高级图像处理与分析课程实验报告4

学号	姓名	日期
SA18225428	许强	2019.04.04

实验名称	灰度变换
实验内容	1、均值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用算术均值滤 波器、几何均值滤波器、谐波和逆谐波均值滤波器进行图像去噪。模板大小为 55。（注：请分别为图像添加高斯噪声、胡椒噪声、盐噪声和椒盐噪声，并观察 滤波效果） 2、中值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用 55 和 99 尺寸的模板对图像进行中值滤波。（注：请分别为图像添加胡椒噪声、盐噪声和 椒盐噪声，并观察滤波效果） 3、自适应均值滤波。 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应局部降 低噪声滤波器去噪算法。模板大小 77（对比该算法的效果和均值滤波器的效果） 4、自适应中值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应中值滤波算 法对椒盐图像进行去噪。模板大小 77（对比中值滤波器的效果） 5、彩色图像均值滤波 具体内容：利用 OpenCV 对彩色图像 RGB 三个通道的像素进行操作，利用算 术均值滤波器和几何均值滤波器进行彩色图像去噪。模板大小为 55。
实验完成情况（包括完成的实验内容及每个实验的完成程度。注意	5个模块全部完成

要贴出每个实验的核心代码）	
实验中的问题（包括在实验中遇到的问题，以及解决问题的方法）	参考了网上的自适应滤波算法
实验结果（实验完成后的源码和打包文件的说明）	代码注释中含有部分说明

```
//
// Created by XQ on 2019-03-29.
//
```

```cpp

#include<iostream>
#include<string>

#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/core/types_c.h>
#include <opencv2/core/core_c.h>
#include <opencv2/highgui/highgui.hpp>
#include <zconf.h>

using namespace std;
using namespace cv;


bool addSaltOrPepper(Mat &image, int flag,int n);
int GaussianNoise(double mu, double sigma);
bool addGaussianNoise(Mat& image);
Mat digitalMeanFilter(Mat& image, int size);
Mat geometryMeanFilter(Mat &image, int size);
Mat harmonicMeanFilter(Mat &image, int size);
Mat inverseHarmonicMeanFilter(Mat &image, int size, double Q);

Mat medianFilter(Mat &image, int size);
Mat selfAdaptMeanFilter(Mat &image, int size);
Mat selfAdaptMedianFilter(Mat &image, int size);

//1.均值滤波
bool meanFilter(string &src, int flag){
    Mat image = imread(src,flag);
    imshow("input", image);

    Mat noise = image.clone();
    addSaltOrPepper(noise,0,1000);
    imshow("with 1000 salt",noise);
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
    imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
    imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
    imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise, 5, 1));
    waitKey(0);
    destroyAllWindows();

    imshow("input", image);
    noise = image.clone();
    addSaltOrPepper(noise,1,1000);
    imshow("with 1000 pepper",noise);
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
    imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
    imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
    imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise, 5, 1));
    waitKey(0);
    destroyAllWindows();

    imshow("input", image);
    noise = image.clone();
    addGaussianNoise(noise);
    imshow("with gaussi",noise);
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
    imshow("geometryMeanFilter", geometryMeanFilter(noise ,5));
    imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
    imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise, 5, 1));
    waitKey(0);
    destroyAllWindows();

    imshow("input", image);
    noise = image.clone();
    addSaltOrPepper(noise,0,1000);
    usleep(500);
    addSaltOrPepper(noise,1,1000);
    imshow("with 1000 pepper and 1000 salt",noise);
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
    imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
    imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
    imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise, 5, 1));
    waitKey(0);
    destroyAllWindows();
    return true;
}
//2.中值滤波
bool medianFilter(string &src, int flag){
    Mat image = imread(src,flag);
    imshow("input", image);

    Mat noise = image.clone();
    addSaltOrPepper(noise,0,1000);
    imshow("with 1000 salt",noise);
    imshow("with 1000 salt, 5*5 median",medianFilter(image,5));
    imshow("with 1000 salt, 9*9 median",medianFilter(image,9));
```

```cpp
    waitKey(0);
    destroyAllWindows();

    imshow("input", image);
    noise = image.clone();
    addSaltOrPepper(noise,1,1000);
    imshow("with 1000 pepper",noise);
    imshow("with 1000 pepper, 5*5 median",medianFilter(image,5));
    imshow("with 1000 pepper, 9*9 median",medianFilter(image,9));
    waitKey(0);
    destroyAllWindows();

    imshow("input", image);
    noise = image.clone();
    addSaltOrPepper(noise,0,1000);
    usleep(500);
    addSaltOrPepper(noise,0,1000);
    imshow("with 1000 salt and 1000 pepper",noise);
    imshow("with 1000 salt and 1000 pepper, 5*5 median",medianFilter(image,5));
    imshow("with 1000 salt and 1000 pepper, 9*9 median",medianFilter(image,9));
    waitKey(0);
    destroyAllWindows();

    return true;
}
//3.自适应均值滤波
bool selfAdaptMeanFilter(string &src, int flag){
    Mat image = imread(src,flag);
    imshow("input", image);

    Mat noise = image.clone();
    addSaltOrPepper(noise, 0,1000);
    usleep(500);
    addSaltOrPepper(noise, 1,1000);
    imshow("with 1000 pepper and 1000 salt",noise);
    imshow("selfAdaptMeanFilter",selfAdaptMeanFilter(noise,7));
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 7));

    waitKey(0);
    destroyAllWindows();
    return true;
}
//4.自适应中值滤波
bool selfAdaptMedianFilter(string &src, int flag){
    Mat image = imread(src,flag);
    imshow("input", image);

    Mat noise = image.clone();
    addSaltOrPepper(noise, 0,1000);
    usleep(500);
    addSaltOrPepper(noise, 1,1000);
    imshow("with 1000 pepper and 1000 salt",noise);
    imshow("selfAdaptMedianFilter", selfAdaptMedianFilter(noise,7));
    imshow("medianFilter", medianFilter(noise, 7));

    waitKey(0);
    destroyAllWindows();
    return true;
}
//5.彩色图像均值滤波
bool colorMeanFilter(string &src, int flag){
    Mat image = imread(src,flag);
    imshow("input", image);

    Mat noise = image.clone();
    addGaussianNoise(noise);
    imshow("with gaussi",noise);
    imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
    imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));

    waitKey(0);
    destroyAllWindows();
    return true;
}

int main(){
    string str = "/Volumes/数据/图片/2k/lostwall.jpg";
    cout << "1.meanFilter:" << meanFilter(str,0) << endl;
    cout << "2.medianFilter:" << medianFilter(str,0) << endl;
    cout << "3.selfAdaptMeanFilter:" << selfAdaptMeanFilter(str,0) << endl;
    cout << "4.selfAdaptMedianFilter:" << selfAdaptMedianFilter(str,0) << endl;
    cout << "5.colorMeanFilter:" << colorMeanFilter(str,1) << endl;
}

/*添加椒盐盐噪声
 *
 * flag = 0 盐噪声
 * flag = 1 椒噪声
 * */
bool addSaltOrPepper(Mat &image, int flag,int n){
    srand((unsigned)time(NULL));
    for (int k = 0; k < n; k++)//将图像中n个像素随机置零
    {
        int i = rand() % image.rows;
        int j = rand() % image.cols;
        //将图像颜色随机改变
        if (image.channels() == 1){
            if(flag == 0)   image.at<uchar>(i, j) = 255;
            if(flag == 1)   image.at<uchar>(i, j) = 0;
        }

        else{
            for (int t = 0; t < image.channels(); t++){
                if(flag == 0)   image.at<Vec3b>(i, j)[t] = 255;
                if(flag == 1)   image.at<Vec3b>(j, i)[t] = 0;
            }
        }
    }
    return true;
}
/*高斯噪声*/
int GaussianNoise(double mu, double sigma){
    //定义一个特别小的值
    const double epsilon = numeric_limits<double>::min();//返回目标数据类型能表示的最逼近1的正数和1的差的绝对值
    static double z0, z1;
    static bool flag = false;
    flag = !flag;
    //flag为假，构造高斯随机变量
    if (!flag)  return z1 * sigma + mu;
    double u1, u2;
    //构造随机变量

    do{
        u1 = rand()*(1.0 / RAND_MAX);
        u2 = rand()*(1.0 / RAND_MAX);
    } while (u1 <= epsilon);
    //flag为真构造高斯随机变量X
    z0 = sqrt(-2.0*log(u1))*cos(2 * CV_PI * u2);
    z1 = sqrt(-2.0*log(u1))*sin(2 * CV_PI * u2);
    return z1 * sigma + mu;
}
/*添加高斯噪声*/
bool addGaussianNoise(Mat &image){
    int channels = image.channels();    //获取图像的通道
    int rows = image.rows;    //图像的行数

    int cols = image.cols*channels;   //图像的总列数
    //判断图像的连续性
    if (image.isContinuous()){    //判断矩阵是否连续，若连续，我们相当于只需要遍历一个一维数组{
        cols *= rows;
        rows = 1;
    }
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){ //添加高斯噪声
            int val = image.ptr<uchar>(i)[j] + GaussianNoise(2, 0.8) * 32;
            if (val < 0)    val = 0;
            if (val > 255)  val = 255;
            image.ptr<uchar>(i)[j] = (uchar)val;
        }
    }
    return true;
}

/*均值滤波
 *
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用算术均值滤波器、几何均值滤波器、谐波和逆谐波均值滤波器进行图像去噪。
 * 模板大小为 5*5。（注：请分别为图像添加高斯噪声、胡椒噪声、盐噪声和椒盐噪声，并观察滤波效果）
 *
 *
 * */
//算数均值
Mat digitalMeanFilter(Mat &image, int size) {
    Mat dst = image.clone();
    int rows = dst.rows, cols = dst.cols;
    int start = size / 2;
    for (int m = start; m < rows - start; m++) {
        for (int n = start; n < cols - start; n++) {
            if (dst.channels() == 1)                //灰色图
            {
                int sum = 0;
                for (int i = -start + m; i <= start + m; i++)
                {
                    for (int j = -start + n; j <= start + n; j++) {
                        sum += dst.at<uchar>(i, j);
                    }
                }
                dst.at<uchar>(m, n) = uchar(sum / size / size);
            }
            else
            {
                Vec3b pixel;
                int sum1[3] = { 0 };
                for (int i = -start + m; i <= start + m; i++)
                {
                    for (int j = -start + n; j <= start + n; j++)
                    {
                        pixel = dst.at<Vec3b>(i, j);
                        for (int k = 0; k < dst.channels(); k++)
                        {
                            sum1[k] += pixel[k];
                        }
                    }

                }
                for (int k = 0; k < dst.channels(); k++)
                {
                    pixel[k] = sum1[k] / size / size;
                }
                dst.at<Vec3b>(m, n) = pixel;
            }
        }
    }
    return dst;
}

//几何均值
Mat geometryMeanFilter(Mat &image, int size)
{
    Mat dst = image.clone();
    int row, col;
    int h = image.rows;
    int w = image.cols;
    double mul;
    double dc;
    int mn;
    //计算每个像素的去噪后 color 值
    for (int i = 0; i < image.rows; i++)
    {
        for (int j = 0; j < image.cols; j++)
        {

            int left = -size/2;
            int right = size/2;
            if (image.channels() == 1)                  //灰色图
            {
                mul = 1.0;
                mn = 0;

                //统计邻域内的几何平均值，邻域大小 5*5
                for (int m = left; m <= right; m++) {
                    row = i + m;
                    for (int n = left; n <= right; n++) {
                        col = j + n;
                        if (row >= 0 && row < h && col >= 0 && col < w) {
                            int s = image.at<uchar>(row, col);
                            mul = mul * (s == 0 ? 1 : s); //邻域内的非零像素点相乘，最小值设定为1
                            mn++;
                        }
                    }
                }
                //计算 1/mn 次方
                dc = pow(mul, 1.0 / mn);
                //统计成功赋给去噪后图像。
                int res = (int)dc;
                dst.at<uchar>(i, j) = res;
            }
            else
            {
                double multi[3] = { 1.0,1.0,1.0 };
                mn = 0;
                Vec3b pixel;

                for (int m = left; m <= right; m++)
                {
                    row = i + m;
                    for (int n = left; n <= right; n++)
                    {
                        col = j + n;
                        if (row >= 0 && row < h && col >= 0 && col < w)
                        {
                            pixel = image.at<Vec3b>(row, col);
                            for (int k = 0; k < image.channels(); k++)
                            {
                                multi[k] = multi[k] * (pixel[k] == 0 ? 1 : pixel[k]);//邻域内的非零像素点相乘，最小值设定为1
                            }
                            mn++;
                        }
                    }
                }
                double d;
                for (int k = 0; k < image.channels(); k++)
                {
                    d = pow(multi[k], 1.0 / mn);
                    pixel[k] = (int)d;
                }
                dst.at<Vec3b>(i, j) = pixel;
            }
        }
    }
    return dst;
}

//谐波均值
Mat harmonicMeanFilter(Mat &image, int size)
{
    //IplImage* dst = cvCreateImage(cvGetSize(image), image->depth, image->nChannels);
    Mat dst = image.clone();
    int row, col;
    int h = image.rows;
    int w = image.cols;
    double sum;
    double dc;
    int mn;
    //计算每个像素的去噪后 color 值
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            sum = 0.0;
            mn = 0;
            //统计邻域,5*5 模板
            int left = -size/2;
            int right = size/2;
            for (int m = left; m <= right; m++) {
                row = i + m;
                for (int n = left; n <= right; n++) {
                    col = j + n;
                    if (row >= 0 && row < h && col >= 0 && col < w) {
                        int s = image.at<uchar>(row, col);
                        sum = sum + (s == 0 ? 255 : 255.0 / s); //如果是0，设定为255
                        mn++;
                    }
                }
            }
            int d;
            dc = mn * 255.0 / sum;
            d = dc;
            //统计成功赋给去噪后图像。
            dst.at<uchar>(i, j) = d;
        }
    }
    return dst;
}

//逆谐波均值
Mat inverseHarmonicMeanFilter(Mat &image, int size, double Q){
    Mat dst = image.clone();
    int row, col;
    int h = image.rows;
    int w = image.cols;
    double sum;
    double sum1;
    double dc;
    //double Q = 2;
    //计算每个像素的去噪后 color 值
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            sum = 0.0;
            sum1 = 0.0;
            //统计邻域
            int left = -size/2;
            int right = size/2;
            for (int m = left; m <= right; m++) {
                row = i + m;
                for (int n = left; n <= right; n++) {
                    col = j + n;
                    if (row >= 0 && row < h && col >= 0 && col < w) {

                        int s = image.at<uchar>(row, col);
                        sum = sum + pow(s, Q + 1);
                        sum1 = sum1 + pow(s, Q);
                    }
                }
            }
            //计算 1/mn 次方
            int d;
            dc = sum1 == 0 ? 0 : (sum / sum1);
            d = (int)dc;
            //统计成功赋给去噪后图像。
            dst.at<uchar>(i, j) = d;
        }
    }
    return dst;
}
```

```
/*中值滤波
 *
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用 5*5 和 9*9 尺寸的模板对图像进行中值滤波。
 * （注：请分别为图像添加胡椒噪声、盐噪声和 椒盐噪声，并观察滤波效果）
 *
 * */
Mat medianFilter(Mat &image, int size) {
    Mat dst = image.clone();
    int rows = dst.rows, cols = dst.cols;
    int start = size / 2;
    for (int m = start; m < rows - start; m++) {
        for (int n = start; n < cols - start; n++) {
            vector<uchar> model;
            for (int i = -start + m; i <= start + m; i++) {
                for (int j = -start + n; j <= start + n; j++) {
                    model.push_back(dst.at<uchar>(i, j));
                }
            }
            sort(model.begin(), model.end());     //采用快速排序进行
            dst.at<uchar>(m, n) = model[size*size / 2];
        }
    }
    return dst;
}

/*自适应均值滤波
 *
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应局部降 低噪声滤波器去噪算法。
 * 模板大小 7*7（对比该算法的效果和均值滤波器的效果）
 *
 * */
Mat selfAdaptMeanFilter(Mat &image, int size)
{
    Mat dst = image.clone();
    blur(image, dst, Size(size, size));
    int row, col;
    int h = image.rows;
    int w = image.cols;
    int mn;
    double Zxy;
    double Zmed;
    double Sxy;
    double Sl;
    double Sn = 100;
    for (int i = 0; i < image.rows; i++)
    {
        for (int j = 0; j < image.cols; j++)
        {
            int Zxy = image.at<uchar>(i, j);
            int Zmed = image.at<uchar>(i, j);
            Sl = 0;
            mn = 0;
            int left = -size/2;
            int right = size/2;
            for (int m = left; m <= right; m++) {
                row = i + m;
                for (int n = left; n <= right; n++) {
                    col = j + n;
                    if (row >= 0 && row < h && col >= 0 && col < w) {
                        int Sxy = image.at<uchar>(row, col);
                        Sl = Sl + pow(Sxy - Zmed, 2);
                        mn++;
                    }
                }
            }
            Sl = Sl / mn;
            int d = (int)(Zxy - Sn / Sl * (Zxy - Zmed));
            dst.at<uchar>(i, j) = d;
        }
    }
    return dst;
}


/*自适应中值滤波
 *
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应中值滤波算 法对椒盐图像进行去噪。
 * 模板大小 7*7（对比中值滤波器的效果）
 *
 *
 * */
Mat selfAdaptMedianFilter(Mat &image, int size) {
    Mat dst = image.clone();
    int row, col;
    int h = image.rows;
    int w = image.cols;
    double Zmin, Zmax, Zmed, Zxy, Smax = size;
    int wsize;
    //计算每个像素的去噪后 color 值
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            //统计邻域
            wsize = 1;
            while (wsize <= size / 2) {
                Zmin = 255.0;
                Zmax = 0.0;
                Zmed = 0.0;
                int Zxy = image.at<uchar>(i, j);
                int mn = 0;
                for (int m = -wsize; m <= wsize; m++) {
                    row = i + m;
                    for (int n = -wsize; n <= wsize; n++) {
                        col = j + n;
                        if (row >= 0 && row < h && col >= 0 && col < w) {
                            int s = image.at<uchar>(row, col);
                            if (s > Zmax) {
                                Zmax = s;
                            }
                            if (s < Zmin) {
                                Zmin = s;
                            }
                            Zmed = Zmed + s;
                            mn++;
                        }
                    }
                }
                Zmed = Zmed / mn;
                int d;
                if ((Zmed - Zmin) > 0 && (Zmed - Zmax) < 0) {
                    if ((Zxy - Zmin) > 0 && (Zxy - Zmax) < 0) {
                        d = Zxy;
                    } else {
                        d = Zmed;
                    }
                    dst.at<uchar>(i, j) = d;
                    break;
                } else {
                    wsize++;
                    if (wsize > size / 2) {
                        int d;
                        d = Zmed;
                        dst.at<uchar>(i, j) = d;
                        break;
                    }
                }
            }
        }
```

```
    }
    return dst;
}


/*彩色图像均值滤波
 *
 * 具体内容：利用 OpenCV 对彩色图像 RGB 三个通道的像素进行操作，利用算 术均值滤波器和几何均值滤波器进行彩色图像去噪。
 * 模板大小为 5*5。
 *
 *
 * */
```