

# 调色板图像编辑 实验报告

潘藤予 2019011280

## 1 实现的得分点

### 1.1 调色板计算

基础功能（6分）：划分 bin、RGB 空间均值计算、K-means 聚类、升序排列

附加分（2分）：使用改进的K-means算法

### 1.2 重着色

基础功能（6分）：实现L空间的亮度线性变化、实现AB空间单个颜色变换

附加功能（4分）：实现AB空间多个颜色变换、实现加速计算

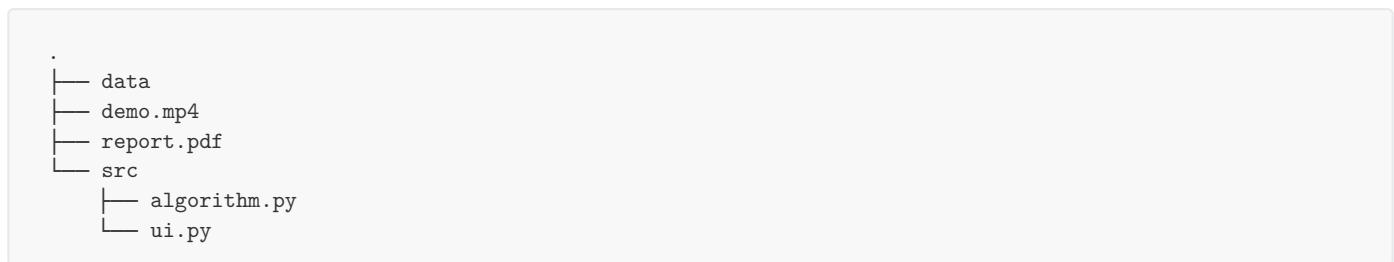
### 1.3 交互界面

基础功能（2分）：实现载入图片、展示调色板、调整调色板、结果展示

附加功能（2分）：通过图形界面 tkinter 交互

## 2 运行方式

项目结构如下，其中 `algorithm.py` 实现主要算法，`ui.py` 实现 GUI 界面。



进入 `src` 文件夹后，运行 `python ui.py` 即可开启交互界面。

## 3 实现

### 3.1 聚类获取调色板

本次实验实现了改进的通过聚类获取调色板算法，代码详见 `algorithm.py/kmeans()`，具体步骤如下：

#### 3.1.1 划分 RGB 空间

对于 RGB 空间中的颜色，共有  $256^3 = 2^{24}$  种组合方式，对全部颜色空间进行聚类需要的计算量过大，因此将 RGB 空间均匀分成  $16 \times 16 \times 16$  个 bins，每个 bin 的范围均为  $16 \times 16 \times 16$ 。此时可将图像中的每一个像素映射到唯一的 bin 上，遍历像素可以统计每个 bin 在图像中对应的像素个数和平均 RGB 值。

由于图像无法覆盖全部 RGB 空间，因此部分 bins 不对应输入图像的任意一个像素。为了进一步加速聚类算法，筛选出在 4096 个 bins 中计数不为零的部分，完成之后的计算。

### 3.1.2 初始化中心点

使用上节筛选后的 bins 作为输入点，将对应的像素个数作为每一个 bin 的初始权值，执行以下迭代：

1. 选择权值最大的点作为新筛选出的中心
2. 计算所有的点对应的平均颜色和新中心之间的 LAB 距离  $d$ ，使用  $d$  更新其权重，其中  $\sigma_a = 80$

$$w = w \times \left(1 - \exp\left(\frac{-d^2}{\sigma_a^2}\right)\right)$$

重复以上步骤直到所有的中心点均被选出，代码请见 [algorithm.py/select\\_init\\_center\(\)](#)。

### 3.1.3 聚类算法

使用上节的初始中心作为当前的  $K$  个中心，并将纯黑色作为第  $K + 1$  个中心，迭代一下步骤：

1. 计算所有点到所有聚类标签之间的 LAB 空间距离，并将距离最近的中心点作为此点的分类标签
2. 按照新的分类计算  $K$  个标签的加权平均值（使用每个 bin 中像素的个数作为每个点的权重），作为其新的聚类标签
3. 若新的中心与旧中心标签相等，则判定分类收敛，结束迭代。

为了使重着色更加方便，将计算后的  $K + 1$  个聚类标签按照 LAB 空间中的 L 值升序排序，并取后  $K$  个作为最终调色板的颜色（由于纯黑色亮度最低，此步骤自动去除了此前手动添加的纯黑色）。

## 3.2 重着色

本次实验实现了多颜色调节的调色板重着色，并使用 color grid 对其进行加速。

### 3.2.1 加速

为了让计算量与输入图片大小解偶，将 RGB 空间划分为  $16 \times 16 \times 16$  个颜色网格（grid），每一个单元的 RGB 范围也为  $16 \times 16 \times 16$ 。对于 RGB 空间中的任意颜色  $[i, j, k]$ ，均可以找到其在 grid 中对应的单元  $[x, y, z]$ 。此时可以通过  $[i, j, k]$  的值和单元  $[x, y, z]$  中的 8 个顶点进行三维线性插值。将每个像素对应的 8 个格点的权重记录在 [ImagePalette.linear\\_weights](#) 中，权重的计算公式如下图所示：

$$\begin{aligned} x_d &= (x - x_0) / (x_1 - x_0) \\ y_d &= (y - y_0) / (y_1 - y_0) \\ z_d &= (z - z_0) / (z_1 - z_0) \\ c &= c_{000}(1 - x_d)(1 - y_d)(1 - z_d) + \\ &\quad c_{100}x_d(1 - y_d)(1 - z_d) + \\ &\quad c_{010}(1 - x_d)y_d(1 - z_d) + \\ &\quad c_{001}(1 - x_d)(1 - y_d)z_d + \\ &\quad c_{101}x_d(1 - y_d)z_d + \\ &\quad c_{011}(1 - x_d)y_dz_d + \\ &\quad c_{110}x_dy_d(1 - z_d) + \\ &\quad c_{111}x_dy_dz_d \end{aligned}$$

<http://blog.csdn.net/webzhuce>

代码请见 [algorithm.py/cal\\_pixel\\_linear\\_weights](#)。

在划分格点并完成权值的计算后，重着色任务变为将  $(16 + 1)^3$  个 grid 顶点颜色进行重着色，完成此步骤之后，将权值矩阵与重着色的格点进行矩阵乘法即可得到像素的重着色结果。

由于本次实现中多颜色重着色和计算加速是同时完成的，因此不能完全控制变量地展示加速效果。在加速前，单颜色地处理一张 demo 中大小为  $760 \times 505$  的图片，耗时约 1 分钟左右；在加速后，多颜色地处理一张图片，耗时约 6 秒左右。

### 3.2.2 调节 L 空间

#### 3.2.2.1 修改调色板

由于在 k-means 算法得到的聚类中心按照 L 通道升序排列，即  $L(0) \leq L(1) \leq \dots \leq L(K - 1)$ ，假设调节第  $i$  个调色板颜色，且亮度改为  $L^*(i)$ 。

则对于所有  $i$  之前的调色板颜色，若其 L 值大于  $L^*(i)$ ，则修改为  $L^*(i)$ ；对于所有  $i$  之后的调色板颜色，若其 L 值小于  $L^*(i)$ ，则修改为  $L^*(i)$ 。

在多次调整同一块调色板颜色时，由于其亮度值 L 的改变序列并非单调，在初始实现时会出现如下的 bug：

1.  $L(2) = 12, L(3) = 15$ : 首先将第二个调色板亮度通道修改为  $L^*(2) = 20$ ，此时  $L'(3) = 20$ 。
2. 再次调节第二个调色板，将其亮度改为  $L^*(2) = 10$ ，此时  $L'(3) = 20$  满足相应的偏序关系，无需再次修改。

但在上例中，调色板 3 在第二次调节后的亮度应该回到最初的  $L(3) = 15$ ，而非  $L'(3) = 20$ 。否则与直接将调色板 2 改为  $L^*(2) = 10$  的结果不一致。

#### 3.2.2.2 修改色彩格点

使用线性插值完成颜色格点的 L 空间颜色修改：使用原聚类中心的 L 通道计算插值比例  $t$ ，并使用修改后的聚类中心完成插值。

### 3.2.3 调节 A、B 空间

依照课件及原论文，实现非线性的 AB 空间颜色修改。记待修改的调色板的 LAB 空间颜色为  $C$ ，修改后的颜色为  $C'$ 。延长  $\overrightarrow{CC'}$  后交色域边界于  $C_b$ 。对于每个色彩格点的 LAB 表示  $x$ ，令  $x_0 = x + C' - C$ ：

1. 若  $x_0$  在色域内，则计算  $\overrightarrow{xx_0}$  与色域的边界  $x_b$
2. 若  $x_0$  在色域外，则计算  $\overrightarrow{C'x_0}$  与色域的边界  $x_b$

使用如下公式计算  $x$  的最终修改颜色  $x'$ ：

$$x' = x + (x_b - x) \left( \frac{\|C' - C\|}{\|x_b - x\|} \times \min(1, \frac{\|x_b - x\|}{\|C_b - C\|}) \right)$$

使用二分查找计算向量与色域的边界；在判断某个 LAB 颜色是否在 RGB 色域内时，由于主流的 LAB 转 RGB 的色彩库都自动把转换结果截取到有效色域内，故无法使用其作为判断依据。本实验使用了他人在 Github 上开源的 RGBtoLAB 转换代码，详情请见 [1](#)。

### 3.2.4 多颜色变换

对于每一个调色板，单颜色的变化与 1.2.3 节计算方式一致，在 `ImagePalette.editted_grid_color` 中分别记录 K 个调色板对应的修改后的格点颜色。在计算多个调色时，需要将 K 个颜色结果  $f_i(x)$  与 K 个权值  $w_i(x)$  点积得到最终结果。

权值  $w_i(x) = \sum_j^K \lambda_{ij} \phi(\|x - C_j\|)$ ，其中  $\phi(r) = \exp(-r^2/2\sigma_r^2)$ ， $\sigma_r = \text{average}(\|C_i - C_j\|)$  ( $i \neq j$ )，即所有调色板点对距离的平均值。

对于  $\lambda_{ij}$  矩阵，由  $w_i(C_i) = \sum_j^K \lambda_{ij} \phi(\|C_i - C_j\|)$ ，且  $w_i(C_i) = 1$ 、 $w_i(C_j) = 0$  ( $i \neq j$ )。可由  $\phi(\|C_i - C_j\|)$  的矩阵求逆得到  $\lambda_{ij}$  矩阵。

在求解得到  $w_i$  后，将权值的负项设为 0 并完成归一化  $\Sigma w_i = 1$ 。

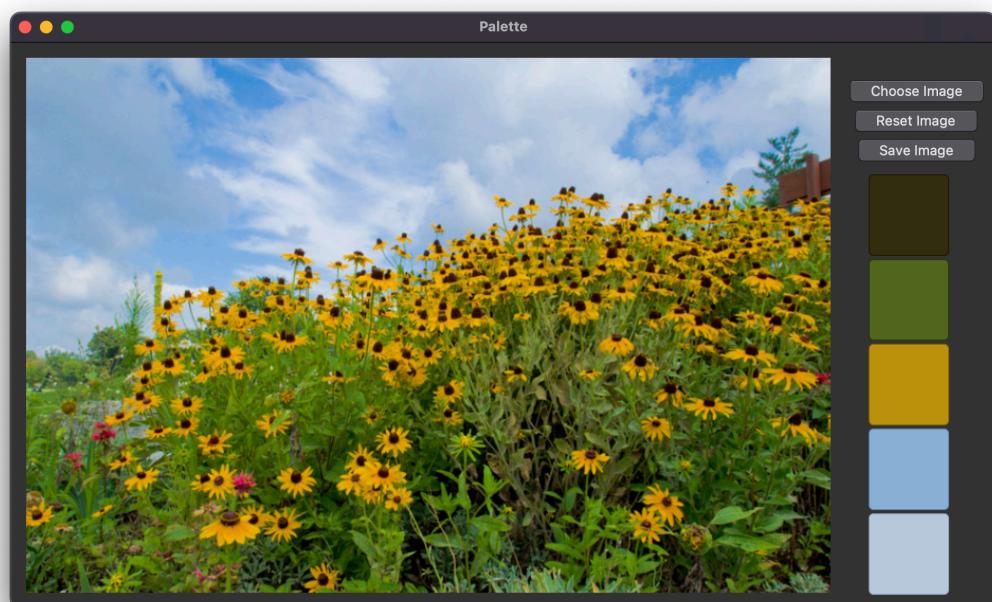
注意到上述步骤与设定的调色板新颜色无关。因此对于同一张图，其权值矩阵是唯一固定的，可以在图像预处理时完成，无需每次重新计算。

### 3.3 UI 界面

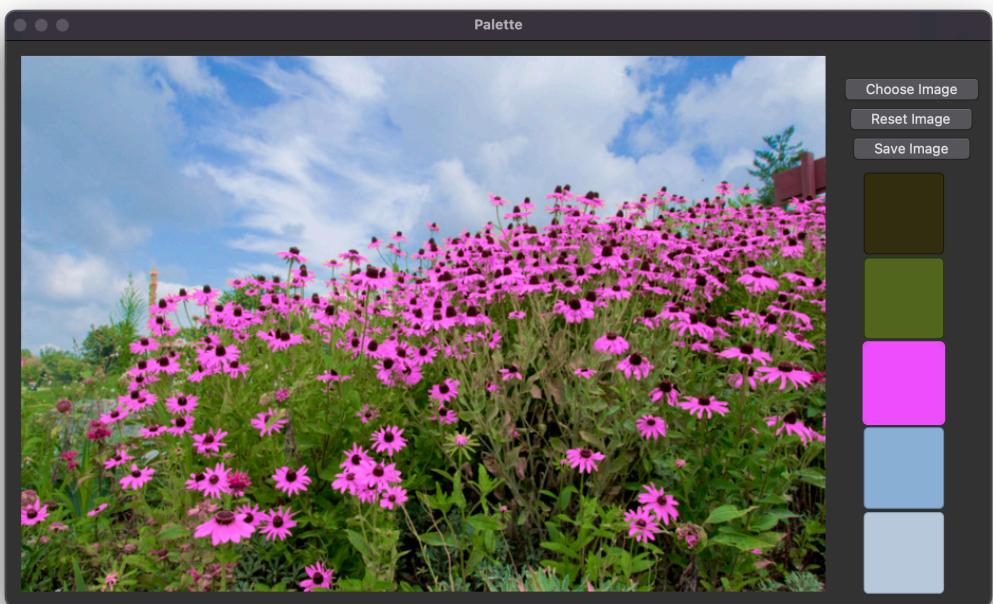
使用 `tkinter` 工具包完成 UI 的设计即交互，基本仿照官网中 Demo 的设计。功能包括选择图片、重置图片、保存图片；以及修改相应的调色板颜色。由于界面并非本次作业的核心任务，故不在此详细介绍，请助教参考提交的视频。

## 4 运行结果

### 4.1 单一颜色调节



原图及调色板



修改后的图片及调色板

注意到第二张修改的图片将蓝色调暗后，天空的对比度明显增高，生成效果类似 HDR 相片。调色板中其他颜色的亮度也相应改变。

## 4.2 多色调调节



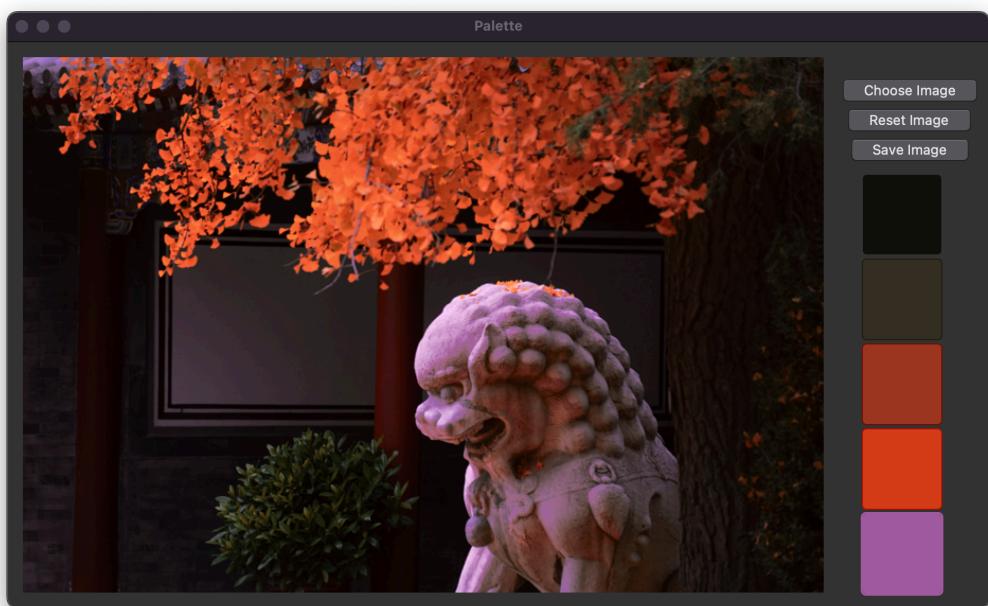
原图及调色板



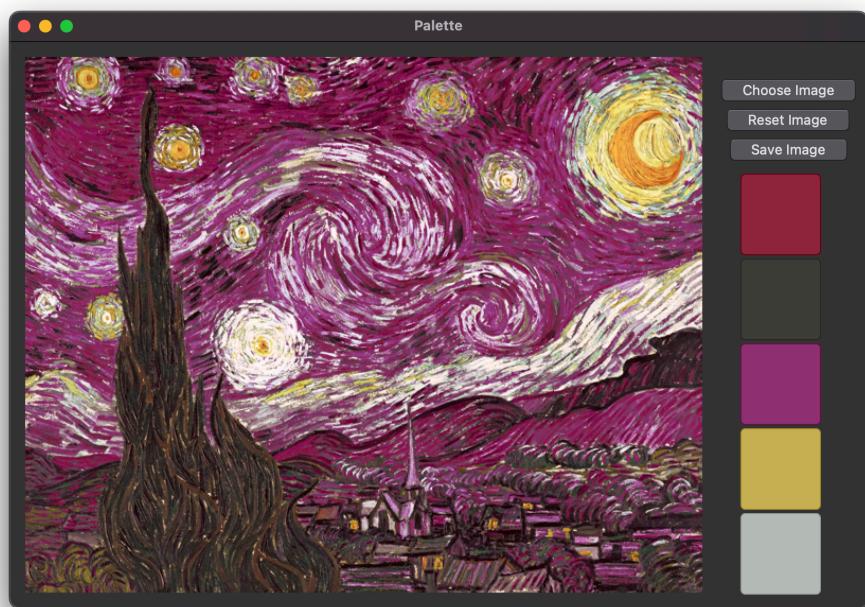
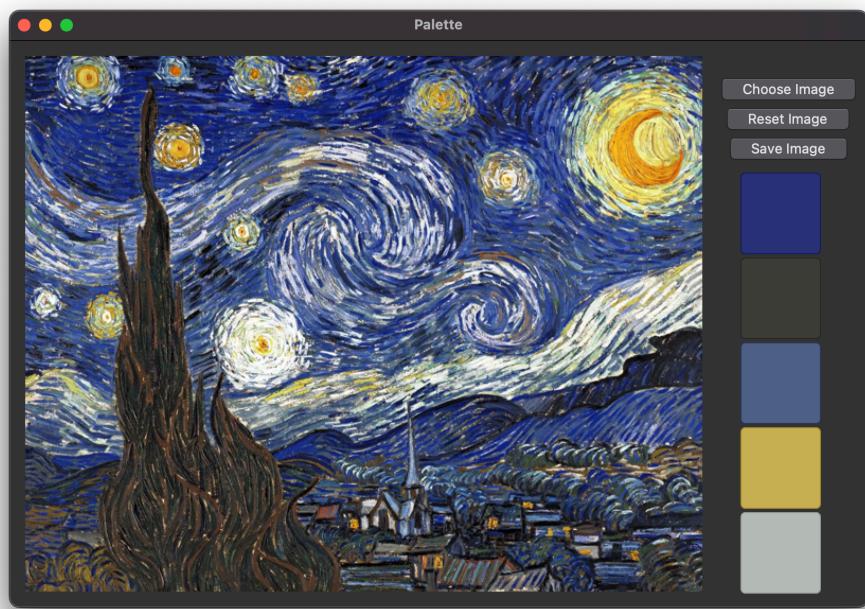
修改后的图片及调色板

### 4.3 个性化图片

- 将工字厅的银杏变成枫叶，并把石狮子变紫：



- 粉红版星月夜



1. [https://github.com/pipi3838/DIP\\_Final/blob/0516f95f9386ade75fb5fe55f81a890adaec0d2b/util.py ↵](https://github.com/pipi3838/DIP_Final/blob/0516f95f9386ade75fb5fe55f81a890adaec0d2b/util.py)