实验：HSV 色彩空间直方图均衡

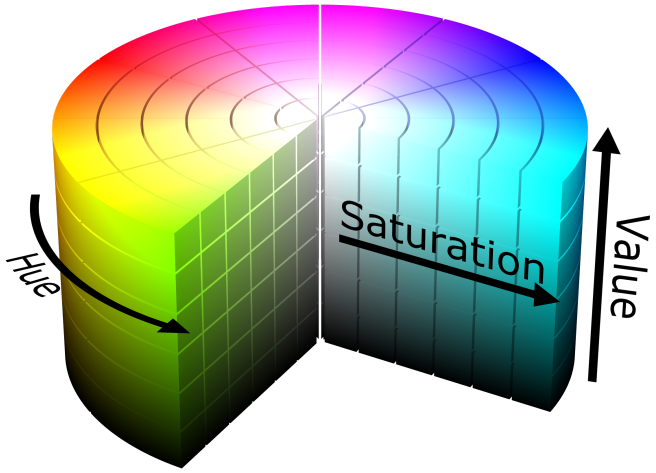
实验概要

在之前的实验中，我们看到将直方图均衡应用于 R，G 和 B 平面会导致颜色通道的修改，从而导致图像的颜色失真 —— 原因是：对于每个像素，其三个颜色通道之间存在相关性（R、G、B 三种颜色最后要重新混合到一起才能决定最终呈现的颜色）。而且，对其进行独立更改会改变颜色色度，从而使其看起来不自然。

尽管 CLAHE 产生了相对较好的结果，但仍然可以更好。

最佳实践是在 HSV 或 LAB 颜色空间中处理图像。在这两种方法中，颜色信息都与强度信息分开，并且颜色平面不变。

在 HSV 中， 色相 平面（H 或平面 0）具有有关基本颜色的信息， 饱和度 平面（S 或平面 1）具有有关该颜色的鲜艳度的信息，而 明度 平面（V 或平面 2）具有有关该颜色的亮度信息。因此，在这里，我们将仅均衡 V 平面。



实验目标

在本实验中，您将在 HSV 空间中的处理直方图均衡化，操作步骤如下：

1. 将图像从 RGB/BGR 转换为 HSV 颜色空间。
2. 将图像拆分为：H, S 和 V 平面。
3. 均衡 V 平面的直方图。
4. 将：H, S 和 V 平面合并回去。
5. 转换回 RGB/BGR。

我们将对 HSV 平面中的彩色图像应用直方图均衡化。使用与之前的实验中相同的图像进行比较。同样，首先使用直方图均衡，然后再使用 CLAHE。

1. 导入依赖库

In [1]:

```
import cv2                          # 导入OpenCV
import matplotlib.pyplot as plt     # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline
```

2. 加载图片

确认读取图片的文件名和路径与之前的实验一样，确保采用的是相同的图片进行增强。

In [2]:

```
# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab3/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

img = cv2.imread('./data/bloom.jpg')  # 读取图片
```

我们将使用 Matplotlib 显示图像。 Matplotlib 拍摄 RGB 格式的图像。因此，要使用它，我们将首先将 BGR 图像转换为 RGB。

In [3]:

```
# 将图像转换为RGB模式
origrgb = cv2.cvtColor(img,cv2.COLOR_BGR2RGB);
# 指定显示转换后的图像
imgplot = plt.imshow(origrgb)
# 指定图像标题
plt.title('Original')
# 显示图像
plt.show()
```

3. 将图像色彩空间转换为 HSV

使用以下代码将图像转换为 HSV 空间

In [4]:

```
# 将图像转换为HSV模式
imgHSV = cv2.cvtColor(img,cv2.COLOR_BGR2HSV);
```

4. 分离 HSV 通道

将图像拆分为 H, S 和 V 平面:

In [5]:

```
h,s,v = cv2.split(imgHSV); # 分离HSV通道
```

5. 对 V 通道执行直方图均衡化

仅将直方图均衡化应用于 V 通道

In [6]:

```
# 仅将直方图均衡化应用于V通道,其余两个通道保持不变
v = cv2.equalizeHist(v);
```

6. 合并通道

将三个平面堆叠在一起，以获得更新后的 HSV 图像

In [7]:

```
# 将执行完直方图均衡化的V平面与其余两个没有被修改平面合并，输出更新后的HSV图像`hsv`
hsv = cv2.merge([h,s,v]);
```

7. 使用 Matplotlib 显示合并后的图像

In [8]:

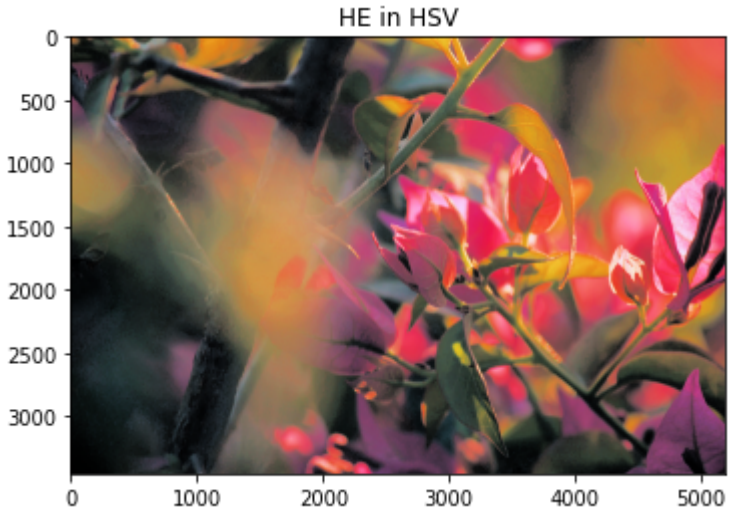
```
# 将图像转换为RGB模式
rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2RGB);
```

注意：如果您要使用 cv2.imshow 显示图像或使用 cv2.imwrite 保存图像，或与 OpenCV 进行其他任何操作，则不使用 RGB 颜色空间，而需要使用 BGR 颜色。可以使用 cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR) 命令实现转换。

将直方图均衡化到 V 通道，然后合并它们后的结果图像如下所示:

In [9]:

```
plt.imshow(rgb)                    # 指定显示转换后的图像
plt.title('HE in HSV')             # 指定图像标题
plt.show()                         # 显示图像
```



8. 再次分离 HSV 通道

In [10]:

```
h,s,v = cv2.split(imgHSV);    # 分离HSV通道
```

9. 创建 CLAHE 对象并应用

In [11]:

```
# 创建一个阈值为4，区快尺寸为16x16的CLAHE对象
clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(16,16))
```

In [12]:

```
# 仅将CLAHE对象应用至V通道
v = clahe.apply(v)
```

10. 再次合并通道

In [13]:

```
# 将执行了clahe的V通道与其余两个没有修改的平面进行合并
hsv = cv2.merge([h,s,v]);
```

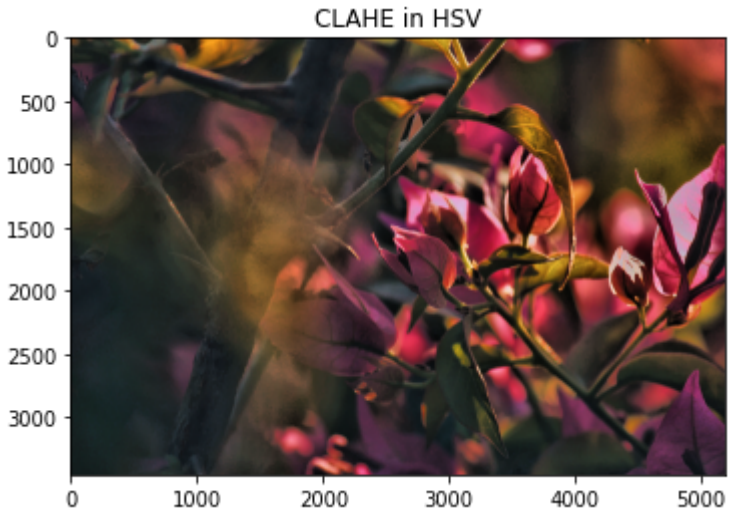
11. 使用 Matplotlib 显示合并后的图像

In [14]:

```
# 将图像转换为RGB显示
rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2RGB);
```

In [15]:

```
# 指定显示转换后的图像
plt.imshow(rgb)
# 指定图像标题
plt.title('CLAHE in HSV')
# 显示图像
plt.show()
```



根据上面的执行结果，我们可以推断出 CLAHE 在 HSV 中的表现也优于直方图均衡化。当然，图像的审美与观感是很主观性的，并没有绝对的标准。

12. 优化技巧

这里有个小提示需要你注意的是： `cv2.split` 命令会消耗较高的计算资源，可能减慢代码的速度。

有一种更快的方法是直接访问所需的平面，并使用 NumPy 索引在一行代码中对其进行修改。

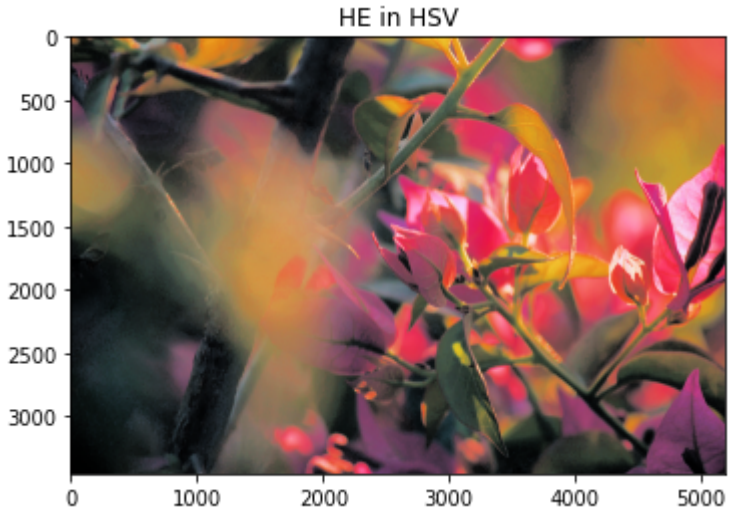
In [16]:

```
# 直接对imgHSV图像的V平面（2）进行直方图均衡化，无需分离/均衡化/合并，如此繁复的操作
imgHSV[:, :, 2] = cv2.equalizeHist(imgHSV[:, :, 2]);
```

转换为 RGB 后，再通过 Matplotlib 进行显示，同样能获得与之前分离通道，直方图均衡化，合并通道相同的效果：

In [17]:

```
# 将图像转换为RGB模式
rgbNew = cv2.cvtColor(imgHSV,cv2.COLOR_HSV2RGB);
# 指定显示转换后的图像
plt.imshow(rgbNew)
# 指定图像标题
plt.title('HE in HSV')
# 显示图像
plt.show()
```



实验小结

在本实验中，你了解了为何针对 HSV 色彩空间进行直方图均衡化能获得相对于 RGB 色彩空间更好的效果；了解了如何对 HSV 的 V 平面执行直方图均衡化与 CLAHE；最后，我们分享了如何通过更快的方式实现对 HSV 图像的直方图均衡化。