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();
```

```
50      addSaltOrPepper(noise,1,1000);
51      imshow("with 1000 pepper",noise);
52      imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
53      imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
54      imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
55      imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise,
    5, 1));
56      waitKey(0);
57      destroyAllWindows();
58
59      imshow("input", image);
60      noise = image.clone();
61      addGaussianNoise(noise);
62      imshow("with gaussi",noise);
63      imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
64      imshow("geometryMeanFilter", geometryMeanFilter(noise ,5));
65      imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
66      imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise,
    5, 1));
67      waitKey(0);
68      destroyAllWindows();
69
70      imshow("input", image);
71      noise = image.clone();
72      addSaltOrPepper(noise,0,1000);
73      usleep(500);
74      addSaltOrPepper(noise,1,1000);
75      imshow("with 1000 pepper and 1000 salt",noise);
76      imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
77      imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
78      imshow("harmonicMeanFilter", harmonicMeanFilter(noise, 5));
79      imshow("inverseHarmonicMeanFilter", inverseHarmonicMeanFilter(noise,
    5, 1));
80      waitKey(0);
81      destroyAllWindows();
82      return true;
83  }
84  //2.中值滤波
85  bool medianFilter(string &src, int flag){
86      Mat image = imread(src,flag);
87      imshow("input", image);
88
89      Mat noise = image.clone();
90      addSaltOrPepper(noise,0,1000);
91      imshow("with 1000 salt",noise);
92      imshow("with 1000 salt, 5*5 median",medianFilter(image,5));
93      imshow("with 1000 salt, 9*9 median",medianFilter(image,9));
```

```
94      waitKey(0);
95      destroyAllWindows();
96
97      imshow("input", image);
98      noise = image.clone();
99      addSaltOrPepper(noise,1,1000);
100     imshow("with 1000 pepper",noise);
101     imshow("with 1000 pepper, 5*5 median",medianFilter(image,5));
102     imshow("with 1000 pepper, 9*9 median",medianFilter(image,9));
103     waitKey(0);
104     destroyAllWindows();
105
106     imshow("input", image);
107     noise = image.clone();
108     addSaltOrPepper(noise,0,1000);
109     usleep(500);
110     addSaltOrPepper(noise,0,1000);
111     imshow("with 1000 salt and 1000 pepper",noise);
112     imshow("with 1000 salt and 1000 pepper, 5*5
    median",medianFilter(image,5));
113     imshow("with 1000 salt and 1000 pepper, 9*9
    median",medianFilter(image,9));
114     waitKey(0);
115     destroyAllWindows();
116
117     return true;
118 }
119 //3.自适应均值滤波
120 bool selfAdaptMeanFilter(string &src, int flag){
121     Mat image = imread(src,flag);
122     imshow("input", image);
123
124     Mat noise = image.clone();
125     addSaltOrPepper(noise, 0,1000);
126     usleep(500);
127     addSaltOrPepper(noise, 1,1000);
128     imshow("with 1000 pepper and 1000 salt",noise);
129     imshow("selfAdaptMeanFilter",selfAdaptMeanFilter(noise,7));
130     imshow("digitalMeanFilter", digitalMeanFilter(noise, 7));
131
132     waitKey(0);
133     destroyAllWindows();
134     return true;
135 }
136 //4.自适应中值滤波
137 bool selfAdaptMedianFilter(string &src, int flag){
138     Mat image = imread(src,flag);
```

```
139         imshow("input", image);
140 
141         Mat noise = image.clone();
142         addSaltOrPepper(noise, 0,1000);
143         usleep(500);
144         addSaltOrPepper(noise, 1,1000);
145         imshow("with 1000 pepper and 1000 salt",noise);
146         imshow("selfAdaptMedianFilter", selfAdaptMedianFilter(noise,7));
147         imshow("medianFilter", medianFilter(noise, 7));
148 
149         waitKey(0);
150         destroyAllWindows();
151         return true;
152     }
153     //5.彩色图像均值滤波
154     bool colorMeanFilter(string &src, int flag){
155         Mat image = imread(src,flag);
156         imshow("input", image);
157 
158         Mat noise = image.clone();
159         addGaussianNoise(noise);
160         imshow("with gaussi",noise);
161         imshow("digitalMeanFilter", digitalMeanFilter(noise, 5));
162         imshow("geometryMeanFilter", geometryMeanFilter(noise, 5));
163 
164         waitKey(0);
165         destroyAllWindows();
166         return true;
167     }
168 
169     int main(){
170         string str = "/Volumes/数据/图片/2k/lostwall.jpg";
171         cout << "1.meanFilter:" << meanFilter(str,0) << endl;
172         cout << "2.medianFilter:" << medianFilter(str,0) << endl;
173         cout << "3.selfAdaptMeanFilter:" << selfAdaptMeanFilter(str,0) <<
    endl;
174         cout << "4.selfAdaptMedianFilter:" << selfAdaptMedianFilter(str,0)
    << endl;
175         cout << "5.colorMeanFilter:" << colorMeanFilter(str,1) << endl;
176     }
177 
178     /*添加椒盐盐噪声
179      *
180      * flag = 0 盐噪声
181      * flag = 1 椒噪声
182      * */
183     bool addSaltOrPepper(Mat &image, int flag,int n){
```

```
184         srand((unsigned)time(NULL));
185         for (int k = 0; k < n; k++)//将图像中n个像素随机置零
186         {
187             int i = rand() % image.rows;
188             int j = rand() % image.cols;
189             //将图像颜色随机改变
190             if (image.channels() == 1){
191                 if(flag == 0)   image.at<uchar>(i, j) = 255;
192                 if(flag == 1)   image.at<uchar>(i, j) = 0;
193             }
194 
195             else{
196                 for (int t = 0; t < image.channels(); t++){
197                     if(flag == 0)   image.at<Vec3b>(i, j)[t] = 255;
198                     if(flag == 1)   image.at<Vec3b>(j, i)[t] = 0;
199                 }
200             }
201         }
202         return true;
203     }
204     /*高斯噪声*/
205     int GaussianNoise(double mu, double sigma){
206         //定义一个特别小的值
207         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
208         static double z0, z1;
209         static bool flag = false;
210         flag = !flag;
211         //flag为假，构造高斯随机变量
212         if (!flag)  return z1 * sigma + mu;
213         double u1, u2;
214         //构造随机变量
215 
216         do{
217             u1 = rand()*(1.0 / RAND_MAX);
218             u2 = rand()*(1.0 / RAND_MAX);
219         } while (u1 <= epsilon);
220         //flag为真构造高斯随机变量X
221         z0 = sqrt(-2.0*log(u1))*cos(2 * CV_PI * u2);
222         z1 = sqrt(-2.0*log(u1))*sin(2 * CV_PI * u2);
223         return z1 * sigma + mu;
224     }
225     /*添加高斯噪声*/
226     bool addGaussianNoise(Mat &image){
227         int channels = image.channels();    //获取图像的通道
228         int rows = image.rows;    //图像的行数
229 
```

```
230     int cols = image.cols*channels;   //图像的总列数
231     //判断图像的连续性
232     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{
233         cols *= rows;
234         rows = 1;
235     }
236     for (int i = 0; i < rows; i++){
237         for (int j = 0; j < cols; j++){ //添加高斯噪声
238             int val = image.ptr<uchar>(i)[j] + GaussianNoise(2, 0.8) * 32;
239             if (val < 0)    val = 0;
240             if (val > 255)  val = 255;
241             image.ptr<uchar>(i)[j] = (uchar)val;
242         }
243     }
244     return true;
245 }
246
247 /*均值滤波
248  *
249  * 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。
250  * 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）
251  *
252  *
253  * */
254 //算数均值
255 Mat digitalMeanFilter(Mat &image, int size) {
256     Mat dst = image.clone();
257     int rows = dst.rows, cols = dst.cols;
258     int start = size / 2;
259     for (int m = start; m < rows - start; m++) {
260         for (int n = start; n < cols - start; n++) {
261             if (dst.channels() == 1)                //灰色图
262             {
263                 int sum = 0;
264                 for (int i = -start + m; i <= start + m; i++)
265                 {
266                     for (int j = -start + n; j <= start + n; j++) {
267                         sum += dst.at<uchar>(i, j);
268                     }
269                 }
270                 dst.at<uchar>(m, n) = uchar(sum / size / size);
271             }
272             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;
```

```
320                     mn = 0;
321
322                     //统计邻域内的几何平均值，邻域大小 5*5
323                     for (int m = left; m <= right; m++) {
324                         row = i + m;
325                         for (int n = left; n <= right; n++) {
326                             col = j + n;
327                             if (row >= 0 && row < h && col >= 0 && col < w)
    {
328                                 int s = image.at<uchar>(row, col);
329                                 mul = mul * (s == 0 ? 1 : s); //邻域内的非零像
    素点相乘，最小值设定为1
330                                 mn++;
331                             }
332                         }
333                     }
334                     //计算 1/mn 次方
335                     dc = pow(mul, 1.0 / mn);
336                     //统计成功赋给去噪后图像。
337                     int res = (int)dc;
338                     dst.at<uchar>(i, j) = res;
339                 }
340                 else
341                 {
342                     double multi[3] = { 1.0,1.0,1.0 };
343                     mn = 0;
344                     Vec3b pixel;
345
346                     for (int m = left; m <= right; m++)
347                     {
348                         row = i + m;
349                         for (int n = left; n <= right; n++)
350                         {
351                             col = j + n;
352                             if (row >= 0 && row < h && col >= 0 && col < w)
353                             {
354                                 pixel = image.at<Vec3b>(row, col);
355                                 for (int k = 0; k < image.channels(); k++)
356                                 {
357                                     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
358                                 }
359                                 mn++;
360                             }
361                         }
362                     }
363                     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;
```

```
409                 //统计成功赋给去噪后图像。
410                 dst.at<uchar>(i, j) = d;
411             }
412         }
413         return dst;
414     }
415 
416     //逆谐波均值
417     Mat inverseHarmonicMeanFilter(Mat &image, int size, double Q){
418         Mat dst = image.clone();
419         int row, col;
420         int h = image.rows;
421         int w = image.cols;
422         double sum;
423         double sum1;
424         double dc;
425         //double Q = 2;
426         //计算每个像素的去噪后 color 值
427         for (int i = 0; i < image.rows; i++) {
428             for (int j = 0; j < image.cols; j++) {
429                 sum = 0.0;
430                 sum1 = 0.0;
431                 //统计邻域
432                 int left = -size/2;
433                 int right = size/2;
434                 for (int m = left; m <= right; m++) {
435                     row = i + m;
436                     for (int n = left; n <= right; n++) {
437                         col = j + n;
438                         if (row >= 0 && row < h && col >= 0 && col < w) {
439 
440                             int s = image.at<uchar>(row, col);
441                             sum = sum + pow(s, Q + 1);
442                             sum1 = sum1 + pow(s, Q);
443                         }
444                     }
445                 }
446                 //计算 1/mn 次方
447                 int d;
448                 dc = sum1 == 0 ? 0 : (sum / sum1);
449                 d = (int)dc;
450                 //统计成功赋给去噪后图像。
451                 dst.at<uchar>(i, j) = d;
452             }
453         }
454         return dst;
455     }
```

```
/*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++) {
```

```
547                 //统计邻域
548                 wsize = 1;
549                 while (wsize <= size / 2) {
550                     Zmin = 255.0;
551                     Zmax = 0.0;
552                     Zmed = 0.0;
553                     int Zxy = image.at<uchar>(i, j);
554                     int mn = 0;
555                     for (int m = -wsize; m <= wsize; m++) {
556                         row = i + m;
557                         for (int n = -wsize; n <= wsize; n++) {
558                             col = j + n;
559                             if (row >= 0 && row < h && col >= 0 && col < w)
    {
560                                 int s = image.at<uchar>(row, col);
561                                 if (s > Zmax) {
562                                     Zmax = s;
563                                 }
564                                 if (s < Zmin) {
565                                     Zmin = s;
566                                 }
567                                 Zmed = Zmed + s;
568                                 mn++;
569                             }
570                         }
571                     }
572                     Zmed = Zmed / mn;
573                     int d;
574                     if ((Zmed - Zmin) > 0 && (Zmed - Zmax) < 0) {
575                         if ((Zxy - Zmin) > 0 && (Zxy - Zmax) < 0) {
576                             d = Zxy;
577                         } else {
578                             d = Zmed;
579                         }
580                         dst.at<uchar>(i, j) = d;
581                         break;
582                     } else {
583                         wsize++;
584                         if (wsize > size / 2) {
585                             int d;
586                             d = Zmed;
587                             dst.at<uchar>(i, j) = d;
588                             break;
589                         }
590                     }
591                 }
592             }
```

```
593         }
594         return dst;
595     }
596 
597 
598     /*彩色图像均值滤波
599      *
600      * 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。
601      * 模板大小为 5*5。
602      *
603      *
604      * */
```