CS6384 Computer Vision 2018S

Report of Project 1

Hao WAN(hxw161730)

Decisions:

1. When we do the transformation from XYZ to Luv, we do 4X /(X + 15Y + 3Z) and there is a chance that the divisor could be 0. Since the value of b, g, r and X, Y, Z are all in the range of [0, 1], it would be the case that when we have a black pixel (value of bgr and XYZ all 0). In Luv system, for the black we have L=0 and u, v undefined. In the program, I simply add an if statement to deal with it, making L, u, v all 0:

```
d = X + 15.0 * Y + 3.0 * Z
if(d == 0.0):
    L = u = v = 0.0
else:
    uprime = 4.0 * X / d
    vprime = 9.0 * Y / d
    u = 13.0 * L * (uprime - uw)
    v = 13.0 * L * (vprime - vw)
```

2. When we do the inverse transformation from Luv back to XYZ, we may encounter the same division-by-0 situation as the above:

Firstly, u'=(u+13uwL)/13L and v'=(v+13vwL)/13L. If L is 0, then the pixel is black and X, Y, Z are all 0. Therefore, I coded as:

```
if(L == 0.0):
    X = Y = Z = 0.0
else:
    uprime = (u + 13 * uw * L) / (13 * L)
    vprime = (v + 13 * vw * L) / (13 * L)
```

Secondly, we divide a certain value by v' to obtain X and Y. According to the formula, if v'=0, Z and X are both 0 too:

```
if(vprime == 0.0):
    X = Z = 0.0
else:
    X = Y * 2.25 * uprime / vprime
    Z = Y * (3.0 - 0.75 * uprime - 5.0 * vprime) / vprime
```

3. When we do the transformation from XYZ to Linear BGR (the matrix multiplication), and the transformation from Linear BGR to Non-linear BGR (the gamma correction), there is a chance that we

get a result that passes the limit of sRGB value range. The RGB values should always be in the range of [0,1]. So, we just clip the out-of-range values: if it's bigger than 1, we make it 1, and if it's less than 0, we make it 0:

```
def checkValue(x):
    if(x>1.0):
        x=1.0
    if(x<0.0):
        x=0.0
    return x
```

```
# matrix multiplication
r = 3.240479 * X + (- 1.53715) * Y + (-0.498535) * Z
g = (- 0.969256) * X + 1.875991 * Y + 0.041556 * Z
b = 0.055648 * X + (-0.204043) * Y + 1.057311 * Z
r = checkValue(r)
g = checkValue(g)
b = checkValue(b)

# gamma correction
b = gammaSingle(b)
g = gammaSingle(g)
r = gammaSingle(r)
b = checkValue(b)
g = checkValue(g)
r = checkValue(r)
```

4. In the linear scaling part, there is a situation that we may divide by 0 as well:

```
L=round((L-L_min)*100.0/(L_max-L_min))
```

If the image has only one luminance, like a pure color image, we'll have L_max = L_min = all the L values. In this case, we can't do the above division. I add an if statement before this:

```
if(L_max==L_min):
    outputImage = inputImage
```

If we have an image with a unique luminance value, there is no need to do the linear scaling, the output is simply the input image.

In the histogram equalization part, there was no such division step, but I kept this if statement as well, because it was the same case – there is no need to do the equalization if we only have one luminance level.

5. In the histogram equalization part, when calculating the equalized values table, we have the possibility of getting an equalized value larger than the biggest number of gray level. During our homework, we simply made it equal to the largest gray level. So, in the code I do the same thing:

```
for i in range(1,101):
    hist_tmp[i] = (f[i-1]+f[i])*0.5*m
    if(hist_tmp[i]>100.0):
        hist_tmp[i]=100.0
```

If we get an equalized value bigger than 100, we clip and make it 100.

6. As the project required, “all L values below the smallest L value in the window should be mapped to 0, and all L value above the largest L value in the window should be mapped to 100”.

In linear scaling part I coded as:

```
if(L>L_max):
    L=100.0
elif(L<L_min):
    L=0.0
else:
    L=round((L-L_min)*100.0/(L_max-L_min))
```

In histogram equalization part I coded as:

```
if(L>L_max):
    L=100.0
elif(L<L_min):
    L=0.0
else:
    L=hist_floor[int(L)]
```

7. In the histogram equalization part, to calculate the equalization table, I go from 0 to 100 as the luminance/gray level (101 levels in total) and stored the equalized values in a list. (hist_tmp) And then applied np.floor() function to it, to get a new list of floored values. This hist_floor list is the result of the equalization. The values are the equalized values, and the indexes of the list are the original luminance levels to be equalized.

```
hist_floor = np.floor(hist_tmp)
```

Results:

Results totally depend on the image and the window we choose. Sometimes the window specified has a very limited number of luminance levels, so that the result image could be considered as “bad”: it would lose a lot of details in the very dark or/and bright area, and some of the colors in the output image would seem incorrect. Generally, histogram equalization method gives us a little more intense result.

Result examples:

Left one is by linear scaling and right one is by histogram equalization. The third shows the window with the original image.

Linear scaling result is not obvious in this set ↓

L_max=92, L_min=0

linear scaling histogram eq original (window in the center)

Details in dark and bright area lost largely for both two methods, could consider as "bad"↓

L_max=61, L_min=33

linear scaling histogram eq original (tiny window in the red circle)

Histogram equalization enhances more the contrast, and linear scaling has almost no effect in this case↓

L_max=100, L_min=0

linear scaling histogram eq original (window in the center)

Details in bright area disappeared a lot, could consider as “bad”↓

L_max=75, L_min=1,

linear scaling histogram eq original (tiny window in the red circle)

As the programs will output directly the input image when there is only one level of luminance, these two programs will do nothing on a “pure color” image↓

L_max=L_min=58

linear scaling

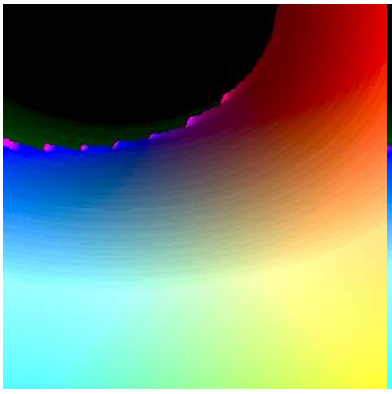
histogram eq

original (window in the center)

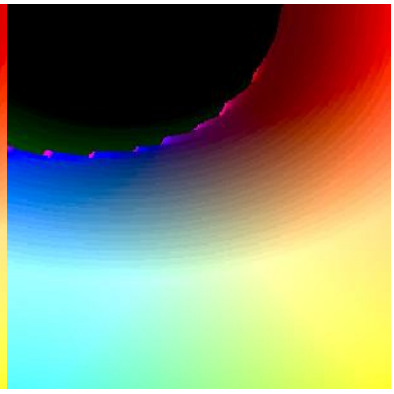
Can’t handle the image without distinct objects or edges on it, like this image of gradient colors, some colors returned are not correct↓

L_max=71, L_min=38

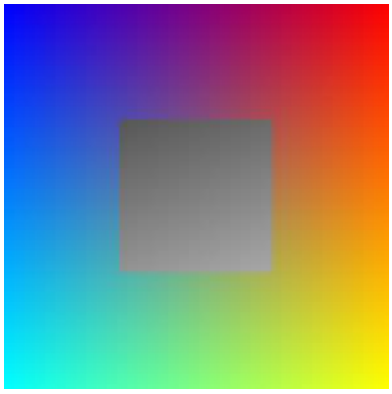
linear scaling



histogram eq



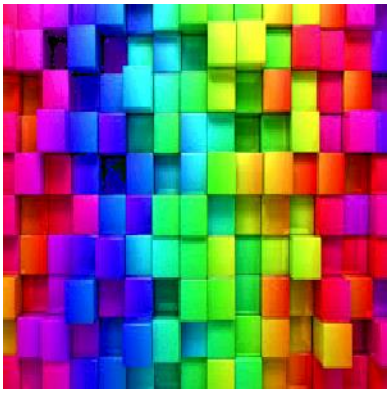
original (window in the center)



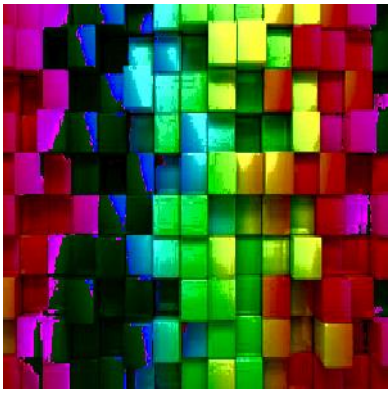
Works better for an image with edges on it, but still, especially for the histogram equalization method, some colors not correct - blue/purple colors turned to green↓

L_max=98 L_min=3

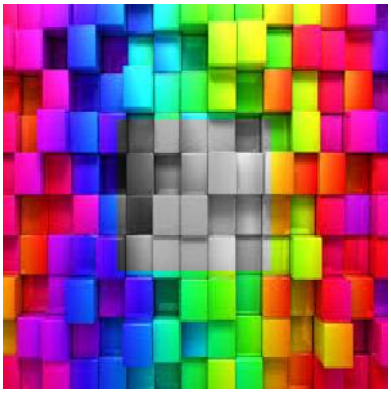
linear scaling



histogram eq



original (window in the center)



(END OF THE REPORT)