R13922136 周洛君

Computer Vision Homework 3: Report

This report will include the 3 parts of the problem, and each with 2 pictures, one is the resulting image, and the other is the corresponding histogram.

Things that are almost the same in the 3 questions:

In each part of my code, the names are given in a similar form, the names of the image, and their corresponding histogram are:

(a) orig_lena, hist_orig_lena

(b) lena_divided_by_3, hist_lena_divided_by_3

(c) lena_divided_by_3_equalized, hist_ lena_divided_by_3_equalized

Also, in each part, before cv2.imshow is the part to generate the image, and after that, the histogram (with the names stated above), will first be initialized by np.zeros, then two layers of for loop will loop over the height and width to include all the pixels in the resulting image to sum up the number of pixels for each intensity value.

Some more detailed information:

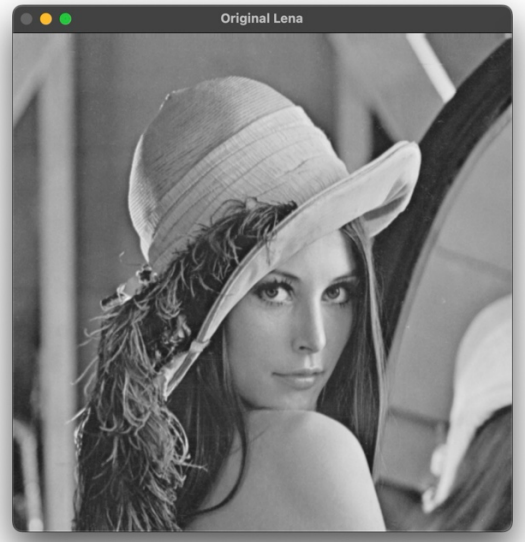
- To show the image, I use 3 cv2 functions:

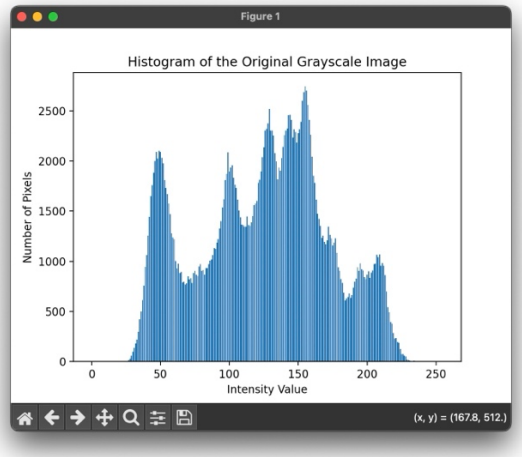
1. cv2.imshow()
2. cv2.waitkey()
3. cv2.destroyAllWindows()

- To plot the histogram, I use 5 plt functions:

1. plt.xlabel()
2. plt.ylabel()
3. plt.title()
4. plt.bar()
5. plt.show()

(a) original image and its histogram





(The histogram in this part is the same as the one in HW2, so I didn't use a bigger picture in order to maintain a pretty layout, if bigger image is need, please let me know!)

In this part, nothing is needed to do to the image, we only need to use cv2.imread() to read in the image, and while setting the flag to 0, we got the greyscale image.

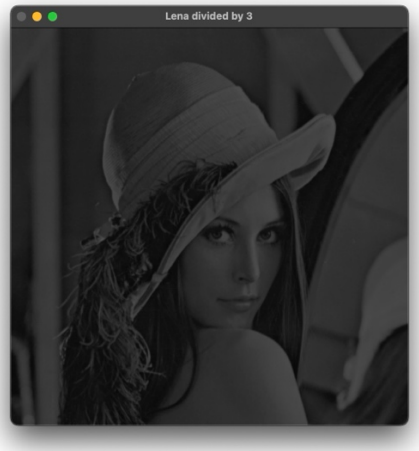
Since we're calculating the number of each intensity, hist_orig_lena is initialized by 256 zeros (from 0 to 255), and the data type is 32 bits unsigned integer. This is the same for the next two parts.

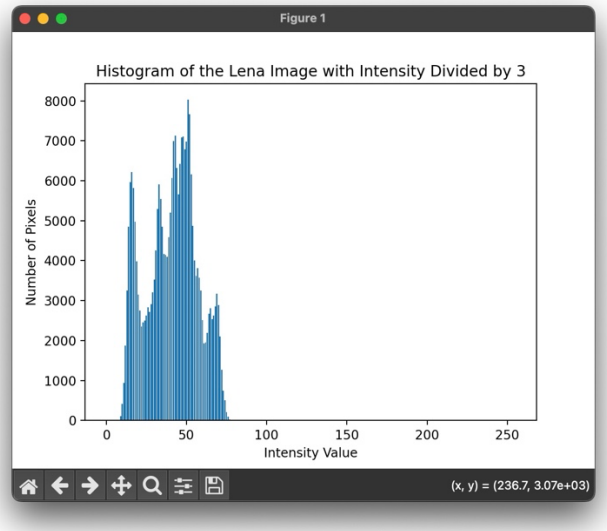
To calculate the number of pixels for each intensity value:

```
15      hist_orig_lena = np.zeros(256, dtype = np.uint32)
16      for i in range(orig_lena.shape[0]):
17          for j in range(orig_lena.shape[1]):
18              hist_orig_lena[orig_lena[i][j]] += 1
```

The orig_lena[i][j] in line 18 is the intensity value of each pixel, by using it as an index, we can increase the number corresponding to this intensity value by one.

(b) image with intensity divided by 3 and its histogram





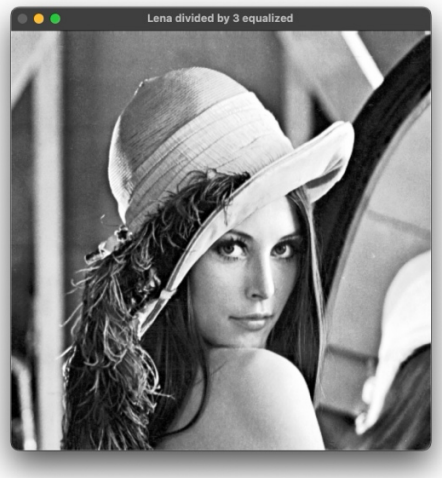
We only need one line to get this lena_divided_by_3 image, which is:

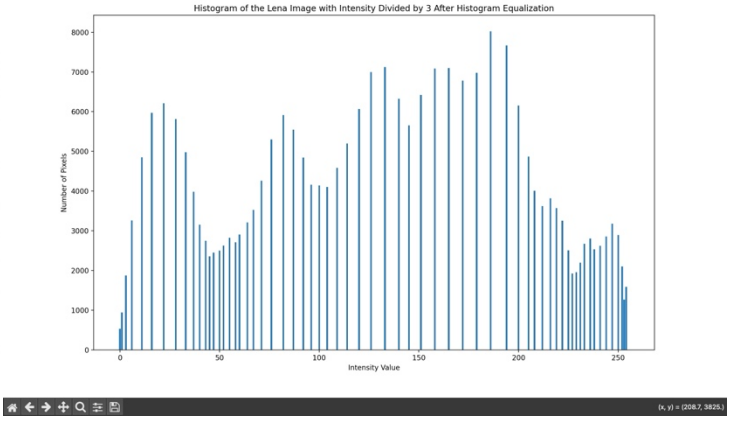
```
28        lena_divided_by_3 = orig_lena // 3
```

Here, integer division is used (//), since pixel values are integers.

The histogram generating process is the same as the previous part, just change the histogram name to hist_lena_divided_by_3.

(c) image after applying histogram equalization to (b) and its histogram





This part is a little bit different from the others.

First, we have to calculate the cdf of the image generated in (b) (lena_divided_by_3), and this is done by using np.cumsum(). The result is saved in a ndarray named cdf:

```
50      cdf = hist_lena_divided_by_3.cumsum()
```

Since hist_ lena_divided_by_3 saves the number of pixels for each intensity value, using cumsum() will accumulate the number of pixels, letting cdf able to tells us, for any intensity, how many pixels have intensities ≤ that value. of intensity. For example, if hist_ lena_divided_by_3 is [3,2,5], then cdf would be [3,5,10]([3, 3+2, 3+2+5]).

Second, we have to do normalization:

```
52      # subaim: normalize cdf
53      # explain: we scale the cdf to the full range of intensity values (0 to 255),
54      # explain: and divide by the total number of pixels(which is the last element of cdf)
55      cdf_normalized = cdf * 255 / cdf[-1]
```

As I wrote in the comments, we scale the cdf by multiplying 255, and then we do the division using the total number of pixels (which we got by accessing the last element in the ndarray cdf.) But the result in cdf_normalized is not of integer type, so I save the integer version of cdf_normalized in the variable lookup_table, which also makes the variable name clearer to represent its usage.

```
58      lookup_table = np.uint8(cdf_normalized)
```

Last, after the previous step, we have the lookup_table, with each index corresponding to the original intensity. For example, if a pixel in the original image has intensity value = 3, we can see lookup_table[3] to see which intensity value it should be mapped to.

Now, we have lena_divided_by_3, which is an array of the original pixel intensities, we then do mapping to all the pixels at the same time to get the equalized image.

```
66      lena_divided_by_3_equalized = lookup_table[lena_divided_by_3]
```