CSC3014 Computer Vision

Lab 4: Region Splitting-and-Merging and Watershed Segmentation

A. Region Splitting

This is another region-based approach for image segmentation. You may choose to run it independently instead of combining it with region merging. The steps involved are straight-forward. First, take the image and assume the entire image forms a single quadrant. Then, apply homogeneity test to the region. If it failed the test, then split the quadrant into four separate quadrants. Next, you repeat the same process of testing and splitting to every single quadrant, until no further splitting is possible or when the stop criteria are met.

Exercise

Similarly, there is no existing function from OpenCV that we can use to perform region splitting. Hence, your task here is to implement your own region splitting function. There are many ways to implement this, but I think it would be easier if you use the list from Python. Please take note that you are only required to write a function that will work on square images (e.g. 8x8, 16x16, etc.). I have listed the general steps below to assist you with the implementation. Feel free to make changes to the steps if you think it would be easier to do it in another way.

Steps:
1) Create two empty list (e.g. toProcessList, doneSplittingList).
2) Add the image to the toProcessList for initialization purpose. You do not have to add to entire image to the list, only add certain information enough for you to identify the region (for example, the information you need would be the starting row coordinate, starting column coordinate, and length).
3) Take an element from the list.
4) Perform homogeneity test (find the difference between the maximum and minimum intensity values of the region).
5) If the region is uniform, add the region to doneSplittingList. Otherwise, remove the current element (the one you are processing) from the toProcessList, split the region into four quadrants and add these to the toProcessList.
6) Repeat step (3) and (5) until no further splitting is possible (or when the toProcessList is empty).

B. Region Merging

Region splitting often splits the image into many small regions that can be merged together. However, you should take note that it is not always necessary to perform merging after splitting. Again, the steps to perform region merging is straight-forward. First, you select one of the regions that you have, and then try to find other homogenous regions from the surrounding. If an adjacent region passed the homogeneity test that you set, then merge it with the region you selected at the beginning. Repeat the testing and merging process until no further merging is possible or when the stop criteria are met.

Exercise

Again, there is no existing function from OpenCV that we can use to perform region merging. Hence, your task here is to implement your own region merging function. Assume that you used list to complete the previous exercise, the general steps to assist you with the implementation are shown below. Likewise, the function is only required to work on square images (e.g. 8x8, 16x16, etc.).

Steps:

1) Extract an element from the doneSplittingList.
2) Based on the extracted region from step (1), search from doneSplittingList, all the regions that fulfills the homogeneity test.
3) Check whether those regions found in step (2) are connected to the extracted region from step (1).
4) If connected, then merge the regions by giving them the same label. Otherwise, do nothing. Mark those regions that you have merged so that they will not be merged again with other regions in subsequent iterations.
5) Repeat step (1) to (4) until all the elements in the doneSplittingList has been processed.

C. Watershed Segmentation

The general idea of watershed segmentation is to imagine an image as a topological surface, where the intensity value of a pixel is used to represent the height from the surface. Assume the surface is not flat and there are several catchment basins, start filling water in every catchment basin, until the water is going to overspill from one to another. At this point, a dam is constructed to prevent the water from overspilling into another. At the end, we will only be able to see all the dams from the top, and they are known as the watershed lines. In general, the watershed lines are the lines that help us to segment the image into multiple regions.

Note

There are many ways to implement watershed segmentation algorithm. However, the key point of using watershed segmentation is not on the algorithm, but how to change the region-of-interest (the regions that we would like to extract) into the "catchment basins", and obtain a set of markers that can be used to control the segmentation process.

There is only one watershed segmentation function provided in OpenCV, and it is based on a marker-based watershed segmentation algorithm proposed in [1]. Before we can use this function, we need to define (of find) three types of markers, (i) markers for regions that very likely belong to the foreground objects (the region-of-interest), (ii) markers for regions that very likely belong to the background, (iii) markers for unknown regions (unsure if the regions should be part of foreground objects or background).

As shown below is a sample code [2] of applying distance transform and watershed segmentation to identify the circles in the given test image [3]. Generally, the process can be divided into three stages. The first stage is to find the three types of regions (so that we can set the internal markers) as mentioned above. In this case, the image is first converted to grayscale before global thresholding is applied to binarize the image in line 8. In this case, we used Otsu's method to determine the optimum threshold.

However, if the threshold adopted does not give you the regions that you are looking for, then you must choose another threshold or switch to another segmentation approach (e.g. edge detection). For example, if you are looking for circle objects in an image, then the output from thresholding need to contain those regions that cover the circle objects. The output from thresholding does not need to be perfect, but the extracted regions must at least cover the objects that you are searching.

```
1   import numpy as np                                                          #import OpenCV
2   import cv2                                                                  #import numpy package
3   from matplotlib import pyplot as pt                                         #import pyplot for plotting
4
5   imgColour = cv2.imread('differentColour.jpg')                               #Read the test image
6   imgGray = cv2.cvtColor(imgColour,cv2.COLOR_BGR2GRAY)                        #Convert to grayscale
7
8   _, thrImg = cv2.threshold(imgGray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)  #Apply Otsu’s method
9
10  sE = np.ones((3,3),np.uint8)                                                #Create a square SE
11
12  sureBg = cv2.dilate(thrImg,sE,iterations=3)                                 #Find background region
13
14  distTransform = cv2.distanceTransform(thrImg,cv2.DIST_L2,5)                 #Use distance transform
15  _, sureFg = cv2.threshold(distTransform,0.7*distTransform.max(),255,0)      #Find foreground region
16
17  sureFg = np.uint8(sureFg)                                                   #Convert to unit8
18  unknownRegion = cv2.subtract(sureBg,sureFg)                                 #Find unknown region
19
20  _, markers = cv2.connectedComponents(sureFg)                                #Label all the regions
21  markers = markers+1                                                         #Background label as 1
22  markers[unknownRegion==255] = 0                                             #Unknown label as 0
23
24  watershedMarkers = cv2.watershed(imgColour,markers.copy())                  #Apply watershed
25  imgColour[watershedMarkers == -1] = [255,255,0]                             #Overlay markers on image
26
27  pt.figure()                                                                 #Create figure window
28  pt.imshow(imgColour)                                                        #Show the image
```

Once we have the binary version of the image (it’s using 0 and 255), dilation by a square SE is applied to grow the regions we have identified in line 12. The purpose of doing this is to identify regions that are outside of our extracted regions. Because the output from thresholding gives us the circle objects we are searching for, we will be able to get the surrounding regions of the circle objects by using dilation (enlarge the circle regions). This gives us the second type of regions we need for the marker-based segmentation algorithm since the regions outside of the circle objects are very likely belong to the background.

After this, distance transform is applied in line 14 by using the distanceTransfrom() function from OpenCV. This function returns the distance of every pixel to the nearest zero value pixel (not non-zero value pixel). Next, the output is binarized using thresholding in line 15. The purpose of applying distance transform and thresholding is to obtain the center of the extracted regions. Although regions extracted from thresholding might not be perfect, the center of the extracted regions are very likely belong to the foreground objects. Finally, we can determine the third type of regions, the unknown regions by taking a subtraction between the two in line 18.

When comes to the second stage, the aim is to label those foreground, background, and unknown regions, so that they can be used as markers. This is achieved by using the connectedComponents() function from OpenCV in line 20. The function will automatically label all the connected components (pixels that are connected form a connected component or region). A few adjustments are applied in line 21-22, so that all the unknown regions are labelled as ‘0’, all background regions are label as ‘1’, and foreground regions are label as ‘2’, ‘3’, ‘4’, …, ‘n’. After we have finished preparing the markers, we can move to the third stage, that is to apply watershed segmentation to the image based on the set of markers.

Exercise

Plot sureFg, sureBg, markers, watershedMarkers, and imgColour in the form of image to understand the entire process.

You may also try to modify the code so that you can apply it to another image, “waterCoins.jpg” [2]. The task is to segment those coins in the image. Overall, the steps remain the same. But in this case, you have to modify the thresholding so that it can help you to segment out the coins. You need to do this before finding the regions that are very likely belong to the foreground objects. Please remember that we need to turn the region-of-interest into the “catchment basins” before we can apply watershed segmentation. This is analogous to the process of searching for the regions that are very likely belong to the foreground objects.

References

[1] F. Meyer, “Color Image Segmentation”. In Proc. International Conference on Image Processing and its Applications, 1992, pp. 303–306.

[2] OpenCV, “Image Segmentation with Watershed Algorithm”, OpenCV-Python Tutorials, Website Link (accessed 1st May 2020).

[3] GameN, “Lyto Different Colour”, Facebook (accessed 1st May 2020).