# Assignment 4 Documentation

**Submitted by**
**Soumava Paul (16EE10056)**

- In this assignment, first I have computed the parameters x and y of the **CIE Chromaticity Model** using a crop of a 3 channel RGB image as input. The (x, y) matrix for the entire cropped region was then clustered using the **KMeans algorithm**. The cluster with the highest number of points represented the **dominant color** in the patch and the corresponding pixels were colored white and displayed in a separate window. Then two images were chosen as *source* and *target* pair. For the final task, the RGB color distribution of the **source patch was modified to match the target illuminant distribution** via an intermediate **l-α-β** color space transformation. Details of those are given below.
- Different helper functions for image display and matrix calculations are implemented in the **utils.py** file. The three broad tasks, namely, **click and crop**, **dominant color finding** and **transfer of dominant color** from **target to source** are implemented in the **modules.py** file.
- All modules are sequentially called from the **main.py** file.

## In `utils.py` file-

1. `def bgr_img2rgb_matrix(img)` - Takes an **BGR** *[m, n, 3]* shaped color image and returns a matrix of shape *[3, m*n]*. Each column of the output matrix corresponds to [R, G, B] values belonging to a pixel.
2. `def bgr2xy(img)` - Takes a **BGR** *[m, n, 3]* shaped color image and computes its XYZ color space values using the above function and doing a single matrix multiplication with a pre-defined conversion matrix. Finally it computes **x = X / (X + Y + Z)** and **y = Y / (X + Y + Z)** for each pixel and returns a *[2, m*n]* shaped array having the 2D chromatic characteristics of the original image.
3. `def disp_2imgs(img1, img2, str1, str2, save_flag, save_name)` - This function is used for displaying a pair of images, one the original and the other a transformed version of it after doing some operations. The two images are stacked side by side. The save_flag also handles the saving of this image to disk. Examples are (img, img with dominant pixels whitened); (source img, source img with target illumination).
4. `def bgr2l_alpha_beta(img, img_flag = True)` -Takes a BGR image and converts it to **l-α-β** color space through intermediate **LMS color space**. The parameter **img_flag** is used for handling the special case when the img is already in the flattened [3, m*n] array representation. This happens during dominant color transfer where the target patch is in the flattened representation with only the relevant dominant color pixels.

5. `def modify_l_alpha_beta(source_lab, target_lab)` - This function **modifies the source l-α-β color space** by a series of transformations (source mean subtraction, standard deviation scaling, and target mean addition).
6. `def l_alpha_beta2bgr(lab, m, n)` - Performs the inverse operation of `bgr2l_alpha_beta` and renders the source crop with target illumination in BGR format.

### In `modules.py` file-

1. `def click_and_crop(img)` - This function implements a **mouse callback routine** that relates mouse **left button down** and **up events** to the selection of **corner points** of a rectangle. While cropping, press at one location, then move along the **principal diagonal** of the rectangle (left to right, top to bottom) and release at the other corner point.
2. `def find_dominant_color(img)` - Using the 2D chromatic characteristics found with `bgr2xy`, this function performs K-Means clustering (with **num_clusters = 3**). The cluster with the maximum no. of pixels is chosen as the modal cluster. Then from the *labels* and *indices* of each point in this cluster, we back-calculate the pixel coordinates in the original image and mark them white. It finally returns a **BGR** image **showing the dominant color region in white**.
3. `def transfer_dom_color(source_img, target_img, target_indices)` - This function combines the functions `bgr2l_alpha_beta`, `modify_l_alpha_beta` and `l_alpha_beta2bgr` sequentially to modify the cropped region of the source image with dominant color transferred from the cropped target image.

## Running Instructions

From the terminal, run the `main.py` file as **"`python3 main.py -img1 img_name1 -img2 imag_name2 -save save_flag(0/1)`"**. The image names should also contain the proper file extension.

The 3 .py files are in the **codes** folder, all test images should in **image_folder** and all results are saved to the **sample_results** folder (if save_flag=1). While running different modules sequentially, mostly you will need to **press 'Esc' to close** the image display windows. Only while cropping regions from the source and target images, **press 'c' to complete** the cropping operation, **'r'** if you couldn't crop properly the first time. The window can be reset any number of times you want by **pressing 'r'**.

## Results

Image crops with the **dominant colors whitened** are saved in the **sample_results** folder for each of the 3 sample images. Moreover, the output of the color transfer algorithm is also saved for 6 possible **source-target** combinations of the 3 given images.

## Packages Required

Python3, NumPy, cv2, sklearn (for K-Means clustering algorithm).