**REPORT FOR BOUNDARY SMOOTHING PROBLEM OF SEGMENTED IMAGES (SLIC SEGMENTATION USED)**

Shavarsh Melikyan

6 February 2023

The following report represents information related to Boundary Smoothing of already segmented picture(s), as a continuation of pebble mosaic paper’s work.

Used libraries:

* Skimage (segmentation, util, io, filters, color); functions: slic, mark\_boundaries, img\_as\_float, butterworth, sobel, watershed, rgb2gray, find\_boundaries
* Argparse\*
* Matplotlib (pyplot)
* Numpy
* Pillow (Image Processing); functions: Image, ImageFilter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Main Problem*

The aim of this report is to show/test/inform about outcomes of ways for boundary smoothing applied to segmented picture

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Imported Libraries*

Some libraries’ usage is already described in a previous report, so only new ones are covered here:

1. Find\_boundaries – finds boundaries between segments of pictures/superpixels and return a boolean value representing it
2. Butterworth – digital image filtering algorithm, optimal variant in comparison with ideal and gaussian low pass filters
3. Sobel – another filtering algorithm which detects grayscale pictures’ edges
4. Watershed – region-based segmentation method in mathematical morphology, based on elevation

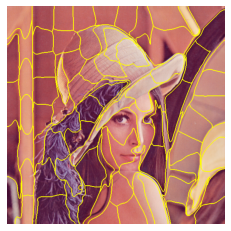
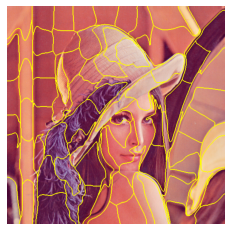
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Process*

First of all there are represented 3 codes, the main one of which is the butterworth.py; so I will describe what is done there and then will discuss other codes in the below section.

The beginning part, which is, using hard-coding techniques to attain the image for usage, is the same as in previous code, same with SLIC’s implementation and codes showing through.

Noticeable differences are in addition to lines with butterworth function and (commented) line of transforming picture to a grayscale function (rgb2gray). Let’s start with the butterworth function; it takes the image as an input, and filters it in a low-pass manner to get returned a filtered image. Note that the filter is used before SLIC function is called; in case of calling the function after SLIC there are being no changes, because due to formatting it is impossible to pass segments as “images” for filtering, so the image should be filtered initially and segmented after with more smooth edges because pixels are being already bloored. Note that there are some parameters inside the function, such as cutoff\_frequency ratio, high\_pass, order, etc. cutoff\_frequency\_ratio determines the position of cutoff relative to the shape of fast fourier transform, high\_pass determines whether the function should apply a high or low pass filter and order defines steepness of slope in frequency space. Note that as low value the cutoff\_frequency\_ratio gets, as more observable changes and more smoothness is being made, however, due to its nature, it is changing the colour of the original image by trying to grayscale it, so the tradeoff is the filtering quality and the colour, and, optimally, 0.5 is a normal value for it (if bigger: worse filtering, lower: worse colour). Order has same type affections on the image, so value of 2 is considered optimal.



*First picture is the output before implementing the butterworth filter and the second one afterwards using it.*

Now, in case if to activate the rgb2gray function which will turn the picture into a grayscale version of it it’ll be noticeable that all segments are squares; the reason of it is because the compactness factor is high in SLIC function; from 10 it should be lowered to as low as to values like 0.1. However, in this case the cutoff\_frequency\_ratio of filter should be lowered as much as possible (in the example value used for it was 0.1) because filtering something more “monotone” is harder for the filter, so quality, by default, drops whenever trying to filter grayscale images.



*First image is one used with high compactness, second one is with low compactness before applying the filter and the third one is the output after applying the filter*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Observations and other possible codes*

Although the above mentioned code works for the programme, there may be better solutions for the highlighted problem; for example, why filter the picture before applying the segmentation algorithm if only edges after segmentation can be modified so the quality will not drop?

For that reason, there are few steps before which should be done. First of all, for filtering the image (and for segmentation algorithms which include filtering) libraries usually require turning images into grayscale format, because otherwise rgb noise is going to be filtered and output may be not the most pleasant one (image can turn into fully red or fully green colour with its shades).

For example, the used in histogram\_leg.py file sobel() function is measuring the varying pixel intensity in an image and hence, it is easiest to do if the image is grayscale. So, in that code, after getting an image as an input, histogram of grey values is being computed and shown, after which the sobel filter shows the elevation map which corresponds to the edges of the picture (without segmenting initially); then, markers are used after which a segmentation algorithm called watershed, which uses such called watershed lines for segmenting pieces (this algorithm works ultimately with grayscale images).

Another example is in pillow\_edges.py code, which, again, takes an image, turns it into grayscale, and then detects edges by image.filter() function (unknown which algorithm used).

So, finding edges is not a big deal, but is it possible to filter ONLY edges? I am not sure in this part, but I found a paper where implicitly my question was answered: No; i*n a paper of A performance Comparison of Edge Detection Techniques for Printed and Handwritten Document Images* (p. 2, V. Mohan) there are given steps (1.1) for detecting an edge; and the first step is an Image Smoothing, then comes Enhancement step which is followed by Detection and finished by Localisation step, from which can be concluded that the normal practice is at first smooth an image and then try to detect edges. Although the paper is not explicitly linking something with this project, however, steps mentioned for edge detection are important I guess.

Also, I personally think there might be some consensus path for solving a problem (not tested yet). A duplicate of an image may be made and instead of whole image or edges there can be chosen some area which is as small so its blooring can be neglected and it is as big so it can be considered as something which is not an edge at all (e.g. edge and neighbouring few lines of pixels). Stackoverlow article is attached in the part of links related to this.

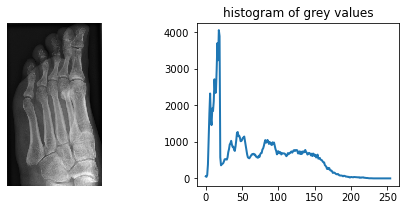
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Conclusion*

Boundary Smoothing works if to smooth an image before segmentation; possibly (but not for sure)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Examples of Results (observations and other possible codes)*





Besides the last picture, all are mentioned parts of histogram\_leg\_py code; the last one is edged lenna.png from pillow\_edges.py

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Further Steps*

1. Understand the complexity of the code and compare it with others and other parameters too for knowing the trade-off
2. Try to find another way of smoothing boundaries without affecting segments’ quality
3. Continuing working through the paper of Pebble Mosaics by then examining the pebble geometry and 3D look of images