

数字图象处理

小作业四

2019010485 自 91 刘祖炎*

2021 年 11 月 11 日

1 任务一：颜色滤镜

1.1 算法原理及代码分析

1.1.1 不变 LUT 图生成

首先生成不变滤镜的 3D LUT 图，代码如下：

```
1 function result = genLUT()
2     block = zeros(64, 64, 2);
3     block(:, :, 1) = meshgrid(0:63, 0:63);
4     block(:, :, 2) = transpose(block(:, :, 1));
5     result = zeros(512, 512, 3);
6     for i = 1:8
7         for j = 1:8
8             index = (i - 1) * 8 + (j - 1);
9             left = (i - 1) * 64 + 1;
10            right = left + 63;
11            top = (j - 1) * 64 + 1;
12            down = top + 63;
13            result(left:right, top:down, 1:2) = block;
14            result(left:right, top:down, 3) = index;
15        end
16    end
17    result = result * 4;
18    result = result / 256;
19 end
```

根据 LUT 图的原理，需要将 [512, 512] 的图片分为 64 个小块，每个小块的尺寸均为 [64, 64, 3]。对每一个小块利用 *meshgrid* 函数生成满足条件的 R G 值（在横、竖方向上均为 0 ~ 63 递增）。此后，[512, 512, 3] 尺寸大小的图片，根据索引确定小块的上下左右四点坐标，遍历每一个小块，对其进行赋值即可。其中，第 1、2 维（R 通道、G 通道）均采用此前生成好的小块数据，第 3 维（B 通道）赋予相同的值。

*liuzuyan19@mails.tsinghua.edu.cn

最后，需要将 $0 \sim 63$ 范围内的颜色数值线性映射到 $[0, 1]$ 区间内以使得颜色数据正确显示。

根据程序运行结果，其 3D LUT 图生成结果如图1所示。可以看到，生成结果与课件中所示相同。

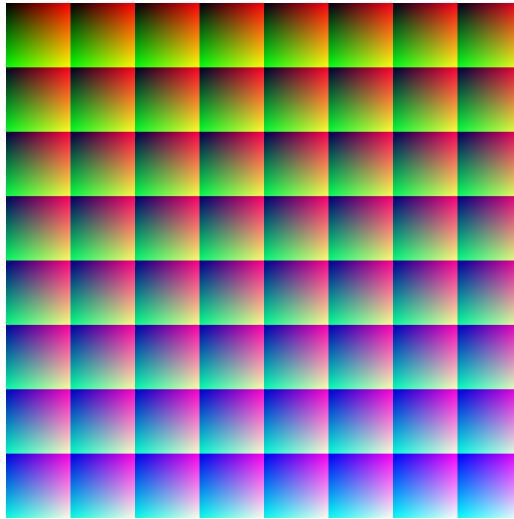


图 1: 不变滤镜的 3D LUT 图

1.1.2 滤镜应用

完成第一步的 LUT 图生成后，需要将该 LUT 图发送至手机上，并应用对应的滤镜得到该滤镜的 3D LUT 图。以美图秀秀 App 中“电影”分类的“橡树”滤镜为例，生成的 3D LUT 图如图2所示。

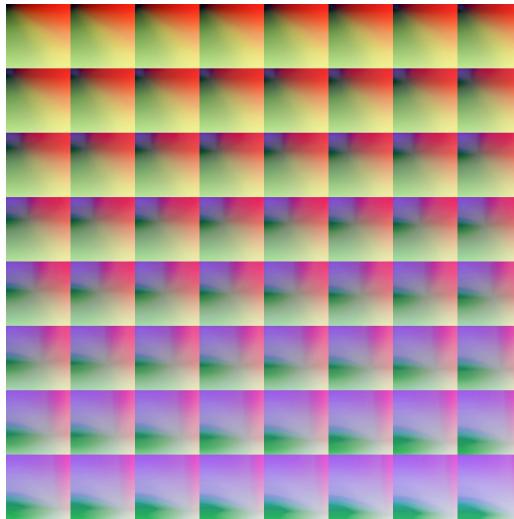


图 2: 滤镜 1 3D LUT 图

此后，根据该步骤的输出结果进行滤镜复现。在代码中加载图2、待修改的图片后，首先将 $[512, 512, 3]$ 的压缩后的 LUT 图扩展为 $[256, 256, 256, 3]$ 的全尺寸 LUT 图，前 3 维各表示一个通道。代码如下：

```
1 function result = resize(filter_input)
2     result = uint8(zeros(256,256,256,3));
3     filterResized= uint8(zeros(64,64,64,3));
4     for i = 1:64
5         index_i = floor((i - 1) / 8) * 64;
6         index_j = int16(mod(i - 1,8)) * 64;
```

```

7      for j = 1:64
8          for k = 1:64
9              filterResized(j,k,i,:)=filter_input(index_i+k,index_j+j,:);
10         end
11     end
12   end
13 result(:,:,:,:,1)=imresize3(filterResized(:,:,:,:,1),[256,256,256]);
14 result(:,:,:,:,2)=imresize3(filterResized(:,:,:,:,2),[256,256,256]);
15 result(:,:,:,:,3)=imresize3(filterResized(:,:,:,:,3),[256,256,256]);
16 result=uint8(round(result));
17 end

```

代码通过 2 个子步骤进行。第一步需要根据 B 通道的数值，将 $[64, 64, 3]$ 的小块赋值进入 $[64, 64, 64, 3]$ 的全尺寸数组中，对原压缩储存的 LUT 图进行重组解码。第二步利用 *imresize3* 函数对 LUT 图像进行插值扩充，得到 $[256, 256, 256, 3]$ 的全尺寸 LUT 图。

此后，对原图的每一个像素 $[R, G, B]$ 值，在 $[256, 256, 256, 3]$ 的表中进行索引即可得到应用滤镜后的 $[R, G, B]$ 图。代码如下：

```

1 function result = Filter2(img, LUT)
2 [height, width, ~] = size(img);
3 result = zeros(height, width, 3);
4 for i = 1:height
5     for j = 1:width
6         result(i,j,:)=LUT(round(img(i,j,1)+1),round(img(i,j,2)+1)
7             ,round(img(i,j,3)+1), :);
8     end
9 end
10 end

```

由此，即完成了任务一的滤镜复现任务。

1.2 变换结果分析

1.2.1 滤镜一：电影-橡树

生成的 3D LUT 图如图2所示。

图像 1 变换结果如图3、4、5所示。可以看到，复现滤镜与原滤镜完全相同。



图 3: 原图



图 4: 滤镜生成图



图 5: 滤镜复现图

图像 2 变换结果如图6、7、8所示。可以看到，复现滤镜与原滤镜完全相同。



图 6: 原图



图 7: 滤镜生成图



图 8: 滤镜复现图

1.2.2 滤镜二：旅行-古都

生成的 3D LUT 图如图9所示。

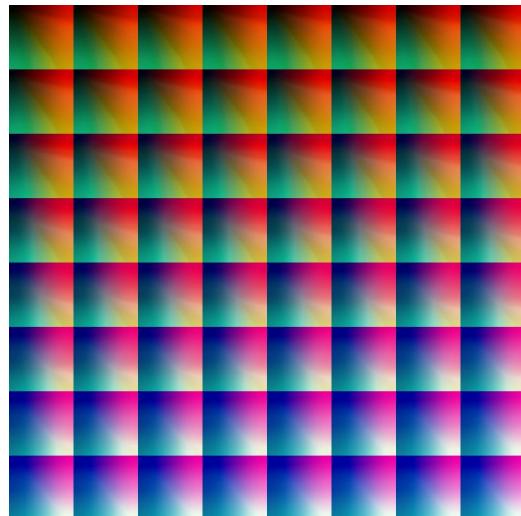


图 9: 滤镜 2 3D LUT 图

图像 1 变换结果如图10、11、12所示。可以看到，复现滤镜与原滤镜完全相同。



图 10: 原图



图 11: 滤镜生成图



图 12: 滤镜复现图

图像 2 变换结果如图13、14、15所示。可以看到，复现滤镜与原滤镜完全相同。



图 13: 原图



图 14: 滤镜生成图



图 15: 滤镜复现图

1.2.3 滤镜三：轻胶片-5207

生成的 3D LUT 图如图16所示。

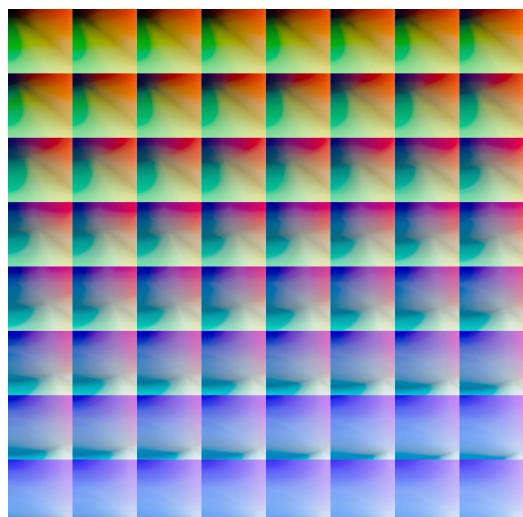


图 16: 滤镜 2 3D LUT 图

图像 1 变换结果如图17、18、19所示。可以看到，复现滤镜与原滤镜完全相同。



图 17: 原图



图 18: 滤镜生成图



图 19: 滤镜复现图

图像 2 变换结果如图20、21、22所示。可以看到，复现滤镜与原滤镜完全相同。



图 20: 原图



图 21: 滤镜生成图



图 22: 滤镜复现图

2 任务二：换装

2.1 UI 设计

UI 界面如图23所示。

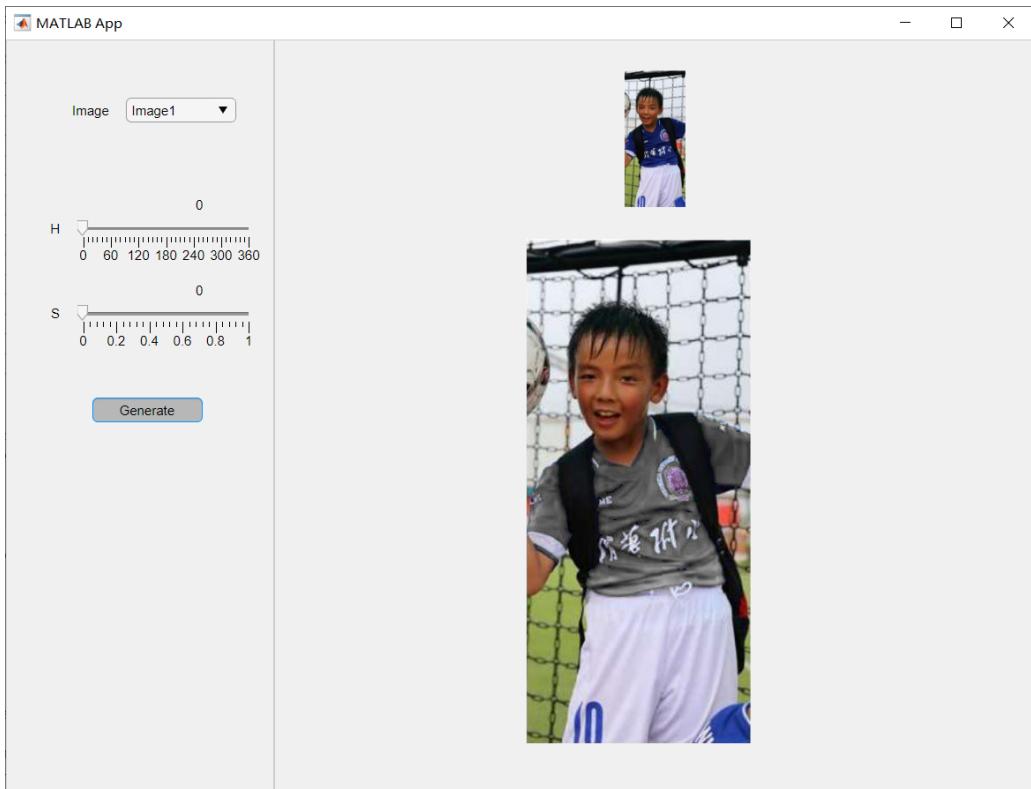


图 23: UI 界面

点击 *Generate* 按钮即可启动程序，再次点击 *Generate* 按钮即可暂停程序。通过调节 *H*、*S* 的滑条可调节色度和饱和度，调节后界面如图24所示。

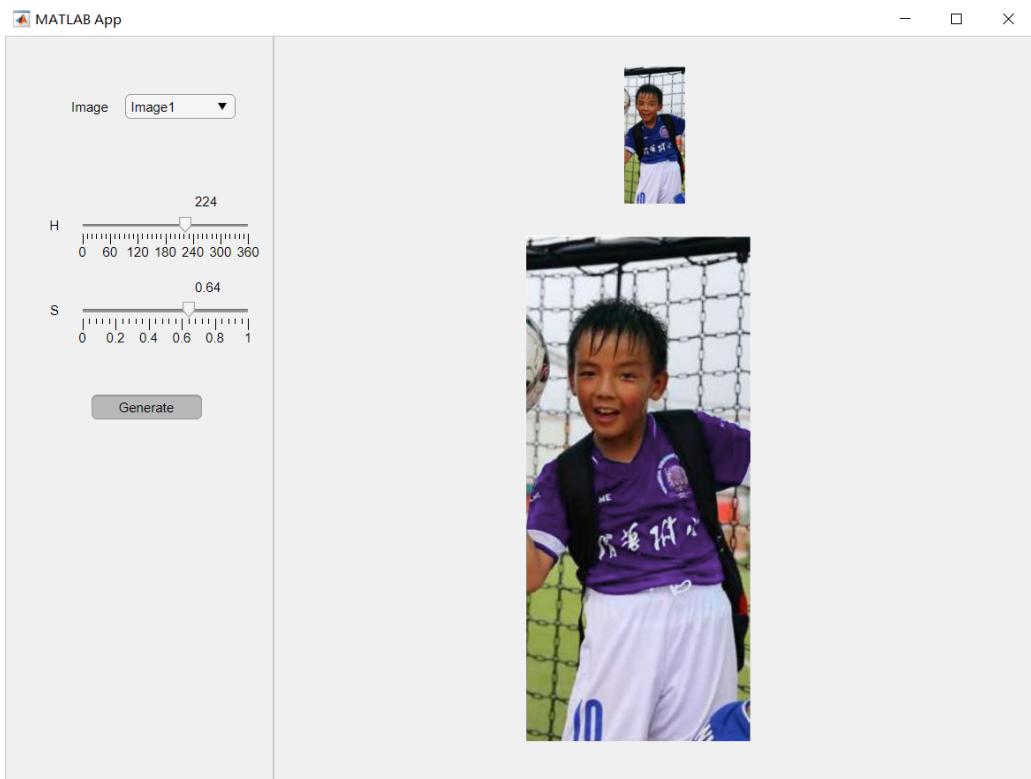


图 24: UI 界面 (2)

通过切换上方的下拉框，可切换当前显示的图片。切换后界面如图25所示。

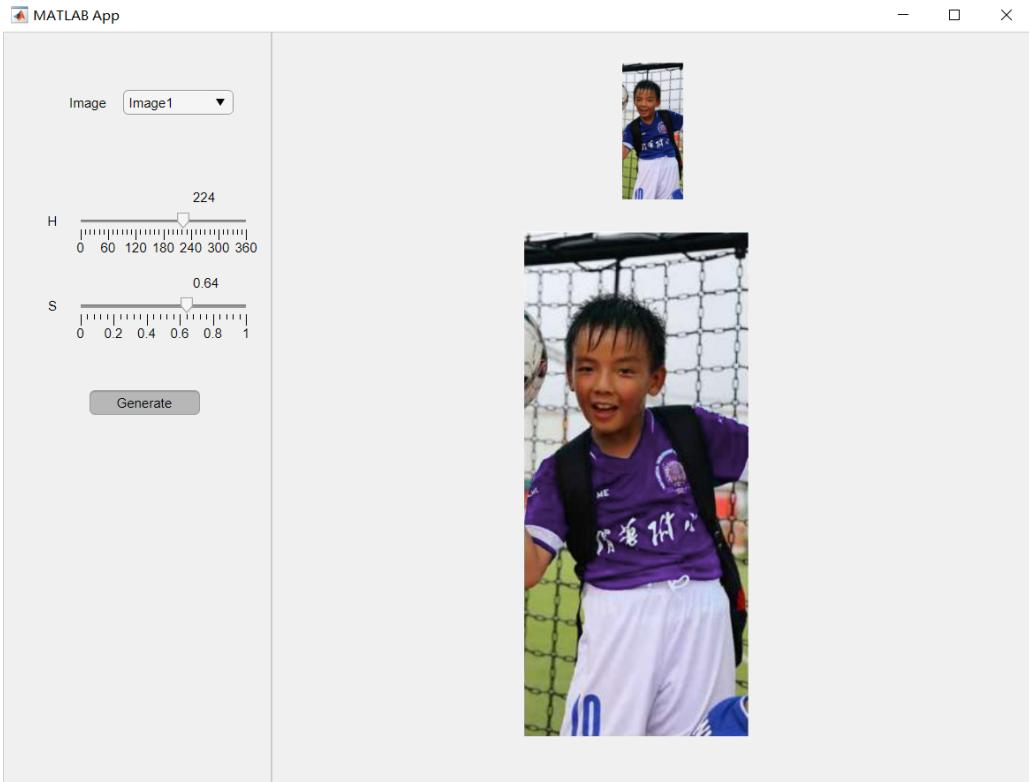


图 25: UI 界面 (3)

2.2 算法原理及代码分析

首先定义类变量，包括如下：

```

1 properties (Access = private)
2     currentHSV;
3     currentMask;
4     srcimg;
5     RecROI;
6     hsvROI
7 end

```

其中，*currentHSV* 为当前图片的 *HSV* 值，*currentMask* 为当前图片前景分割结果，*srcimg* 为原图，*RecROI* 为当前图片预设的矩形区域，*hsvROI* 为当前图片预设的 *HSV* 阈值。

定义实现了如下函数：

```

1 function mask = genMask(app, img)
2     [height, width, ~] = size(img);
3     RecMask = zeros(height, width);
4     RecMask(app.RecROI(1):app.RecROI(2), app.RecROI(3):app.RecROI(4)) = 1;
5     hsvMask = (img(:, :, 1) >= app.hsvROI(1)).*(img(:, :, 1) <= app.hsvROI(2));
6     hsvMask = hsvMask.* (img(:, :, 2) >= app.hsvROI(3))
7     .* (img(:, :, 2) <= app.hsvROI(4));
8     hsvMask = hsvMask.* (img(:, :, 3) >= app.hsvROI(5))
9     .* (img(:, :, 3) <= app.hsvROI(6));
10    mask = RecMask.* hsvMask;

```

```

11 se = strel('disk', 1);
12 mask = bwareaopen(mask, 64, 8);
13 mask = imdilate(mask, se);
14 end

```

genMask 函数根据当前 *RecROI*、*hsvROI* 的值生成对应的前景分割遮罩。矩形分割直接利用索引即可，*HSV* 分割需要利用大小判断比较，共得到 M_1 、 M_2 、 M_3 、 M_4 四个遮罩。将四个遮罩相乘即得到最终遮罩 M 。

$$M = M_1 \times M_2 \times M_3 \times M_4 \quad (1)$$

```

1 function result = genHSV(app)
2 [height, width, ~] = size(app.currentHSV);
3 newHSV = app.currentHSV;
4 newHSV(find(app.currentMask)) = (app.HSlider.Value) / 360;
5 newHSV(find(app.currentMask) + height * width) = app.SSlider.Value;
6 newRGB = hsv2rgb(newHSV);
7 app.Image.ImageSource = newRGB;
8 end

```

genHSV 函数用于根据当前设定的 H, S 值生成预期的 *RGB* 图片。该部分较为简单，直接根据公式进行转换即可。

定义实现了如下回调函数：

- 启动回调函数

```

1 function startupFcn(app)
2     app.CurrentSLabel.Text = "0";
3     app.CurrentHLabel.Text = "0";
4     app.srcimg = imread('1.png');
5     app.RecROI = [200, 500, 1, 310];
6     app.hsvROI = [0.6, 0.8, 0.40, 0.95, 0.15, 0.7];
7 end

```

启动程序时，将对应标签初始化为 0，设置需要显示的图片为第一张图，并对应设置 *RecROI*、*hsvROI*。

- S 滑条变化回调函数

```

1 function SSliderValueChanging(app, event)
2     changingValue = event.Value;
3     app.CurrentSLabel.Text = num2str(roundn(changingValue, -2));
4     if app.GenerateButton.Value == 1
5         app.genHSV();
6     end
7 end

```

当 S 滑条变化时，将对应的结果保留两位小数显示在标签中，并刷新当前图片即可。

- H 滑条变化回调函数

```

1 function HSliderValueChanging(app, event)
2     changingValue = event.Value;
3     app.CurrentHLabel.Text = num2str(round(changingValue));
4     if app.GenerateButton.Value == 1
5         app.genHSV();
6     end
7 end

```

当 H 滑条变化时，将对应的结果保留整数显示在标签中，并刷新当前图片即可。

- 下拉框变化回调函数

```

1 function ImageDropDownValueChanged(app, event)
2     value = app.ImageDropDown.Value;
3     if strcmp(value, 'Image1')
4         app.srcimg = imread('1.png');
5         app.RecROI = [200, 500, 1, 310];
6         app.hsvROI = [0.6, 0.8, 0.4, 0.95, 0.15, 0.7];
7     elseif strcmp(value, 'Image2')
8         app.srcimg = imread('2.png');
9         app.RecROI = [300, 1000, 400, 1000];
10        app.hsvROI = [0, 1, 0.6, 1, 0.25, 1];
11    elseif strcmp(value, 'Image3')
12        app.srcimg = imread('3.jpg');
13        app.RecROI = [200, 853, 90, 640];
14        app.hsvROI = [0.5, 0.85, 0.5, 1, 0.1, 1];
15    elseif strcmp(value, 'Image4')
16        app.srcimg = imread('4.png');
17        app.RecROI = [150, 600, 1, 400];
18        app.hsvROI = [0, 0.3, 0.36, 1.0, 0.65, 1.0];
19    elseif strcmp(value, 'Image5')
20        app.srcimg = imread('5.png');
21        app.RecROI = [430, 800, 200, 600];
22        app.hsvROI = [0.2, 0.5, 0.2, 1.0, 0.0, 1.0];
23    elseif strcmp(value, 'Image6')
24        app.srcimg = imread('6.png');
25        app.RecROI = [1, 818, 1, 700];
26        app.hsvROI = [0.9, 1.0, 0.2, 1.0, 0.4, 1.0];
27    elseif strcmp(value, 'Image7')
28        app.srcimg = imread('7.png');
29        app.RecROI = [200, 1000, 200, 800];
30        app.hsvROI = [0.15, 0.7, 0.0, 1.0, 0.0, 1.0];
31    end
32    app.currentHSV = rgb2hsv(app.srcimg);
33    app.Image2.ImageSource = app.srcimg;

```

```

34     app.currentMask = app.genMask(app.currentHSV);
35     if app.GenerateButton.Value == 1
36         app.genHSV();
37     end
38 end

```

当改变下拉框的选取值时，根据当前选取的内容，加载对应的图片，并对应设置 *RecROI*、*hsvROI* 即可。(此处的 *RecROI*、*hsvROI* 为提前调节参数所得，具体参数选取参见后一部分)

完成设置后，需要修改图像模块的 *ImageSource*，并生成当前图片对应的遮罩 *M*。若程序为启动状态，则调用 *genHSV* 函数更新变换后的图像。

- 按钮按下回调函数

```

1 function GenerateButtonValueChanged(app, event)
2     value = app.GenerateButton.Value;
3     if value == 1
4         app.currentHSV = rgb2hsv(app.srcimg);
5         app.Image2.ImageSource = app.srcimg;
6         app.currentMask = app.genMask(app.currentHSV);
7         app.genHSV();
8     else
9         app.Image.ImageSource = app.srcimg;
10    end
11 end

```

当改变按钮状态时，根据改变后的按钮状态进行对应的动作。若为启动，则更新 *Mask* 值与当前显示的变换后图像。若为关闭，则将当前图像复原为原图。

2.3 参数选取

通过参数调节，选取如下参数作为前景分割参数：

表 1: 参数设置

图片	矩形框参数	HSV 参数
1	[200, 500, 1, 310]	[0.6, 0.8, 0.40, 0.95, 0.15, 0.7]
2	[300, 1000, 400, 1000]	[0, 1, 0.6, 1, 0.25, 1]
3	[200, 853, 90, 640]	[0.5, 0.85, 0.5, 1, 0.1, 1]
4	[150, 600, 1, 400]	[0, 0.3, 0.36, 1.0, 0.65, 1.0]
5	[430, 800, 200, 600]	[0.2, 0.5, 0.2, 1.0, 0.0, 1.0]
6	[1, 818, 1, 700]	[0.9, 1.0, 0.2, 1.0, 0.4, 1.0]
7	[200, 1000, 200, 800]	[0.15, 0.7, 0.0, 1.0, 0.0, 1.0]

其中，矩形框参数四个参数分别表示左侧、右侧、上侧、下侧像素位置；HSV 参数六个参数分别表示 H、S、V 的最小、最大值。

2.4 运行结果 (部分)

为节省篇幅，笔者此处仅展示图片 1(即作业要求中提供的样例图片) 的各颜色替换结果与其余图片的单个颜色替换结果，完整换装结果请参见随报告提交的文件。



图 26: 原图



图 27: 蓝-> 灰



图 28: 蓝-> 红



图 29: 蓝-> 黄



图 30: 蓝-> 绿



图 31: 蓝-> 紫



图 32: 图片 2(黄色)



图 33: 图片 3(黄色)



图 34: 图片 4(黄色)



图 35: 图片 5(黄色)



图 36: 图片 6(黄色)



图 37: 图片 7(黄色)

3 遇到的困难与解决方案

- 任务一颜色精度问题

一开始，笔者在对待修改图像进行 HSV 查表时，采用的是直接将图像的 *RGB* 颜色映射到所得 $[512, 512, 3]$ 的对应位置得到滤镜处理后图片的方式。代码如下：

```
1 function result = Filter(img, LUT)
2     [height, width, ~] = size(img);
3     result = zeros(height, width, 3);
4     for i = 1:height
5         for j = 1:width
6             indexR = floor(double(img(i, j, 1)) / double(4)) + 1;
7             indexG = floor(double(img(i, j, 2)) / double(4)) + 1;
```

```

8      indexB = floor(double(img(i, j, 3)) / double(4)) + 1;
9      index_i = int16(ceil(double(indexB) / double(8)));
10     index_j = int16(mod(indexB, 8) + 1);
11     left = (index_i - 1) * 64 + 1;
12     right = left + 63;
13     top = (index_j - 1) * 64 + 1;
14     down = top + 63;
15     block = LUT(left:right, top:down, :);
16     result(i, j, 1) = block(indexG, indexR, 1);
17     result(i, j, 2) = block(indexG, indexR, 2);
18     result(i, j, 3) = block(32, 32, 3);
19   end
20 end
21 end

```

然而，通过该方式得到的图片存在明显的分界线，与原滤镜结果有一定差异。如图38、39所示。



图 38: 滤镜生成图



图 39: 滤镜复现图

经过分析，笔者认识到其原因是对色彩空间进行了压缩，实际上生成的图像为 6 位图像，不能够表示 0 ~ 255 之间的颜色值。笔者因而修改代码，利用 *imresize3* 函数将 [512, 512, 3] 的 LUT 图像通过插值扩充为 [256, 256, 256, 3] 的图像，较好地解决了该问题。

- 任务二 RGB 与 HSV 转换问题

开始时，笔者利用 PPT 中提供的公式自行实现 *rgb2hsv* 函数，但在利用系统提供的函数 *hsv2rgb* 进行向回转换时，发现 PPT 中的公式与 Matlab 库函数使用的公式不完全一致，从而导致无法正确复原图像。笔者此后采用系统实现的 *rgb2hsv* 完成该任务。

4 收获与心得体会

- 对彩色图像有了进一步的认识。
- 进一步锻炼了调参能力。
- 对滤镜的原理有了较为透彻的理解。
- 锻炼了发现问题、解决问题的能力。

5 可能的改进方向

- 优化算法逻辑，提高程序运行效率。
- 程序目前对接近人体皮肤的衣服颜色较难进行区分，因此应用时存在一定的局限性。未来可以考虑通过其他方式解决这一问题。