Image and Video Processing Assignment 4

Arseni Loika

May 2024

1 Color Palette Extraction from Images

1.1 RGB Palette

To extract the RGB palette, first the image was linearized using the same operation from assignment 1. Then, the individual channels were extracted from the image and appended into a 3-column matrix where each row represents one image pixel. This format is accepted by the MATLAB's `kmeans` function, which, after specifying the number of clusters (7), yielded a 7x3 matrix, where each row represents a centroid for each cluster, as well as the indices of the pixels closest to those centroids.
After that, the indices were used to reconstruct the parts of the image that belonged to each cluster, and colored in using information from the original image. During the partial reconstructions, each layer was split into HSV space to enable easy editing. The exact code can be seen in the file `RGB_Palette.m`.

The code yields the following results:

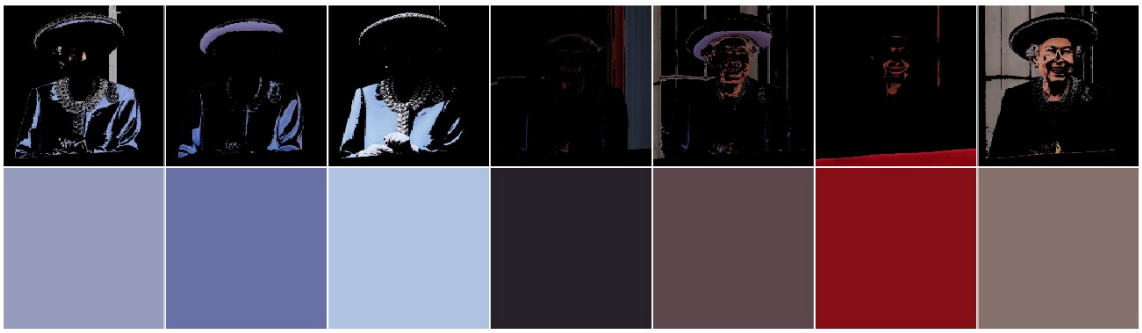

Figure 1: RGB Palette of 7 clusters extracted from queen.jpg

1.2 CIELab Color Palette

The process of extracting the CIELab palette was very similar to that of RGB palette, with the exception that the image was converted to the lab color space for clustering. The exact code can be seen in `CIELab_Palette.m`. It is clear that the clusters produced with this color space are a lot more perceptually-uniform.

The code yields the following results:

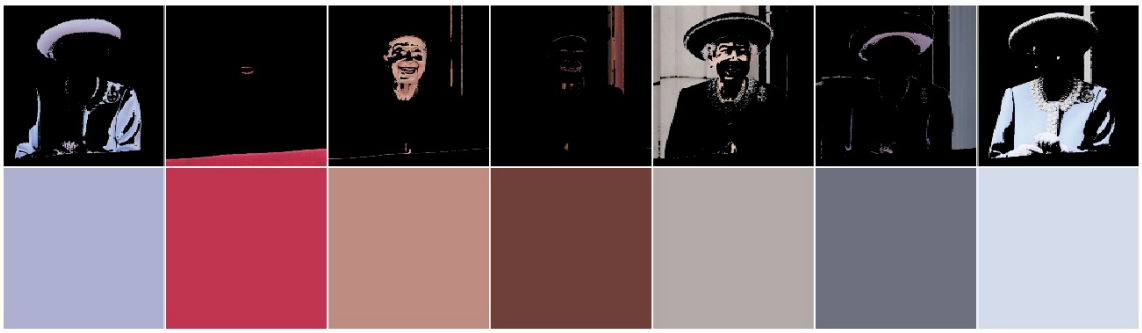

Figure 2: RGB Palette of 7 clusters extracted from queen.jpg

In regards to editing, here are the results produced by modifying the HSV values of the image layers after clustering:

Figure 3: Original and edited queen.jpg with a glossy look

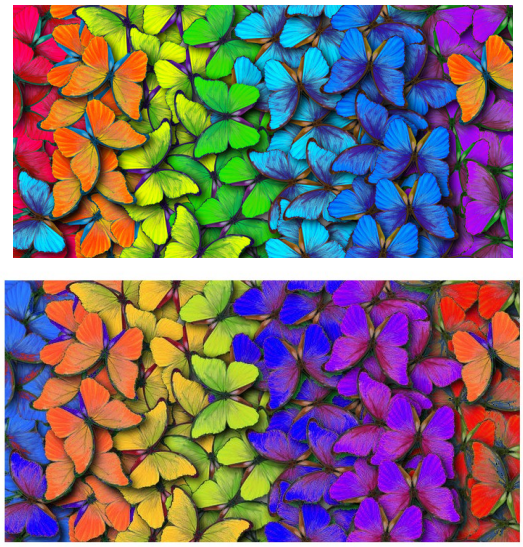

Figure 4: Original and edited color.jpg with partially changed colors

2 Color Quantization and LUTs

The first step of color quantization was to extract the most prominent colors through clustering. As a result, the functions made for previous exercises (RGB_Palette and CIELab_Palette - for bonus) were used to get 32 color clusters from the image. After that, each part of the image that corresponded to a specific cluster was colored in with the corresponding color, which was the main step in the quantizing process. Then, for converting the image to greyscale, a very simple LUT was used in the form of mapping each channel onto a linear transformation into grey. The implementation details can be seen in file `imquantize.m`.
The results of the code are as follows:

Figure 5: Color Quantization with RGB Clusters

Figure 6: Bonus - Color Quantization with CIELab Clusters

3 Gaussian and Laplacian Pyramids

In order to construct the Laplacian pyramid, first the Gaussian pyramid was constructed by applying the Gaussian filter on each subsequent layer of the image. Then, starting at the first layer (the original image), each level of the Gaussian pyramid had its next (upscaled) level subtracted from it, yielding a level of Laplacian pyramid. The exact code can be seen below:

```
function [lapPyramid] = laplacianPyramid(impath, nLevels)
    close all;
    img = im2double(imread(impath));
    % make gaussian pyramid
    gaussPyramid = cell(1,nLevels+1);
    gaussPyramid{1} = img;
    for i = 2:nLevels+1
        cur_lvl = imgaussfilt(gaussPyramid{i-1},1);
        cur_lvl_sub = imresize(cur_lvl, 0.5,'nearest');
        gaussPyramid{i} = cur_lvl_sub;
    end

    % use gaussian pyr to make laplacian pyr
    lapPyramid = cell(1,nLevels+1);
    for i = 1:nLevels
        cur = gaussPyramid{i};
        prev = gaussPyramid{i+1};
        prev = imresize(prev, [size(cur, 1), size(cur, 2)], "nearest");
        out = cur - prev;
        lapPyramid{i} = out;
    end
    lapPyramid{nLevels+1} = gaussPyramid{nLevels+1};
end
```

The code yielded the following results:

Figure 7: Laplacian Pyramid of happy.jpg

Figure 8: Laplacian Pyramid of sad.jpg

The function `laplacianVerifier.m` can be used to confirm that each layer (**together with the last layer that is equal to the last layer of the Gaussian pyramid - which was omitted here**) combine to recreate the original image.

4 Hybrid Images

To first step to create a hybrid image from files `happy.jpg` and `sad.jpg` was to apply a Gaussian filter on latter and the complementary high pass filter on the former. That was done by applying the Gaussian filter to the file, then subtracting it from the original, yielding the high-passed version of the image. The value of the sigma parameter for the filtering was decided through a brief trial, which determined that the value of 3 produces a well-balanced image. Then, the two images were combined in a simple addition. The exact code can be seen below:

```
close all;
impath_happy = "../happy.jpg";
impath_sad = "../sad.jpg";

img_happy = im2double(imread(impath_happy));
img_sad = im2double(imread(impath_sad));

sigma = 3;
img_happy = img_happy - imgaussfilt(img_happy, sigma);

img_sad = imgaussfilt(img_sad, sigma);
img_sad = imresize(img_sad, [size(img_happy, 1), size(img_happy, 2)]);

res = img_happy + img_sad;

imshow(res); title("Hybrid image, size = " ...
    + size(res, 1) + " x " + size(res, 2));
```

And the result is as follows:

Figure 9: Hybrid Image

After some measurements, it was determined that the distance to see happy face was approx. **50cm** from the monitor. The distance to see sad face was determined to be from **1.5m** and further.