SEM slices segmentation – Guidelines

Scope and summary

The scope of this code is the analysis of SEM electrode slices to segment the different phases (*i.e.*, assigning every pixel to a given phase – here cathode active material, carbon-binder domain/solid electrolyte, or pores), and, for each phase, distinguish the single particles/agglomerates/pores (instance segmentation), and calculate the associated properties (interfaces, volume fractions, particles size distribution and aspect ratio, *etc.*).

The code has been originally developed for the case of classical Li—Li-ion battery electrodes, but it can be directly translated to the solid-state scenario (by knowing that what is called “carbon-binder domain – CBD” by the code, in this scenario would rather be the solid electrolyte -SE-).

This code has functionalities that at the moment cannot be included in the executable, therefore it should be run directly the code – but don’t worry, it is really easy and you want a section to explain you how to do it below.

To run the code, you should fill in the associated input Excel (“Inputs_SEM_segmentation.xlsx”), which offers also a description of each of the parameters you can control. In the “Example” folder, you can also find a set of parameters that you can take as a first reference. Afterward, you can simply run the Python code (the Excel and the code should be placed in the same folder). This will lead to a first message (that will close itself automatically after 5 seconds) informing you that the code started, and a second message (which will not close automatically) when the analysis is finished.

An example of an image suited for this analysis is reported in Figure 1.

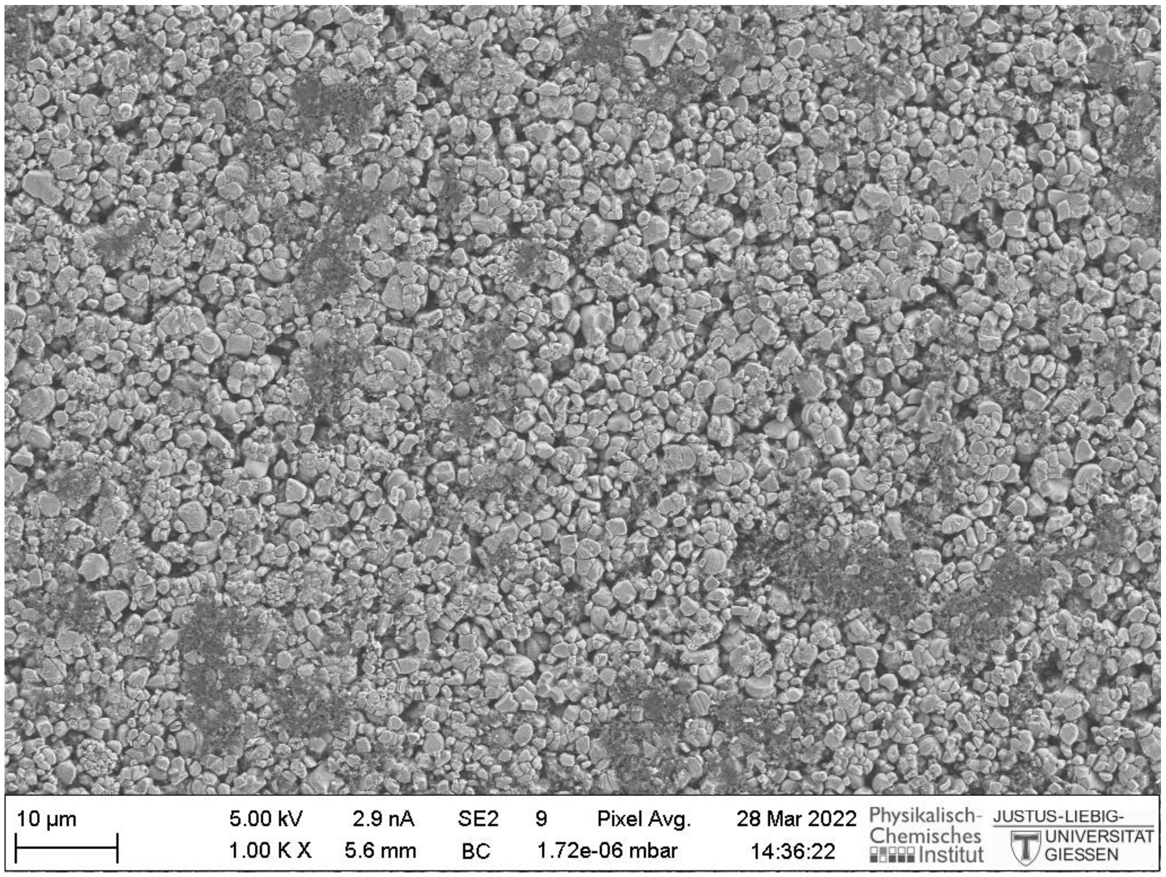


Figure 1. Example of SEM image suited for the analysis performed through the code discussed here. The scalebar is automatically detected (parameters in the Excel) to calculate the pixel resolution.

Phases Segmentation

The segmentation is performed in two steps:

1) An Otsu thresholding to distinguish the cathode active material (CAM) from the rest (carbon-binder domain + pores);
2) A Gabor filter + manual thresholding to discriminate the pores from the carbon-binder domain phase (CBD), *i.e.*, the agglomerates of conductive additive and binder. The CBD phase location is then determined by exclusion.

The procedure above is exemplified in Figure 2 (using as input the image in Figure 1).

First, the original image is divided in two (CAM, and CBD+pores) through an Otsu thresholding (an approach based on identifying the threshold value, in terms of pixel brightness, between different phases in the image – as shown in Figure 3). Afterward, a Gabor Filter (Figure 4) is applied on the reversed pore+CBD image (now the CBD and pores are brighter, while the CAM is fully dark – *i.e.*, with an imposed pixel intensity of 0). The Gabor filter is dependent on a number of parameters (Figure 4) and can be indeed considered an “infinite” filter (meaning that with an infinite combination of parameters, you will get an infinite number of different filters), and can make arise certain image features and hide others. This filter is very powerful, but there is no way to know in advance which set of parameters can work for your specific scenario. In my case I got lucky after a few hundred tests I found a set of parameters that allow spotting the pore position while hiding all the CBD (Figure 1, “4) Pores – Gabor”). Once having identified also the pores, it is possible to define the CBD by exclusion with respect to the already identified location of CAM and pores.

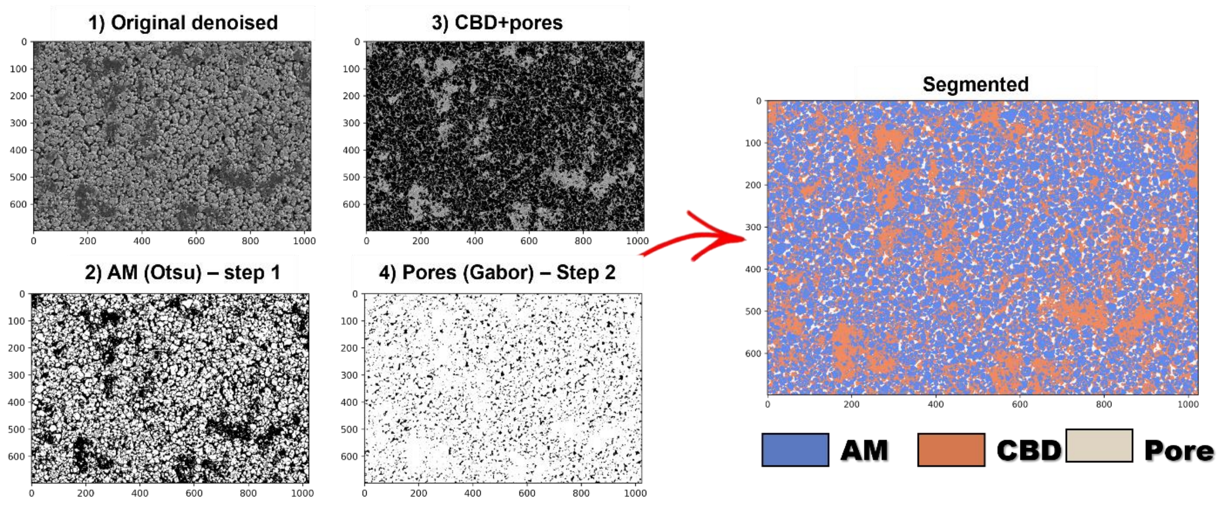


Figure 1. Example of segmentation for the case of LiB electrode slice. The same procedure can be applied for solid-state electrodes, with the difference that “CBD’ here should be read as the solid electrolyte.

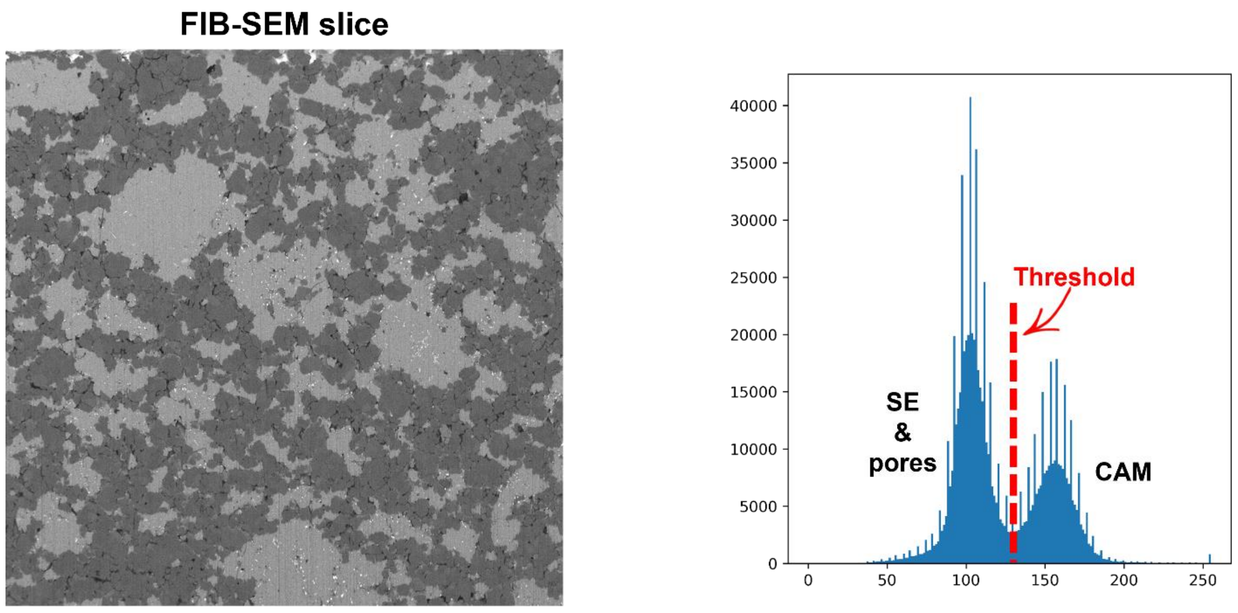
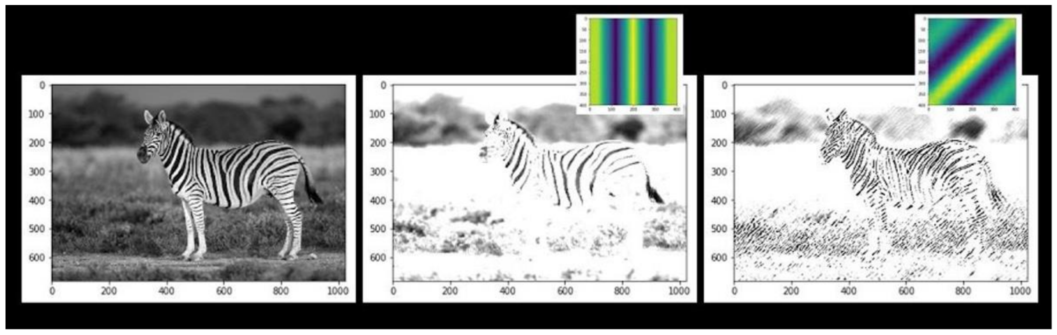


Figure 3. Example of Otsu thresholding for the case of a FiB-SEM slice of a solid-state electrode (but the same principle is applied for the case of SEM image of LiB electrodes). If you want to discover more about this approach, you can check the following YT video: https://youtu.be/YdhhiXDQDl4



$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

Figure 4. Example of Gabor filter, with the equation behind the definition this class of filter on the bottom right of the image. If you want to discover more about this type of digital filter, you can check the following YT video: https://youtu.be/QEz4bG9P3Qs

In case this segmentation approach does not lead to satisfactory results, you can also play with the “k_size_CBD” parameter (only in the code, not in the Excel). If also this does not work, I leave you also a convolutional neural network + Extreme boosting (decision tree-based) code that you can use to train a model for doing the segmentation you need ("CNN + ML code for alternative segmentation approach" folder). For understanding how this code works and how to implement/use it, you can refer to this YT video: https://youtu.be/vgdFovAZUzM

Instance (particles/agglomerates/pores) Segmentation

The code is also able to discriminate between different particles (or agglomerates) and pores, allowing to access the property (surface area, aspect ratio, size, *etc.*) of each of them. This process is based on a watershed-based algorithm, whose working principle is depicted in Figure 5.

To understand how this works, it is needed to briefly mention what is it a digital image at first. A grayscale digital image is nothing else than a 2D grid, in which every pixel is associated with a number representing an intensity (I). Typically, the possible value range between (extremes included) 0 and 255 (8-bit image, *i.e.*, 2^8 possible values, but you can have also 16- or 32-bit images with much greater shades of gray), where 0 is black, while 255 is white. Therefore, a 2D image is actually a 3D object (x, y, I). In the case of a colored image, the concept is the same but, instead of having only one channel, you have three (three grids, one for red, one for green, and one for blue) – meaning that an RGB image is a 5-dimensional object (x, y, I_r, I_g, I_b).

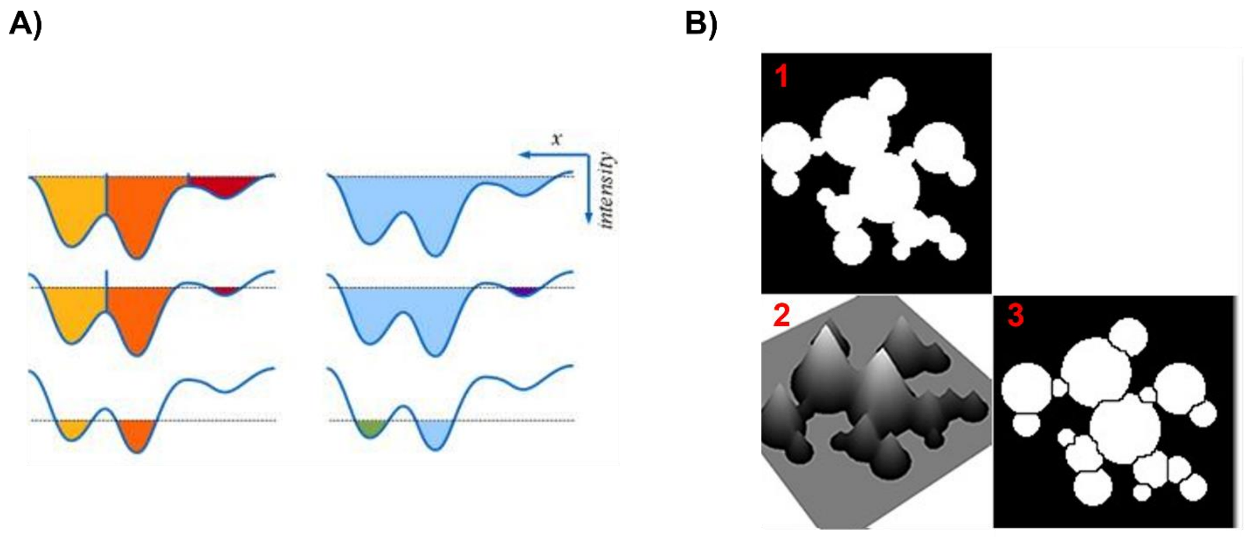


Figure 5. Example of watershed-based instance segmentation. For more information on how this approach works, you can check the following YT video: https://youtu.be/M1mJsJ5M4iE

Coming back to our grayscale image (for instance Figure 5 B1), this can be represented in 3D, where the third axis represents the intensity (Figure 5 B2) –the brighter regions being identified with mountains and the darker regions with valleys. Now, let’s turn it over (figure 5 A) - the mountains become valleys and vice versa.

At this point, we can imagine filling these valleys with water. At the beginning, two different valleys will be filled separately (bottom of Figure 5 A), until the water level raise high enough to overcome the mountain between them, leading to the (previously) separated water reservoirs entering into contact (Figure 5 A center and top). This is exactly how the watershed algorithm works, which defined the borders between different particles as the first point at which the (previously) separated water reservoirs start to touch each other. Performing this operation all over the original image allows for separating the different objects/particles (Figure 5 B3).

Of course, there are some other (more technical) steps in between (that lead to the need of some parameters, which are reported in the Excel and that you can then control – I also left you an example file with some reference values). You can just play with them to see the effect (a bit of a trial-and-error approach). If you want to better understand, however, I advise you to watch this short YT video: https://youtu.be/M1mJsJ5M4iE

An example of identified CAM particles (circled in yellow) and the associated determined properties are reported in Figures 6 and 7, respectively.

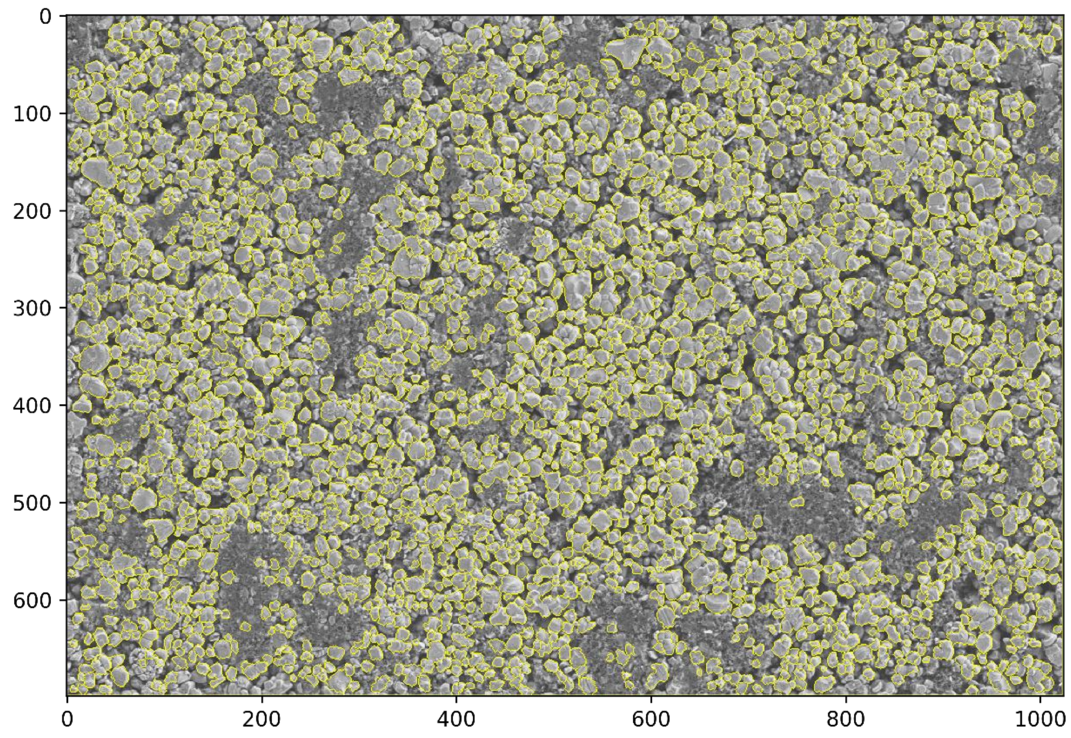


Figure 6. Example of CAM particle identification (circled in yellow) through the watershed-based approach.

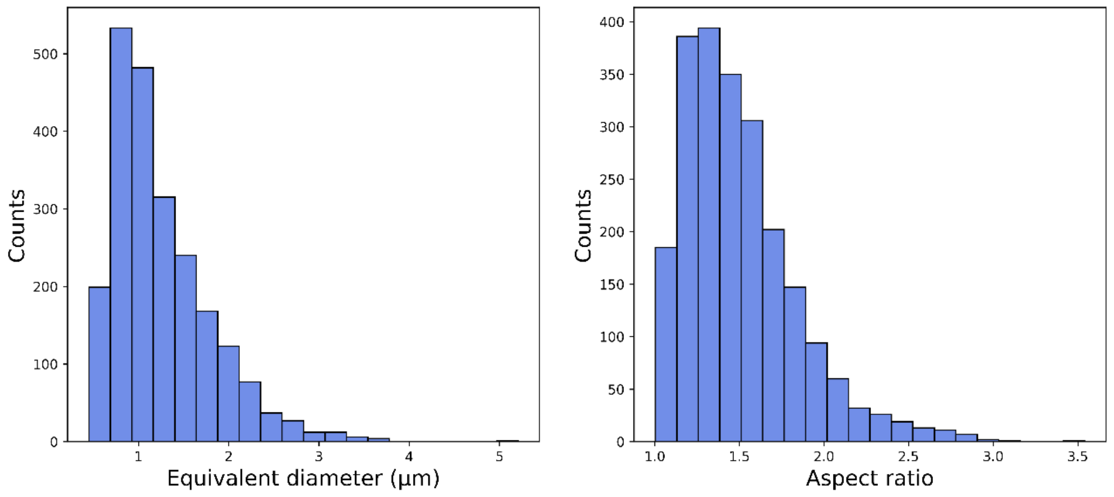


Figure 7. Particle size distribution (left) and aspect ratio distribution (right) of the identified CAM particles shown in Figure 6. The aspect ratio is simply defined as the ratio between the major and minor axis of each particle.

Results

In addition to the segmented image (Figure 1) and the particles/agglomerated/pore identification for all the phases involved, *i.e.*, CAM, CBD/SE, and pore (Figure 6), the code outputs also the main information on the phases in terms of volumes fraction and interfaces (surface percentages in contact with the other phases) in two dedicated csv files ("Interfaces.csv" and "Phase volume fraction.csv"). The code also outputs the calculated properties for each particle/agglomerate/pore identified for all the phases in your image, both as an image (example in Figure 7) and a csv file.

In addition to the particle size and aspect ratio distributions, other two results automatically plotted are the solidity and Wadell circularity (both reported as a function of the particle size - reported in the x-axis as well as in the data point size and color – both directly proportional to the particles' effective radius).

Basically, the solidity tries to quantify how "regular" your particles are (how well their size fits within a regular polygon of any number of sides. The value reported refers to the calculation corresponding to the regular polygon whose number of sizes fits the best that given particle).

The Wadell circularity is a quantification of the sphericity (and then regularity) of a particle.

Lastly, you will also find a "Checks" folder, in which a series of images associated with the different steps of the image analysis (segmentation, particles' identification) are reported. This can be useful for you during the code parametrization (in particular for the watershed step).

Running the code

This code has functionalities that at the moment cannot be included in the executable, therefore it should be run directly the code – but don't worry, it is really easy.

I advise you to use the Jupyter Notebook version of the code, as this will allow you to re-launch (if needed) only some part of the code and not all of it (saving time) and because the Jupyter version of the code offers some interactive plotting to check the goodness of each processing step (cropping and alignment, light enhancement, segmentation).

The easiest path for using Jupyter Notebook is probably installing Anaconda (containing Jupyter Notebook, among other things):

https://www.anaconda.com/products/distribution

After that, you can launch the Anaconda Navigator (just search it after the installation) and from there you can run Jupyter Notebook.

The libraries to be installed are the ones reported in the very top of the code ("import Xxx" or "from Xxx"). It is very easy to install them (you can also google, library by library, how to install them, but typically it works for all in the same way):

https://youtu.be/Yr_ihLKq_yY

If you want to understand a bit better and learn more about Python (that can really be handy, so I definitely advise it!), you can look at the following videos series:

https://youtu.be/7uE6hypji0o

Contact for problems/doubts

If you have any problem using the code, feel free to contact me on my personal email: teo.lombardo3@gmail.com