Pore Analyzer overview

Miguel Pardo & José Contreras – Jan 2023

# Dependencies:

* Python 3.9
* NumPy 1.23
* Pandas 1.5
* OpenCV 4.6
* Matplotlib 3.6

# Python structure:

* pore\_analyzer\_<date>.py
  + Wrapper program that handles folder paths and initializes, for each, the analyzer function. Also re-runs folders queued up for second tries and saves up the final information
* analyzer\_funcs.py
  + Main code, holding all the important function definitions, and some other auxiliary functions
* analyzer\_figures.py
  + Code that graphs all the obtained information, optional

# Detailed step-by-step code overview:

Low/Med/High: Importance of the step withing the general scope

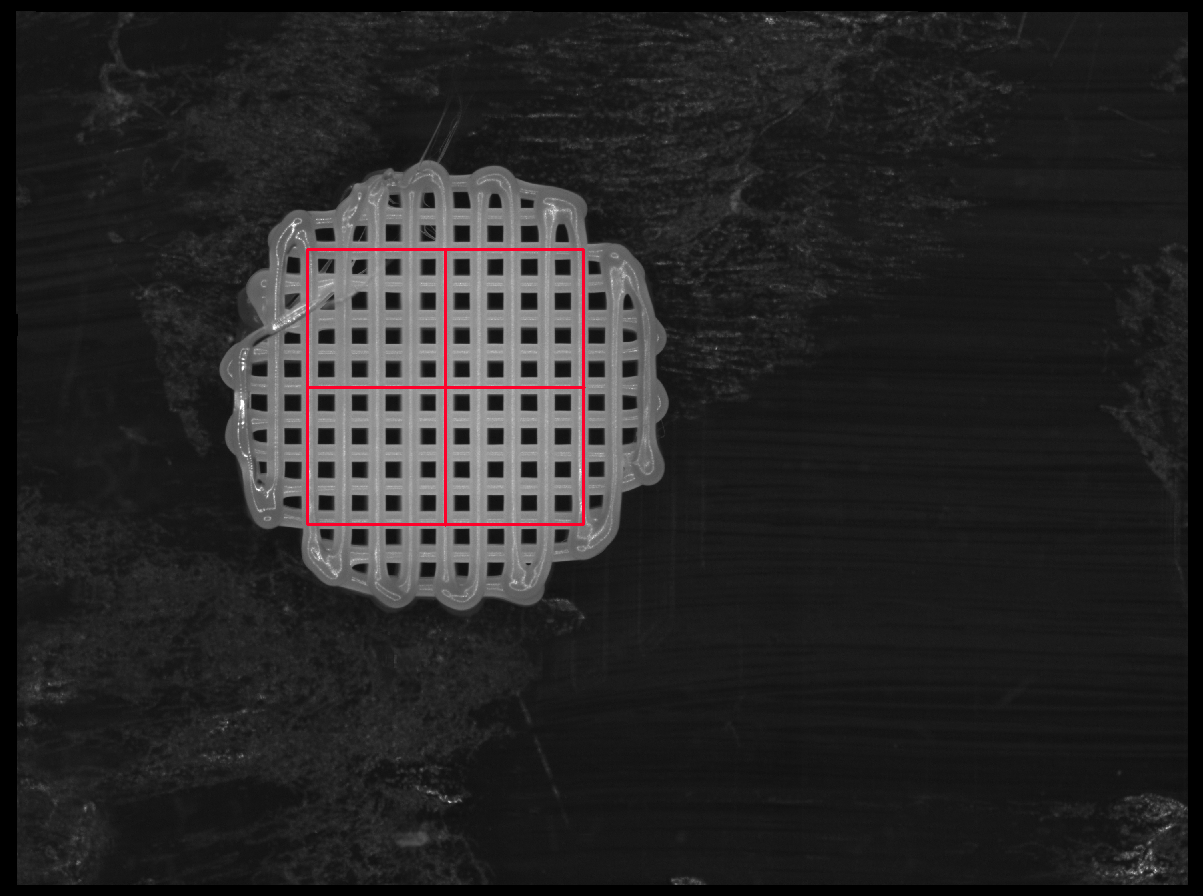
Steps with \* are specific to our usage, steps with + have hardcoded values

## pore\_analyzer.py

1. (Low) Importing of relevant modules and dependencies
2. (Low) Change working directory to the file’s directory
3. (Med) Look for “paths.txt” in folder

Each line in this file should point to a directory holding the Bioplotter’s image-containing folders

1. (Med) Briefly run first images from first folder to obtain a starting ROI and initial pore count



Process better explained in analyzer\_funcs. Only square (NxN) pore counts are allowed

1. (Med) Parse the folders from “paths.txt”, check for L/R and add each subfolder to a list
2. (High) Process each of the found folders with the *analyzer* function and temporarily save the results
3. (High) Re-run the folders automatically marked as “crop retry” by the program

Folders failing for other reasons (empty, corrupted, etc.) are not analyzed again

1. (High) Save each successful folder’s output as a separate .csv
   1. (Low) In case of error, a python buffer dump can be saved as well for salvaging

## analyzer\_funcs.py

#### Local initialization

1. (Low) Importing of relevant modules and dependencies
2. (Low) Defining used functions
3. (Low) Normalizing folder path name

I don’t know if this will work on Mac/Linux

1. (Med) Fetching the images from the given path name folder
   1. (Low) Sort them alphabetically
   2. (Low) Check for empty folders and notify accordingly if so
2. +\*(Med) Check folder name for tagging

Recommended folder naming system:  
<Main folder>\<print name>\_<YYYY-MM-DD>\_<HH.MM> AABBBCCC\<L or R>  
Where AA is the %BG (00, 10, 20), BB is the material and CCC is the temperature

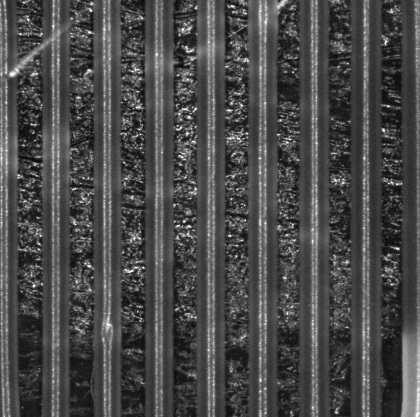
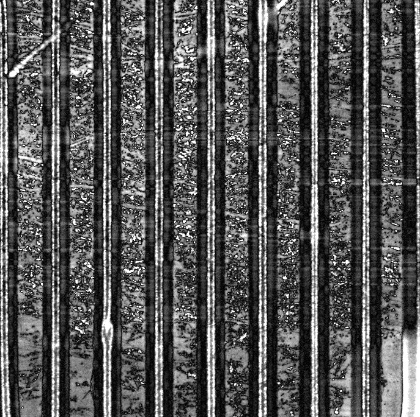
* 1. “ACE”, “DCM”, “MINI” or “PCL” materials are allowed, and those names must be somewhere in the full folder path, case insensitive
  2. “90”, “110” and “130” temperatures are allowed, and those numbers must be somewhere in the full folder path

Both Temperature and material names are searched anywhere in the full folder path, so false identifications can occur (e.g.: main folder has “90” in the name it even if that’s not the temperature)

* 1. Repetition number is automatically detected following the recommended naming system
  2. “ 10” (note the space in order not be identified with “110”) and “20” strings are located for the Bioglass content. If none found, it is assumed to be 0%BG

Any issues with folder tagging means that the folder will not be analyzed

1. (Low) Check the number of layers using the last image’s name as reference
2. (Med) Load up all the images on the memory
   1. Missing or otherwise corrupt images are skipped, and a blank placeholder will be loaded instead
3. +\*(Low) It is preliminarily assumed that each pore is 85px apart (0.8mm in our case)
4. (Low) Image stack is cropped so it includes one additional pore than ROI indicates
5. (HIGH) Image luminance correction stack is calculated as each image but box blurred 85px in all directions
6. +(Med) Image stack’s first image is automatically detected for printing direction
   1. The first image is blurred 301 px either vertically or horizontally, and the total sum of absolute difference between the original image with blurred result is evaluated
   2. Whichever value is lower is set to be the printing direction, and it is assumed that printing alternates each layer

    
From left to right: Original first layer, horizontally blurred, vertically blurred.

Reasoning: the first layer only has strands in one direction, so blurring it in that direction results in a comparatively more similar image than blurring it in the perpendicular direction. That difference is quantitatively evaluated here as a darker overall image.

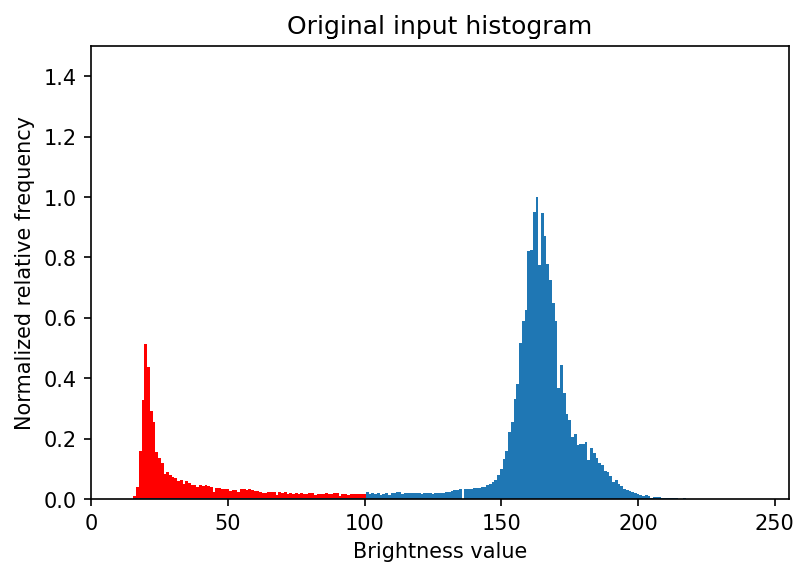
1. (HIGH) Image stack is converted to NumPy array and normalized by dividing each layer by its luminance correction layer
2. (Low) Image stack is cropped to match the ROI

#### Calculating threshold

Threshold values may be inherited from a previous run if it was rescheduled for a retry, skipping this part.

All steps are for each individual image in the stack, unless otherwise stated

1. (Med) Every image is thresholded by Otsu’s method

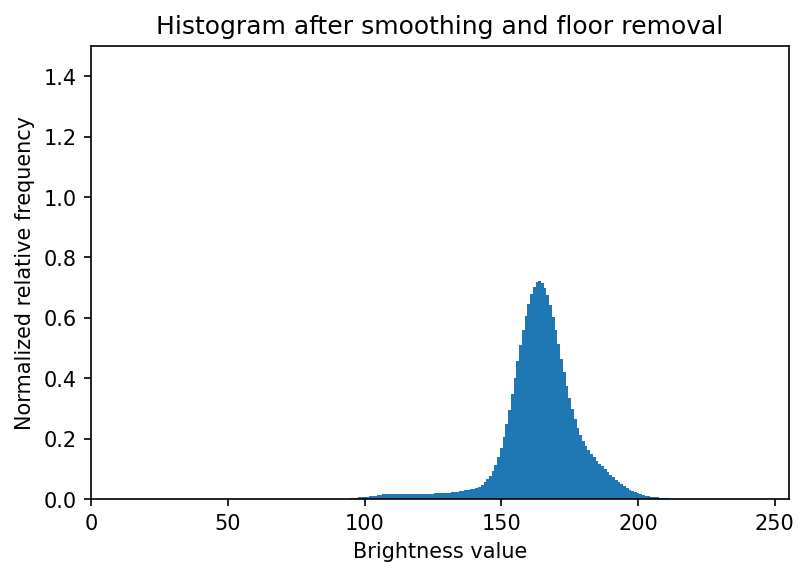


Zone eliminated by Otsu in red

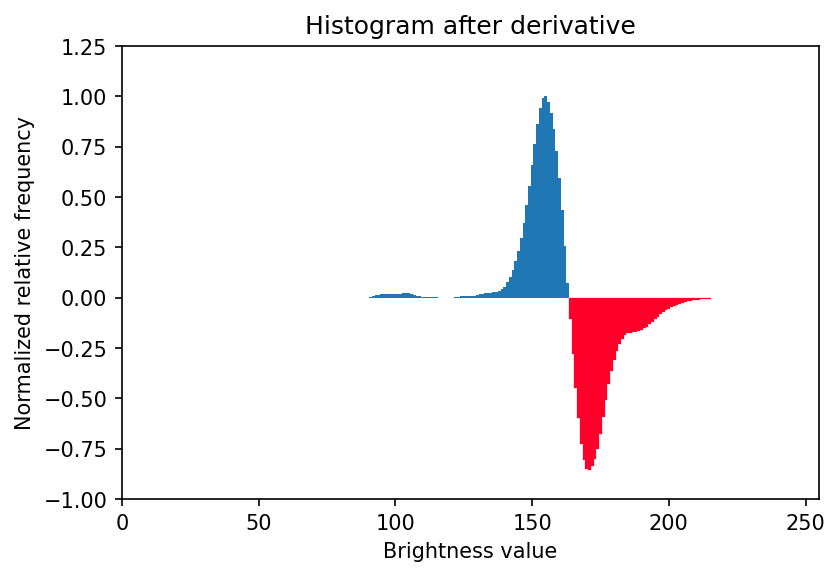
1. (Med) Image histogram is calculated
2. (Low) Values in histogram are smoothed (averaged) with 1 neighboring bins (to each side)
3. +(Low) Histograms are “normalized” by subtracting the lowest 15th percentile value (floor removal)

This step is probably not even necessary, as the values are differentiated later on

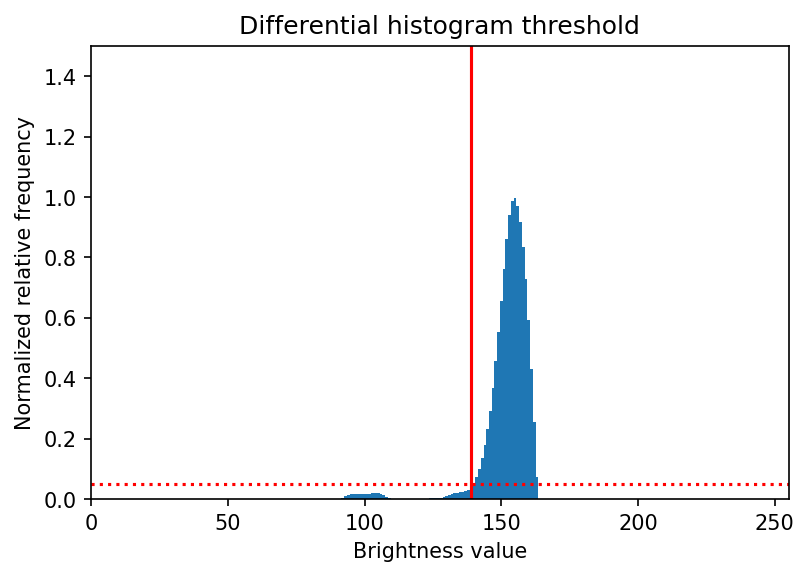
1. +(Low) Histograms are smoothed first with 7 neighboring bins, and then three times with 1



1. (High) Histograms are differentiated, clipped positively, and normalized to the 0-1 range



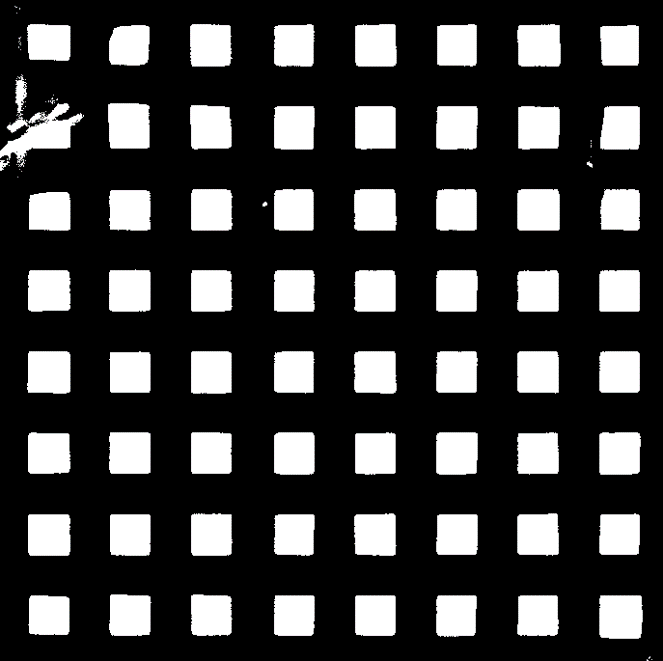
1. +(Low) Histograms are subtracted lowest 10th percentile value (floor removal)
2. +(Med) Relevant levels are set at a normalized relative frequency of 0.05
3. (High) Threshold value is set wherever two relevant values in a row are detected



1. +(Low) Threshold value is multiplied by 0.9

#### Applying thresholds

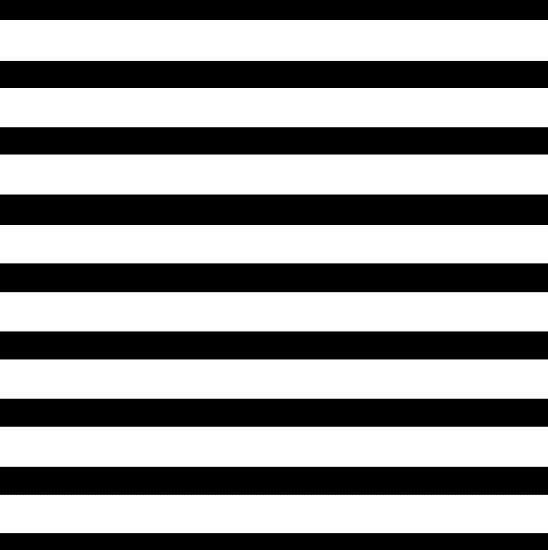
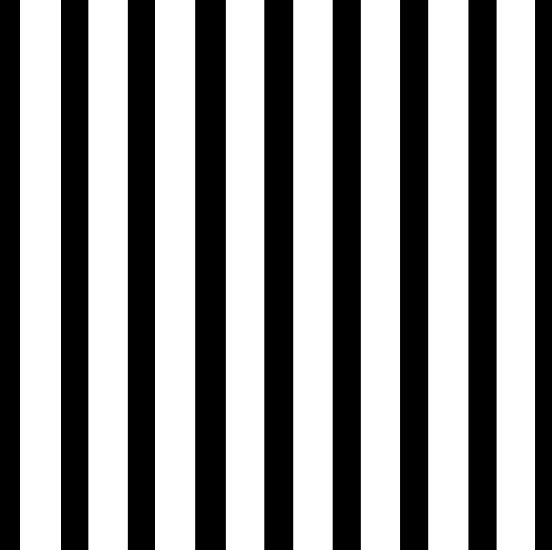
1. (High) Images are thresholded by the obtained value and inverted



1. (Low) First two images (bottom layers) are discarded
2. +(High) A grid is created. It checks for crop rescheduling and pore counting

The steps *a-c* are repeated horizontally and vertically

* 1. A row or column strip is created, with white values where at least 15% of the column or row’s values are white (pores). E.g.: For the column strip, each pixel will be white if the associated row strip has at least 15% white pixels in the image
  2. The row or column is morphologically processed
     1. Open with a kernel of 7 px
     2. Closed with 15 px
     3. Eroded with 33 px
     4. Dilated with 39 px
  3. Another morphology processing, but between layers
     1. Closed with 9 px (in this case, 9 layers)

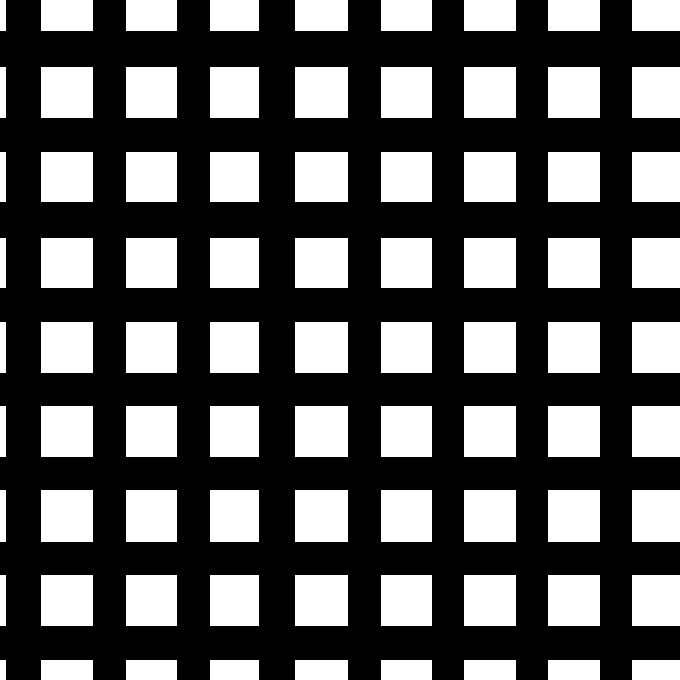
* 1. Column and row strips are resized and multiplied together to form a grid

A picture containing background pattern

Description automatically generated

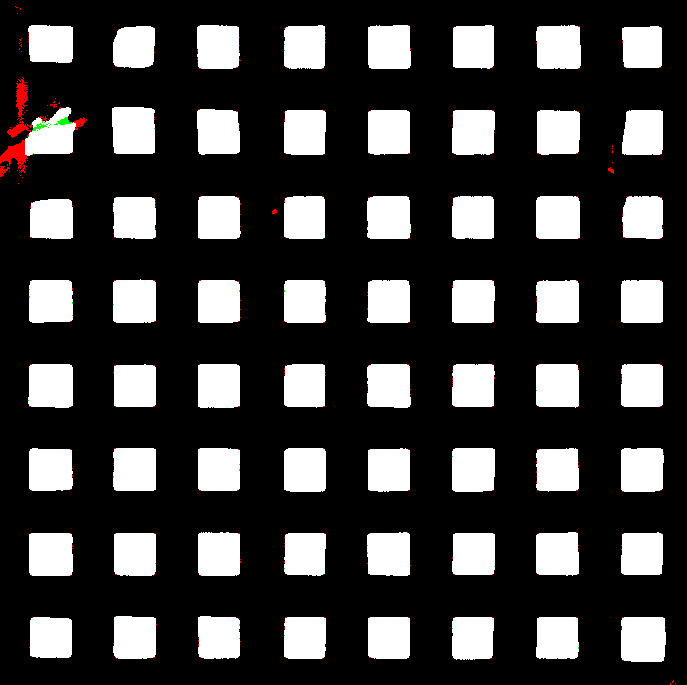
1. (Med) Grid is checked for crop scheduling. If there are any white parts touching the edges, it is assumed that the ROI is not correctly placed

These steps are only done if white parts are detected, as in the example image



* 1. A first try is done by attempting to calculate how much should the ROI be displaced, horizontally and vertically
  2. If a displacement can be calculated, the analyzer will run again, skipping the threshold part
  3. If a displacement value couldn’t be calculated, or if the re-run didn’t work, the image stack is labeled as “crop retry” for a next run, with a new manual ROI selection

1. (Low) Image stack is multiplied with the grid, to eliminate potentially large white spots outside the pore area
2. +(Med) Image stack is morphologically processed
   1. Opened with 9 px
   2. Closed with 9 px



1. (Low) Missing layers are added with the grid as placeholder

#### Extracting contours

1. (Low) Contours are extracted from grid image
2. (Med) Edge contours are counted, and if they don’t match with the global pore count, the layer is marked as failed

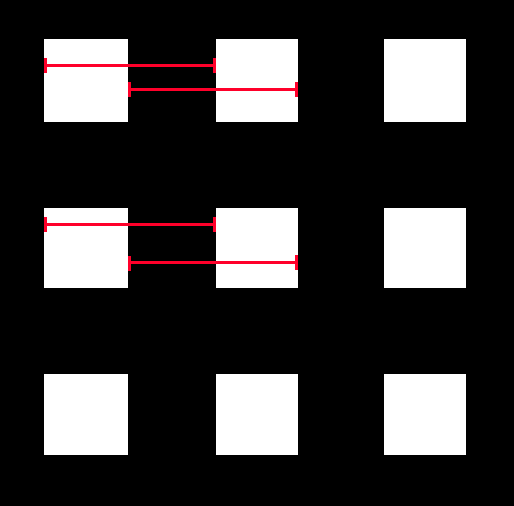
Probably here is where most of the “invalid” pores are coming from, with problems coming from the thresholding stage

1. (High) Contours are extracted from the thresholded image stack
   1. For each layer, the position of each pore is determined by its grid image, with a pad of 10px to every side
   2. During this process, if, for each grid image pore, no contours are found, the pore is labeled as “EMPTY”; and if there are more than one contour, it is labeled as “SPLIT”

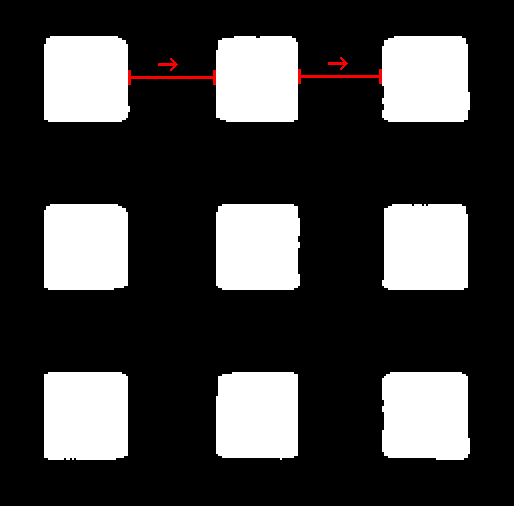
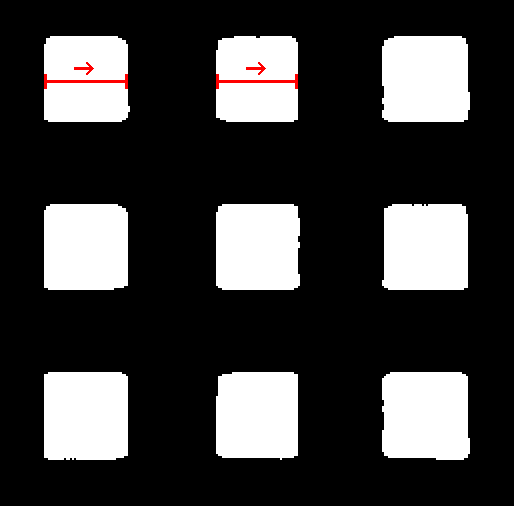
#### Calculating pore information

1. (Low) Vertical and horizontal edge strips outermost edges are set to black, just in case something weird happens
2. (Med) From the strips, position is recorded whenever values change from 0 to 1 (pore to strand) and vice-versa.
3. \*+(Med) The mean from the horizontal and vertical differences is set as the conversion factor (px to mm), divided by 0.8 because our strands are 0.8 mm apart

The conversion factor is global for each sample, not specific for each layer as before



1. \*+(High) Contours from thresholded images are iterated and areas and perimeters are calculated
   1. If any pore is smaller than 0.09 mm2, it is labeled as “SMALL”
2. +(High) Circularity is calculated from area and perimeter, and any pore with a circularity 8% over or under that of a perfect square is labeled as invalid with “CIRC”
3. +(High) Pore distance and size are calculated in the printed direction, using each pore’s 10th and 90th percentile X-Y pixels coordinates as references

Pore position and size, respectively

#### Returns

1. All info is returned to the main wrapper in a convenient Pandas format