高级图像处理与分析课程实验报告2

学号	姓名	日期
SA18225428	许强	2019.04.02

实验名称	灰度变换
实验内容	1、计算灰度图像的归一化直方图。 具体内容：利用 OpenCV 对图像像素进行操作，计算归一化直方图.并在 窗口中以图形的方式显示出来 2、灰度图像直方图均衡处理 具体内容：通过计算归一化直方图,设计算法实现直方图均衡化处理。 3、彩色图像直方图均衡处理 具体内容： 在灰度图像直方图均衡处理的基础上实现彩色直方图均衡 处理。
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。 注意要贴出 每个实验的 核心代码）	3个模块全部完成
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）	rectangle绘制方形图像用于显示直方图。
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>
#include<string>
#include<cmath>

#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
```

```cpp
#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 normalizeHistogramImage(string &, int flag = 0);
bool equalizeHistogramImage(string &,int flag = 0);
Mat equalizeHistogram(Mat&);
bool HisRGB(Mat &);
bool HisGray(Mat &);
bool His2D(Mat &);


int main(){
    string str = "/Volumes/数据/图片/2k/lostwall.jpg";
    cout << "1.normalizeHistogramImage:" << normalizeHistogramImage(str) << endl;
    cout << "2.equalizeHistogramImage-gray:" << equalizeHistogramImage(str,0) << endl;;
    cout << "3.equalizeHistogramImage-color:" << equalizeHistogramImage(str,1) << endl;

}

/*计算灰度图像的归一化直方图
 *
 * 具体内容:
 *  利用 OpenCV 对图像像素进行操作，计算归一化直方图.并在窗口中以图形的方式显示出来
 *
 * */
bool normalizeHistogramImage(string &src, int flag){
    Mat image = imread(src,flag);
    HisGray(image);

    image = imread(src,1);
    HisRGB(image);
    His2D(image);

    return true;
}

/*图像直方图均衡处理
```

```
 *
 * 具体内容:
 *  通过计算归一化直方图,设计算法实现直方图均衡化处理.
 *  在灰度图像直方图均衡处理的基础上实现彩色直方图均衡处理。
 *
 * */
bool equalizeHistogramImage(string &src,int flag){

    if(flag == 0){
        Mat image = imread(src,flag);
        imshow("input", image);

        Mat res = equalizeHistogram(image);
        imshow("equalizeHistogramImageGray", res);
        waitKey(0);
        destroyAllWindows();
        HisGray(res);
    } else if (flag == 1){
        Mat image = imread(src,flag);
        imshow("input", image);
        Mat res;
        vector<Mat> t;
        split(image, t);
        for (int i = 0; i < 3; i++){
            t[i] = equalizeHistogram(t[i]);
        }
        merge(t, res);  //对RGB三通道各自均衡化后，再组合输出结果

        imshow("equalizeHistogramImageColor", res);
        waitKey(0);
        destroyAllWindows();
        HisRGB(res);
    }
    return true;
}


/*函数*/
Mat equalizeHistogram(Mat& input){
    Mat res = input.clone();
    int gray[256] = {0}; //记录每个灰度级别下的像素个数

    double gray_prob[256] = {0}; //记录灰度密度
    double gray_count[256] = {0}; //记录累计密度

    int gray_equalized[256] = {0}; //均衡化后的灰度值
    int gray_sum = res.rows * res.cols; //像素总数

    for(int i = 0; i < res.rows; i++){
        auto * p = res.ptr<uchar>(i);
        for(int j = 0; j < res.cols; j++){
            //统计每个灰度下的像素个数
            gray[p[j]]++;
        }
    }

    for(int i = 0; i < 256; i++){
        //统计灰度频率
        gray_prob[i] = (double)gray[i]/gray_sum;
    }

    gray_count[0] = gray_prob[0];
    for(int i = 1; i < 256; i++){
        //计算累计密度
        gray_count[i] = gray_count[i-1] + gray_prob[i];
    }
    for (int i = 0; i < 256; i++){
        //重新计算均衡化后的灰度值，四舍五入。参考公式：(N-1)*T+0.5
        gray_equalized[i] = (int)(gray_count[i] * 255 + 0.5);
    }

    for(int i = 0; i < res.rows; i++){
        auto * p = res.ptr<uchar>(i);
        for(int j = 0; j < res.cols; j++){
            //直方图均衡化,更新原图每个点的像素值
            p[j] = gray_equalized[p[j]];
        }
    }
    return res;
}

bool HisRGB(Mat &image)
{
    imshow("input", image);
    int bins = 256;

    int hist_size[] = { bins };
    float range[] = { 0, 256 };
    const float* ranges[] = { range };

    MatND hist_r, hist_g, hist_b;
    int channels_r[] = { 0 };
    calcHist(&image, 1, channels_r, Mat(), // do not use mask
             hist_r, 1, hist_size, ranges,
             true, // the histogram is uniform
             false);

    int channels_g[] = { 1 };
    calcHist(&image, 1, channels_g, Mat(), // do not use mask
             hist_g, 1, hist_size, ranges,
             true, // the histogram is uniform
             false);

    int channels_b[] = { 2 };
    calcHist(&image, 1, channels_b, Mat(), // do not use mask
             hist_b, 1, hist_size, ranges,
             true, // the histogram is uniform
             false);

    double max_val_r, max_val_g, max_val_b;
    minMaxLoc(hist_r, 0, &max_val_r, 0, 0);
    minMaxLoc(hist_g, 0, &max_val_g, 0, 0);
    minMaxLoc(hist_b, 0, &max_val_b, 0, 0);
    int scale = 1;

    int hist_height = 256;
    Mat colorHis = Mat::zeros(hist_height, bins * 3, CV_8UC3);
    for (int i = 0; i < bins; i++)
    {
        float bin_val_r = hist_r.at<float>(i);
        float bin_val_g = hist_g.at<float>(i);
        float bin_val_b = hist_b.at<float>(i);
        int intensity_r = cvRound(bin_val_r*hist_height / max_val_r); //要绘制的高度
        int intensity_g = cvRound(bin_val_g*hist_height / max_val_g); //要绘制的高度
        int intensity_b = cvRound(bin_val_b*hist_height / max_val_b); //要绘制的高度
        rectangle(colorHis, Point(i*scale, hist_height - 1),
                  Point((i + 1)*scale - 1, hist_height - intensity_r),
                  CV_RGB(255, 0, 0));

        rectangle(colorHis, Point((i + bins)*scale, hist_height - 1),
                  Point((i + bins + 1)*scale - 1, hist_height - intensity_g),
                  CV_RGB(0, 255, 0));

        rectangle(colorHis, Point((i + bins * 2)*scale, hist_height - 1),
                  Point((i + bins * 2 + 1)*scale - 1, hist_height - intensity_b),
                  CV_RGB(0, 0, 255));

    }
    namedWindow("HisRGB", WINDOW_AUTOSIZE); // Create a window for display.
    imshow("HisRGB", colorHis);
    waitKey(0);
    destroyAllWindows();
    return true;

}

bool HisGray(Mat &image){

    imshow("input", image);
    int bins = 256;

    int hist_size[] = { bins };
    float range[] = { 0, 256 };
    const float* ranges[] = { range };

    MatND hist;
    int channels[] = { 0 };
    calcHist(&image, 1, channels, Mat(), // do not use mask
             hist, 1, hist_size, ranges,
             true, // the histogram is uniform
             false);

    double max_val;
    minMaxLoc(hist, 0, &max_val, 0, 0);

    int scale = 2;
    int hist_height = 256;
    Mat hist_img = Mat::zeros(hist_height, bins*scale, CV_8UC3);
    for (int i = 0; i < bins; i++)
    {
        float bin_val = hist.at<float>(i);
        int intensity = cvRound(bin_val*hist_height / max_val);  //要绘制的高度
        rectangle(hist_img, Point(i*scale, hist_height - 1),
                  Point((i + 1)*scale - 1, hist_height - intensity),
                  CV_RGB(255, 255, 255));
    }

    imshow("HistogramGray", hist_img);
    waitKey(0);
    destroyAllWindows();
    return true;
}

bool His2D(Mat &image){
    imshow("input", image);

    int hbins = 256, sbins = 256;
    int histSize[] = { hbins, sbins };

    float hranges[] = { 0, 256 };
    float sranges[] = { 0, 256 };
    const float* ranges[] = { hranges, sranges };
    MatND hist;

    int channels[] = { 0, 1 };
    calcHist(&image, 1, channels, Mat(), // do not use mask
             hist, 2, histSize, ranges,
             true, // the histogram is uniform
             false);
    double maxVal = 0;
    minMaxLoc(hist, 0, &maxVal, 0, 0);
    int scale = 2;
    Mat histImg = Mat::zeros(sbins*scale, hbins*scale, CV_8UC3);
    for (int h = 0; h < hbins; h++)
        for (int s = 0; s < sbins; s++)
        {
            float binVal = hist.at<float>(h, s);
            int intensity = cvRound(binVal * 255 / maxVal);
            rectangle(histImg, Point(h*scale, s*scale),
                      Point((h + 1)*scale - 1, (s + 1)*scale - 1),
                      Scalar::all(intensity),
                      FILLED);
        }
    imshow("His2D", histImg);
    waitKey(0);
    destroyAllWindows();
    return true;
}
```