This program implements the histogram matching algorithm that loads the **source image**(lena.bmp) for matching its histogram to the one of **target image**(license_plate.bmp).

programmer: Shih-Shinh Huang

date:: 2021.10.19

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [3]:

```
import google.colab.patches as colab
import cv2
import numpy
import matplotlib.pyplot as plt
```

load the source image (lena.bmp) and display it

In [4]:

```
source_mat = cv2.imread("/content/drive/MyDrive/Colab Notebooks/histogram ma
colab.cv2_imshow(source_mat)
```

load the target image (license_plate.bmp) and display it

```
In [5]: target_mat = cv2.imread("/content/drive/MyDrive/Colab Notebooks//histogram m
        colab.cv2_imshow(target_mat)
```

Step 1: find transformation function $T(.)$ that maps source histogram to uniform distribution $U(.) = 1$

compute the histogram of source image

In [6]:

```
source_hist = numpy.zeros((256), dtype=float)

rows = source_mat.shape[0]
cols = source_mat.shape[1]

for r in range(rows):
  for c in range(cols):
    gray_value = source_mat[r, c]
    source_hist[gray_value] = source_hist[gray_value] + 1

for i in range(256):
  source_hist[i] = source_hist[i] / (rows * cols)
```

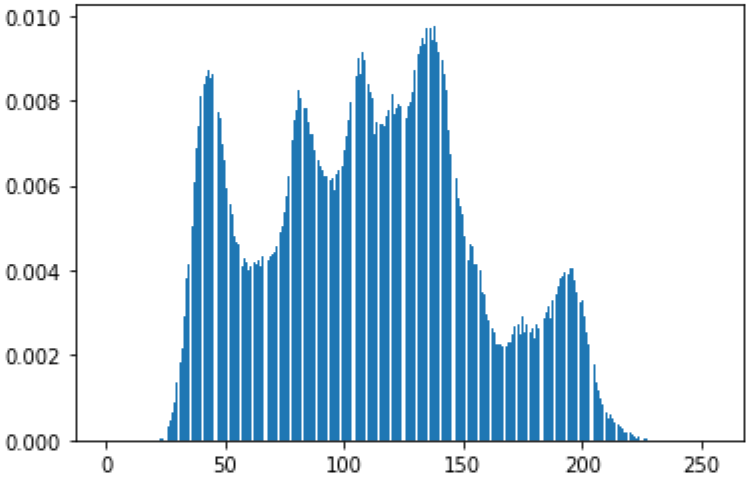
display the histogram of the source image

In [7]:

```
plt.bar(range(source_hist.shape[0]), source_hist)
```

Out[7]:

```
<BarContainer object of 256 artists>
```



compute the transformation function $T(.)$

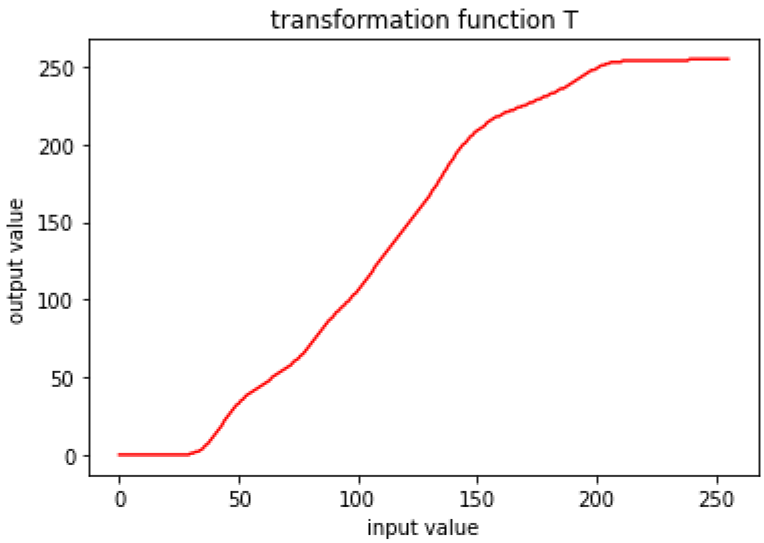
In [8]:

```
# T: transformation function
T = numpy.zeros((256), dtype=int)

# acc_pr: the accumulated probability of the input histogram
acc_pr = 0.0;

for x in range(256):
  acc_pr = acc_pr + source_hist[x]
  T[x] = int(255.0 * acc_pr)
```

display the transformation function $T(.)$

In [9]:

```
plt.title('transformation function T')
plt.xlabel('input value')
plt.ylabel('output value')
plt.plot(range(T.shape[0]), T, 'red')
plt.show()
```



Step 2: find the transformation function $G(.)$ that maps target histogram to uniform distribution $U(.) = 1$

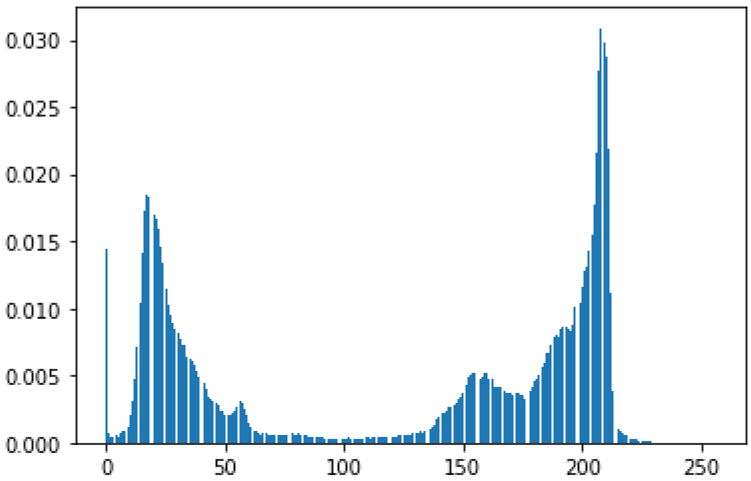
compute the histogram of target image

```
In [10]: target_hist = numpy.zeros((256), dtype=float)

         rows = target_mat.shape[0]
         cols = target_mat.shape[1]

         for r in range(rows):
           for c in range(cols):
             gray_value = target_mat[r, c]
             target_hist[gray_value] = target_hist[gray_value] + 1

         for i in range(256):
           target_hist[i] = target_hist[i] / (rows * cols)
```

display the histogram of the target image

```
In [11]: plt.bar(range(target_hist.shape[0]), target_hist)
```

```
Out[11]: <BarContainer object of 256 artists>
```



compute the transformation function $G(.)$

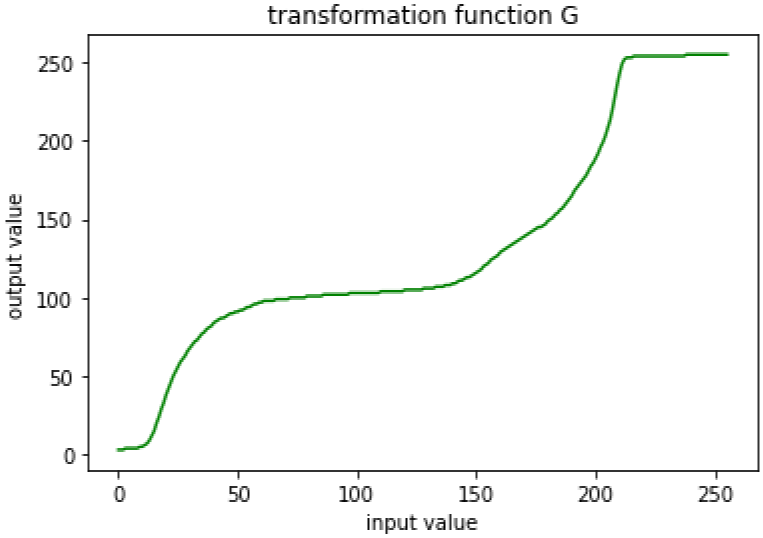
In [12]:

```
# G: transformation function
G = numpy.zeros((256), dtype=int)

# acc_pr: the accumulated probability of the input histogram
acc_pr = 0.0;

for x in range(256):
  acc_pr = acc_pr + target_hist[x]
  G[x] = int(255.0 * acc_pr)
```

display the transformation function $G(.)$

In [13]:

```
plt.title('transformation function G')
plt.xlabel('input value')
plt.ylabel('output value')
plt.plot(range(G.shape[0]), G, 'green')
plt.show()
```



Step 3: compute $z = G^{-1}(u)$

case 1: mapping is not unique.

scan over all y for finding the smallest x that makes $y = G(x)$

In [14]:

```
G_inv = numpy.zeros((256), dtype=int)

for y in range(256):

  # initially assume that the mapping of inverse does not exist
  G_inv[y] = -1

  # find the x that outputs y, that is, y = G(x)
  for x in range(256):

    # find the mapping of G_inv[y]
    if G[x] == y:
      G_inv[y] = x

      # make the x is the smallest one
      break;
```

case 2: no mapping exists

use the output of the value that is the closet to the current one

In [15]:

```
for y in range(256):

  # check if no mapping exists
  if G_inv[y] == -1:

    for offset in range(256):
      # check if the mapping at y - offset exists
      if y - offset >=0 and G_inv[y - offset] != -1:
        G_inv[y] = G_inv[y - offset]
        break;

      # check if the mapping at y + offset exists
      if y + offset < 255 and G_inv[y + offset] != -1:
        G_inv[y] = G_inv[y + offset]
        break;
```

display the inverse function $G^{-1}(.)$

In [ ]:

```
plt.title('transformation function G inverse')
plt.xlabel('input value')
plt.ylabel('output value')
plt.plot(range(G_inv.shape[0]), G_inv, 'blue')
plt.show()
```

Step 4: form the function $Z(.) = G^{-1}(T(r))$ for mapping

compute the Z function

In [17]:

```
Z = numpy.zeros((256), dtype=int)

for x in range(256):
  y = T[x]
  Z[x]  = G_inv[y]
```

display the transformation function Z(.)

In [ ]:

```
plt.title('transformation function Z=G_inv(T(.))')
plt.xlabel('input value')
plt.ylabel('output value')
plt.plot(range(Z.shape[0]), Z, 'yellow')
plt.show()
```

do the histogram mapping

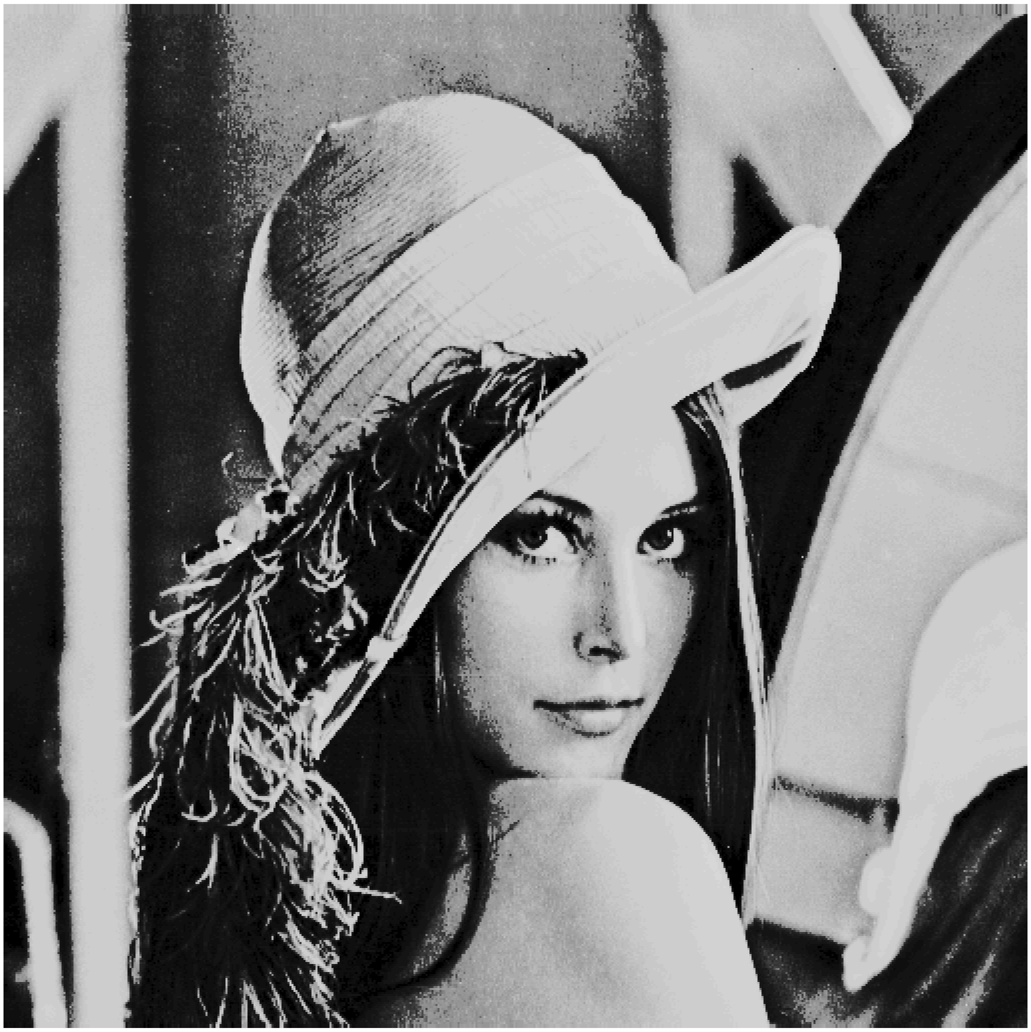
In [19]:

```
# create an output image
out_mat = numpy.zeros(source_mat.shape, dtype=numpy.uint8)

rows = source_mat.shape[0]
cols = source_mat.shape[1]

for r in range(rows):
  for c in range(cols):
    gray_value = source_mat[r, c]
    out_value = Z[gray_value]
    out_mat[r, c] = out_value
```

diplay the image after histogram matching

In [20]:

```
colab.cv2_imshow(out_mat)
```



display the histogram of the image after histogram matching

In [21]:

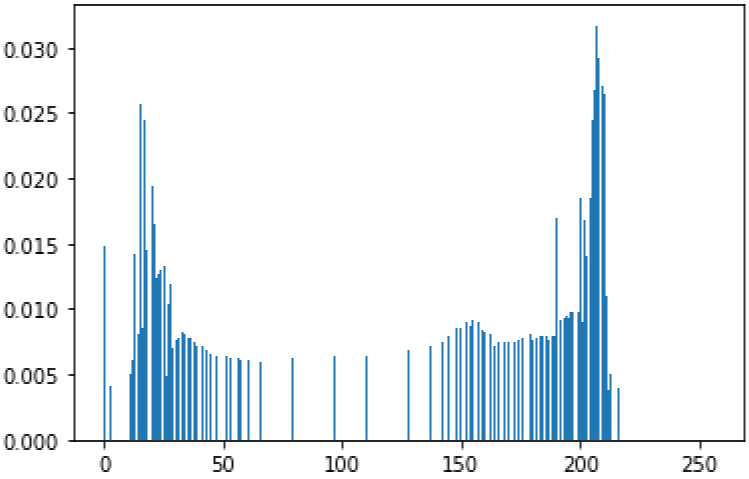
```
out_hist = numpy.zeros((256), dtype=float)

for r in range(rows):
  for c in range(cols):
    gray_value = out_mat[r, c]
    out_hist[gray_value] = out_hist[gray_value] + 1

for i in range(256):
  out_hist[i] = out_hist[i] / (rows * cols)
```

In [22]:

```
plt.bar(range(out_hist.shape[0]), out_hist)
```

Out[22]:

```
<BarContainer object of 256 artists>
```


0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250