图像处理 HW1：基本图像色彩处理

李天宇　　2200013188　　信息科学技术学院

1　调库

调库这事咱们熟啊，只用写一个调整 HSL 三个参数的函数就可以了。

Code Listing 1: 简单调个 OpenCV 的库

```
hls_image = cv.cvtColor(input_image, cv.COLOR_RGB2HLS)
hls_image = HSL_adjust(hls_image, pH, pL, pS)
output_image = cv.cvtColor(hls_image, cv.COLOR_HLS2RGB)

def HSL_adjust(input_image: gr.Image, pH, pL, pS):
    output_image = np.zeros_like(input_image)
    H = input_image[..., 0]
    L = input_image[..., 1]
    S = input_image[..., 2]
    output_image[..., 0] = H * (1+pH) if pH<0 else H + (180-H) * pH
    output_image[..., 1] = L * (1+pL) if pL<0 else L + (255-L) * pL
    output_image[..., 2] = S * (1+pS) if pS<0 else S + (255-S) * pS
    return output_image
```

这里我的理念是，如果 Slider 拉到最左或最右，就把这些值调成范围边界值；0 则保持不变。至于中间，那就线性好了。于是写了这样的式子：

$$y = \begin{cases} a \cdot (1+x), & -1 \le x < 0 \\ a + (b-a) \cdot x, & 0 \le x \le 1 \end{cases} \tag{1}$$

教学网的作业要求中，给出的直接调用是`cvt2.cvtColor(img_np, cv2.COLOR_BGR2HLS)`，为什么是 BGR？我查阅了 opencv 的文档，找到了这段话：“请注意，OpenCV 中的默认颜色格式通常被称为 RGB，但实际上它是 BGR（字节顺序相反）。因此，标准（24 位）彩色图像的第一个字节是一个 8 位蓝色组件，第二个字节是绿色，第三个字节是红色。下一个四个字节将是第二个像素（蓝色，然后是绿色，然后是红色），依此类推。”但我实际操作后，发现似乎 RGB2HLS 更符合色相图。大概 numpy/gradio 中，图像的第三维仍按照 R，G，B 的顺序排列。

2 自己造轮子

既然有公式，轮子应该不难造。numpy 中关于数组的一些操作较为诡异，如不能用常规的 if-else，而是要用`np.where()`，不能用`min()`，而要用`np.minimum()`等。另外，关于`np.where()`，我在使用时也遇到了问题。在运行这段代码时就报错了：

Code Listing 2: 这里为什么会出 bug 呢

```
1    inter = R*R + G*G + B*B - R*G - R*B - B*G
2    theta = np.where(inter==0,0,np.arccos((2*R-G-B)/(2*np.sqrt(inter))))
```

报错信息如下：

```
pic_proc\functions.py:46: RuntimeWarning: invalid value encountered in divide
theta = np.where(inter==0, 0, np.arccos((2*R - G - B)/(2 * np.sqrt(inter))))
```

根据`np.where(condition, x, y)`的定义，它返回一个`ArrayLike`，当某位置的`condition`为`True`，就以`x`计算，否则为`y`。这里当`inter=0`时，返回的是 0 而非后面的式子，为 0 的`inner`不应该出现在分母的位置上，也排除了`inner`为很小的数字的可能性。这很奇怪，但不影响计算。

幸运的，当我将所有公式转为代码后，经过一些 debug 工作，$RGB \to HSL \to RGB$ 的如蜜传如蜜路径成功了。我得到了与原始图像基本一致的图像。

3 难道公式有问题？

当我拉动滑动条时，自己的轮子并没能跟标准答案——也就是 opencv 的实现达成一致。我反复检查了代码，把一些不安全的、模棱两可的代码进行了重写，但 de 破了天也没能对上标准答案。请教了包含 ChatGPT 先生在内的各色人，也没能解决问题。最后我终于怀疑到了公式头上：纳尼？情报是假的？

于是我找到了 opencv 的文档，这里给出了 $RGB \to HLS$ 的转换公式。

$$\begin{aligned} V_{max} &\leftarrow \max(R,G,B) \\ V_{min} &\leftarrow \min(R,G,B) \end{aligned} \tag{2}$$

$$\begin{aligned} L &\leftarrow \frac{V_{max}+V_{min}}{2} \\ S &\leftarrow \begin{cases} \dfrac{V_{max}-V_{min}}{V_{max}+V_{min}} & if\ L < 0.5 \\ \dfrac{V_{max}-V_{min}}{2-(V_{max}+V_{min})} & if\ L \geq 0.5 \end{cases} \\ H &\leftarrow \begin{cases} 60(G-B)/(V_{max}-Vmin) & if\ V_{max}=R \\ 120+60(B-R)/(V_{max}-Vmin) & if\ V_{max}=G \\ 240+60(R-G)/(V_{max}-V_{min}) & if\ V_{max}=B \\ 0 & if\ R=G=B \end{cases} \end{aligned} \tag{3}$$

如果 $H<0$ 则 $H \leftarrow H+360$。输出时 $0 \leq L \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$。

又从 Wikipedia（虽然这个不该作为学术参考吧）翻到了 $HSL \to RGB$ 的公式：

$$
\begin{aligned}
C &= (1 - |2L - 1|) \times S_L \\
H' &= \frac{H}{60^\circ} \\
X &= C \times (1 - |H' \mod 2 - 1|
\end{aligned} \tag{4}
$$

$$
(R_1, G_1, B_1) = \begin{cases}
(C, X, 0) & if\ 0 \le H' < 1 \\
(X, C, 0) & if\ 1 \le H' < 2 \\
(0, C, X) & if\ 2 \le H' < 3 \\
(0, X, C) & if\ 3 \le H' < 4 \\
(X, 0, C) & if\ 4 \le H' < 5 \\
(C, 0, X) & if\ 5 \le H' < 6
\end{cases} \tag{5}
$$

$$
\begin{aligned}
m &= L - \frac{C}{2} \\
(R, G, B) &= (R_1 + m, G_1 + m, B_1 + m)
\end{aligned} \tag{6}
$$

这显然与课程 ppt 中给出的公式大相径庭：标准库的实现更像是经验公式，而课程 ppt 中的公式是根据几何计算而来的。

这里找一个极端的例子：对白色而言，$L = 1$（这里 L 的范围是 $0 - 1$）时，H 与 S 都无意义，无论 H 与 S 取何值，这个 HSL 格式的色块转为 RGB 后都是`#ffffff`。显然课件中的转换公式没有考虑这一点。在下示的图中可以较清楚地看到这一点：

图 1: 示例转换

可以看到，将饱和度 S 稍稍上调 0.01 就使得画面中的白色部分出现了大量失真，“分裂”出了许多色彩。这大约是公式未考虑极端情况导致的。

我又尝试翻了 opencv 的源代码，在`opencv-4.10.0/modules/imgproc/src/color.cpp:192`找到了`cvtColor()`函数。之后顺藤摸瓜找到了一条调用链，但没能看懂算法的具体实现。最终，我也没能确定这是否为 bug，是公式的不完善导致的偏差还是代码写错导致的错误。

另外较为奇怪的一点是：在 $RGB \leftrightarrow HSL$ 中均用 opencv 接口或自己的轮子，可以实现 $I \to I$，这不奇怪。在 $RGB \to HSL$ 时用自己的轮子，在 $HSL \to RGB$ 时调用 opencv 接口，也可以成功转换回原图；然而如果反过来，用 opencv 接口实现 $RGB \to HSL$，再用自己的轮子转回来的时候，图像便出现了差异，如下图所示：

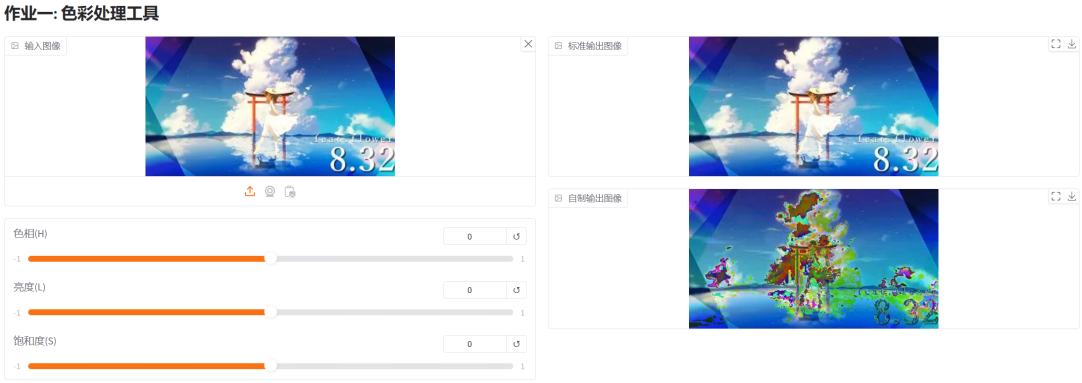

图 2: 无法实现 $I \to I$

从这里可以看出，应该是自己造的 $HSL \to RGB$ 轮子出了问题，但能力有限，未继续研究。

4 成果展示

以下是程序运行过程中的截图:

图 3: 色相调整

图 4: 亮度调整

图 5: 饱和度调整

以上是挑了几个自己的函数表现比较好的区间做了展示。这时我不禁怀疑，难道是`HLS_adjust`函数写错了？但由于标准输出很正常，这个可能也被排除了。后我又发现，在某个情况下，我的输出似乎优于 opencv 实现：

图 6: 我击败了 opencv?

然而打开 PhotoShop，导入图片，调整亮度后，是上面的图像。不过我还是认为，下面的更贴近“降低亮度”的概念。就当我们的公式更好吧。

至此，这个小作业做完了。