高级图像处理与分析课程实验报告3

学号	姓名	日期
SA18225428	许强	2019.04.03

实验名称	灰度变换
实验内容	1、利用均值模板平滑灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 33、55 和 99 尺寸的均值模板平滑灰度图像 2、利用高斯模板平滑灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 33、55 和 99 尺寸的高斯模板平滑灰度图像 3、利用 Laplacian、Robert、Sobel 模板锐化灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 Laplacian、Robert、 Sobel 模板锐化灰度图像 4、利用高提升滤波算法增强灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，设计高提升滤波算法增 强图像 5、利用均值模板平滑彩色图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，利 用 33、55 和 99 尺寸的均值模板平滑彩色图像 6、利用高斯模板平滑彩色图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分 别利用 33、55 和 99 尺寸的高斯模板平滑彩色图像 7、利用 Laplacian、Robert、Sobel 模板锐化灰度图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分 别利用 Laplacian、Robert、Sobel 模板锐化彩色图像
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。	7个模块全部完成

注意要贴出 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）	参考了网上的高提升滤波算法
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 2019-03-28.
//

#include<iostream>
```

```cpp
#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>

using namespace std;
using namespace cv;


bool Filter(string &, int, int, int, string);
bool enhanceFilter(string &, int flag = 0);

int main(){
    string str = "/Volumes/数据/图片/2k/lostwall.jpg";
    cout << "3*3:" << Filter(str, 3, 3, 0, "mean") << endl;
    cout << "5*5:" << Filter(str, 5, 5, 0, "mean") << endl;
    cout << "9*9:" << Filter(str, 9, 9, 0, "mean") << endl;
    cout << "3*3:" << Filter(str, 3, 3, 0, "Gaussi") << endl;
    cout << "5*5:" << Filter(str, 5, 5, 0, "Gaussi") << endl;
    cout << "9*9:" << Filter(str, 9, 9, 0, "Gaussi") << endl;
    cout << "Laplacian:" << Filter(str, 0, 0, 0, "Laplacian") << endl;
    cout << "Robert:" << Filter(str, 0, 0, 0, "Robert") << endl;
    cout << "Sobel:" << Filter(str, 0, 0, 0, "Sobel") << endl;

    cout << "enhanceFilter:" << enhanceFilter(str)<< endl;
    cout << "enhanceFilter:" << enhanceFilter(str,1)<< endl;

    cout << "3*3:" << Filter(str, 3, 3, 1, "mean") << endl;
    cout << "5*5:" << Filter(str, 5, 5, 1, "mean") << endl;
    cout << "9*9:" << Filter(str, 9, 9, 1, "mean") << endl;
    cout << "3*3:" << Filter(str, 3, 3, 1, "Gaussi") << endl;
    cout << "5*5:" << Filter(str, 5, 5, 1, "Gaussi") << endl;
    cout << "9*9:" << Filter(str, 9, 9, 1, "Gaussi") << endl;
    cout << "Laplacian:" << Filter(str, 0, 0, 1, "Laplacian") << endl;
    cout << "Robert:" << Filter(str, 0, 0, 1, "Robert") << endl;
    cout << "Sobel:" << Filter(str, 0, 0, 1, "Sobel") << endl;

}


```

```
53
54
55
56  /*
57   * 利用均值模板平滑灰度图像。
58   *  具体内容：利用 OpenCV 对图像像素进行操作，分别利用 3*3、5*5 和 9*9 尺寸的均值模板平滑灰度图像。
59   * 利用高斯模板平滑灰度图像。
60   *  具体内容：利用 OpenCV 对图像像素进行操作，分别利用 3*3、5*5 和 9*9 尺寸的高斯模板平滑灰度图像。
61   * 利用 Laplacian、Robert、Sobel 模板锐化灰度图像。
62   *  具体内容：利用 OpenCV 对图像像素进行操作，分别利用 Laplacian、Robert、Sobel 模板锐化灰度图像。
63   * 利用高提升滤波算法增强灰度图像。
64   *  具体内容：利用 OpenCV 对图像像素进行操作，设计高提升滤波算法增 强图像。
65   * 利用均值模板平滑彩色图像。
66   *  具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，利 用 3*3、5*5 和 9*9 尺寸的均值模板平滑彩色图像。
67   * 利用高斯模板平滑彩色图像。
68   *  具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分 别利用 3*3、5*5 和 9*9 尺寸的高斯模板平滑彩色图像。
69   * 利用 Laplacian、Robert、Sobel 模板锐化彩色图像
70   *  具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分 别利用 Laplacian、Robert、Sobel 模板锐化彩色图像。
71   * */
72  bool Filter(String &src, int a, int b, int flag, string model){
73      Mat image = imread(src,flag);
74      imshow("input", image);
75      Mat res = image.clone();
76      if(model == "mean"){
77          blur(image, res, Size(a,b));
78      } else if (model == "Gaussi"){
79          GaussianBlur(image, res, Size(a,b),1);
80      } else if (model == "Laplacian"){
81          Mat kernel = (Mat_<float>(3, 3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
82          filter2D(image, res, image.depth(), kernel);
83      } else if (model == "Robert"){
84          if(flag == 0){
85              for (int i = 0; i < image.rows - 1; i++) {
86                  for (int j = 0; j < image.cols - 1; j++) {
87                      //根据公式计算
88                      int t1 = (image.at<uchar>(i, j) -
89                                image.at<uchar>(i + 1, j + 1))*
90                               (image.at<uchar>(i, j) -
91                                image.at<uchar>(i + 1, j + 1));
92                      int t2 = (image.at<uchar>(i + 1, j) -
```

```
                                image.at<uchar>(i, j + 1))*
                               (image.at<uchar>(i + 1, j) -
                                image.at<uchar>(i, j + 1));
                    //计算g (x,y)
                    res.at<uchar>(i, j) = (uchar)sqrt(t1 + t2);
                }
            }
        } else if(flag == 1){
            CvScalar t1, t2, t3, t4, t;
            auto res2 = IplImage(image);

            for (int i = 0; i < res2.height - 1; i++)
            {
                for (int j = 0; j < res2.width - 1; j++)
                {
                    t1 = cvGet2D(&res2, i, j);
                    t2 = cvGet2D(&res2, i + 1, j + 1);
                    t3 = cvGet2D(&res2, i, j + 1);
                    t4 = cvGet2D(&res2, i + 1, j);


                    for (int k = 0; k < 3; k++)
                    {
                        int t7 = (int)((t1.val[k] - t2.val[k])*(t1.val[k] - t2.val[k]) + (t4.val[k] - t3.val[k])*(t4.val[k] - t3.val[k]));
                        t.val[k] = sqrt(t7);
                    }
                    cvSet2D(&res2, i, j, t);
                }
            }
            res = cvarrToMat(&res2);
        }
    } else if (model == "Sobel"){
        Mat grad_x, grad_y;
        Mat abs_grad_x, abs_grad_y;

        Sobel(image, grad_x, image.depth(), 1, 0, 3, 1, 1, BORDER_DEFAULT);
        convertScaleAbs(grad_x, abs_grad_x);
        imshow("X-Sobel", abs_grad_x);

        Sobel(image, grad_y, image.depth(), 0, 1, 3, 1, 1, BORDER_DEFAULT);
        convertScaleAbs(grad_y, abs_grad_y);
        imshow("Y-Sobel", abs_grad_y);

```

```
136             //【5】合并梯度(近似)
137             addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, res);
138         }
139
140
141
142
143
144
145         imshow(model+"Filter", res);
146         waitKey(0);
147         destroyAllWindows();
148         return true;
149     }
150
151     void EnhanceFilter(Mat img, Mat &dst, double dProportion, int nTempH,
    int nTempW, int nTempMY, int nTempMX, float *pfArray, float fCoef){
152
153
154         int i, j, nHeight = img.rows, nWidth = img.cols;
155         vector<vector<int>> GrayMat1, GrayMat2, GrayMat3;//暂存按比例叠加图像,
    R,G,B三通道
156         vector<int> vecRow1(nWidth, 0), vecRow2(nWidth, 0), vecRow3(nWidth,
    0);
157         for (i = 0; i < nHeight; i++)
158         {
159             GrayMat1.push_back(vecRow1);
160             GrayMat2.push_back(vecRow2);
161             GrayMat3.push_back(vecRow3);
162         }
163
164         //锐化图像，输出带符号响应，并与原图像按比例叠加
165         for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
166         {
167             for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
168             {
169                 float fResult1 = 0;
170                 float fResult2 = 0;
171                 float fResult3 = 0;
172                 for (int k = 0; k < nTempH; k++)
173                 {
174                     for (int l = 0; l < nTempW; l++)
175                     {
176                         //分别计算三通道加权和
177                         fResult1 += img.at<Vec3b>(i, j)[0] *
    pfArray[k*nTempW + 1];
```

```
                    fResult2 += img.at<Vec3b>(i, j)[1] * pfArray[k*nTempW + 1];
                    fResult3 += img.at<Vec3b>(i, j)[2] * pfArray[k*nTempW + 1];
                }
            }

            //三通道加权和分别乘以系数并限制响应范围，最后和原图像按比例混合
            fResult1 *= fCoef;
            if (fResult1 > 255)
                fResult1 = 255;
            if (fResult1 < -255)
                fResult1 = -255;
            GrayMat1[i][j] = dProportion * img.at<Vec3b>(i, j)[0] + fResult1 + 0.5;

            fResult2 *= fCoef;
            if (fResult2 > 255)
                fResult2 = 255;
            if (fResult2 < -255)
                fResult2 = -255;
            GrayMat2[i][j] = dProportion * img.at<Vec3b>(i, j)[1] + fResult2 + 0.5;

            fResult3 *= fCoef;
            if (fResult3 > 255)
                fResult3 = 255;
            if (fResult3 < -255)
                fResult3 = -255;
            GrayMat3[i][j] = dProportion * img.at<Vec3b>(i, j)[2] + fResult3 + 0.5;
        }
    }
    int nMax1 = 0, nMax2 = 0, nMax3 = 0;//三通道最大灰度和值
    int nMin1 = 65535, nMin2 = 65535, nMin3 = 65535;//三通道最小灰度和值
    //分别统计三通道最大值最小值
    for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
    {
        for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
        {
            if (GrayMat1[i][j] > nMax1)
                nMax1 = GrayMat1[i][j];
            if (GrayMat1[i][j] < nMin1)
                nMin1 = GrayMat1[i][j];

            if (GrayMat2[i][j] > nMax2)
                nMax2 = GrayMat2[i][j];
```

```
            if (GrayMat2[i][j] < nMin2)
                nMin2 = GrayMat2[i][j];

            if (GrayMat3[i][j] > nMax3)
                nMax3 = GrayMat3[i][j];
            if (GrayMat3[i][j] < nMin3)
                nMin3 = GrayMat3[i][j];
        }
    }
    //将按比例叠加后的三通道图像取值范围重新归一化到[0,255]
    int nSpan1 = nMax1 - nMin1, nSpan2 = nMax2 - nMin2, nSpan3 = nMax3 - nMin3;
    for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
    {
        for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
        {
            int br, bg, bb;
            if (nSpan1 > 0)
                br = (GrayMat1[i][j] - nMin1) * 255 / nSpan1;
            else if (GrayMat1[i][j] <= 255)
                br = GrayMat1[i][j];
            else
                br = 255;
            dst.at<Vec3b>(i, j)[0] = br;

            if (nSpan2 > 0)
                bg = (GrayMat2[i][j] - nMin2) * 255 / nSpan2;
            else if (GrayMat2[i][j] <= 255)
                bg = GrayMat2[i][j];
            else
                bg = 255;
            dst.at<Vec3b>(i, j)[1] = bg;

            if (nSpan3 > 0)
                bb = (GrayMat3[i][j] - nMin3) * 255 / nSpan3;
            else if (GrayMat3[i][j] <= 255)
                bb = GrayMat3[i][j];
            else
                bb = 255;
            dst.at<Vec3b>(i, j)[2] = bb;
        }
    }
}

bool enhanceFilter(string &src, int flag){
    Mat img = imread(src,flag);
    imshow("input", img);
```

```
266         Mat dst = img.clone();
267         //常用滤波模板数组
268         //平均平滑1/9
269         float Template_Smooth_Avg[9] = { 1, 1, 1, 1, 1, 1, 1, 1, 1 };
270         //Gauss平滑1/16
271         float Template_Smooth_Gauss[9] = { 1, 2, 1, 2, 4, 2, 1, 2, 1 };
272         //Sobel垂直边缘检测
273         float Template_Smooth_HSobel[9] = { -1, 0, 1, -2, 0, 2, -1, 0, 1 };
274         //Sobel水平边缘检测
275         float Template_Smooth_VSobel[9] = { -1, -2, -1, 0, 0, 0, 1, 2, 1 };
276         //LOG边缘检测
277         float Template_Log[25] = { 0, 0, -1, 0, 0, 0, -1, -2, -1, 0, -1, -2,
    16, -2, -1, 0, -1, -2, -1, 0, 0, 0, -1, 0, 0 };
278         //Laplacian边缘检测
279         float Template_Laplacian1[9] = { 0, -1, 0, -1, 4, -1, 0, -1, 0 };//
    对90度各向同性
280         float Template_Laplacian2[9] = { -1, -1, -1, -1, 8, -1, -1, -1, -1
    };//对45度各向同性
281
     /*************************************************************************
    ************************************
282         高提升滤波
283         dProportion: 高提升滤波中原图像的混合比例
284         nTempH: 模板高度, nTempW: 模板宽度
285         nTempMY: 模板中心元素坐标, nTempMX: 模板中心元素坐标
286         fpArray: 指向模板数组的指针, 可以选取不同模板实现不同滤波的高提升版本
287         fCoef: 模板系数
288
     *************************************************************************
    ************************************/
289         EnhanceFilter(img, dst, 1.8, 3, 3, 1, 1, Template_Laplacian2, 1);
290
291         imshow("Enhance Laplacian", dst);
292         waitKey(0);
293         destroyAllWindows();
294         return true;
295     }
```