目录

1 实验目标

利用课上学习的空间滤波方法对图像进行处理。具体要求包括:

- 输入一张校园图像。
- 分别添加高斯噪声和椒盐噪声。
- 分别选择高斯滤波或中值滤波对图像进行处理。

提交内容:

- a.源码。
  完成(2)(40分)，使用OpenCV函数完成(3)(20分)，自行写(3)中的两种(30分，其中一种给15分)，使用c++，提交cpp文件。
- b.报告。
  包含实验结果分析及运行结果截图。原图+(2）中任意一种效果图+(3）效果图Pdf文件，10分

2 程序

2.1 程序简介

此代码使用OpenCV库并且使用C++编写。它主要执行了以下任务：

- 使用OpenCV的`imread()`函数读取名为“tju.jpg”的图像。并且使用`imshow()`函数在窗口中显示图像。
- 为图像添加椒盐噪声和高斯噪声。
- 借助OpenCV库中的`medianBlur`函数和`GaussianBlur`函数分别去除椒盐噪声和高斯噪声。
- 自行实现两个图像过滤函数`myMedianBlur`和`myGaussianBlur`分别去除椒盐噪声和高斯噪声。

2.2 算法

2.2.1 中值滤波

中值滤波能够有效地去除图像中的椒盐噪声和斑点噪声，因为它不会受到极端值的影响。中值滤波的核心思想是在一个固定大小的邻域内，将像素灰度值按照大小排序，取中间的值作为该像素的输出值。

具体而言，对于输入图像中的每个像素，取其周围的一个固定大小的邻域，例如 3×3 的邻域，然后将该邻域内的像素值排序，取中间的值作为输出像素的灰度值。具体数学公式如下：

$$I_{\text{out}}(i,j) = \text{median}\left\{I_{\text{in}}(x,y) : x = i-k,\ldots,i+k, y = j-k,\ldots,j+k\right\}$$

其中，I_{in} 表示输入图像，I_{out} 表示输出图像，(i,j) 表示当前像素的坐标，k 表示邻域大小。

2.2.2 高斯滤波

高斯滤波是一种线性滤波方法，它可以有效地去除图像中的高斯噪声。高斯滤波通过对像素周围的邻域进行加权平均来平滑图像，权值是一个高斯函数。高斯函数能够根据像素与中心点的距离来决定其权重，距离越近的像素，其权重越大。具体数学公式如下：

$$I_{\text{out}}(i,j) = \frac{1}{\sum_{k,l} G_{k,l}} \sum_{k,l} G_{k,l} I_{\text{in}}(i+k, j+l)$$

其中，I_{in} 表示输入图像，I_{out} 表示输出图像，(i,j) 表示当前像素的坐标，G 表示高斯核，$\sum_{k,l} G_{k,l}$ 表示归一化常数，保证输出像素的灰度值在 $[0,255]$ 范围内。高斯核的大小和标准差是高斯滤波的两个重要参数，它们会影响滤波的效果和速度。

2.3 实现细节

2.3.1 输入一张校园图像

使用 `imread()` 函数读取 "tju.jpg" 图片，并创建一个窗口来显示它。

```
    //读取tju.jpg
    Mat src = imread("tju.jpg");
    //创建窗口
    namedWindow("tju", WINDOW_AUTOSIZE);
    //显示图像
    imshow("tju", src);
    //等待按键
    waitKey(0);
    return 0;
```

2.3.2 分别添加高斯噪声和椒盐噪声

部分关键步骤如下：

- 使用 `RNG()` 函数来生成随机数。
- 使用 `uniform()` 函数来生成指定范围内的随机数。
- 使用 `at<>` 函数来修改图像的像素值。
- 使用 `gaussian()` 函数来生成服从均值为0，标准差为25的高斯分布的随机数。
- 使用 `imwrite()` 函数将处理后的两张图片保存为文件。

```
    Mat dst1, dst2;
    src.copyTo(dst1);
    src.copyTo(dst2);
    //定义随机数生成器
    RNG rng;
    //定义噪声的范围
    int width = src.cols;
    int height = src.rows;
    //定义噪声的数量
    int num = width * height / 20;
    //定义噪声的类型
    int type = 0;
    //定义噪声的强度
    int intensity = 255;
    //向dst1添加椒盐噪声
    for (int i = 0; i < num; i++)
    {
        //随机生成噪声的位置
        int x = rng.uniform(0, width);
```

```
        int y = rng.uniform(0, height);
        //随机生成噪声的类型
        type = rng.uniform(0, 2);
        //随机生成噪声的强度
        intensity = rng.uniform(0, 255);
        //添加椒盐噪声
        if (type == 0)
        {
            dst1.at<Vec3b>(y, x)[0] = 0;
            dst1.at<Vec3b>(y, x)[1] = 0;
            dst1.at<Vec3b>(y, x)[2] = 0;
        }
        else
        {
            dst1.at<Vec3b>(y, x)[0] = 255;
            dst1.at<Vec3b>(y, x)[1] = 255;
            dst1.at<Vec3b>(y, x)[2] = 255;
        }
    }
    //向dst2添加高斯噪声
    //向dst2添加高斯噪声
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            Vec3b& pixel = dst2.at<Vec3b>(i, j);
            for (int k = 0; k < 3; k++)
            {
                float noise = rng.gaussian(25);//rng.gaussian(val)表示生成一个服从均值为
0,标准差为val的高斯分布的随机数
                int val = pixel[k] + static_cast<int>(noise);
                val = std::max(val, 0);
                val = std::min(val, 255);
                pixel[k] = static_cast<uchar>(val);
            }
        }
    }
    //显示图像
    namedWindow("tju-SPN", WINDOW_AUTOSIZE);
    imshow("tju-SPN", dst1);
    namedWindow("tju-GN", WINDOW_AUTOSIZE);
    imshow("tju-GN", dst2);
    //等待按键
    waitKey(0);
    //将两张图片分别保存为tju-SPN和tju-GN
    imwrite("tju-SPN.jpg", dst1);
    imwrite("tju-GN.jpg", dst2);
```

2.3.3 分别选择高斯滤波或中值滤波处理图像

a) 调用OpenCV相关接口处理

使用的中值滤波函数是OpenCV中的`medianBlur`函数，第二个参数3表示滤波器大小为3x3。

使用`GaussianBlur`函数对dst2进行高斯滤波，并将结果保存到dst4中。`GaussianBlur`函数的第一个参数是输入图像，第二个参数是输出图像，第三个参数是高斯核的大小，第四个参数和第五个参数分别表示在X和Y方向的高斯核标准差，如果都设置为0，则函数会自动计算标准差。

```
    //对dst1进行中值滤波
    Mat dst3;
    medianBlur(dst1, dst3, 3);
    //显示图像
    namedWindow("tju-SPN-MedianBlur", WINDOW_AUTOSIZE);
    imshow("tju-SPN-MedianBlur", dst3);
    waitKey(0);

    //对dst2进行高斯滤波
    Mat dst4;
    GaussianBlur(dst2, dst4, Size(5, 5), 0, 0);
    namedWindow("tju-GN-GaussianBlur", WINDOW_AUTOSIZE);
    imshow("tju-GN-GaussianBlur", dst4);
    waitKey(0);
    //保存图像
    imwrite("tju-SPN-MedianBlur.jpg", dst3);
    imwrite("tju-GN-GaussianBlur.jpg", dst4);
```

b) 2.3.2 自行编写函数处理

自行编写的中值滤波和高斯滤波函数，分别为`myMedianBlu`和`myGaussianBlur`。

- `myMedianBlur`函数：

将输入的src图像中的每个像素用它周围ksize* ksize个像素的中值来替换。其中，边界不做处理。具体实现方法是：遍历图像中每个像素的周围ksize* ksize个像素，将这些像素的RGB通道的值分别存入values数组中，对每个通道分别进行排序，取每个通道中间的值作为当前像素的值，最后将这个值赋给dst图像对应位置的像素。

- `myGaussianBlur`函数：

将输入的src图像中的每个像素用它周围ksize* ksize个像素的加权平均值来替换。其中，边界不做处理。具体实现方法是：生成一个长度为ksize的高斯核，每个位置的权重系数为高斯分布函数的值，然后遍历图像中每个像素的周围ksize个像素，将这些像素在每个通道上的值分别与高斯核的对应位置的权重系数相乘并求和，将结果赋给dst图像对应位置的像素。

```
Mat myMedianBlur(const Mat& src, int kernel_size) {
    int border_size = kernel_size / 2;
    Mat dst(src.size(), CV_8UC3);
    for (int i = border_size; i < src.rows - border_size; i++) {
        for (int j = border_size; j < src.cols - border_size; j++) {
            Vec3b median_value(0, 0, 0);
```

```
            vector<int> values[3];
            for (int m = -border_size; m <= border_size; m++) {
                for (int n = -border_size; n <= border_size; n++) {
                    Vec3b pixel = src.at<Vec3b>(i + m, j + n);
                    values[0].push_back(pixel[0]);
                    values[1].push_back(pixel[1]);
                    values[2].push_back(pixel[2]);
                }
            }
            sort(values[0].begin(), values[0].end());
            sort(values[1].begin(), values[1].end());
            sort(values[2].begin(), values[2].end());
            median_value[0] = static_cast<uchar>(values[0][kernel_size * kernel_size
/ 2]);
            median_value[1] = static_cast<uchar>(values[1][kernel_size * kernel_size
/ 2]);
            median_value[2] = static_cast<uchar>(values[2][kernel_size * kernel_size
/ 2]);
            dst.at<Vec3b>(i, j) = median_value;
        }
    }
    return dst;
}

Mat myGaussianBlur(const Mat& src, int ksize, double sigma) {
    int border = ksize / 2; // 边界宽度
    double* kernel = new double[ksize]; // 滤波器核数组
    double sum = 0.0; // 核数组元素之和

    // 生成高斯滤波核
    for (int i = 0; i < ksize; i++) {
        kernel[i] = exp(-(i - border) * (i - border) / (2 * sigma * sigma));
        sum += kernel[i];
    }
    for (int i = 0; i < ksize; i++) {
        kernel[i] /= sum;
    }

    // 处理图像
    Mat dst(src.size(), CV_8UC3);
    dst.create(src.size(), src.type());
    for (int y = border; y < src.rows - border; y++) {
        for (int x = border; x < src.cols - border; x++) {
            for (int c = 0; c < src.channels(); c++) {
                double val = 0.0;
                for (int i = 0; i < ksize; i++) {
                    val += kernel[i] * src.at<Vec3b>(y, x - border + i)[c];
                }
                dst.at<Vec3b>(y, x)[c] = static_cast<uchar>(val + 0.5);
            }
        }
    }
```

```
    delete[] kernel;
    return dst;
}
```

3 结果与讨论

3.1 实验结果

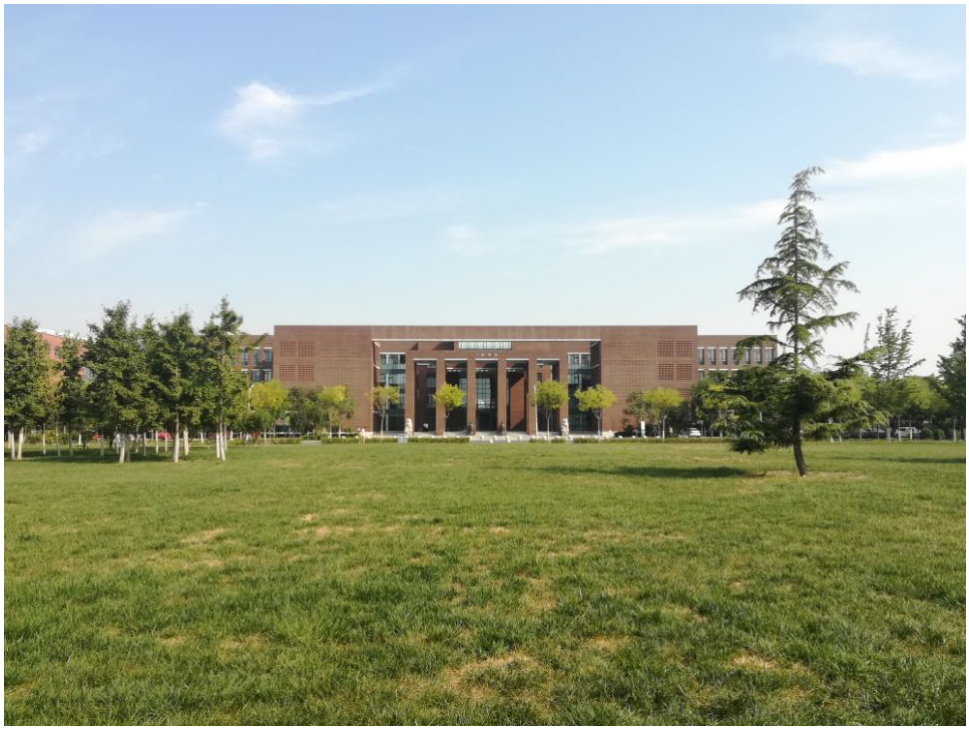

图1 校园风景图

3.1.1 分别添加椒盐噪声和高斯噪声

图2 添加噪声后图像（上图为添加椒盐噪声，下图为添加高斯噪声）

3.1.2 调用OpenCV相关接口处理图像

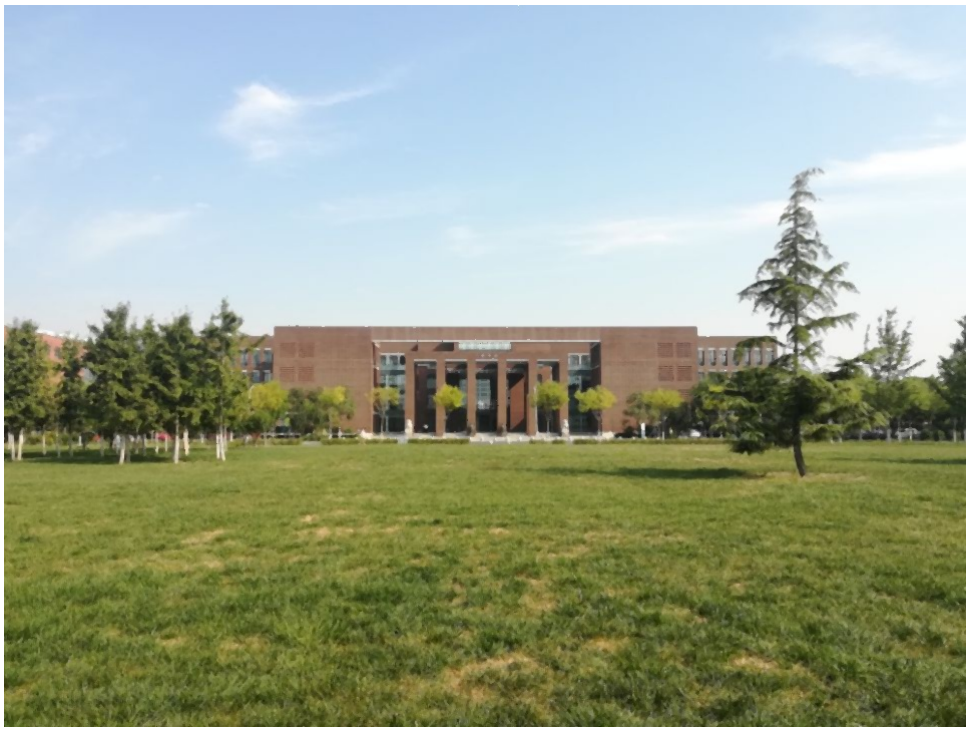

图3 借助OpenCV库函数对图像降噪（上图为对椒盐噪声中值滤波后图像，下图为对高斯噪声高斯滤波后图像）

3.1.3 自行编写函数处理图像

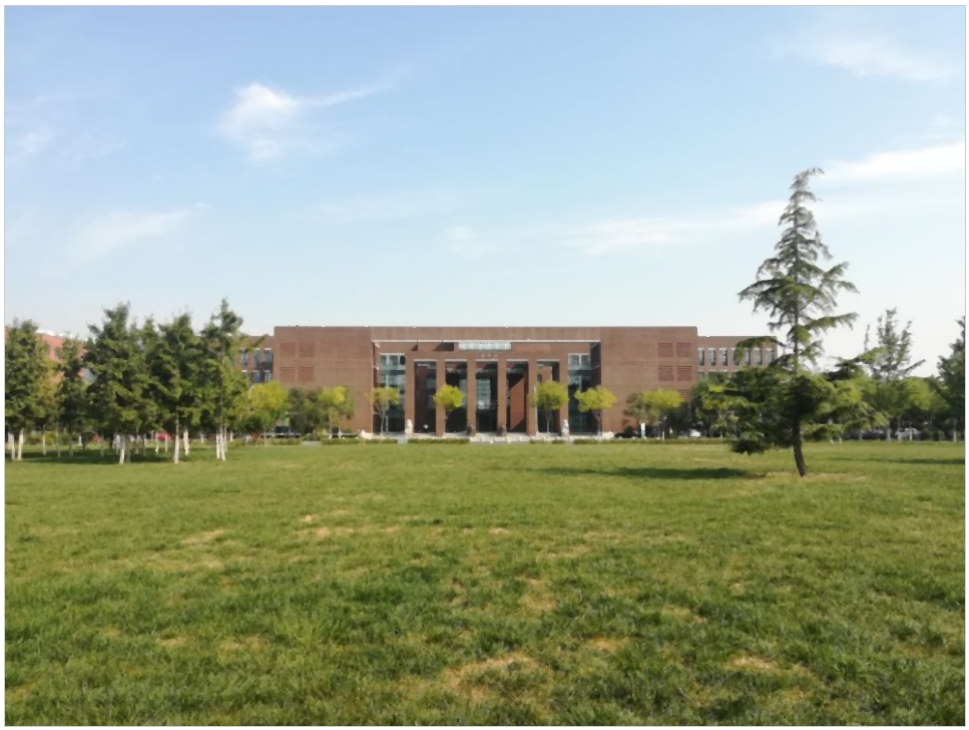

图4 自行编写函数对图像降噪（上图为对椒盐噪声中值滤波后图像，下图为对高斯噪声高斯滤波后图像）

3.2 实验分析

可以看到，添加椒盐噪声后图像出现了大量黑白噪点，导致图像质量严重下降。使用中值滤波处理后噪声被有效地去除，图像质量得到了明显的提升。添加高斯噪声后图像出现了模糊和颜色偏差，导致图像质量下降。使用高斯滤波后一定程度上能减缓这种噪声，但无法完全消除。

3.3 讨论

噪声是影响图像质量的一个重要因素。在实际的应用中，需要根据具体情况选择合适的去噪方法和参数。例如，如果需要去除椒盐噪声，则中值滤波更适合；如果需要保留更多的图像细节，则可以考虑使用高斯滤波。总之，掌握图像去噪的基本原理和方法，对于理解和应用计算机视觉算法都有很大的帮助。通过这次实验，我对椒盐噪声、高斯噪声、中值滤波和高斯滤波的理解更加深刻，从中收益匪浅。