



Master in Science (Artificial Intelligence)
(MSAI)

AI-6121

Assignment_01: Histogram Equalization

Name of Student:
Teo Lim Fong

Matriculation No: G2101964G

Table of content

Contents

Table of content	2
Methodology	3
Histogram Equalization	3
Enhanced Histogram Equalization	5
Pros and Cons of Histogram Equalization	9
Result	10
Reference	14
Annex	15

Methodology

Histogram Equalization

Step 1: The given testing images are in RGB channels. Therefore, we need to convert RGB into colour domain such as HSV or YCRCB which contains intensity channel.

I wrote a path to call all testing images stored in MSAI file and save it into another folder. Then the coding will read the image and convert it from BGR to YUV colour domain. Only Y channel contain intensity so we have to spilt all the 3 channels.

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import os

path = 'C:/Users/LimFong/Desktop/MSAI/'
save_path = 'C:/Users/LimFong/Desktop/MSAI/CV1'

for image in glob.glob(path + "/*.JPG"): #change to JPEG
    filename = os.path.basename(image)
    new_filename = filename[:-4] + "_HE" + filename[-4:] # change to [-3:] if the images are in JPEG format
    img = cv2.imread (image)
    ycbcr = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    y,cb,cr = cv2.split(ycbcr)
```

Step 2: Plot and show the original histogram (Y channel)

In order to plot the histogram, I need to create an empty array (H_array) with a dimension of 1 X 256. Then, I create a while loop to get all the pixel values into the array. Since there is no 0 pixel value, I used np.count_nonzero to count the pixel. This mean that if k == 5, it will read the 5th pixel value in Y channel and place it into the empty array. Lastly, I normalised the histogram by dividing the resolution of the image

```
(n,m)= y.shape

k= 0
kk=0
L= 255
H_array = np.zeros((256), dtype=int)

while k<256:
    H_array[k] = np.count_nonzero(y==k)
    k+=1

res = n*m
H_normalise = H_array/res
```

Step 3: Using the given formula, sum all the normalised pixel and multiple by the number of bins (L).

$$s_k = (L - 1) \sum_{j=0}^k p_j$$

I create an empty list and using for loop to save every index after the summation. I convert the updated list into an array. To find the transform function, I just need to multiple by the number of bins (L).

I create an empty array to be replaced by the new updated intensity pixel.

```
new_list=[]
j=0
for i in range(0,len(H_normalise)):
    j+=H_normalise[i]
    new_list.append(j)

new_array = np.array(new_list)

sk = np.uint8(L * new_array) #

new_y = np.zeros_like(y)

for ii in range(0, n):
    for jj in range(0, m):
        new_y[ii, jj] = sk[y[ii, jj]]
```

Step 4: Combine all the three channels and convert it to BGR (using cv2 to save)

Final step is to combine all the three channels (new Y channel) together and convert it to BGR.

```
new_ycbcr = cv2.merge([new_y, cb, cr])
new_rgb = cv2.cvtColor(new_ycbcr, cv2.COLOR_YUV2BGR)

new_H_array = np.zeros((256), dtype=int)
print(new_filename)
while kk<256:
    new_H_array[kk] = np.count_nonzero(new_y==kk)
    kk+=1

cv2.imwrite(save_path + "/" + new_filename, new_rgb)
```

Enhanced Histogram Equalization

This Enhanced Histogram Equalization algorithm is called ‘Contrast enhancement of brightness-distorted images by improved adaptive gamma correction’ [1]

One of the cons of HE is the overall abnormal brightness in the image. When HE is applied to dim images, the image may look excessive enhanced especially its brightness. Another contrast enhancement method is Gamma Correction. The problem with Gamma Correction is that it is hard to find the correct gamma coefficient for each pixel. Therefore, the author uses CDF to find the ideal gamma coefficient. With the help of adaptive gamma correction, it will enhance the contrast of the image especially on dimmed image.

The author divide image into two sections, dim and bright.

For dimmed image, truncation is used to avoid over saturation and over-enhancement when the existing pixels are already closed to 255. Not to mention when the pixel is exponentially increased by the gamma coefficient. Setting a threshold can greatly improve and also prevent bright regions from degrading the image. It can also prevent any loss information.

For bright image, it is first subtracted from 255 to get a negative image. The pixel intensity distribution in negative image is somehow similar to dim image. However, the weighting factor, alpha, used in bright image is 0.25 instead of 0.75. With such configuration, the bright image will be enhanced accordingly. If bright image is not converted, the weighting factor will increase the intensity value of the image such that even more pixels are closed or equal to 255.

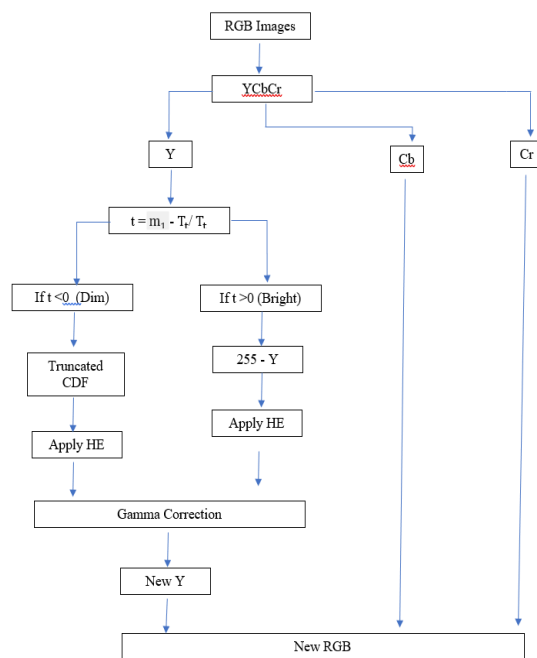


Figure 1: Flowchart of Enhanced HE with Adaptive Gamma Correction

Step 1: Create directory and read the images in the folder. Then convert RGB images into YCrCb.

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
import glob
import os

path = 'C:/Users/LimFong/Desktop/MSAI/'
save_path = 'C:/Users/LimFong/Desktop/MSAI/GAMMA/'

for image in glob.glob(path + "/*.JPEG"):
    filename = os.path.basename(image)
    new_filename = filename[:-4] + "_GAMMA" + filename[-5:]
    print(new_filename)
    img = cv2.imread(image)
    ycrb = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

    y,cr,cb = cv2.split(ycrb)
    m,n = y.shape
    res = m*n
```

Step 2: From the article, set the global brightness (112) and find the mean value in Y channel. t is to determine if the image is dim (-ve) or bright (+ve).

```
gb = 112 # global brightness

mean = np.sum(y/res)
t = (mean - gb) / gb # if negative mean dim, positive mean bright
##### UC #####
```

Step3: Sum all the normalised pixel and multiple by the number of bins (L).

```
k = 0
kk = 0
kkk = 0
L = 255
img_new = y.copy()

intensity_matrix = np.arange(256)

H_array = np.zeros((256), dtype=int)
new_dim = np.zeros((256), dtype=int)
new_bri = np.zeros((256), dtype=int)

while k<256:
    H_array[k] =np.count_nonzero(y == k)
    k+=1

H_normalise = H_array/res
```

Step 4a: If the value of t is less than 0, then set the alpha value 0.75. The next four lines are just the formula given by the author to find the normalised weighting distribution function. Next, find the summation of normalised weighting distribution function, which also called the CDF (Cumulative Distribution Function) For dim images, apply another formula given by the author to find the gamma coefficient. Last, apply gamma correction on each pixel and save on a new array and combine the other two channels and convert it to new enhanced RGB.

```
if t <0: # if image is dim

    a = 0.75
    H_max = max(H_normalise)
    H_min = min(H_normalise)
    formula_a = (H_normalise - H_min) / (H_max - H_min)
    H_w = H_max*(formula_a**a)
    sum_H_w = sum(H_w)
    normalised_H_W = H_w/sum_H_w

    for i in range(0, len(normalised_H_W)):
        j+=normalised_H_W[i]
        new_list.append(j)

    new_array = np.array(new_list)

    inverse = np.maximum(0.5,1 - new_array)

    for w in intensity_matrix:
        pixel_in = np.round(255 * (w / 255)**inverse[w])
        img_new[y==w] = pixel_in

    new_ycbcr1 = cv2.merge((img_new, cb, cr))
    new_rgb1 = cv2.cvtColor(new_ycbcr1, cv2.COLOR_YUV2RGB)
    cv2.imwrite(save_path + "/" + new_filename, new_rgb1)
```

$$p_w(l) = p_{\max} \left(\frac{p(l) - p_{\min}}{p_{\max} - p_{\min}} \right)^\alpha$$

$$\gamma'_w(l) = \max(\tau, 1 - c_w(l))$$

Step 4b: If the value of t is more than 0, then set the alpha parameter 0.25. For bright image, it is required to subtract 255 from the Y channel. After that, same as **Step 4a**, using the given formula to find the normalised weighting distribution function and the CDF. To find the gamma coefficient, simply take $1 - \text{CDF}$ and apply gamma correction on each pixel. Before merging the channel, the new Y channel need to be subtracted from 255 and convert it back to new enhanced RGB image.

```

if t > 0: # if image is bright
    a = 0.25
    y = 255 - y # negative image
    H_max = max(H_normalise)
    H_min = min(H_normalise)
    formula_a = (H_normalise - H_min) / (H_max - H_min)
    H_w = H_max * (formula_a ** a)
    sum_H_w = sum(H_w)
    normalised_H_W = H_w / sum_H_w
    normalised_H_W = np.flipud(normalised_H_W) #negative image

    for i in range(0, len(normalised_H_W)):
        j += normalised_H_W[i]
        new_list.append(j)

    new_array = np.array(new_list)
    inverse = 1 - new_array

    for w in intensity_matrix:
        pixel_in = np.round(255 * (w / 255) ** inverse[w])
        img_new[y==w] = pixel_in

    img_new = 255 - img_new

    new_ybcr1 = cv2.merge((img_new, cb, cr))
    new_rgb1 = cv2.cvtColor(new_ybcr1, cv2.COLOR_YUV2RGB)

    cv2.imwrite(save_path + "/" + new_filename, new_rgb1)
    while kk < 256:
        new_bri[kk] = np.count_nonzero(img_new == kk)
        kk += 1

```


Pros and Cons of Histogram Equalization

Pros:

- Enhance the quality of the image by spreading the intensity between 0 to 255 instead of one lump at either 0 or 255
- Histogram Equalization improved the global contrast of an image. It is useful when the image is either too bright or too dark.

Cons:

- Histogram Equalization focus on global contrast but not local contrast. Hence, some information will be loss.
- When applying Histogram Equalization, it will not only increase the contrast of the image but also increase the background noise and sometimes will decrease the foreground image too.
- Histogram Equalization only can be applied in intensity channel such as Gray, Value in HSV and Y in YCrCb.
- If the image is too bright, after applying Histogram Equalization, part of the image will still remain bright

Result

These are the results after Histogram Equalization.




	Original	Histogram Equalization	Enhanced HE
Sample 1			

Table 1: Results of Histogram Equalization and Enhanced HE on sample 1 image

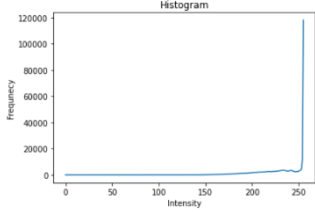
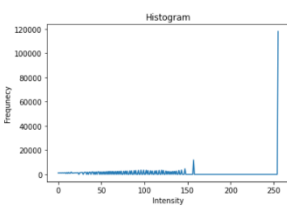
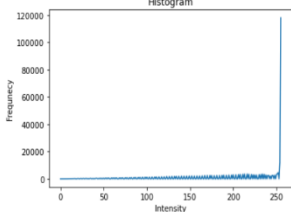
	Original	Histogram Equalization	Enhanced HE
Sample 1			

Table 2: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 1 image




	Original	Histogram Equalization	Enhanced HE
Sample 2			

Table 3: Results of Histogram Equalization and Enhanced HE on sample 2 image

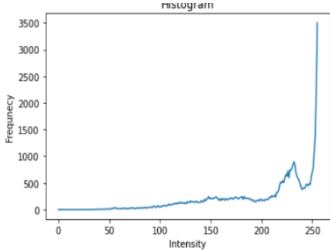
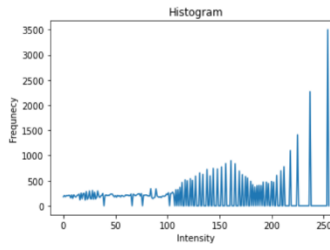
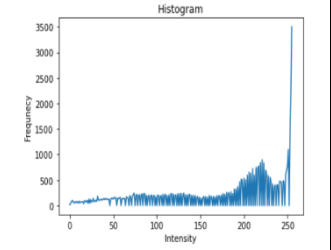
	Original	Histogram Equalization	Enhanced HE
Sample 2			

Table 4: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 2 image



	Original	Histogram Equalization	Enhanced HE
Sample 3			

Table 5: Results of Histogram Equalization and Enhanced HE on sample 3 image

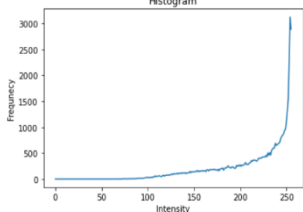
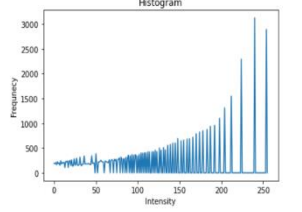
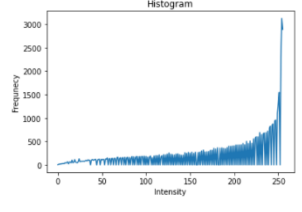
	Original	Histogram Equalization	Enhanced HE
Sample 3			

Table 6: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 3 image




	Original	Histogram Equalization	Enhanced HE
Sample 4			

Table 7: Results of Histogram Equalization and Enhanced HE on sample 4 image

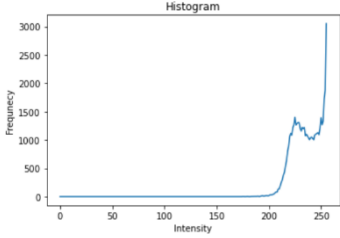
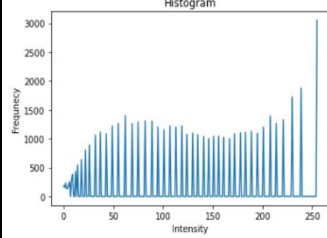
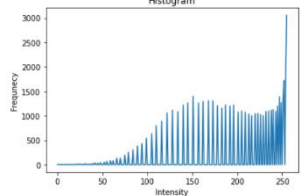
	Original	Histogram Equalization	Enhanced HE
Sample 4			

Table 8: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 4 image

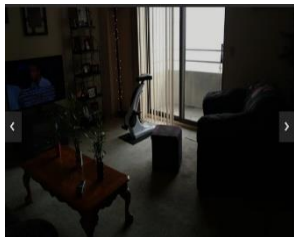


	Original	Histogram Equalization	Enhanced HE
Sample 5			

Table 9: Results of Histogram Equalization and Enhanced HE on sample 5 image

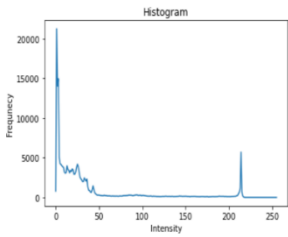
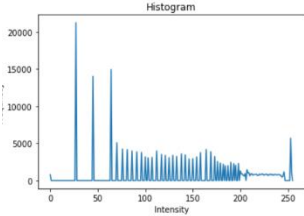
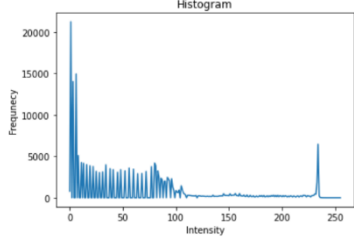
	Original	Histogram Equalization	Enhanced HE
Sample 5			

Table 9: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 5 image



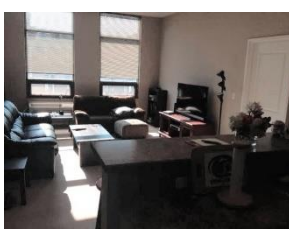
	Original	Histogram Equalization	Enhanced HE
Sample 6			

Table 11: Results of Histogram Equalization and Enhanced HE on sample 6 image

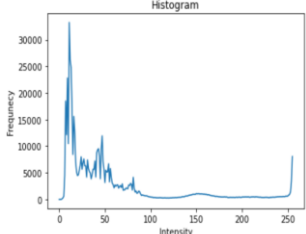
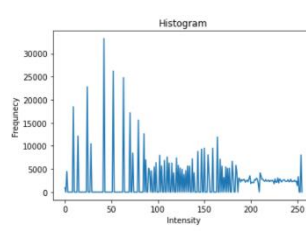
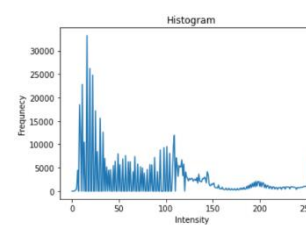
	Original	Histogram Equalization	Enhanced HE
Sample 6			

Table 12: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 1 image

	Original	Histogram Equalization	Enhanced HE
Sample 7			

Table 13: Results of Histogram Equalization and Enhanced HE on sample 7 image

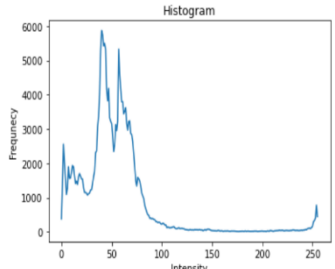
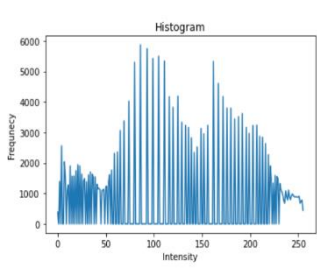
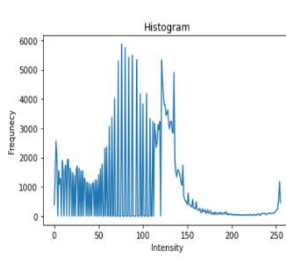
	Original	Histogram Equalization	Enhanced HE
Sample 7			

Table 14: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 7 image




	Original	Histogram Equalization	Enhanced HE
Sample 8			

Table 15: Results of Histogram Equalization and Enhanced HE on sample 8 image

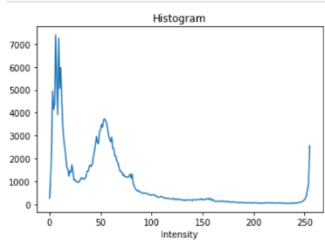
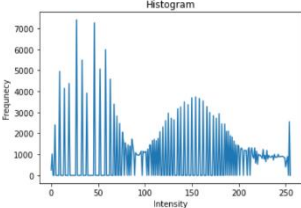
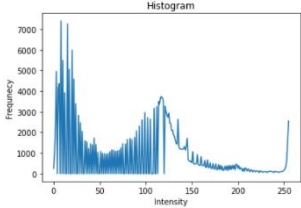
	Original	Histogram Equalization	Enhanced HE
Sample 8			

Table 16: Distribution of the intensity values using Histogram Equalization and Enhanced HE on sample 8 image

Reference

[1] Gang Cao, Li Hui, HuaWei Tan. Contrast enhancement of brightness-distorted images by improved adaptive gamma correction 2018

Annex

Histogram Equalization

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import os

path = 'C:/Users/LimFong/Desktop/MSAI/a/'
save_path = 'C:/Users/LimFong/Desktop/MSAI/CV1'

for image in glob.glob(path + "/*.JPG"): #change to JPEG
    filename = os.path.basename(image)
    new_filename = filename[:-4] + "_HE" + filename[-4:] # change to [-3:] if the images are in JPEG format
    img = cv2.imread(image)
    ycbcr = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    y,cb,cr = cv2.split(ycbcr)

    (n,m)= y.shape

    k= 0
    kk=0
    L= 255
    H_array = np.zeros((256), dtype=int)

    while k<256:
        H_array[k] = np.count_nonzero(y==k)
        k+=1

    res = n*m
    H_normalise = H_array/res

    new_list=[]
    j=0
    for i in range(0,len(H_normalise)):
        j+=H_normalise[i]
        new_list.append(j)

    new_array = np.array(new_list)

    sk = np.uint8(L * new_array) #

    new_y = np.zeros_like(y)

    for ii in range(0, n):
        for jj in range(0, m):
            new_y[ii, jj] = sk[y[ii, jj]]

    new_ycbcr = cv2.merge([new_y, cb, cr])
    new_rgb = cv2.cvtColor(new_ycbcr, cv2.COLOR_YUV2BGR)

    new_H_array = np.zeros((256), dtype=int)
    print(new_filename)
    while kk<256:
        new_H_array[kk] = np.count_nonzero(new_y==kk)
        kk+=1

    cv2.imwrite(save_path + "/" + new_filename, new_rgb)
```


Enhanced Histogram Equalization

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
import glob
import os

path = 'C:/Users/LimFong/Desktop/MSAI/'
save_path = 'C:/Users/LimFong/Desktop/MSAI/GAMMA/'

for image in glob.glob(path + "/*.JPEG"):
    filename = os.path.basename(image)
    new_filename = filename[:-4] + "_GAMMA" + filename[-5:]
    print(new_filename)
    img = cv2.imread(image)
    ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

    y,cr,cb = cv2.split(ycrcb)
    m,n = y.shape
    res = m*n
    gb = 112 # global brightness

    mean= np.sum(y/res)
    t = (mean- gb)/ gb # if negative mean dim, postive mean bright
    ##### HE #####

    k = 0
    kk = 0
    kkk= 0
    L = 255
    img_new = y.copy()

    intensity_matrix = np.arange(256)

    H_array = np.zeros((256), dtype=int)
    new_dim = np.zeros((256), dtype=int)
    new_bri = np.zeros((256), dtype=int)

    while k<256:
        H_array[k] =np.count_nonzero(y == k)
        k+=1

    H_normalise = H_array/res

    new_list= []
    j = 0

    if t <0: # if image is dim

        a = 0.75
        H_max = max(H_normalise)
        H_min = min(H_normalise)
        formula_a = (H_normalise - H_min) / (H_max - H_min)
        H_w = H_max*(formula_a**a)
        sum_H_w = sum(H_w)
        normalised_H_w = H_w/sum_H_w

        for i in range(0, len(normalised_H_w)):
            j+=normalised_H_w[i]
            new_list.append(j)

        new_array = np.array(new_list)

        inverse = np.maximum(0.5,1 - new_array)

    for w in intensity_matrix:
        pixel_in = np.round(255 * (w / 255)**inverse[w])
        img_new[y==w] = pixel_in

    new_ybcr1 = cv2.merge((img_new, cb, cr))
    new_rgb1 = cv2.cvtColor(new_ybcr1, cv2.COLOR_YUV2RGB)
    cv2.imwrite(save_path + "/" + new_filename, new_rgb1)
    while kkk<256:
        new_dim[kkk] =np.count_nonzero(img_new == kkk)
        kkk+=1
```



```

if t > 0: # if image is bright
    a = 0.25
    y = 255 - y # negative image
    H_max = max(H_normalise)
    H_min = min(H_normalise)
    formula_a = (H_normalise - H_min) / (H_max - H_min)
    H_w = H_max * (formula_a ** a)
    sum_H_w = sum(H_w)
    normalised_H_w = H_w / sum_H_w
    normalised_H_w = np.flipud(normalised_H_w) #negative image

    for i in range(0, len(normalised_H_w)):
        j += normalised_H_w[i]
        new_list.append(j)

    new_array = np.array(new_list)
    inverse = 1 - new_array

    for w in intensity_matrix:
        pixel_in = np.round(255 * (w / 255) ** inverse[w])
        img_new[y == w] = pixel_in

    img_new = 255 - img_new

    new_ycbcr1 = cv2.merge((img_new, cb, cr))
    new_rgb1 = cv2.cvtColor(new_ycbcr1, cv2.COLOR_YUV2RGB)

    cv2.imwrite(save_path + "/" + new_filename, new_rgb1)
    while kk < 256:
        new_bri[kk] = np.count_nonzero(img_new == kk)
        kk += 1

```

