

计算摄影学作业报告

Lab 2 —— 图像滤波和傅里叶变换

Jessie Peng

2019/03/12

1 实验内容

1.1 空域滤波

实现盒状均值滤波、高斯滤波、中值滤波和简单的双边滤波，具体函数为：

盒状均值滤波：BoxFilter <input-image> <output-image> <w> <h>

高斯滤波：GaussianFilter <input-image> <output-image> <sigma>

中值滤波：MedianFilter <input-image> <output-image> <w> <h>

双边滤波：BilateralFilter <input-image> <output-image> <sigma-s> <sigma-r>

1.2 傅里叶变换

利用傅里叶变换完成图像的频域变换

2 实验环境

编程语言：C++

开发环境：CLion 2018.3

操作系统：macOS 10.14.1 (18B75)

3 实验原理

3.1 均值滤波

均值滤波是以像素邻域内的平均值代替原先的像素：

$$\bar{I}(x, y) = \frac{1}{|N|} \sum_{(u,v) \in N} I(x + u, y + v)$$

引入卷积核后，用空域卷积表示为：

$$\bar{I}(x, y) = \sum_{u,v} K(u, v) I(x + u, y + v)$$

本次实验中，通过手动定义卷积核，然后调用 cv::filter2D 函数可以完成均值滤波。

3.2 高斯滤波

高斯滤波是使用二维高斯核函数代替均值滤波中的卷积核来进行滤波，适用于图像带有高斯噪声的情况下去噪：

$$\bar{I}(x, y) = \sum_{u,v} G(u, v) I(x+u, y+v)$$

本次实验中同样是先定义卷积核，然后调用 `cv::filter2D` 函数完成滤波。

3.3 中值滤波

中值滤波是一种序统计滤波器，使用邻域内像素亮度的中值来取代原本的像素值，适用于过滤椒盐噪声：

$$\bar{I}(x, y) = \text{Median}\{I(x+u, y+v) | (u, v) \in N\}$$

不同于均值滤波和高斯滤波，中值滤波器是非线性的，因此不能通过卷积核的方式来实现。本次实验中，采用最简单的遍历邻域像素取中位数的方式来实现。

3.4 双边滤波

双边滤波提供了一种降噪的同时保持边界（前三种滤波会让边界模糊）的方法，它是在高斯滤波加权平均的基础上引入一个新的项反应亮度差带来的加权：

$$\bar{I}(p) = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I(p) - I(q)|) I(q)$$

其中的权重则是由空间上的高斯加权和灰度距离上的高斯加权组合而成：

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I(p) - I(q)|)$$

本实验中，采用简单的遍历邻域内像素计算权重的方式来实现。

3.5 傅里叶变换

大小为 $W \times H$ 的图像 I 的二维傅里叶变换定义为：

$$\hat{I}(\omega_1, \omega_2) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x, y) e^{-j2\pi(\frac{x\omega_1}{W} + \frac{y\omega_2}{H})}$$

其实部用于表示图像。

本实验中利用函数 `cv::dft` 来实现傅里叶变换，并且将变换后的结果中的零频率从矩阵的左上角移动到中心，然后取对数将图像可视化显示。

4 实现过程与代码分析

4.1 盒状均值滤波

```
void BoxFilter(const Mat &src, Mat &dst, int w, int h)
{
    // Kernel size = (2w+1) x (2h+1)
    Mat kernel = Mat::ones(w * 2 + 1, h * 2 + 1, CV_64F);
    // Normalization
    kernel /= kernel.rows * kernel.cols;
    // Convolution
    filter2D(src, dst, -1, kernel);
}
```

参数说明：

- `src` 输入图像
- `dst` 输出图像
- `w` 卷积核中心到左右边界的距离
- `h` 卷积核中心到上下边界的距离

盒状均值滤波代码非常简单，第一行定义卷积核，第二行进行归一化保证卷积核的权重之和为 1，最后一行做卷积运算。

下图从上至下从左至右分别是 $(w, h) = (3, 3), (5, 5), (1, 5), (5, 1)$ 时的滤波结果：



可以看到卷积核越大，图像越模糊，并且横纵方向上模糊的程度与卷积核的形状相符合。

4.2 高斯滤波

```
void GaussianFilter(const Mat &src, Mat &dst, float sigma)
{
    // Kernel size = (2[5 sigma]+1) x (2[5 sigma]+1)
    int a = cvFloor(5 * sigma);
    Mat kernel(2 * a + 1, 2 * a + 1, CV_64F);
    for (int i = -a; i <= a; i++)
    {
        for (int j = -a; j <= a; j++)
        {
            kernel.at<double>(i+a, j+a) = exp( -(i * i + j * j) / (2 * sigma * sigma) );
        }
    }
    // Normalization:
    // Since 5 times sigma is large enough to cover most area in Gaussian distribution
    // Normalization coefficient can be estimated by 1 / (2 PI sigma^2)
    kernel /= 2 * CV_PI * sigma * sigma;
    // Convolution
    filter2D(src, dst, -1, kernel);
}
```

参数说明：

src 输入图像
dst 输出图像
sigma 高斯分布中的标准差 sigma 参数

高斯滤波与均值滤波流程相似，也是做卷积运算，只是卷积核各元素的值遵循高斯分布。本实验中取 5 倍 sigma 作为核中心到边界的距离，可以覆盖高斯分布中的大部分面积，因此权重可以直接用 $1 / (2 \pi \sigma^2)$ 来估计。

下图分别是 $\sigma = 1, 3, 5, 7$ 时的结果：



从图中可以看出， σ 越大图像越模糊。

4.3 中值滤波

```
void MedianFilter(const Mat &src, Mat &dst, int w, int h)
```

参数说明：

src	输入图像
dst	输出图像
w	邻域窗口中心到左右边界的距离
h	邻域窗口中心到上下边界的距离

该中值滤波器支持单通道图像和三通道图像。对于三通道图像，直接将其拆分成三张单通道的灰度图，然后对每张单通道灰度图分别滤波，最后再将滤波得到的三个通道合并到一张图像中。

对于单通道图像，作如下处理：

```
Mat img;
// Converts to float
src.convertTo(img, CV_64F);
dst = img.clone();
// Extends borders by replication
// Kernel size = (2w+1) x (2h+1)
copyMakeBorder(img, img, h, h, w, w, BORDER_REPLICATE);
vector<double> arr;
```

```

for (int i = 0; i < dst.rows; i++)
{
    for (int j = 0; j < dst.cols; j++)
    {
        arr.clear();
        for (int x = i; x < i + 2 * h + 1; x++)
        {
            for (int y = j; y < j + 2 * w + 1; y++)
            {
                arr.push_back(img.at<double>(x, y));
            }
        }
        // Calculates median
        sort(arr.begin(), arr.end());
        dst.at<double>(i, j) = arr[arr.size() / 2];
    }
}
// Convert to int
dst.convertTo(dst, CV_8U);

```

下图从上至下从左至右分别是 $(w, h) = (3, 3), (5, 5), (1, 5), (5, 1)$ 时的滤波结果：



与均值滤波类似，邻域窗口越大，图像越模糊，并且横纵方向上模糊的程度与窗口的形状相符合。

4.4 双边滤波

```
void BilateralFilter(const Mat &src, Mat &dst, float sigma_s, float sigma_r)
```

参数说明：

src	输入图像
dst	输出图像
sigma_s	空间上的高斯权重的标准差
sigma_r	亮度上的高斯权重的标准差

本实验中取 sigma_s 和 sigma_r 中更大的一个的 5 倍作为窗口中心到边界的距离：

```
// Kernel size = (2[5m]+1) x (2[5m]+1), where m = max{sigma_s, sigma_r}
int m = sigma_s > sigma_r ? cvFloor(5 * sigma_s) : cvFloor(5 * sigma_r);
int channels = src.channels();
```

该滤波器支持单通道和三通道图像滤波，流程相似，下面仅展示三通道滤波代码：

```
// Converts to float
src.convertTo(img, CV_64FC3);
dst = img.clone();
// Extends borders by reflection
int n = 2 * m + 1;
copyMakeBorder(img, img, m, m, m, m, BORDER_REFLECT);
double weight, w;
Vec3d arr;
for (int i = 0; i < dst.rows; i++)
{
    for (int j = 0; j < dst.cols; j++)
    {
        arr = 0;
        weight = 0;
        for (int x = i; x < i + n; x++)
        {
            for (int y = j; y < j + n; y++)
            {
                // Spatial difference: |x1-x2| + |y1-y2|
                // Intensity difference: |B(x1,y1)-B(x2,y2)| + |G(x1,y1)-G(x2,y2)|
+ |R(x1,y1)-R(x2,y2)|
                w = exp( -(abs(i+m-x) + abs(j+m-y)) / (2 * sigma_s * sigma_s)
                    -(abs(img.at<Vec3d>(i+m, j+m)[0] - img.at<Vec3d>(x, y)[0])
+ abs(img.at<Vec3d>(i+m, j+m)[1] - img.at<Vec3d>(x, y)[1])
+ abs(img.at<Vec3d>(i+m, j+m)[2] - img.at<Vec3d>(x,
y)[2])) )
                    / (2 * sigma_r * sigma_r) );
                weight += w;
                arr += img.at<Vec3d>(x, y) * w;
            }
        }
        dst.at<Vec3d>(i, j) = arr / weight;
    }
}
// Converts to int
dst.convertTo(dst, CV_8UC3);
```

在计算权重时，空间上用 $|x_1-x_2|+|y_1-y_2|$ 来加权，亮度上用每个通道的亮度差的绝对值之和来加权。与高斯滤波不同的是，这里的归一化系数不直接估计，而是精确计算。

下图分别是 $(\text{sigma}_s, \text{sigma}_r) = (3, 3), (5, 5), (3, 5), (5, 3)$ 时的滤波结果：

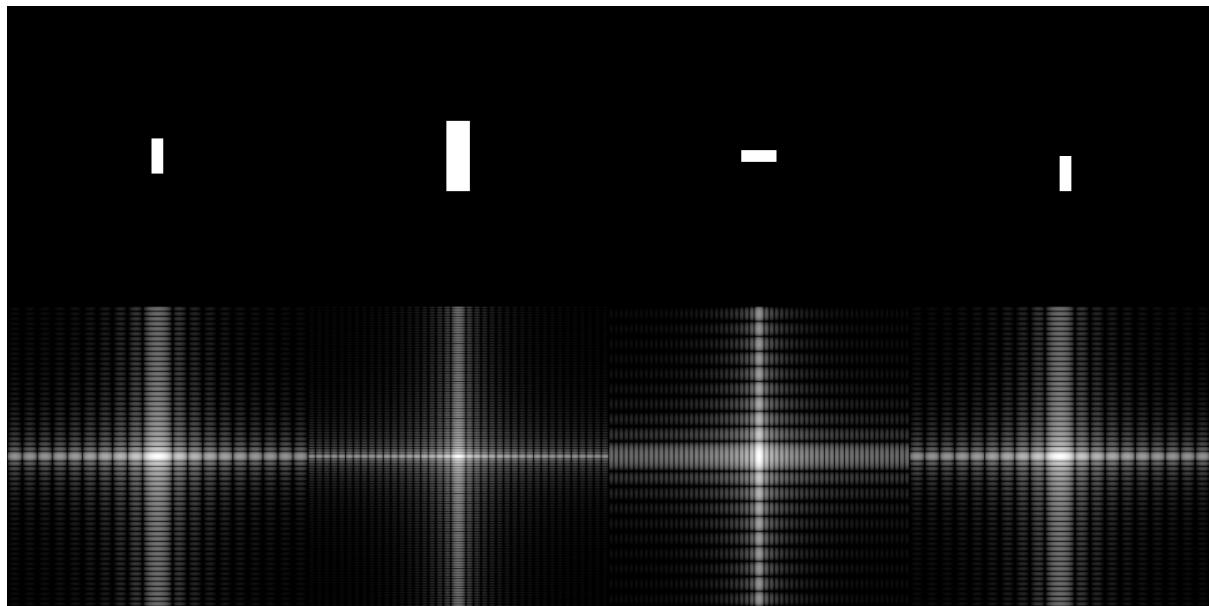


可以看到，`sigma_s` 对滤波的效果影响不是很明显，但是仔细对比左上($\text{sigma_s} = 3$)和右下两幅图($\text{sigma_s} = 5$)可以看出，更大的值会使大块的色彩（比如沙滩）更加均匀，图像更加真实。

而 `sigma_r` 对滤波的效果影响十分显著，对比左边两幅图（`sigma_r` 分别为 3 和 5）发现，更大的值使得图像中同一色块区域更加“光滑”，能更好地消除噪点。

4.5 傅里叶变换

根据课程网站上的要求绘制矩形图进行傅里叶变换并可视化，然后分别改变矩形的大小、旋转矩形、平移矩形再次傅里叶变换，比较所得的图像：



上图中第一排为输入图像（对原矩形进行了缩放、旋转、平移操作），第二排为对应的傅里叶变换后的可视化输出，图像的中心为零频率。

第一组和第二组输出没有差异，因为平移并没有改变频域上的分布；矩形的长宽不等，因此旋转会改变不同方向上的频域分布；缩放也会改变频域分布。

5 结果分析与实验总结

5.1 四种滤波器比较

下图分别是均值、高斯、中值、双边滤波的结果（所有参数均取 3）：



对比可以看出，第一排的均值滤波和高斯滤波使图像各个部分无差别地变模糊，而第二排的中值滤波和双边滤波则保留了相对独立的色块。

进一步比较前两幅图发现，均值滤波容易出现类似“重影”的现象（比如汽车的窗户），而高斯滤波虽然使图像相当模糊，但却容易保留图像原本的结构。

再比较后面两幅图：中值滤波能很好地消除噪点（比如沙滩上的大小沙粒基本上全部被消除），但是会让边界变形（比如汽车的窗户边角变圆滑），使得图像呈现一种卡通化的效果；而双边滤波可以保留非常清晰的边界，能使颜色相近的色块变得更加均匀统一，但却使一些明显的噪点更加突出（沙滩上的大沙粒更加明显了）。

综上，高斯滤波主要用于消除高斯噪声，中值滤波适用于消除椒盐噪声和产生卡通化的艺术效果，双边滤波则具有“磨皮祛斑”等人像美颜功能。

5.2 问题与解决

本次作业遇到的主要问题是由于对 `cv::Mat` 的内存管理机制不熟悉导致的图像数据混乱，通过阅读手册明确了其内存管理机制，需要在以后的实验中记住：`Mat A = B` 这样的语句相当于指针赋值，实际上 `A` 与 `B` 是共享同一块内存，因此对 `A` 的改动也会体现在 `B` 上；而 `Mat A = B.clone()` 和 `B.copyTo(A)` 这两种拷贝的方式则是另外开辟一块内存给 `A`，因此 `A` 是独立于 `B` 存在的。

另外，我发现中值滤波和双边滤波的程序运行非常缓慢，通过分析发现，这两种函数的时间复杂度都达到 $O(nm)$ ，其中 n 为输入图像的大小， m 为邻域窗口的大小，因此效率运算很低。解决办法是优化算法，可以采用维护一个数组结构、以空间换时间的方式来计算每次的邻域窗口的像素值。虽然本次实验中仍然使用最简单的遍历方式，并没有对算法进行优化，但是必须记住：算法的效率对于图像处理是非常重要的，在以后的学习中要时刻留意算法的优化问题。

6 参考文献

课程网站：<http://www.cad.zju.edu.cn/home/gfzhang/course/computational-photography/lab2-filtering/filtering.html>

中值滤波算法的优化：<https://www.cnblogs.com/yoyo-sincerely/p/6058944.html>

双边滤波算法的优化：<https://www.cnblogs.com/cpuimage/p/7629304.html>