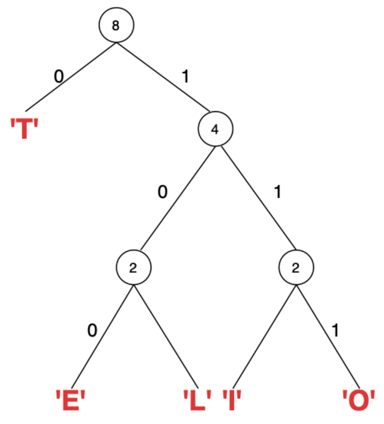
MINI PROJECT 3

Eliott Simon
Department of Data Science and Knowledge Engineering
Introduction to Image and Video Processing

May 2020

1 Huffman's Coding

Huffman's coding is a method which aims to compress the data. The method that I used to solve this problem is building a Binary Search Tree, which sort of acts like a trie. The first step is (as redundancy of character is important) to find the frequency of each character in the string. Since my name does not contain any redundancy, the frequency of each character equals to 1 and the algorithm won't be able to compress the data. However, if we consider that my last name is 'eliotttt', we can already see some re-occurences in the characters. Now we have our string $\{s = eliotttt\}$. The bit-length of s is equal to 64. First of all, a good way of reducing these characters is to transform each of these characters into a small integer. Say e = 0, l = 1, i= 2, o= 3, t= 4. Hence we can rewrite our result as 0 1 2 3 4 4 4 4, which gives a bit of length 21. Now comes the Huffman's coding part. I built a BST which has edges of value 0,1, corresponding to bit-values. The idea is to put all the characters inside the BST, such that the most occurring characters 't' has the smallest depth out of all the other nodes. This means that, when traversing the tree (to retrieve the word), the character which occurs the most will have the shortest bit-length. The fact that my last name (simon) did not contain any re-occurring character means that the results were going to be poor since the tree would have been balanced. Now for the string $s = eliotttt$, I implemented the resulting BST, which is unbalanced:



NB: a 1 disappeared between node (2) and leaf node ('L') and a 0 disappeared between node (2) and leaf node ('I').

Each internal node stores the sums of the weights of its descending nodes. If we now want to traverse the tree to find 'eliotttt', the result would be: 100 101 110 111 0 0 0 0 . This bit has length 16, which is smaller than the 21 previously computed, thus the data got compressed.

2 Morphological image processing

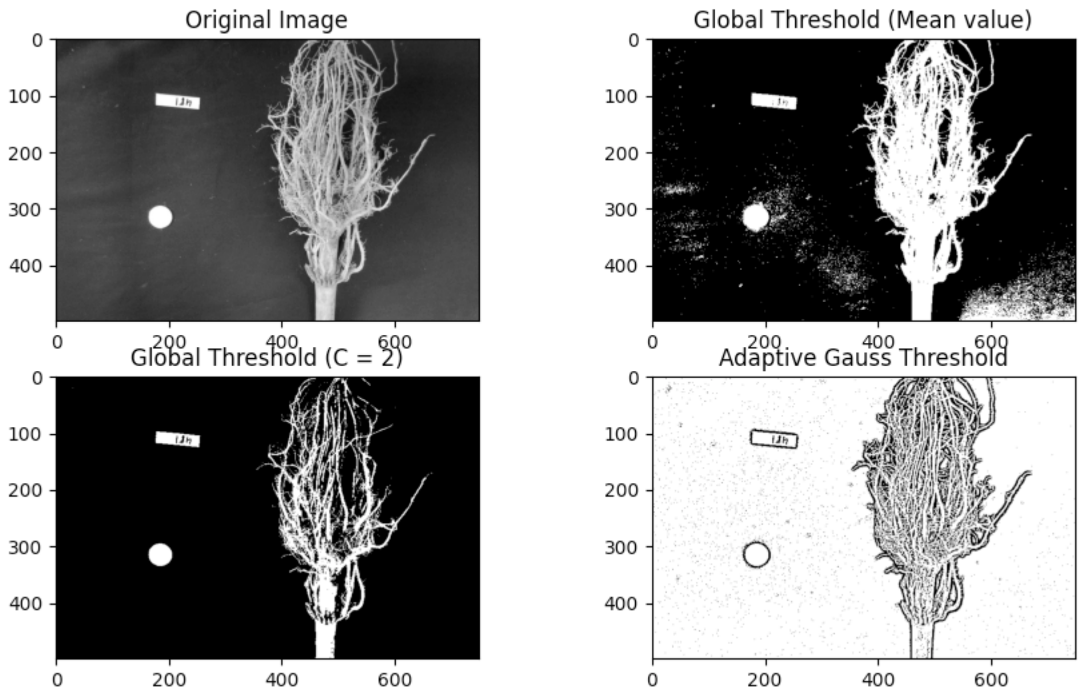
2.1 Image Binarization and Thresholding

The image that I chose for this exercise is the following:

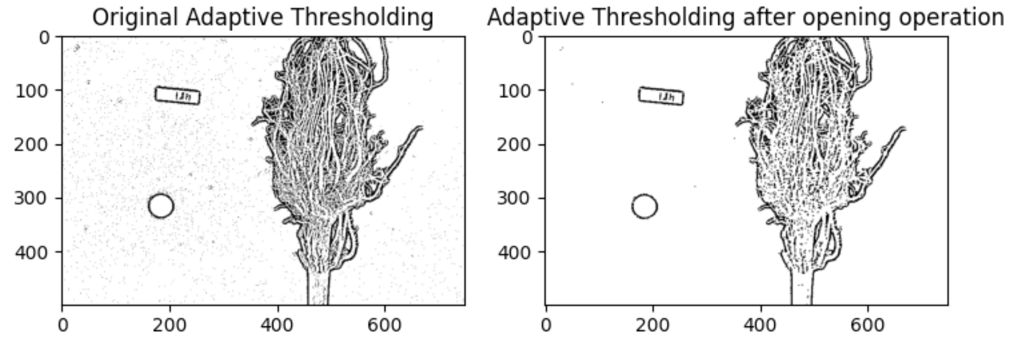
The image has an intensity mean $\mu = 60.6$, which means that it is rather dark. Thus, we can take this μ as a threshold for binarization:

$$f(x,y) = \begin{cases} 255 & f(x,y) \geq \mu \\ 0 & else \end{cases}$$

We could also take different strategies, such as taking some standard deviations away from this mean (using a constant factor C). It is very hard to predict which value is going to be the best for thresholding and it requires to make some tests and adjustments. I stuck to a value of $C = 2$, since it gave, in my opinion, the best results
Another approach is to use *adaptive thresholding*. This method, as its name suggests, does not only consider a 'global' threshold like the previous method. Instead, it is going to evaluate a threshold for different regions of the image, to avoid, for instance, shades to distort the result of the binarization.

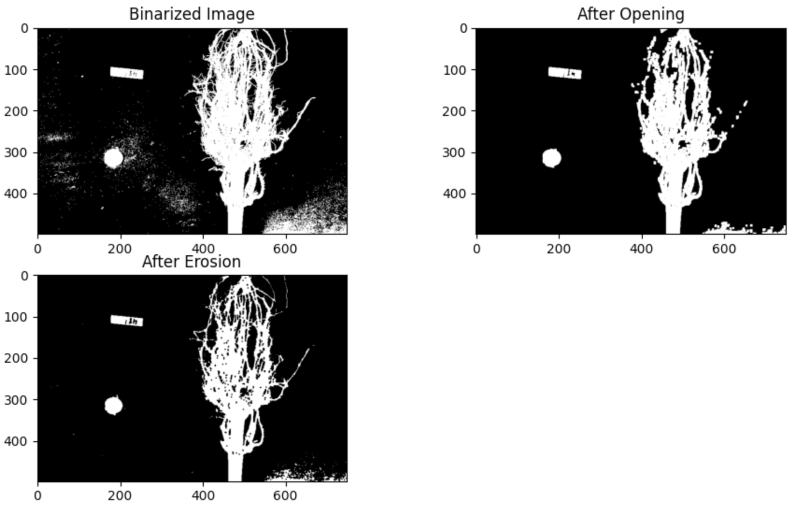


For the global thresholding method (using the mean value μ as threshold), the result is quite bad. It looks like the (white) shades influenced the result and this is why there is a lot of noise.
To improve this result, I had to pick a better threshold value. By calculating the standard deviation of the image, and add it certain amount of time (C) to the mean μ, we get better result, as seen on the bottom left (which was calculated with $C = 2$).
Besides, we see that using adaptive thresholding, the *contrasted* areas are well-defined. Indeed, we can clearly see the contour of each object. However there is still a bit of noise in the background area. To remove this noise, I used the *closing* operations, since it is useful to remove black dots on white background. As results:


Original Adaptive Thresholding
Adaptive Thresholding after opening operation

2.2 Morphological Transformations

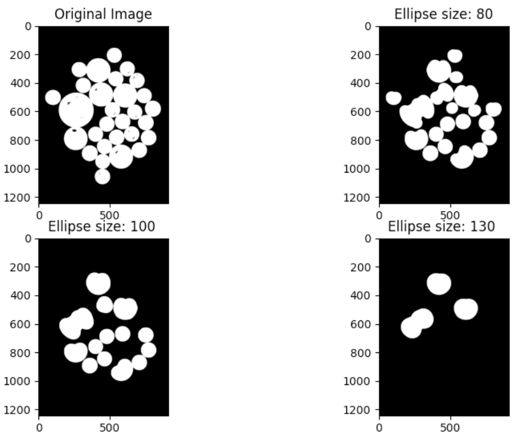
For this exercise, I decided to work on the binarized image with threshold = μ. I thought it would be more interesting, since it is noisy and the contour of some shapes are not clearly defined, which means that it can be drastically improved. The first and most important operation is to remove the white noise. To do so, we can use an erosion (which would remove the white dots), followed by a dilatation (since the image would be shrunk due to the erosion). This whole process is also known as opening.



As expected, the opening operation removed most of the noise that was present in the original image. However, it looks like the dilatation affected the image too much. Indeed, the branches (if these are branches) got too wide. This is why I made another erosion operation, such that the contour of the branches were more correct. Furthermore, we can see that the value contained inside the white rectangle (on the left) is more readable after the last operation(erosion).

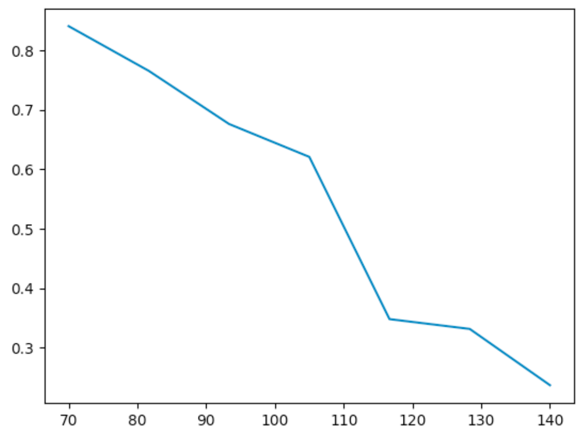
3 Bonus

For this exercise, I implemented the granulometry method. The first step is to binarize the image. I used the global thresholding method previously discussed to get the black and white version of the image. The image contains a lot of circles of different sizes. Hence, the structured elements that we have to implement must be circles. By computing several SE's (circles) whose size are *increasing*, and set them as kernels for making an opening operation on the image, we expect the circles to disappear of the image throughout the iterations. Here is a sample plot of iterations.



As seen, when the size of the SE gets bigger, the image loses his shapes. For the last plot, we can see that there are only elements remaining at the positions where there a big circles in the original image.

Finally , let's plot the size distribution.



This plot represents the difference (in pixel values) between the original image and the new image (whose elements are being removed). At $x = 70$, the value is close to 1 since no element has been removed yet. However, when the value of the ellipses gets bigger, the difference is getting close to 0. This means that the resulting image does not contain any information about the first image.