Particle Size detection and quantifying its distribution

Sneha Bhattacharjee (112001056),
Rapeti Siddhu Neehal (112001034)
Indian Institute of Technology, Palakkad

Open Ended Lab Project

Mahesh R Panicker
Kanmani Subbu

Shanmugapriyan V G

19 May, 2023

ABSTRACT:

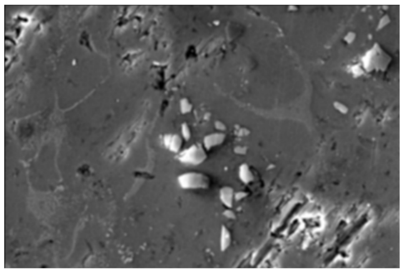
The first step is preprocessing, which consists of removing noise. The second step is finding the particles in the image using thresholding to differentiate between foreground and background. The last step is to remove any impurities that have been falsely detected as particles and then find out the statistics of the image (ie. size distribution and quantifying its distribution)

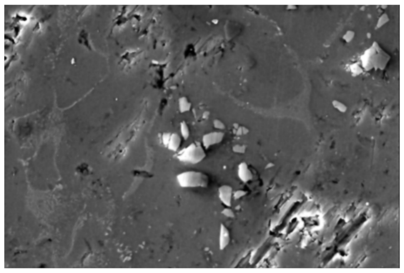
DENOISE:

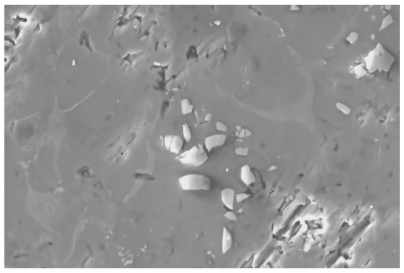
We have worked on upwards of 10 methods for denoising, of which

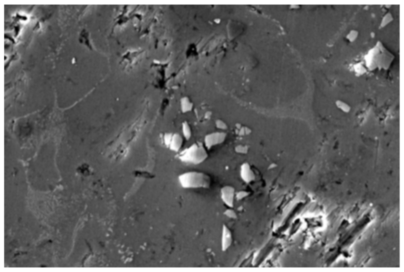
- (*best*)Bilateral blur seems to be the better option as it helps reduce noise without having any drastic effect on the edges. This later causes problems during segmentation(watershed)
- (*most compatible*) Non-local means denoising (nlm), it preserves certain textures. It was able to get rid of much of the Gaussian noise. It also doesn't excessively blur the smaller particles.
- Median blur works in this use case as a good amount of gaussian noise.
- B3MD manages to smoothen the matrix, but it reduces the color contrast between the particles and the medium.
- Total variation denoising: Much like nlm but the edges become less sharp and the image is smoother.
- Other filter include: averaging using a kernel, gaussian blur, denoise_tv_chamboll, etc… but they were ineffective.

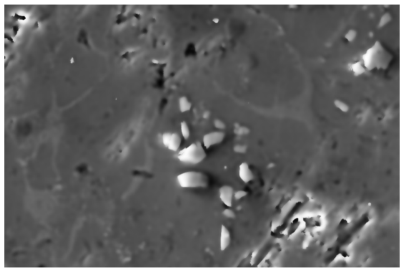
advanced way: using morphological operations to remove noise. A series of erode() and dilate() function calls on the image to reduce the noise while preserving the edges. Particularly useful with watershed.

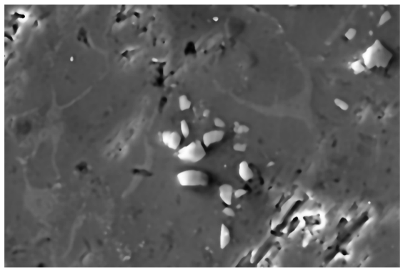

NLM DENOISE


TV_CHAMBOLL DENOISE


B3MD FILTER


GAUSSIAN BLUR

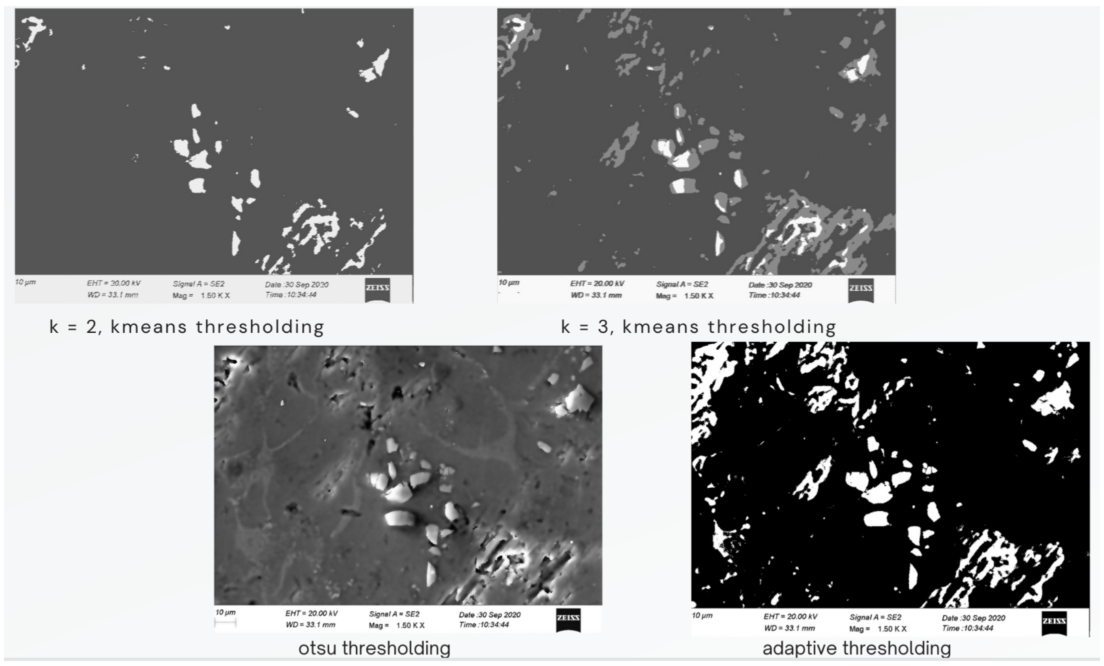

BILATERAL BLUR


MEDIAN BLUR

THRESHOLDING:

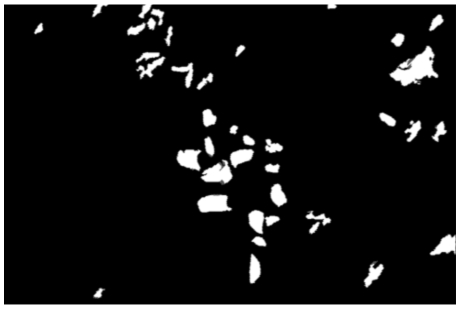
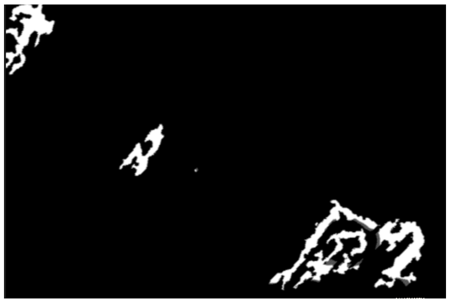
The next step is to differentiate background (polished metal) from foreground (silica, groves and indistinguishable noise). We tested the following thresholding techniques:

- **K-means thresholding:** it clusters the pixels into k clusters and then assigns each pixel to a cluster, the cluster with the highest threshold value that is less than the pixel's intensity. This effectively divides the image into different regions based on their pixel intensities.
  In our trials (k=2) did not work because the foreground (particle) was again split into shadow and light regions. (k=3) gave the best result by assigning one cluster for background, second for shadow and third for light regions in the foreground. Further binary thresholding merged the shadow and light regions to form the particle.
- **Otsu's thresholding:** finds the value that minimizes the intra-class variance of the foreground and background pixels in the image. The threshold value is chosen such that the intra-class variance is minimized, which implies that the foreground and background pixels will be well-separated. otsu's method uses a statistical approach to calculate this threshold.
  We found this method to be the most compatible.
- **Adaptive thresholding:** it partitions the image into small regions, typically using a sliding window or a block-based approach. Then, for each region, the threshold value is calculated based on the local statistics of the image, such as the mean or median pixel intensity values. the threshold will be based on the local properties of the image. This better fits our scenario as all the foreground is not very clearly differentiable and the silica particles (foreground) vary in the intensities, And it does work better, but it was not very compatible with watersheding.



SEGMENTATION and CONTOUR DETECTION:
After thresholding 2 new problems arised, one is segmentation. The first problem was to find the particles that are groves and remove them. We used findContour() to detect the contours. These contours will be stored in a list and we used perimeter^2 / area of the contour as the metric to remove the grooves because the grooves will have this ratio larger when compared to the silica particles as the particles tend to be closer to circle/square and not narrow like grooves.

- The metric to qualify a particle as a grove was best found tobe max(ratio)/3.5 from trial and error.
- We also explored another metric using max(ratio)*10/avg(ratio). This is a more generic check.

The first image has filtered particles and the second image has disqualified particles which are groves.

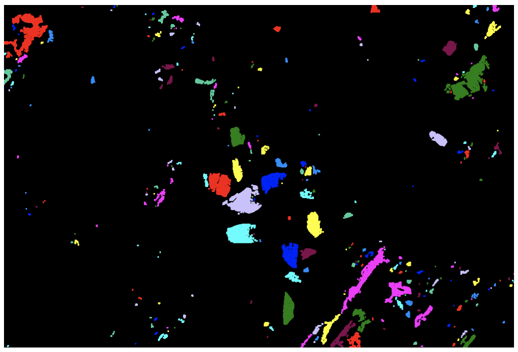
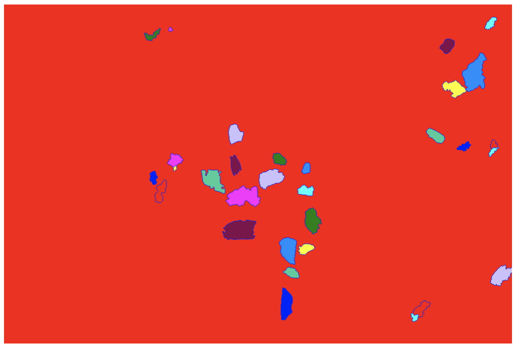
Some of the noise was being clumped together into a particle (the large chunk on the top right of the first image above) and A good portion of the particles that are adjacent are broken chunks from the same parent particle, such chunks must be counted as different particles as once broken their properties differ. To segment such objects in the image we used watershed segmentation. Watershed algorithm has 2 parts and the values in both were manually adjusted by trial and error and the statistics of the image:

1. Marker Generation: The intensity of the pixels is considered the elevation. Initially, markers or seeds are placed at the local minima of the image. These markers represent the starting points of the flooding process.
2. Flooding and Segmentation: The flooding process begins, and the water gradually fills up the basins, merging them at their boundaries. The segmentation is performed by assigning each pixel to a specific basin based on the flooding process. The watershed lines represent the boundaries between different basins.

In combination with morphological functions like erode() and dilute().

- erode() would erode the boundaries causing slightly torn/broken edges. erode() dismantles the particles made up of noise and removes tiny particles.
- dilute() would expand the boundaries and fill in the gaps to enhance a particle.

(a permutation of them both being used over multiple iterations which was decided manually) the particles formed by noise (these are typically smaller than the particles from observation) got broken/granular edges which disintegrated into very small particles (easily removable). The same function also helps strengthen the edge/crevices in actual particles to better segment them.

The first image depicts the foreground distinguished from the background which contains a lot of noise and particles clumped together. The second image is after watershed and morphing.

SCALE DETECTION:
First tried to extract metadata from the .tif files. But the method is highly dependent on the kind of files given (one we tried works only for Zeiss microscope images). Following which I tried using OCR followed by object detection to be able to draw a precise rectangular bounding box around the scale line after isolating the region via OCR on the number. This did not work as it could not detect the scale line. Then we tried another approach using generalized Hough transform to detect the line of the scale, but it did not work since it was detecting several other non-existent lines in the same region. Ultimately have left it to user input.

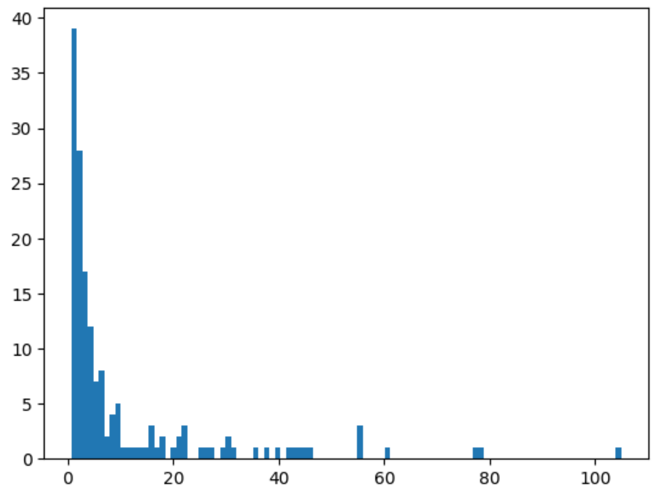
RANDOM FOREST CLASSIFIER:
Key insights:

- Divided particles in images into classes
    - Charred
    - Particles
    - Grooves
    - Shadow
    - Background

Technical problems we faced and insights when implementing random forest in the scope of our project:

- Too little training data to train effectively.
- High variability in image, some images have all classes and some have 1-2 only.
- Ambiguity in classifying the data. Particles very near to an identified groove are less likely to be particles and more likely imperfections adjacent to the groove that are lit well.
- Only bigger particles have shadows. The depth of the shadow gives some idea about the height of the particle.
- Our methods support only identification of raw particles, not coated ones.
- Compared segmented images from the random forest model with a deep learning model built in apeer.
- Manually segmented images, divided them into training and test set and trained a random forest classifier to predict the segmentation using various filters (Gabor filter, sobel, canny, median etc…) as features.

RESULTS:



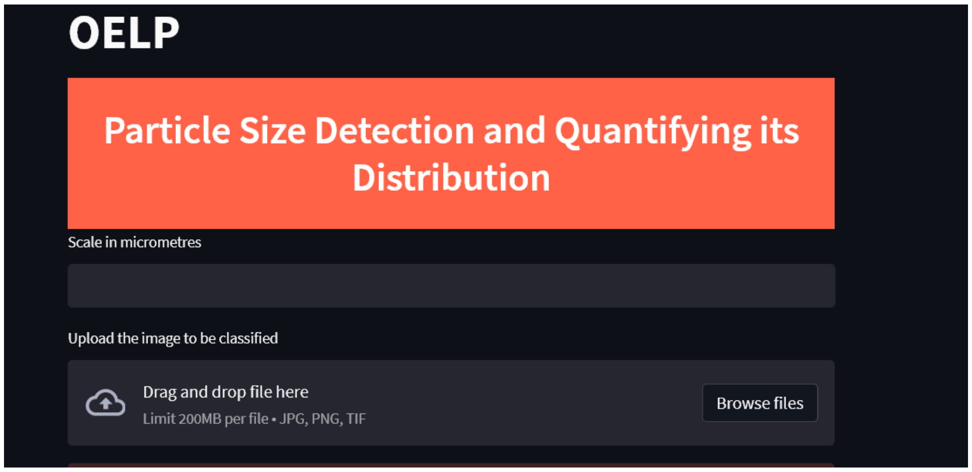
While performing clustering on the distribution, we are not considering the size of the particles here (that can be considered with slight modification). We have used *Hopkins test for cluster tendency* as a metric to test if the distribution is uniform or not.
If the value is between {0.01, ...,0.3}, the data is regularly spaced. If the value is around 0.5, it is random. If the value is between {0.7, ..., 0.99}, it has a high tendency to cluster.

Here, we obtained values around 0.5, which is akin to randomly distributed.

Another metric we used is the Kolmogorov-Smirnov test to compare our distribution to a uniform distribution.

Also it has to be noted that the particles are also embedded in the matrix, all our methods only deal with distribution on the surface and not what is below the tip of the iceberg. A little idea about the height (not depth) of the particles can be obtained by looking at the extent of shadow of each particle.

DEPLOYMENT:
The model did not work that well and gave low accuracy after the contouring was done, even though the segmentation seemed to be better. So we shifted to the previous algorithmic methods and deployed them via streamlit. The interface takes the scale as an input and the image, which can be in the formats .tif, .png or .jpg and displays the Hopkins metric and the particle size histogram.


OELP
Particle Size Detection and Quantifying its Distribution
Scale in micrometres
Upload the image to be classified
Drag and drop file here
Limit 200MB per file • JPG, PNG, TIF
Browse files

References:

1. Validating Clusters using the Hopkins Statistic from IEEE 2004.
2. Application of an improved threshold segmentation method in SEM material analysis: Denghui Li and Yanhong Wang 2018 IOP Conf. Ser.: Mater. Sci. Eng. 322 022057
3. Kolmogorov-Smirnov test on 2D data
4. Genetic algorithms for KMeans Clustering