# Purpose

This project was focused around computer vision color space conversions and image transformations, particularly histogram equalization and linear scaling. This was all done in Python using PyCharm IDE. This project involved creating four separate programs:

***Program 1***

The first program involved creating a specified width x height image of the full color spectrum using the following color values for each pixel at row i and column j:

L = 90, u = 354\*j/width – 134, v = 262\*i/height – 140

This required creating functions for converting between Luv and BGR color spaces.

***Program 2***

The second program involved utilizing the same color space conversions from Program 1, this time to take a specified image and selected boxed region in that image to perform linear scaling on the luminance values (L-values) in that boxed region only. This is all done through a custom function to demonstrate understanding of linear scaling. The remainder of the image should remain untouched.

***Program 3***

The third program is similar to Program 2, only this time the boxed region of the image will undergo histogram equalization on the luminance values (L-values). Again, this will be done through a custom function to demonstrate understanding of histogram equalization. The remainder of the image should remain untouched.

***Program 4***

The last program is identical to Program 3, only this time openCV functions can be used for color conversions and histogram equalization in the boxed region. The remainder of the image should remain untouched.

# Design and Implementation

***Files and Function Layout***

Since this project requires custom implementation of color space conversions and image transformations which are reusable, a design decision was made to create these functions in separate Python files. These files and functions were divided as follows:

*ImageProcessing.py*

* Linear scaling function
* Histogram equalization function

*ColorConversion.py*

* Luv to XYZ function
* XYZ to BGR function
* BGR to XYZ function
* XYZ to Luv function
* Linear to non-linear BGR gamma correction function
* Non-linear to linear BGR gamma correction function
* Clipping function (to ensure valid band ranges)

*constants.py*

* Contained all constants used in color space conversions

Each program was then created in its own Python file, importing ImageProcessing or ColorConversions, as necessary. These files were *Program1.py*, *Program2.py*, *Program3.py*, and *Program4.py*.

***Design Decisions***

Keeping all constants in a separate Python file kept the coding very clean and easy to manage. This included all constants used in the color space conversions which also included the matrix used for BGR to XYZ conversion and visa versa, which used the same matrix or inverted matrix.

For color conversion functions, the clipping function ensured that values passed into and out of each function remained in the value range. Using XYZ color space as an intermediary also simplified the process. Important places were value range validity had to be checked included:

* Luv to XYZ
  + Ensuring L-value in valid range
  + Checking if L = 0, then u’ and v’ were also 0
  + If v’ = 0, then X and Z were 0
* XYZ to Luv
  + If X, Y, and Z were 0, then u’ and v’ were also 0
* BGR to XYZ
  + Ensuring BGR values in valid range

Many of these checks ensured no division by 0 errors as well.

***Debugging and Challenges***

For debugging, all color conversion functions had an optional debug flag that can be passed in as an argument that defaulted to false. If true, debug statements would print out for each calculation in the color space conversion to allow detailed check for errors. This allowed quick debugging when program output gave unexpected results.

Typical challenges that were overcome during debugging were related to division by zero errors and integer division where float division was expected. Other challenges involved becoming more familiar with Python lists and arrays and multi-dimensional lists. With some trial and error and Python documentation, these challenges were overcome.

# Results and Analysis

Starting with **Program 1**, initially, there were some issues getting the output to display correctly due to integer division. Once this was resolved, the program was able to output a beautiful image of the full color spectrum as shown in **Figure 1**.



**Figure 1.** Program 1 output using 300 x 400 input

This program was fairly simple. Keeping the luminance value at a constant value of 90, based on the user’s input for the image size, the full u and v range was stretched over this window allowing all the colors to show up in all the pixels in the image. Because the Luv equations given had to be used, the output required Luv-to-XYZ conversions followed by XYZ-to-BGR.

**Program 2** involved linear scaling of the image passed in which scaled the L-value for a specified boxed region. This involved calculating the pixel indices and traversing the region to count the number of pixels and min/max L-values in the region. Then, passing these values to the linear scaling function that was created, the L-values in this region was scaled.

For most of the tests in Programs 2, 3, and 4, the fruits.jpg image was used. Program 2, however, did not show a very clear difference due to the L-values spanning almost the entire range from 0 to 100. Testing on fruits.jpg resulted in the following as shown in **Figure 2**.



**Figure 2**. Program 2 output using fruits.jpg and converting the top half

of the image. The result is very subtle due to wide L-value coverage.

To get a more vivid linear scaling result, an image that uses mostly the upper range or lower range only would demonstrate how this transformation could make the image clearer. In the next test, an bright image with white clouds was used, whiteclouds.jpg. Using Program 2 here shows how the clouds become much more vivid as shown in **Figure 3**.



**Figure 3**. Program 2 output using whiteclouds.jpg and converting the

top half of the image. The result is must more vivid due to low L-value

coverage in the upper range.

**Program 3** involved a similar approach as Program 4. This time, the L-values had to be discretized by flooring the value to create a histogram for the boxed region. This histogram was passed to the histogram equalization function created along with the number of pixels in the boxed region and the new L-value range of 0-100. The function returned the translation table for old L-value to new L-value which was written back to the image. This resulted in the following when applied to the top half of the fruits.jpg image as shown in **Figure 4**.



**Figure 4.** Program 3 output when applied to fruits.jpg using top half of

the image. Image becomes much more vivid and sharp.

However, not all results using this created a better image. Because histogram equalization flattens the L-value in the given region, if applied to a region that uses a very narrow range, a very distorted is the result as seen in Figure 5.



**Figure 5**. Program 3 output when applied to small region. Histogram

equalization flattens the L-values and causes the darker regions to

become completely black resulting in a poorer improvement.

Finally, Program 4 was identical to Program 3, except here the openCV libraries were used for histogram equalization and color space conversion. This vastly simplified the code being able to use these libraries. Use the boxed regions as shown in Figure 4 and Figure 5 gave the same results which reinforced the confidence that this code was correctly implemented. This concluded the project scope for Project 1.