Datawhale 计算机视觉基础-图像处理（上）-Task03 彩色空间互转

3.1 简介

图像彩色空间互转在图像处理中应用非常广泛，而且很多算法只对灰度图有效；另外，相比RGB，其他颜色空间(比如HSV、HSI)更具可分离性和可操作性，所以很多图像算法需要将图像从RGB转为其他颜色空间，所以图像彩色互转是十分重要和关键的。

3.2 学习目标

- 了解相关颜色空间的基础知识
- 理解彩色空间互转的理论
- 掌握OpenCV框架下颜色空间互转API的使用

3.3 内容介绍

1.相关颜色空间的原理介绍

2.颜色空间互转理论的介绍

3.OpenCV代码实践

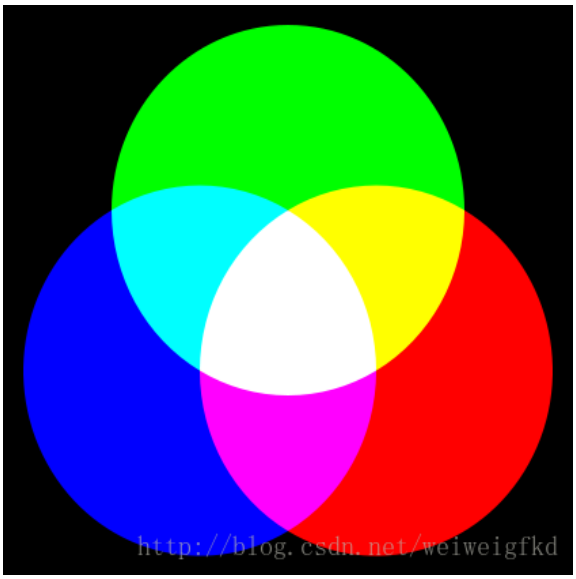
4.动手实践并打卡（读者完成）

3.4 算法理论介绍与资料推荐

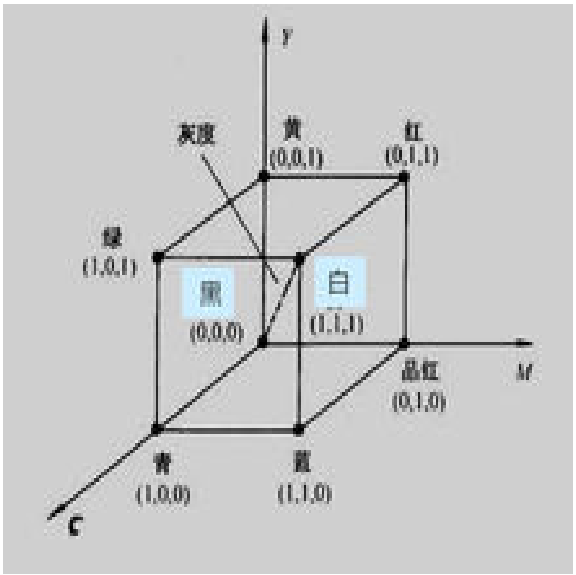
3.4.1 RGB与灰度图互转

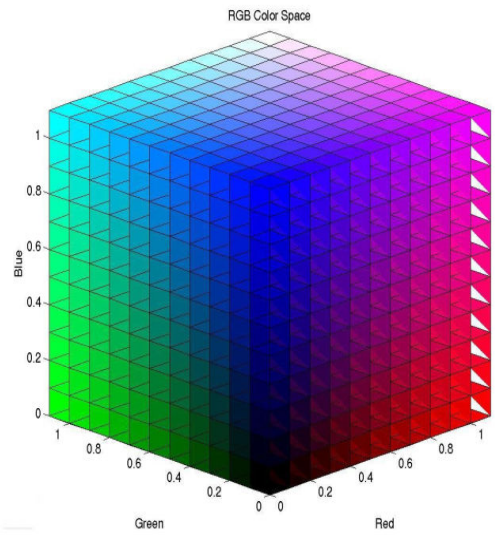
RGB（红绿蓝）是依据人眼识别的颜色定义出的空间，可表示大部分颜色。但在科学研究一般不采用RGB颜色空间，因为它的细节难以进行数字化的调整。它将色调，亮度，饱和度三个量放在一起表示，很难分开。它是最通用的面向硬件的彩色模型。该模型用于彩色监视器和一大类彩色视频摄像。

RGB颜色空间 基于颜色的加法混色原理，从黑色不断叠加Red，Green，Blue的颜色，最终可以得到白色，如图：



将R、G、B三个通道作为笛卡尔坐标系中的X、Y、Z轴，就得到了一种对于颜色的空间描述，如图：





对于彩色图转灰度图，有一个很著名的心理学公式：

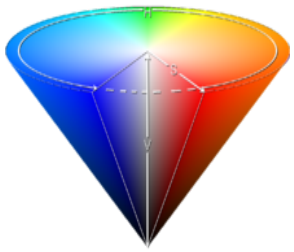
$$Gray = R * 0.299 + G * 0.587 + B * 0.114$$

3.4.2 RGB与HSV互转

HSV是一种将RGB色彩空间中的点在倒圆锥体中的表示方法。HSV即色相(Hue)、饱和度(Saturation)、明度(Value)，又称HSB(B即Brightness)。色相是色彩的基本属性，就是平常说的颜色的名称，如红色、黄色等。饱和度（S）是指色彩的纯度，越高色彩越纯，低则逐渐变灰，取0-100%的数值。明度（V），取0-max(计算机中HSV取值范围和存储的长度有关)。HSV颜色空间可以用一个圆锥空间模型来描述。圆锥的顶点处，V=0，H和S无定义，代表黑色。圆锥的顶面中心处V=max，S=0，H无定义，代表白色。

RGB颜色空间中，三种颜色分量的取值与所生成的颜色之间的联系并不直观。而HSV颜色空间，更类似于人类感觉颜色的方式，封装了关于颜色的信息：“这是什么颜色？深浅如何？明暗如何？

HSV模型

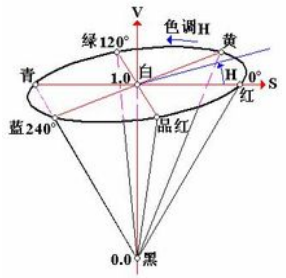


这个模型就是按色彩、深浅、明暗来描述的。

H是色彩；

S是深浅， S = 0时，只有灰度；

V是明暗，表示色彩的明亮程度，但与光强无直接联系。



应用：可以用于偏光矫正、去除阴影、图像分割等

1.RGB2HSV

$$V \leftarrow \max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-\min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-\min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-\min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-\min(R,G,B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \le V \le 1, 0 \le S \le 1, 0 \le H \le 360$.

或

$$h = \begin{cases} 0^\circ & \text{if } max = min \\ 60^\circ \times \frac{g-b}{max-min} + 0^\circ, & \text{if } max = r \text{ and } g \ge b \\ 60^\circ \times \frac{g-b}{max-min} + 360^\circ, & \text{if } max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{max-min} + 120^\circ, & \text{if } max = g \\ 60^\circ \times \frac{r-g}{max-min} + 240^\circ, & \text{if } max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } max = 0 \\ \frac{max-min}{max} = 1 - \frac{min}{max}, & \text{otherwise} \end{cases}$$

$$v = max$$

2.HSV2RGB

$$h_i \equiv \left\lfloor \frac{h}{60} \right\rfloor \pmod 6$$

$$f = \frac{h}{60} - h_i$$

$$p = v \times (1 - s)$$

$$q = v \times (1 - f \times s)$$

$$t = v \times (1 - (1 - f) \times s)$$

对于每个颜色向量 $(r,\ g,\ b)$,

$$(r,g,b) = \begin{cases} (v,t,p), & \text{if } h_i = 0 \\ (q,v,p), & \text{if } h_i = 1 \\ (p,v,t), & \text{if } h_i = 2 \\ (p,q,v), & \text{if } h_i = 3 \\ (t,p,v), & \text{if } h_i = 4 \\ (v,p,q), & \text{if } h_i = 5 \end{cases}$$

3.5 基于OpenCV的实现

- 工具：OpenCV3.1.0+VS2013
- 平台：WIN10

函数原型（c++）

> void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)

- src: 输入图像
- dst: 输出图像
- code: 颜色空间转换标识符
  - OpenCV2的CV_前缀宏命名规范被OpenCV3中的COLOR_式的宏命名前缀取代
  - 注意RGB色彩空间默认通道顺序为BGR
  - 具体可以参考： enum cv::ColorConversionCode部分
- dstCn: 目标图像的通道数，该参数为0时，目标图像根据源图像的通道数和具体操作自动决定

实现示例（c++）

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
// main
int main( int argc, char** argv )
{
    // Load image
    cv::Mat srcImage = cv::imread("1.jpg"), dstImage;

    // RGB2GHSV
    cv::cvtColor(srcImage, dstImage, cv::COLOR_BGR2hHSV);
    imshow("Lab Space", dstImage);

    //RGB2GRAY
    cv::cvtColor(srcImage, dstImage, cv::COLOR_BGR2GRAY);
    imshow("Gray Scale", dstImage);

    cv::waitKey();

    return 0;
}
```

进阶实现(根据原理自己实现)

- 1.RGB2GRAY

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
```

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

cv::Mat RGB2GRAY(cv::Mat src, bool accelerate=false){
        CV_Assert(src.channels()==3);
        cv::Mat dst = cv::Mat::zeros(src.size(), CV_8UC1);
        cv::Vec3b rgb;
        int r = src.rows;
        int c = src.cols;

          for (int i = 0; i < r; ++i){
                 for (int j = 0; j < c; ++j){
                        rgb = src.at<cv::Vec3b>(i, j);
                        uchar B = rgb[0]; uchar G = rgb[1]; uchar R = rgb[2];
                        if (accelerate = false){
                                dst.at<uchar>(i, j) = R*0.299 + G*0.587 + B*0.114;   //原式
                        }
                        else{
                                dst.at<uchar>(i, j) = (R * 4898 + G * 9618 + B * 1868) >> 14;  //优化
                        }
                 }
           }
        return dst;
}

int main(){
        cv::Mat src = cv::imread("I:\\Learning-and-Practice\\2019Change\\Image process
algorithm\\Img\\lena.jpg");

        if (src.empty()){
                return -1;
        }
        cv::Mat dst,dst1;

        //opencv自带
        double t2 = (double)cv::getTickCount(); //测时间
        cv::cvtColor(src, dst1, CV_RGB2GRAY);
        t2 = (double)cv::getTickCount() - t2;
        double time2 = (t2 *1000.) / ((double)cv::getTickFrequency());
        std::cout << "Opencv_rgb2gray=" << time2 << " ms. " << std::endl << std::endl;

        //RGB2GRAY
        double t1 = (double)cv::getTickCount(); //测时间
        dst = RGB2GRAY(src, true);
        t1 = (double)cv::getTickCount() - t1;
        double time1 = (t1 *1000.) / ((double)cv::getTickFrequency());
        std::cout << "My_rgb2gray=" << time1 << " ms. " << std::endl << std::endl;


        cv::namedWindow("src", CV_WINDOW_NORMAL);
        imshow("src", src);
        cv::namedWindow("My_rgb2gray", CV_WINDOW_NORMAL);
        imshow("My_rgb2gray", dst);
        cv::namedWindow("Opencv_rgb2gray", CV_WINDOW_NORMAL);
        imshow("Opencv_rgb2gray", dst1);
        cv::waitKey(0);
        return 0;

}
```

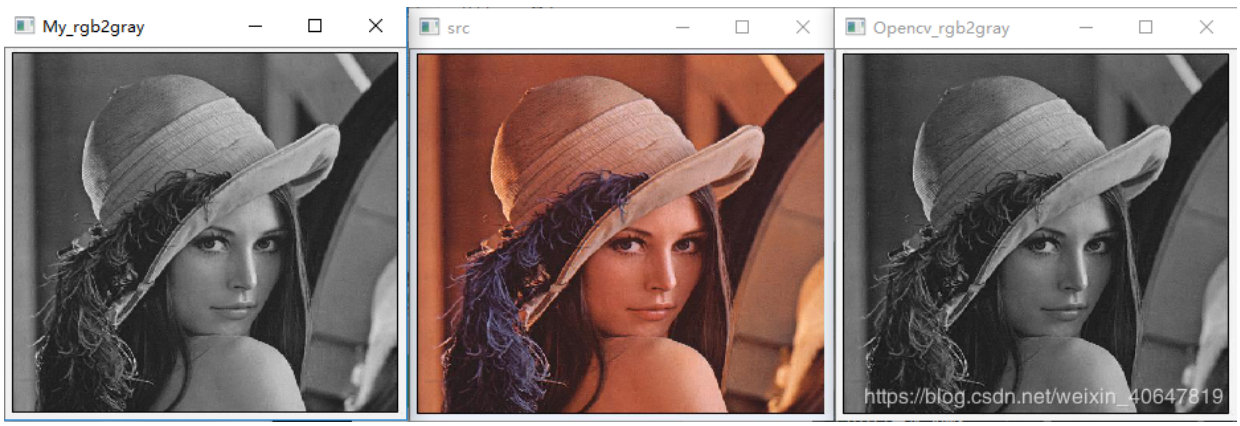
- 2.RGB2HSV/HSV2RGB

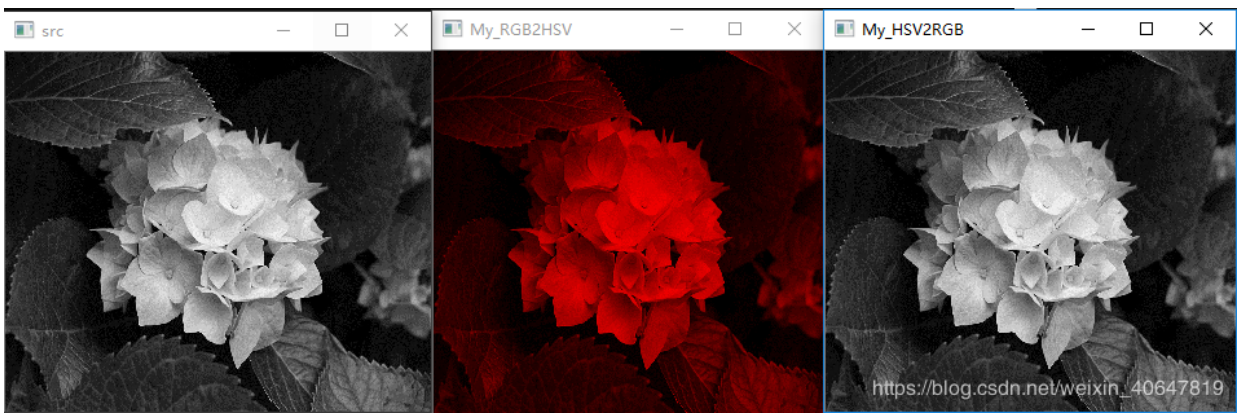
```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
using namespace cv;


Mat RGB2HSV(Mat src) {
        int row = src.rows;
        int col = src.cols;
        Mat dst(row, col, CV_32FC3);
        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        float b = src.at<Vec3b>(i, j)[0] / 255.0;
```

```
                        float g = src.at<Vec3b>(i, j)[1] / 255.0;
                        float r = src.at<Vec3b>(i, j)[2] / 255.0;
                        float minn = min(r, min(g, b));
                        float maxx = max(r, max(g, b));
                        dst.at<Vec3f>(i, j)[2] = maxx; //V
                        float delta = maxx - minn;
                        float h, s;
                        if (maxx != 0) {
                                s = delta / maxx;
                        }
                        else {
                                s = 0;
                        }
                        if (r == maxx) {
                                h = (g - b) / delta;
                        }
                        else if (g == maxx) {
                                h = 2 + (b - r) / delta;
                        }
                        else if (b==maxx) {
                                h = 4 + (r - g) / delta;
                        }
                        else{
                                h = 0;
                        }
                        h *= 60;
                        if (h < 0)
                                h +=  360;
                        dst.at<Vec3f>(i, j)[0] = h;
                        dst.at<Vec3f>(i, j)[1] = s;
                }
        }

        return dst;
}

Mat HSV2RGB(Mat src) {
        int row = src.rows;
        int col = src.cols;
        Mat dst(row, col, CV_8UC3);
        float r, g, b, h, s, v;
        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        h = src.at<Vec3f>(i, j)[0];
                        s = src.at<Vec3f>(i, j)[1];
                        v = src.at<Vec3f>(i, j)[2];
                        if (s == 0) {
                                r = g = b = v;
                        }
                        else {
                                h /= 60;
                                int offset = floor(h);
                                float f = h - offset;
                                float p = v * (1 - s);
                                float q = v * (1 - s * f);
                                float t = v * (1 - s * (1 - f));
                                switch (offset)
                                {
                                case 0: r = v; g = t; b = p; break;
                                case 1: r = q; g = v; b = p; break;
                                case 2: r = p; g = v; b = t; break;
                                case 3: r = p; g = q; b = v; break;
                                case 4: r = t; g = p; b = v; break;
                                case 5: r = v; g = p; b = q; break;
                                default:
                                        break;
                                }
                        }
                        dst.at<Vec3b>(i, j)[0] = int(b * 255);
                        dst.at<Vec3b>(i, j)[1] = int(g * 255);
                        dst.at<Vec3b>(i, j)[2] = int(r * 255);
                }
        }
        return dst;
}
```

```
int main(){
        cv::Mat src = cv::imread("I:\\Learning-and-Practice\\2019Change\\Image process
algorithm\\Img\\lena.JPG");

        if (src.empty()){
                return -1;
        }
        cv::Mat dst, dst1, dst2;

        ////////opencv自带/////////
        cv::cvtColor(src, dst1, CV_RGB2HSV); //RGB2HSV

        //////////RGB2HSV//////////
        dst = RGB2HSV(src); //RGB2HSV
        dst2 = HSV2RGB(dst); //HSV2BGR

        cv::namedWindow("src", CV_WINDOW_NORMAL);
        imshow("src", src);
        cv::namedWindow("My_RGB2HSV", CV_WINDOW_NORMAL);
        imshow("My_RGB2HSV", dst);
        cv::namedWindow("My_HSV2RGB", CV_WINDOW_NORMAL);
        imshow("My_HSV2RGB", dst2);
        cv::namedWindow("Opencv_RGB2HSV", CV_WINDOW_NORMAL);
        imshow("Opencv_RGB2HSV", dst1);
        cv::waitKey(0);
        return 0;

}
```

效果





相关技术文档、博客、书籍、项目推荐

opencv文档：https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html
博客：https://blog.csdn.net/weixin_40647819/article/details/92596879
https://blog.csdn.net/weixin_40647819/article/details/92660320
python版本：https://www.kancloud.cn/aollo/aolloopencv/263731

3.6 总结

该部分主要讲解彩色空间互转，彩色空间互转是传统图像算法的一个关键技术，学习颜色转换有助于我们理解图像的色域，从而为我们从事CV相关工程技术和科学研究提供一些基础、灵感和思路。

Task03 彩色空间互转 END.

--- ***By: 小武***

> 博客：https://blog.csdn.net/weixin_40647819

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。