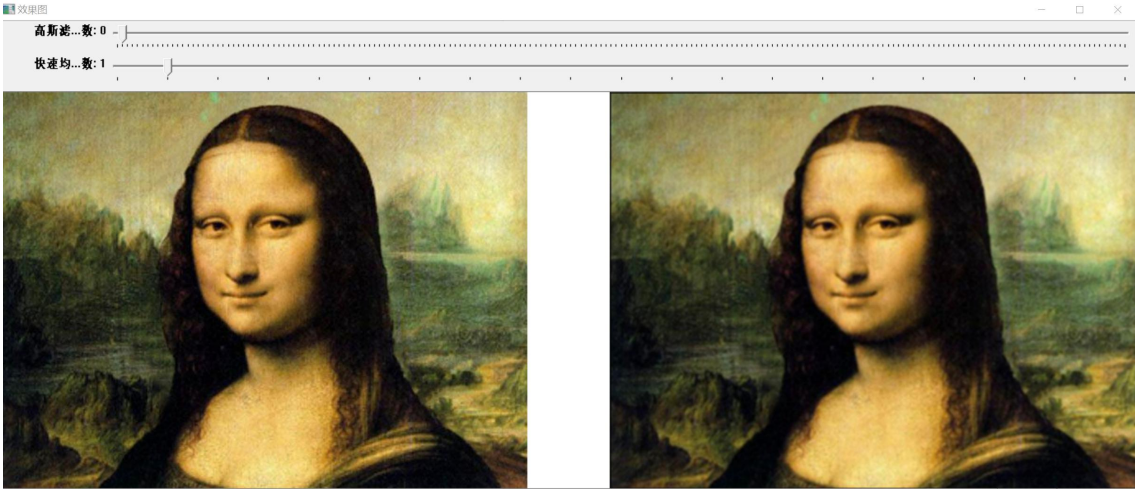
$$O(u,v)=\frac{1}{Z}[S(u+w,v+w)+S(u-w-1,v-w-1)-S(u+w,v-w-1)-S(u-w-1,v+w)]$$

当然对于边界值，也进行对称处理。

```
for (int c = 0; c < outChannels; c++)
{
    //初始化积分图为 0
    for (int x = 0; x < H; x++)
        for (int y = 0; y < W; y++)
            Sum[x][y] = 0;
    //生成积分图
    for (int x = 0; x < H; x++) {
        int tmp = 0;
        for (int y = 0; y < W; y++) {
            tmp += (int) input.ptr(x, y)[c];
            if (x == 0)
                Sum[x][y] = tmp;
            else
                Sum[x][y] = tmp + Sum[x - 1][y];
        }
    }
    //均值滤波
    int x1,x2,y1,y2;
    for (int x = 0; x < H; x++) {
        for (int y = 0; y < W; y++) {
            x1 = x - window_size - 1;
            y1 = y - window_size - 1;
            x2 = x + window_size;
```

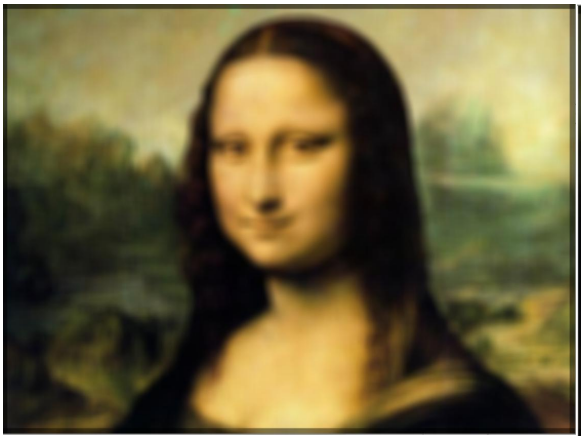
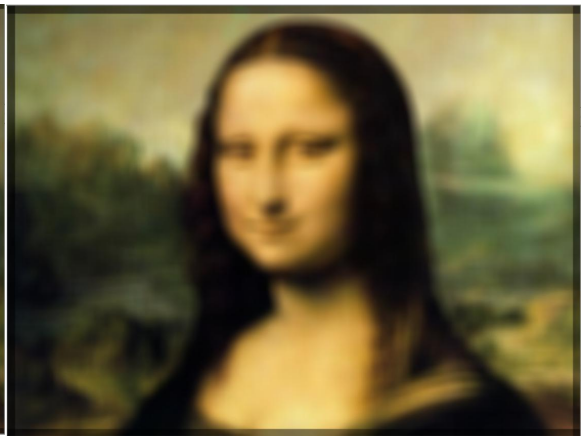
```
                y2 = y + window_size;
                if (x1 < 0)         x1 = x1 + window_size + 1;
                if (y1 < 0)         y1 = y1 + window_size + 1;
                if (x2 >= H)        x2 = x2 - window_size;
                if (y2 >= W)        y2 = y2 - window_size;
                output.ptr(x, y)[c] = (Sum[x1][y1] + Sum[x2][y2] - Sum[x2][y1] - Sum[x1][y2]) / pow(2 * window_size
+ 1, 2);
            }
        }
}
```

效果如下:(左边可以视为原图，右边为快速均值滤波效果图)



w=2,　　w=4

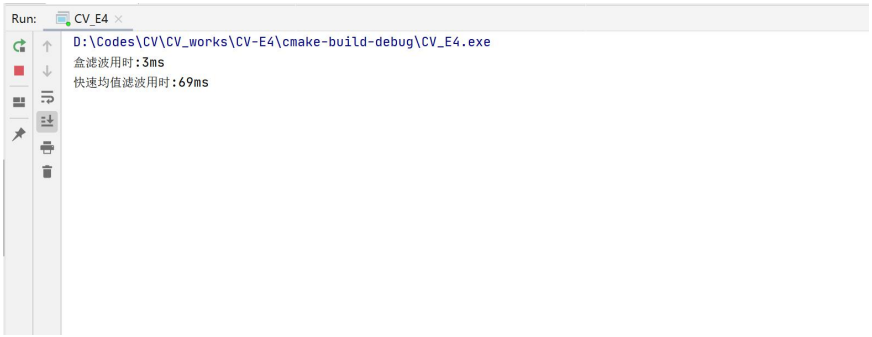
w=6,　　w=8

BoxFilter:

void boxFilter(InputArray src, OutputArray dst, int ddepth, Size ksize, Point anchor=Point(-1,-1), boolnormalize=true, int borderType=BORDER_DEFAULT);

对比时间如下:



结果分析与体会：

高斯滤波和快速均值滤波我都选择用对称处理。

Boxfilter 复杂度比 FastMeanFilter 之所以低那么多，主要是因为在 FastMeanFilter 使用积分图的时候，窗口内的像素和=S[4]+S[1]-S[2]-S[3]，进行加减运算。而 Boxfilter 只需要访问计算结果的数组对应位置。另外 Boxfilter 初始化过程也很迅速，每个矩形的计算基本只需要一加一减，当然初始化过程代码相对比较繁琐。