Color Quantization and Color Transfer

Sanket Sanjaypant Hire Yash Malviya

1 Color Quantization

Color Quantization is quantization applied to color spaces; it is a process that reduces the number of distinct colors used in an image, usually with the intention that the new image should be as visually similar as possible to the original image [3].

(a) (b)

Figure 1: Color Quantization [3]

(a) An example image in 24-bit RGB color.
(b) The same image reduced to a palette of 16 colors specifically chosen to best represent the image; the selected palette is shown by the squares at the bottom of the image.

1.1 Representative Colors Selection

Representative points are the the colors that appear in quantized image. Each representative point will represent (or replace) one or more colors in quantized image.

1.1.1 Popularity Algorithm

Popularity algorithm counts the number of unique RGB value tuples in the entire image, then picks k colors (these are the representative points) with highest occurrences.

1.1.2 Median Cut Algorithm

Median cut algorithm is used to sort data of an arbitrary number of dimensions into series of sets by recursively cutting each set of data at the median point along the longest dimension. Here, we are using Median cut for color quantization.

Median Cut Algorithm works as follows:

1. Find the smallest cube in RGB space containing all the pixel colors in the image.
2. Sort the color codes along the longest axis of the cube (Sort the pixels on the color axis with the highest range).
3. Split the cube into two regions at the median of the sorted list.
4. Repeat the above process until the original color space has been divided into desired regions
5. Representative point is the mean of RGB value of a region

1.1.3 Floyd and Steinberg Algorithm

The Floyd-Steinberg dithering algorithm is based on error dispersion. The error dispersion technique can be understood from the following pseudocode:

Algorithm 1 Floyd-Steinberg algorithm

```
for each x from top to bottom do
    for each y from left to right do
        oldpixel = pixel[x, y]
        newpixel = FindClosestPaletteColor(oldpixel)
        pixel[x, y] = newpixel
        error = oldpixel - newpixel

        pixel[x, y + 1] = pixel[x, y + 1] + error * 7 / 16
        pixel[x + 1, y + 1] = pixel[x + 1, y + 1] + error * 1 / 16
        pixel[x + 1, y] = pixel[x + 1, y] + error * 5 / 16
        pixel[x + 1, y - 1] = pixel[x + 1, y - 1] + error * 3 / 16
    end for
end for
```

1.2 Quantized Color Mapping

Mapping colors to their nearest representative point according to some metric on their RGB values.

1.2.1 Exhaustive Search

In this search, we compute the nearest neighbour for each pixel from each of the K representative points calculated using popularity or median cut algorithm. Basically brute force approach for nearest neighbor. This method is slow since most of the time is wasted in considering distant points which couldn't possibly be the nearest neighbour.

1.2.2 K-D Tree Search

This algorithm is used to query the nearest neighbour for each of the k representative points (calculated using popularity or median cut algorithm) quickly. Using the K-D Tree database to structure the K representative points, search time can be reduced to O(logK). For using K-D tree search, we imported neighbours package from sklearn in order to perform faster nearest neighbour query.

2 Color Transfer to grey level images

Adding chromatic value to grey scale image, these methods aimed to reduce need of human interaction in the process. Both the following algorithms take a gray image to color according to the color scheme in the input colored image.

2.1 $L\alpha\beta$ Space

The $L\alpha\beta$ color space provides three decorrelated, principal channels corresponding to an achromatic luminance channel (L) and two chromatic channels α and β, which roughly correspond to yellow-blue and red-green opponent channels. Thus, changes made in one color channel should minimally affect values in the other channels. The reason the $L\alpha\beta$ color space is selected in the current procedure is because it provides a decorrelated achromatic channel for color images. This allows us to selectively transfer the chromatic α and β channels from the color image to the greyscale image without cross-channel artifacts.[2]

2.2 Global Image Matching

Procedure for Global Image Matching Color Transfer

1. Both gray and color image converted to $L\alpha\beta$ space.
2. Histogram equalization is applied on luminosity (i.e. L values) of gray image according to the luminosity of color image.
3. 200 sample points from the color image are selected from the image using Jittered sampling. 200 square blocks are distributed uniformly over the color image, 1 point is picked randomly for each block.
4. Nearest sample point to each pixel in gray image is calculated according to metric - weighted sum (0.5, 0.5) of L2 norm of luminance and neighborhood standard deviation.
5. Replace chromaticity values (i.e. α and β values) in gray image.

2.3 Swatches

Procedure for User provided Swatch Color Transfer

1. Rectangular swatch pairs are given as input by user. These mention that color in gray swatch should be similar to that of corresponding swatch in color image.
2. Both gray and color image converted to $L\alpha\beta$ space.
3. Equalize on like in the global image matching case.
4. Global Image Matching algorithm runs on cropped swatch pairs. It generates a colored swatch for each gray swatch.
5. 50 points per colored swatch are sampled used jittered sampling. The entire collection of these sampled points is used used to find best chromaticity value for transfer.
6. Best sample point is picked on basis of texture match which is defined as sumation of L2 norm of pixel luminosity in a 5X5 neighborhood.
7. Replace chromaticity values (i.e. α and β values) in gray image.

3 Results

3.1 Color Quantization

3.1.1 Popularity vs Median Cut (with/without Dithering)

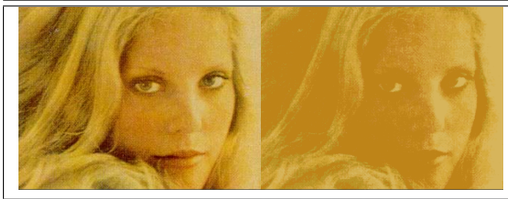
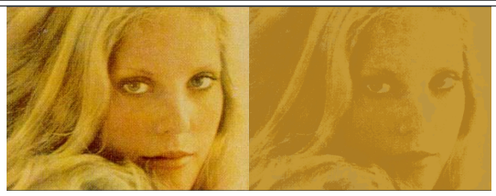
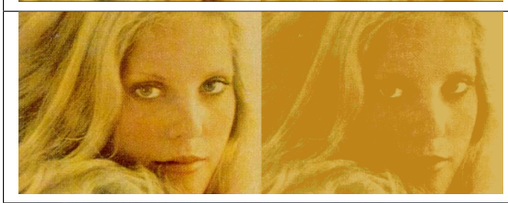
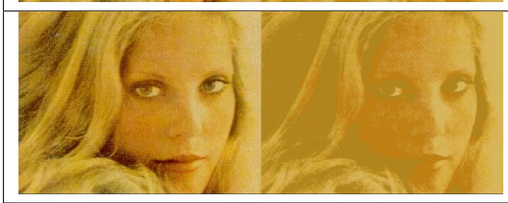
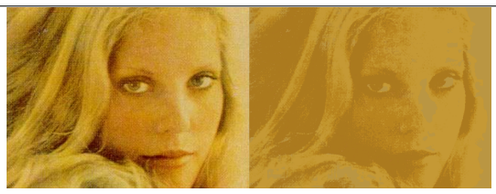
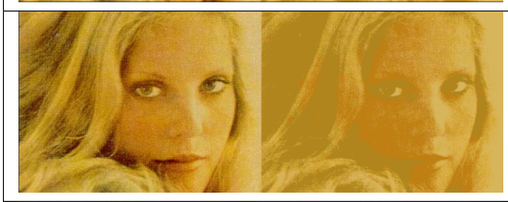
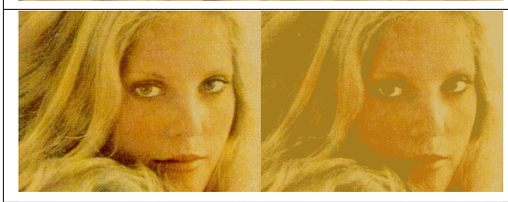
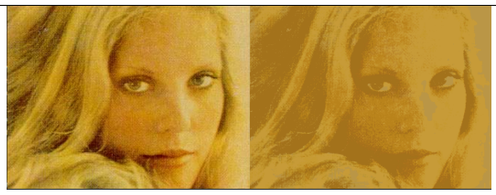
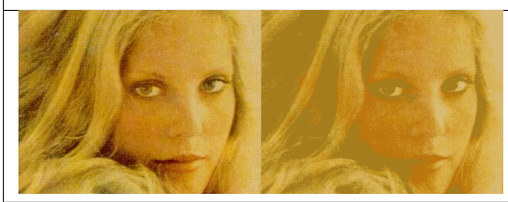
Popularity	Median Cut	Number of representative points	Dithering
		64	OFF
		64	ON
		128	OFF
		128	ON
		256	OFF
		256	ON

Table 1: Popularity vs Median Cut (with/without Dithering)

Here, for higher values of k, popularity algorithm generates output image more similar to input image than that in median cut.

3.1.2 KDTree vs Exhaustive Search

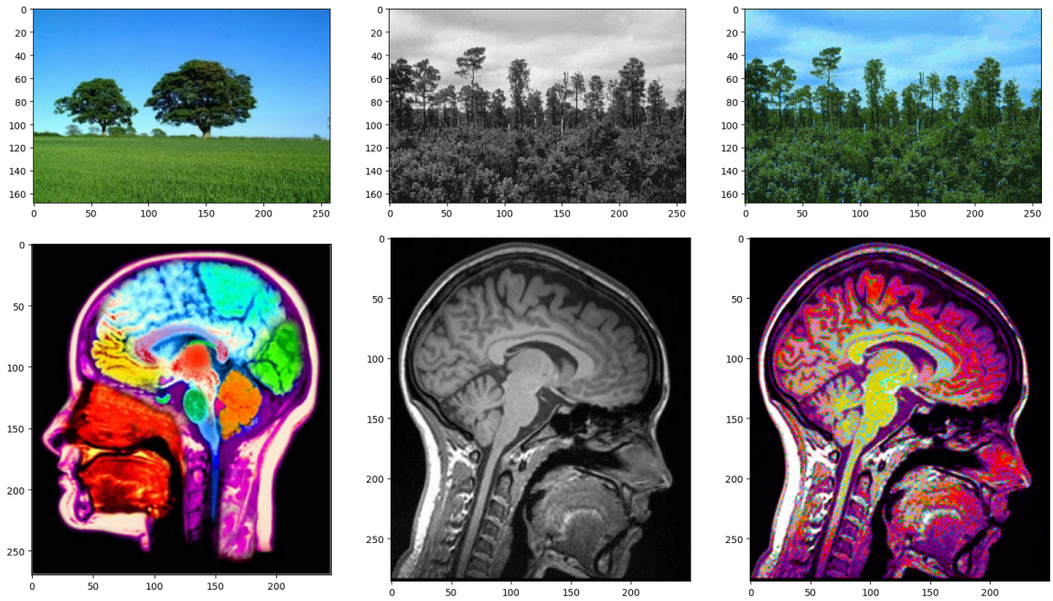
The section of code where searching for nearest neighbor is done was timed (median cut, no dithering, for k = 64, 128, 256 on pamela image). It can be observed that KDTree is taking same in all cases, the O(logK) could be observed for higher values of k, but on these values due to over head it takes same time. While in case of exhaustive search time grows linearly.

KDTree	Exhaustive Search	Number of representative points
7.48	48.87	64
7.35	99.90	128
7.16	198.50	256

Table 2: Execution times (in sec) of KDTree and Exhaustive Search (without Dithering)

3.2 Color Transfer

3.2.1 Global



3.2.2 Swatch

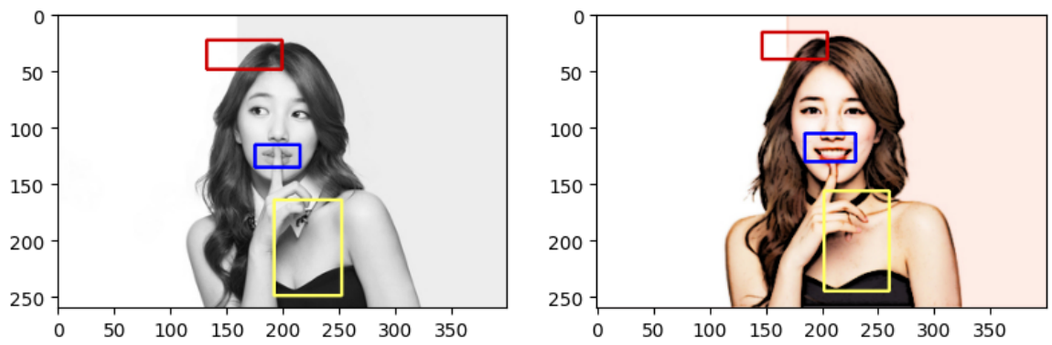


Figure 2: Swatch selection display

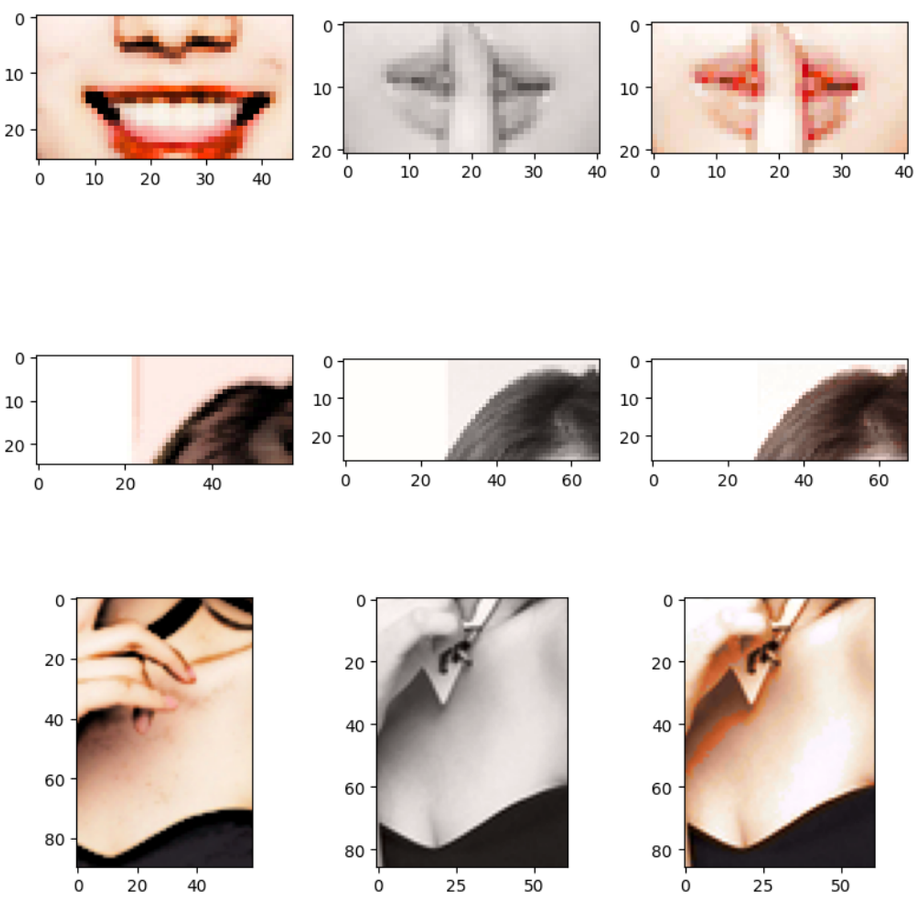


Figure 3: Global coloring ran on individual swatch pairs

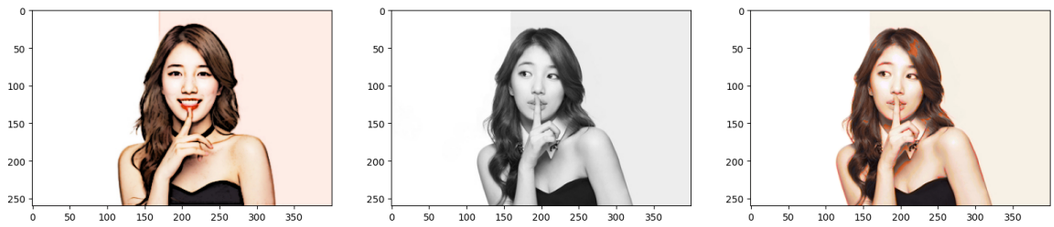


Figure 4: Swatch based coloring after final step

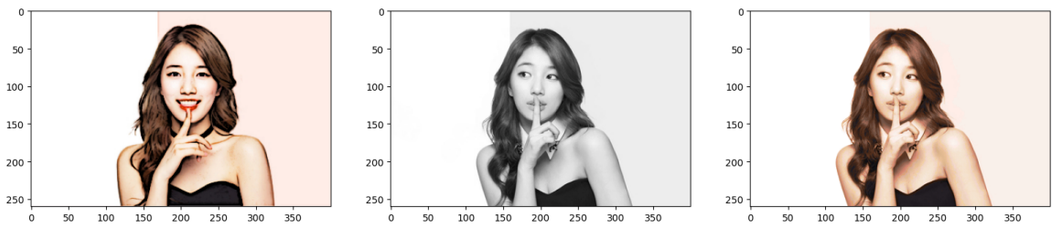


Figure 5: Global transfer

Comparing this to global coloring for same images below. The color range in swatch method is more. This can observed at the mouth region, but this also lead to some wrong coloring of the hair.

4 How to Run

4.1 Color Quantization

Ex: python color_quantize.py -rf median_cut -s kdtree -k 256 -d images/fox.jpg

```
└─$ python3 color_quantize.py --help
usage: color_quantize.py [-h] -rf {popularity,median_cut} -s {exhaustive,kdtree} -k NUM_COLORS [-d] [-o OUTPUT] input_image

Color Quantization

positional arguments:
  input_image           Path of image to quantize

optional arguments:
  -h, --help            show this help message and exit
  -rf {popularity,median_cut}, --representative-finder {popularity,median_cut}
  -s {exhaustive,kdtree}, --search {exhaustive,kdtree}
  -k NUM_COLORS, --num-colors NUM_COLORS
  -d, --dithering
  -o OUTPUT, --output OUTPUT
                        Output file to save the quantized image as
```

4.2 Color Transfer

python color_transfer.py -s images/transfer/gray/girl.jpg images/transfer/color/girl.jpg

```
└─$ python3 color_transfer.py --help
usage: color_transfer.py [-h] [-s] [-o OUTPUT] gray_image color_image

Color Transfer

positional arguments:
  gray_image            Path of grayscale image to colorize
  color_image           Path of color image based on which colorization is done

optional arguments:
  -h, --help            show this help message and exit
  -s, --swatches        To enable user interactive swatch selection
  -o OUTPUT, --output OUTPUT
                        Output file to save the colored image as
```

Without the -s flag Global color transfer works. When -s flag is provided the swatch based color transfer run. After running the command with -s flag, program asks for input of rectangular swatches for grey image and colored image respectively. Input of rectangular swatches is taken as four integer values say x1,y1,x2 and y2. In which (x1,y1) corresponds to the top left coordinate of rectangular swatch, whereas (x2,y2) corresponds to the bottom right coordinate of rectangular swatch. Matplotlib plot opens on the side to get x1, y1, x2 and y2 coordinates.

```
└─$ python color_transfer.py -s images/transfer/gray/mj_dark_bw.png images/transfer/color/mj_light.png
Rectangle for grey image: 63 244 172 293
Rectangle for color image: 103 143 148 169
Rectangle for grey image: 53 166 102 196
Rectangle for color image: 99 99 130 120
Rectangle for grey image: 207 333 233 345
Rectangle for color image: 86 219 110 230
Rectangle for grey image:
Rectangle for color image:
Done inputting
```

References

[1] Color Image Quantization For Frame Buffer Display: https://www.cse.iitd.ac.in/~pkalra/col783/assignment1/p297-heckbert.pdf

[2] Transferring Color to Greyscale Images: https://www.cse.iitd.ac.in/~pkalra/col783-2017/colorize.pdf

[3] Color Quantization: https://en.wikipedia.org/wiki/Color_quantization