

Colorization using Optimization

王世因 2016011246

目录

1 算法简介	2
2 图像处理	2
2.1 图像着色结果	3
2.2 图像换色结果	5
3 从静态图片扩展到视频	6
3.1 准静态图片的像素采样法	6
3.2 位移捕捉和临点检测	6
4 算法的不足	7

1 算法简介

在 YIQ 和 YUV 图片格式中，Y 代表图片的亮度，含有各种阴影细节，单独输出后就可以得到黑白照片。我在算法实现中使用的是 YIQ 格式，是美洲和日韩电视机使用的格式。算法假设距离近的点中，灰度 Y 差距小的点，它们的色彩差距也会小。因此我们可以将这个问题转化成一个最小化 $J(I)$ 和 $J(Q)$ 的优化问题。

$$w_{rs} \propto e^{-\frac{(Y(r)-Y(s))^2}{2\sigma_r^2}} \quad (1)$$

$$J(I) = \sum_r (I(r) - \sum_{s \in N(r)} w_{rs} I(s))^2 \quad (2)$$

$$J(Q) = \sum_r (Q(r) - \sum_{s \in N(r)} w_{rs} Q(s))^2 \quad (3)$$

在艺术家着色的草图中，部分点的色彩是给定的：如果是白色就是说明要保留原图的色彩，如果是其他颜色就是把这部分色彩变成新上的颜色。把白色的点的集合记为 $White$ ，把其他颜色的点的集合记为 $Color$ 。把一个像素相邻的点记为 $N(i, j)$

最小值在 $J'(I) = 0$ 和 $J'(Q) = 0$ 时取到，也就是解下面的方程：

$$W_{i,j} = \begin{cases} 1 & i = j \\ -w_{ij} & i \neq j, i \in N(j), i \notin White \cap Color \\ 0 & otherwise \end{cases} \quad (4)$$

$$b_i = \begin{cases} sketch(i) & i \in Color \\ origin(i) & i \in White \\ 0 & otherwise \end{cases} \quad (5)$$

$$WI = b_1 \quad (6)$$

$$WQ = b_2 \quad (7)$$

注意到因为 W 矩阵只在两个点相邻的情况下才可能取非零的值，每个点的相邻点很少，所以它是一个稀疏的矩阵，我使用了 scipy 中的 csc 格式来存储和求解，提升计算效率。

因为我的笔记本的计算资源有限，我事先对图片进行了 2×2 的压缩，最后把输出再进行差值，提升了运算效率。

2 图像处理

在 frame.py 中，StaticFrame 代表了静态图片的处理，执行脚本是 color.py。使用的方法为：

```
1 python3 color.py images/gray_0.png images/sketch_0.png images/result_0.  
png images/weights/0.pickle
```

依次输入原始图片位置、上色草图位置、输出图片位置和预存的权重的位置（如果事先没有权重的话就是计算好的权重应该存储的位置）。

2.1 图像着色结果



图 2: gray

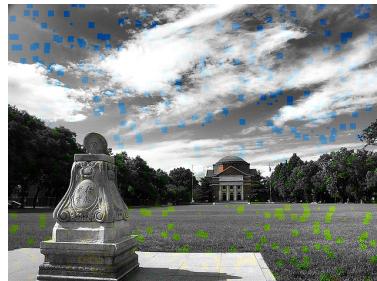


图 3: sketch



图 4: result



图 5: gray



图 6: sketch



图 7: result



图 8: gray



图 9: sketch



图 10: result



图 11: gray

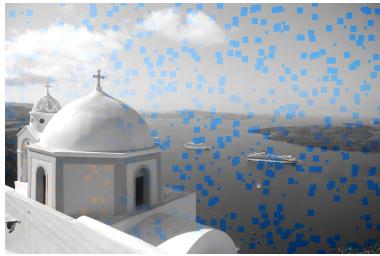


图 12: sketch

图 13: result



图 14: gray



图 15: sketch



图 16: result

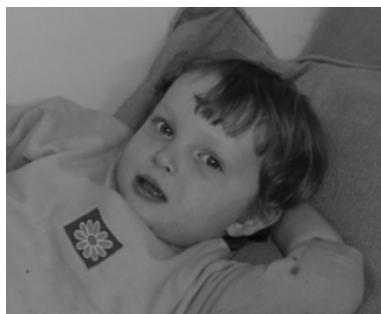


图 17: gray



图 18: sketch



图 19: result



图 20: gray



图 21: sketch



图 22: result

2.2 图像换色结果



图 23: origin



图 24: sketch



图 25: result



图 26: origin



图 27: sketch



图 28: result

3 从静态图片扩展到视频

我首先使用 `cv2.VideoCapture` 把视频分成若干帧的形式。在路径下使用“gray”文件夹来存储所有的原始视频的帧（黑白帧或者是着色前的图片），使用“sketch”来存储艺术家上色的草图帧。然后在命令行中执行命令：

```
1 python3 video_dynamic.py videos/butterfly
```

3.1 准静态图片的像素采样法

这部分算法对应代码文件 `video_color.py`，我通过在上一帧图片中获得一些采样点来，作为本帧的标注色彩。因为我采用的是 30 帧一秒的视频质量，视频中相邻两帧间的变化不大，所以可以直接进行一定的色彩继承分析。因为优化算法最小化的是总体的灰度匹配概率，因为部位微小位移造成的标注误差可以在最优化求解的时候被补偿。相比下面的这种方法，本方法因为没有增加权重矩阵的大小，所以计算起来更快，占用的计算资源也更小。



图 28：相邻帧之间采样标注色彩

3.2 位移捕捉和临点检测

这是原文中提到的方法，通过算法 Lucas-Kanade 计算各个像素点的运动情况，找到相邻两帧中对应的点，拓展静态图片中像素点的临点，再进行求解。

$$Y_t(p) + \nabla Y_{(x,y)} v = 0 \quad (8)$$

$$\begin{bmatrix} Y_x(p_1) & Y_y(p_1) \\ Y_x(p_2) & Y_y(p_2) \\ Y_x(p_3) & Y_y(p_3) \\ \dots & \dots \\ Y_x(p_{25}) & Y_y(p_{25}) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = - \begin{bmatrix} Y_t(p_1) \\ Y_t(p_2) \\ Y_t(p_3) \\ \dots \\ Y_t(p_{25}) \end{bmatrix} \quad (9)$$

$$\|(x_{t+1} - v_x, y_{t+1} - v_y) - (x_t, v_t)\| < T \quad (10)$$

在具体的实现上，我根据上面的公式生成一个包含两帧的权重矩阵，重新跑一边静态图片的上色算法，得到后一帧的色彩。因为权重矩阵是一个很大的稀疏矩阵，这个做法的权重矩阵是静态图片权重

矩阵的四倍，显著地增加了耗时。对比来看，这个方法适用于移动迅速的图片和每秒帧数少的图片。相应的代码实现在 `video_dynamic.py` 中，对应 `frame.DynamicFrame` 类。

4 算法的不足

算法假设 算法假设蕴含的逻辑是，一个色彩相同区域的灰度是基本上一样的，而且不同色块间的灰度有一定的大小差距。然而，在一些表面纹理复杂的物体中，同一个色块的灰度变化挺大；在漫画或者过度曝光的照片中，不同的色块中并没有很多的灰度差距。比如下面的这个画面来自 1928 年的电影《威力号汽船》，会出现大片的白色，缺少灰度的变化，而且白色的内部的灰度的变化没有什么规律。在助教提供的中国地图中，也出现了同样的情况，我认为这个不是一个很好的测例，不适用于这篇文章中提到的算法。



图 29: 《威力号汽船》剧照

工作量 本方法需要艺术家进行人为的色彩标注，对于处理长视频来说是一个任务量很大的工作。而且随着图片精度的提高，权重矩阵的大小也迅速增长，计算的时间变得很长，影响效率。