Image Segmentation Method

Class Assignment for Signal and Image Processing course as a part of Master of Statistics curriculum

Name: Subhrajyoty Roy
Roll: MB1911

8th November, 2020

1 Introduction

Image Segmentation refers to the task to seperate foreground from the background of an image. In general, the objects present in a digital image generally have different distribution of pixel intensities than the distribution of pixel intensities of the background. The task becomes to optimally determine a threshold such that any pixel having lesser intensity than the threshold can be associated with the background while any pixel having higher intensity can be attributed to the objects present in an image.

2 Exploratory Analysis

We shall use `Python` programming language for performing all operations on the images. Firstly, we import three basic packages as follows:

- `opencv`, which is used to read the image from a file and store it into a 2d matrix.
- `matplotlib`, which will be used to display a 2d matrix as a grayscale image.
- `numpy`, which will be used to perform matrix operations.

[1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

We shall be using the popular `cameraman` image for demonstration purpose.

[2]:
```
image = cv2.imread('cameraman.tif')     # read the image into the matrix form
image[:5, :5, 0]     # print first 5x5 submatrix of the matrix for checking
```

[2]:
```
array([[156, 159, 158, 155, 158],
       [160, 154, 157, 158, 157],
       [156, 159, 158, 155, 158],
       [160, 154, 157, 158, 157],
       [156, 153, 155, 159, 159]], dtype=uint8)
```

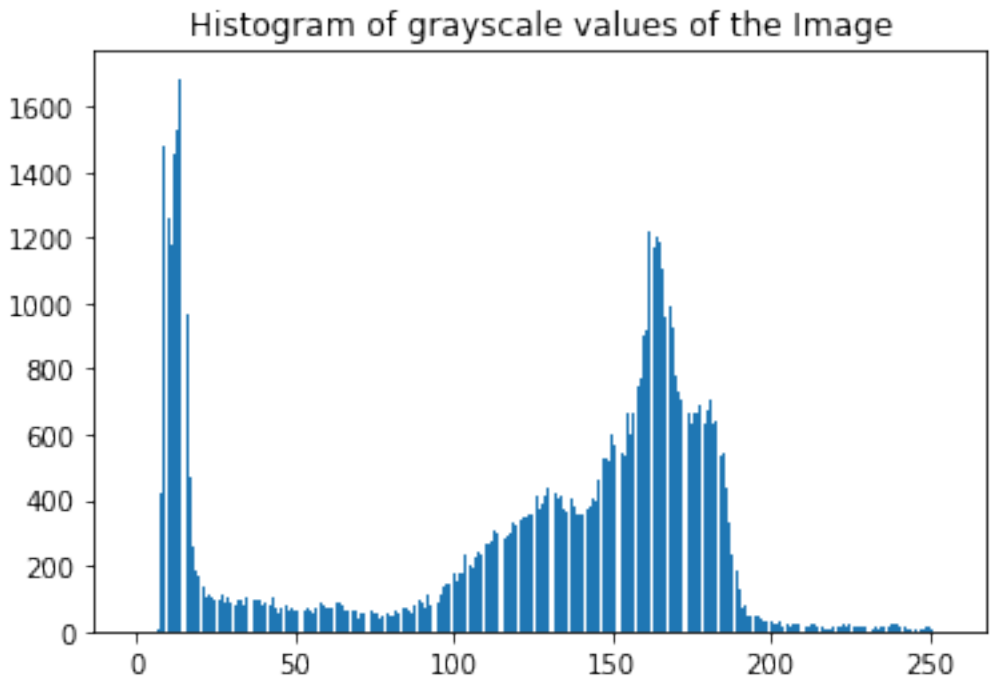
Let us now see how the image actually looks.

[3]:
```
plt.imshow(image)
plt.title('Cameraman Image')
plt.show()
```

In the above image, there is a man with a camera, comprising of two objects. Also, there is a building in the background. The background comprises of both the sky and the field, depicted in two different shades of gray. A good image segmentation algorithm should be able to understand that these changes are minor compared to the object of interest, namely the man with the camera.

In order to understand how image segmentation could be possible based on pixel intensity values, let us look at the histogram of the pixel intensities present in the above image.

[4]:
```
pixel_frequencies = np.zeros(256)   # an array to store the frequencies of the␣
↪pixel values 0-255

# loop over all pixels to count the frequencies
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        pixel_frequencies[image[i, j, 0]] += 1

plt.bar(np.arange(256), pixel_frequencies)
plt.title('Histogram of grayscale values of the Image')
plt.show()
```



Clearly, we see that a bimodal distribution of the pixel intensities. Clearly, since the man and the camera are of darker shade, the peak closer to 0 corresponds to the pixels associated with the object, while the peak around 160 is due to lighter shade pixels from the background.

3 A Simple Algorithm for Image Segmentation

In order to determine the optimal threshold for the image segmentation, we convert the problem mathematically. Let, f_i be the number of pixels with intensity i, where $i = 0, 1, 2, \ldots 255$. Let us assume T is the chosen threshold. Then, we consider the two sets $\{0, 1, 2, \ldots T\}$ and $\{(T+1), (T+2), \ldots 255\}$ as the pixel intensity levels corresponding to two groups, the foreground and the background.

The average intensity values of two groups are

$$\mu_f(T) = \frac{\sum_{i=0}^{T} i f_i}{\sum_{i=0}^{T} f_i} \qquad \mu_b(T) = \frac{\sum_{i=(T+1)}^{255} i f_i}{\sum_{i=(T+1)}^{255} f_i}$$

and the within group variances are

$$s_f^2(T) = \frac{\sum_{i=0}^{T} (i-\mu_f)^2 f_i}{\sum_{i=0}^{T} f_i} \qquad s_b^2(T) = \frac{\sum_{i=(T+1)}^{255} (i-\mu_b)^2 f_i}{\sum_{i=(T+1)}^{255} f_i}$$

In a way, we would like to choose the threshold T such that both of these groups pertaining to the foreground and the background have lesser within group variation of pixel intensities. Hence, the

optimal threshold would be

$$\hat{T} = \arg \min_{T \in \{0,1,2,\ldots 255\}} (\sum_{i=0}^{T} f_i) s_f^2(T) + (\sum_{i=(T+1)}^{255} f_i) s_b^2(T)$$

On the other hand, denoting the total variance $s^2 = \frac{\sum_{i=0}^{255}(i-\mu)^2 f_i}{\sum_{i=0}^{255} f_i}$ where $\mu = \frac{\sum_{i=0}^{255} i f_i}{\sum_{i=0}^{255} f_i}$. With a very simple calculation, it can be shown that

$$s^2 = (\sum_{i=0}^{T} f_i) s_f^2(T) + (\sum_{i=(T+1)}^{255} f_i) s_b^2(T) + (\sum_{i=0}^{T} f_i)(\sum_{i=(T+1)}^{255} f_i)(\mu_f(T) - \mu_b(T))^2$$

Hence, the optimal threshold can also be recovered as

$$\hat{T} = \arg \max_{T \in \{0,1,2,\ldots 255\}} (s^2 - (\sum_{i=0}^{T} f_i) s_f^2(T) + (\sum_{i=(T+1)}^{255} f_i) s_b^2(T)) = (\sum_{i=0}^{T} f_i)(\sum_{i=(T+1)}^{255} f_i)(\mu_f(T) - \mu_b(T))^2$$

The following function calculate this quantity

$$L(T) = (\sum_{i=0}^{T} f_i)(\sum_{i=(T+1)}^{255} f_i)(\mu_f(T) - \mu_b(T))^2$$

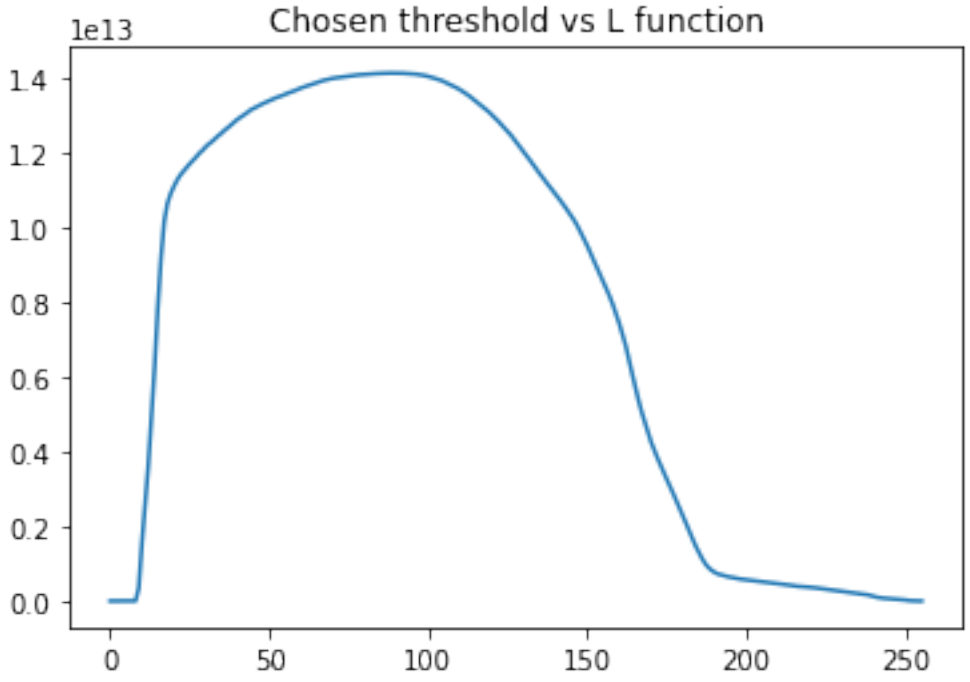
for each $T = 1, \ldots 255$ and outputs their maximum as the optimal threshold for segmentation.

[5]:
```
def optimal_segment(img):
    L = np.zeros(256)   # an array to store L(T)
    freq = np.zeros(256)   # an array to store f_i's

    # obtain f_i's
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            freq[img[i, j]] += 1     # increase corresponding intensity frequency

    for s in range(1, 255):
        w1 = freq[0:s].sum()  # calculate the sum (i=0)^T f_i
        w2 = freq[s:].sum()   # calculate the sum (i=T+1)^255 f_i

        # ensure that both w1 and w2 are nonzero, only then L(T) is nonzero
        if w1!=0 and w2!= 0:
            mu1 = sum( np.arange(s) * freq[0:s] ) / w1   # calculate the mean␣
↪mu_f
            mu2 = sum( np.arange(s, 256) * freq[s:] ) / w2   # calcuale the mean␣
↪mu_b
            L[s] = w1 * w2 * ((mu1 - mu2)**2)
```

```
    return {'Lfunction' : L, 'Threshold' : np.argmax(L)}
```

We call this `optimal_segment` function on the cameraman image and plot the $L(T)$ as a function of the threshold T to see whether there is smooth maximum.

[6]:
```
result = optimal_segment(image)
plt.plot(result['Lfunction'])
plt.title('Chosen threshold vs L function')
plt.show()
```



[7]:
```
result['Threshold']
```

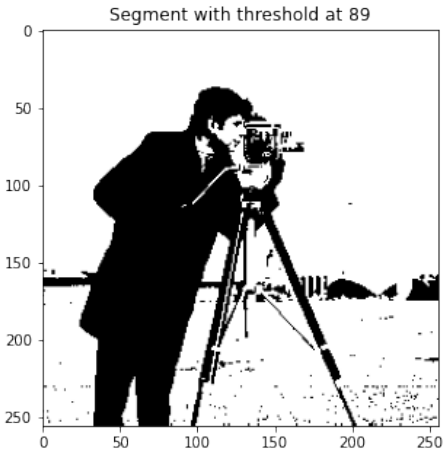
[7]: 89

We see that as expected the $L(T)$ function is smoothly increasing upto a optimal threshold and then decreases afterwards. We find that 89 is the optimal threshold for the cameraman data. The following function segments an image given the threshold value. Then we plot both the original and the segmented image side by side for better comparison.

[8]:
```
def segment_image(image, threshold):
    """
        A function to return the segmented black and white images
        - If a pixel is > threshold, we set it = 255
        - Otherwise, we set it = 0
```

```
    """
    segmented_image = np.zeros(image.shape, dtype = 'int')    # initialize all values of the 3d matrix equal to 0
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            for k in range(image.shape[2]):
                if image[i, j, k] > threshold:
                    segmented_image[i, j, k] = 255     # set the segmented image pixel 255 if original pixel exceeds threshold

    return np.round(segmented_image)
```

[9]:
```
fig, plots = plt.subplots(1,2, figsize = (15, 5))      # subplot with 1 row, 3 columns

# subplot 1
plots[0].imshow(image, cmap='gray')
plots[0].set_title('Original Image')

# subplot 2
plots[1].imshow(segment_image(image, result['Threshold']), cmap='gray')
plots[1].set_title('Segment with threshold at '+str(result['Threshold']) )

plt.show()
```

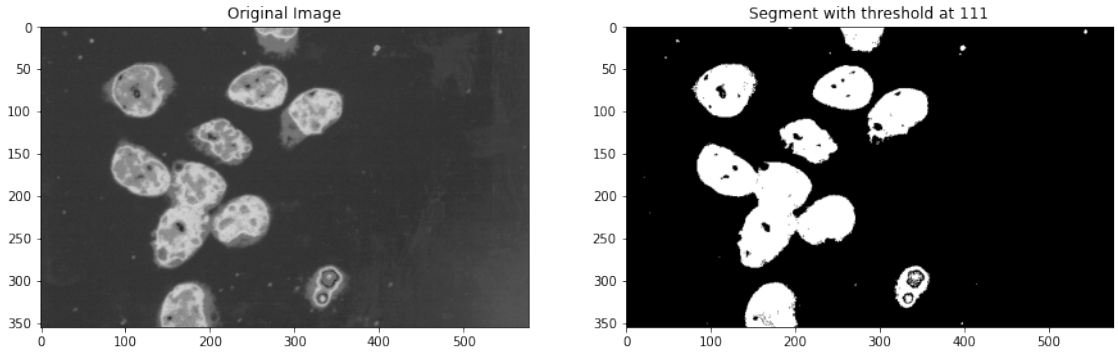


4 Examples

Let us consider some more examples of this segmentation to see how this algorithm of finding optimal threshold performs.

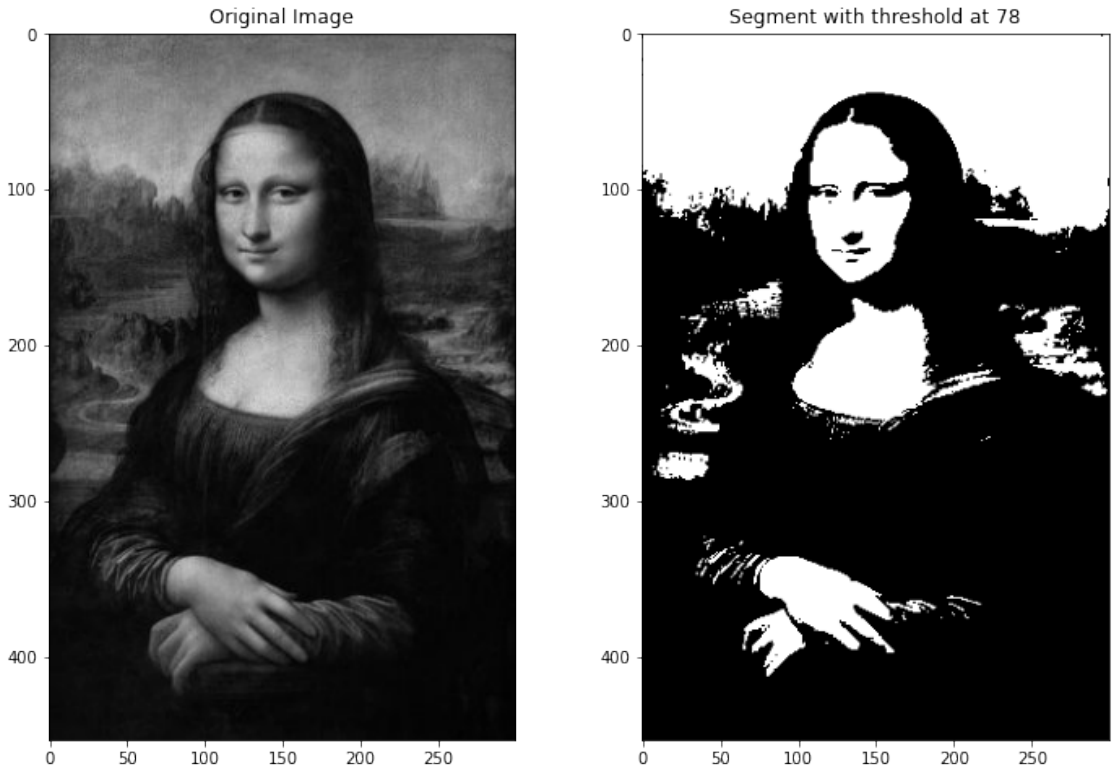
4.0.1 A Medical Image of Cells

[10]:
```
image = cv2.imread('cell.jpg')
result = optimal_segment(image)
fig, plots = plt.subplots(1,2, figsize = (15, 5))      # subplot with 1 row, 2
 ↪columns

# subplot 1
plots[0].imshow(image, cmap='gray')
plots[0].set_title('Original Image')

# subplot 2
plots[1].imshow(segment_image(image, result['Threshold']), cmap='gray')
plots[1].set_title('Segment with threshold at '+str(result['Threshold']) )

plt.show()
```



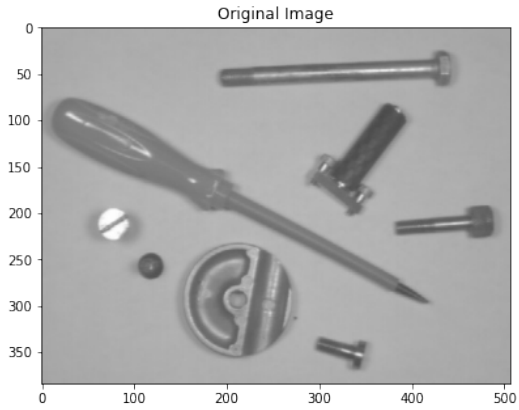
4.0.2 Monalisa Image

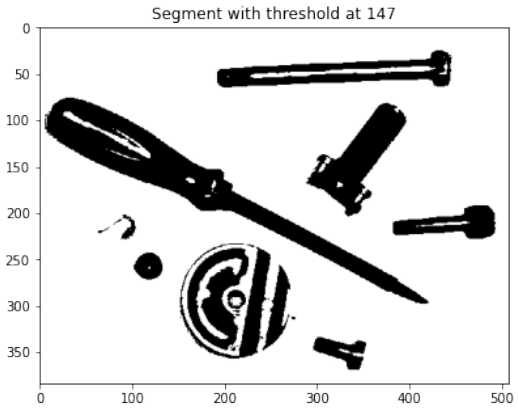
[11]:
```
image = cv2.imread('Mona_Lisa.jpg')
result = optimal_segment(image)
fig, plots = plt.subplots(1,2, figsize = (12, 8))      # subplot with 1 row, 2
 ↪columns

# subplot 1
plots[0].imshow(image, cmap='gray')
plots[0].set_title('Original Image')

# subplot 2
plots[1].imshow(segment_image(image, result['Threshold']), cmap='gray')
plots[1].set_title('Segment with threshold at '+str(result['Threshold']) )

plt.show()
```



4.0.3 An Image of some tools lying on a table

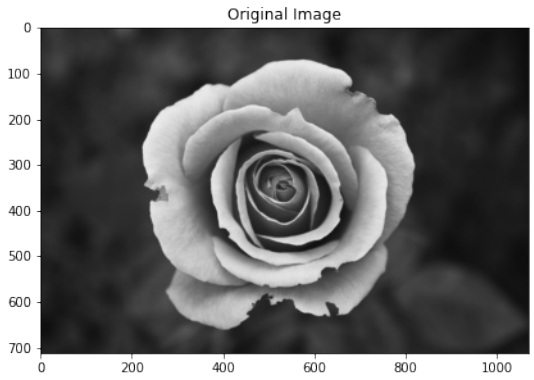
[12]:
```
image = cv2.imread('tools.jpg')
result = optimal_segment(image)
fig, plots = plt.subplots(1,2, figsize = (15, 5))      # subplot with 1 row, 2␣
 ↪columns

# subplot 1
plots[0].imshow(image, cmap='gray')
plots[0].set_title('Original Image')

# subplot 2
plots[1].imshow(segment_image(image, result['Threshold']), cmap='gray')
plots[1].set_title('Segment with threshold at '+str(result['Threshold']) )

plt.show()
```





4.0.4 An image of a Flower

[13]:
```
image = cv2.imread('flower2.jpg')
result = optimal_segment(image)
fig, plots = plt.subplots(1,2, figsize = (15, 5))      # subplot with 1 row, 2␣
 ↪columns

# subplot 1
plots[0].imshow(image, cmap='gray')
plots[0].set_title('Original Image')

# subplot 2
plots[1].imshow(segment_image(image, result['Threshold']), cmap='gray')
plots[1].set_title('Segment with threshold at '+str(result['Threshold']) )

plt.show()
```



THANK YOU