```
#!/usr/bin/env python
# coding: utf-8

import argparse
import os
import time as time

import h5py
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.spatial as spatial
import skimage
from matplotlib import cm
from matplotlib import pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image
from pylab import gca
from scipy.ndimage import rotate
from scipy.ndimage.interpolation import geometric_transform
from skimage import measure, morphology
from skimage.segmentation import flood_fill
from skimage.transform import resize
from sklearn import cluster
from tqdm import tqdm

from analyze_pore_samples.plotting_utils import (animate_data, frame_tick,
                                                 set_axes_equal)
from analyze_pore_samples.polar_cartesian_convert import (linear_polar,
                                                          map_pixel,
                                                          polar_linear)
from utils import *

rootdir =  './sample_dataset'
subfolders = [ f.path for f in os.scandir(rootdir) if f.is_dir() ]


voxelsizes = np.loadtxt('./sample_dataset/VoxelSize.txt', skiprows = 1,
dtype= 'str')
voxelsizes = dict(voxelsizes)




'''
legend:
default location : upper left
default fontsize: 8
Frame is always off
'''
```

```
54 def load_data(folder_index, num=100, start=0, shape = None, verbose = 0):
55     '''
56     Folder index: [int] Index corresponding to the folder of .tiff images to
   be analyzed
57     num: [int] Number of frames to analyze
58     start: [int] Starting frame of analysis. Frames "start" until "start +
   num" will be analyzed.
59     shape: [int] default None -- desired shape of cross-section image in
   pixels, will be cropped to this shape
60     verbose: [int]
61     '''
62     subfolder = subfolders[folder_index]
63     angle_dict= {}
64     for key in ['B05-04', 'B05-05', 'B05-06']:
65         angle_dict[key] = 0.662
66     for key in ['F26-01', 'F26-02', 'F26-03', 'F26-04', 'F26-05']:
67         angle_dict[key] = 164.188
68     for key in ['F26-06', 'F26-07', 'F26-08', 'F26-09', 'F26-10']:
69         angle_dict[key] = -83.302
70     if verbose > 0:
71         print(" Currently reading: " + str(subfolder))
72     pictures = os.listdir(subfolder)
73     pictures.sort()
74 
75     # Get the dimensions of the data from folder name
76     picname = pictures[0]
77     voxel_name = [s for s in picname.split('_')[0].split('H') if s]
78     im_test = Image.open(os.path.join(subfolder, picname))
79     image_test = np.array(im_test)
80     imstack = np.zeros((image_test.shape[0], image_test.shape[1], num),
   'uint8')
81     diameterx = []
82     diametery = []
83     xmaxs = []
84     xmins = []
85     ymaxs = []
86     ymins = []
87     xmin_test = np.min(np.where(image_test == 255)[0])
88     xmax_test = np.max(np.where(image_test == 255)[0])
89     ymin_test = np.min(np.where(image_test == 255)[1])
90     ymax_test = np.max(np.where(image_test == 255)[1])
91     voxelsize = float((voxelsizes)['H-'+voxel_name[0]])
92     if verbose > 1:
93         print(subfolder, 'SUBFOLDER', voxelsize, folder_index, voxelsize)
94     for count, picture in tqdm(enumerate(pictures)):
95         if count < start:
96             continue
97         try:
98             im = Image.open(os.path.join(subfolder, picture))
99             if verbose> 1:
100                 print(os.path.join(subfolder, picture))
101             image = np.array(im, dtype = 'uint8')
102             new_image = rotate(image, -angle_dict[voxel_name[0]], reshape =
    False, order = 0)
103             image = np.array(new_image, dtype = 'uint8')
```

```
104
105             imstack[:, :, count - start] = new_image  # [:,:]
106         except:
107             breakpoint()
108         xmin = np.min(np.where(image == 255)[0])
109         xmax = np.max(np.where(image == 255)[0])
110         ymin = np.min(np.where(image == 255)[1])
111         ymax = np.max(np.where(image == 255)[1])
112
113         xcenter = image.shape[0]//2
114         ycenter = image.shape[1]//2
115
116
117         ymaxs.append(ymax)
118         ymins.append(ymin)
119         xmins.append(xmin)
120         xmaxs.append(xmax)
121
122         diameterx.append(xmax-xmin)
123         diametery.append(ymax-ymin)
124
125         if count >= num + start - 1:
126             break
127
128     if shape is None:
129         return voxelsize, imstack[np.min(xmins):np.max(xmaxs),
    np.min(ymins):np.max(ymaxs), :], voxel_name, angle_dict[voxel_name[0]]
130     else:
131
132         if shape > imstack.shape[0] or shape > imstack.shape[1]:
133             return voxelsize, imstack, voxel_name, angle_dict[voxel_name[0]]
134         else:
135             return voxelsize, imstack[xcenter-shape//2:xcenter+shape//2,
    ycenter-shape//2:ycenter+shape//2, :], voxel_name, angle_dict[voxel_name[0]]
136
137
138
139
140
141 def get_boundary(imstack):
142     '''
143     Return boundary given segmented image.
144
145     Parameters
146     ----------
147     imstack : NumPy array, with boundary listed as pixels with intensity 255.
148
149     Returns
150     -------
151     boundary_imstack: Binary NumPy array indicating boundary pixels with 1,
    and other pixels with 0.
152     '''
153
154     boundary_imstack = np.copy(imstack)
155     boundary_imstack[imstack != 255] = 0
```

```
156     boundary_imstack[imstack == 255] = 1
157     if np.sum(boundary_imstack) == 0:
158         breakpoint()
159     return(boundary_imstack)
160 def replace_boundary(imstack):
161 
162 
163     '''
164     Return rescaled segmented image with pore and boundary phases set to 1.
165 
166     Parameters
167     ----------
168     imstack : NumPy array, with boundary listed as pixels with intensity 255
    and pores listed with intensity 159.
169 
170     Returns
171     -------
172     newstack: Binary NumPy array indicating pore pixels with 1, and all other
    pixels with 0.
173 
174     boundary_imstack: Binary NumPy array indicating boundary pixels with 1,
    and all other pixels with 0.
175 
176     '''
177 
178     boundary_imstack = np.copy(imstack)
179     boundary_imstack[imstack != 255] = 0
180     boundary_imstack[imstack == 255] = 1
181     newstack = np.copy(imstack)
182     newstack[imstack  == 255] = 0
183     newstack[imstack == 159] = 1
184     newstack = np.array(newstack, dtype = 'uint8')
185     return newstack, boundary_imstack
186 def extract_pores(imstack):
187     '''
188     Return list of RegionProps objects representing the pores present in the
    input array.
189 
190     Parameters
191     ----------
192     imstack : NumPy array, with boundary listed as pixels with intensity 255
    and pores listed with intensity 159.
193 
194     Returns
195     -------
196     props: List of RegionProps objects representing the pores present in
    imstack.
197 
198     im: Labeled binary image, with each pore represented by a group of pixels
    with a unique intensity.
199 
200     boundary_imstack: Binary NumPy array indicating boundary pixels with 1,
    and all other pixels with 0.
201 
202     '''
```

```
203
204     boundary_imstack = np.copy(imstack)
205     boundary_imstack[imstack != 255] = 0
206     boundary_imstack[imstack == 255] = 1
207     newstack = np.copy(imstack)
208     newstack[imstack  == 255] = 0
209     newstack[imstack == 159] = 1
210     newstack = np.array(newstack, dtype = 'uint8')
211     im = measure.label(newstack[:,:,:])
212     props = skimage.measure.regionprops(im[:,:,:])
213     return props, im, boundary_imstack
214 def save_data(data, folder, name, count  = 0):
215     '''
216     Return RegionProps objects representing the pores present in the input
    array.
217
218     Parameters
219     ----------
220     imstack : NumPy array, with boundary listed as pixels with intensity 255
    and pores listed with intensity 159.
221
222     Returns
223     -------
224     props: Binary NumPy array indicating pore pixels with 1, and all other
    pixels with 0.
225
226     boundary_imstack: Binary NumPy array indicating boundary pixels with 1,
    and all other pixels with 0.
227
228     '''
229     name = 'threechannel'
230     f = h5py.File(folder+'/'+str(name)+'_'+str(count)+'.hdf5', 'w')
231     f.create_dataset('data', data=data, dtype='i8', compression='gzip')
232     f.close()
233 def create_data_channels(im, props, k = 1, pore_list_attr = [], save =
    False):
234     '''
235     Returns list of pore attributes, saves information about each pore.
236
237     Parameters
238     ----------
239     im: Labeled binary image, with each pore represented by a group of pixels
    with a unique intensity.
240     props:  List of RegionProps objects representing the pores present in im.
241     k: The current folder index that is being processed.
242     pore_list_attr: The previous list of pore_list_attributes that the
    attributes extracted here should be appended to. (Default: [])
243     save: Whether to save the pore attributes extracted here. (Default:
    False)
244
245
246     Returns
247     -------
248     pore_list_attr: A nested list of pore volumes, start indices,
    anisotropies, orientations, phis
```

```
    '''
    im_channels = im
    previous_count = len(pore_list_attr)
    pores_cross = np.unique(im_channels[:,:,0]) + 1

    output_dir = './analyze_pore_samples/results/individual_pore_samples/partsample' + str(k) + '/'
    os.makedirs(output_dir, exist_ok = True)


    size = 32
    count = 0
    name = 'pore_original'
    for p_idx in np.arange(len(props)):
        inertia_eigval =  props[p_idx].inertia_tensor_eigvals
        inertia =  props[p_idx].inertia_tensor
        maxeig = np.argmax(inertia_eigval)
        eigvec = np.linalg.eig(props[p_idx].inertia_tensor)[1]
        eigvals = np.linalg.eig(props[p_idx].inertia_tensor)[0]
        anis = 1 - np.min(eigvals)/np.max(eigvals)
        max_0 = np.max([prop.area for prop in props])
        z_0 = np.max([prop.centroid[2] for prop in props])

        maxvector = eigvec[:, maxeig]
        orientation = angle_between(maxvector, np.array([0,0,1]))
        phi = angle_between(maxvector, np.array([0,1,0]))

        pore_3d = np.zeros((size*2, size*2, size*2))
        xdist = props[p_idx].slice[0].stop - props[p_idx].slice[0].start
        ydist = props[p_idx].slice[1].stop - props[p_idx].slice[1].start
        zdist = props[p_idx].slice[2].stop - props[p_idx].slice[2].start
        if save == True:
            try:

                pore_3d[size - xdist//2: size+(xdist-xdist//2), size-ydist//2:size+(ydist- ydist//2), size-zdist//2:size+(zdist-zdist//2)]  = np.array(props[p_idx].image, dtype = 'float')
                f = h5py.File(str(output_dir)+'/'+str(name)+'_'+str(count+previous_count)+'.hdf5', 'w')
                f.create_dataset('data', data=pore_3d, dtype='i8', compression='gzip')
                f.close()

                count = count + 1
                if count % 100 ==0:
                    print('Saving pores ', count)
                pore_list_attr.append([props[p_idx].area, props[p_idx].slice[2].start, anis, orientation, phi])
            except Exception as e:
                print(props[p_idx].area, e)
                continue
```

```
297         if p_idx % 50 == 0:
298             print("Pore processed: " + str(p_idx) + " pores out of " +
    str(len(props)) + " pores...")
299 
300 
301     pore_matrix =np.array(pore_list_attr)
302     pore_list_dict = {'Volume': pore_matrix[:,0], 'z_start':
    pore_matrix[:,1], 'anisotropy': pore_matrix[:,2], 'orientation':
    pore_matrix[:,3], 'phi': pore_matrix[:,4]}
303     df = pd.DataFrame.from_dict(pore_list_dict)
304     if save:
305         np.savetxt(output_dir+'pore_matrix', pore_matrix)
306     return pore_list_attr
307 
308 def analyze_boundaries(k, limit, index = 0, data_info = None):
309     '''
310     Returns 2-D projection of the surface, following conversion to polar
    coordinates.
311 
312     Parameters
313     ----------
314 
315     k: The current folder index that is being processed.
316     limit: The ending frame index to be loaded for processing.
317     index: The starting frame to be loaded for processing. limit - index
    frames will be processed in this program
318     data_info: Pre-loaded CT data to use for processing. Structure:
    data_info[0] is the voxel conversion to microns, data_info[1] is the 3D array
    defining the part segment.
319 
320 
321     Returns
322     -------
323     profile_2d: The 2-D projection of the surface.
324     '''
325     print("Processing boundaries ...")
326     os.makedirs('analyze_pore_samples/results', exist_ok = True)
327     window = limit
328     profile_2d = []
329     if data_info is None:
330         voxelsize, imstack_test, _, _  = load_data(k, num = limit, start =
    index)
331     elif limit == data_info[1].shape[2]:
332         voxelsize, imstack_test = data_info
333     else:
334         voxelsize, imstack_test, _, _  = load_data(k, num = limit, start =
    index)
335     # while index < limit:
336         # imstack = imstack_test[:,:, index:window+index]
337         # index += window
338 
339     before_time = time.time()
340     boundary_imstack = get_boundary(imstack_test, k = k)
341     if np.sum(boundary_imstack) == 0:
342         print("Boundary imstack is empty")
```

```
343         breakpoint()
344     oldtime = time.time()
345     for sample in range(imstack_test.shape[2]):
346         if sample % 500 == 0:
347             print("processed {} samples out of {}".format(sample,
    imstack_test.shape[2]))
348         center = boundary_imstack[:,:, sample].shape[1]//2
349         polar= linear_polar(boundary_imstack[:,:, sample])
350         line = np.argmax(polar, axis = 0)
351 
352         if 0 in line:
353             # breakpoint()
354             dilated = morphology.dilation(polar)
355             skeleton = morphology.skeletonize(dilated)
356             polar = skeleton
357             line = np.argmax(polar, axis = 0)
358         if 0 in line:
359             idxs = np.where(line == 0)[0]
360             # breakpoint()
361             for case in idxs:
362                 indices = np.where(line)[0]
363                 if len(np.where(line)[0]) == 0:
364                     print("Empty line")
365                     # breakpoint()
366                 nearest = indices[np.argmin(np.abs(np.where(line)[0] -
    case))]
367                 min_edge = np.max([0, nearest-5])
368                 max_edge = np.min([len(line), nearest+5])
369                 neighborhood = line[min_edge:max_edge]
370                 line[case] = int(np.mean(neighborhood[neighborhood>0]))
371 
372         angs = np.array([i*2*np.pi/polar.shape[1] for i in
    range(polar.shape[1])])
373 
374         profile_2d.append(line)
375     after_time = time.time()
376     print(after_time-before_time, "time in seconds")
377     profile_2d = np.array(profile_2d)
378 
379     profile_2d[profile_2d== 0 ] = np.mean(profile_2d[profile_2d > 0 ])
380     plt.clf()
381     plt.yticks(np.arange(3000)[::500], np.round(np.arange(3000)*voxelsize)
    [::500])
382     plt.xticks(np.arange(profile_2d.shape[1])[::500],
    np.round(angs[::500]/np.pi,1))
383     plt.xlabel(r'$\theta$ [$\pi$ rad]')
384     plt.ylabel(r'z [$\mu$ m]')
385     ax = plt.gca()
386     img = plt.imshow(profile_2d*voxelsize - np.mean(profile_2d*voxelsize),
    cmap = 'binary' )
387 
388     np.save('./make_surface/original_profilometry_' + str(k) + '.npy',
    profile_2d)
389     np.save('./analyze_pore_samples/results/profilometry_' + str(k) + '.npy',
    profile_2d)
```

```
390
391     plt.title('Part sample: ' + str(k) + ' Average  elevation:
    {:.1f}'.format(np.mean(np.abs(profile_2d - np.mean(profile_2d)))*voxelsize) +
    r'$\mu$m Maximum =  {:.1f}'.format(np.max(np.abs(profile_2d -
    np.mean(profile_2d)))*voxelsize) +r'$\mu$m', fontsize = 8)
392
393
394     divider = make_axes_locatable(ax)
395     cax = divider.append_axes("right", size="5%", pad=0.05)
396     cbar = plt.colorbar(img, cax=cax)
397     cbar.set_label(r'elevation [$\mu$m]', labelpad = -40, y = 1.4, rotation =
    0)
398     plt.tight_layout()
399     plt.savefig('./analyze_pore_samples/results/profilometry_' + str(k) +
    '.png')
400     plt.clf()
401     plt.close('all')
402
403     return profile_2d
404 def pore_extract(pore_dataset):
405     '''
406     Returns: None
407
408     Computes pore metrics, saves 3-D binary representation of each pore as an
    HDF5 file, given a list of RegionProps objects representing pores.
409
410     Parameters
411     ----------
412
413     pore_dataset: List of RegionProps objects representing pores.
414
415
416     Returns
417     -------
418     None.
419     '''
420     count = 0
421     size = 32
422     output_dir = str(size*2) + '_full_labeled_largepore_threephase_3D'
423     name = 'tiff_threephase'
424     os.makedirs(output_dir, exist_ok=True)
425     stats = compute_statistics(pore_dataset, voxelsize = 1)
426     labels_inst = np.array([stats['anisotropies'], np.array(stats['vols'])**
    (1/3), stats['orientations'], np.arange(len(pore_dataset))]).T
427     np.savetxt(str(output_dir)+'/labels'+str(name)+'_' + str(id) + '.txt',
    labels_inst)
428
429     for index, pore_sample in enumerate(pore_dataset):
430
431         pore_3d = np.zeros((size*2, size*2, size*2))
432         xdist = pore_sample.slice[0].stop - pore_sample.slice[0].start
433         ydist = pore_sample.slice[1].stop - pore_sample.slice[1].start
434         zdist = pore_sample.slice[2].stop - pore_sample.slice[2].start
435
436         try:
```

```
437
438             pore_3d[size - xdist//2: size+(xdist-xdist//2), size-
    ydist//2:size+(ydist- ydist//2), size-zdist//2:size+(zdist-zdist//2)]  =
    np.array(pore_sample.image, dtype = 'float')
439
440             if len(np.where(pore_3d)[0]) == 0:
441                 print("Pore not found")
442                 breakpoint()
443             f =
    h5py.File(str(output_dir)+'/'+str(name)+'_'+str(pore_extract.counter)+'.hdf5'
    , 'w')
444             f.create_dataset('data', data=pore_3d, dtype='i8',
    compression='gzip')
445             f.create_dataset('target', data=labels_inst[index],
    compression='gzip')
446             f.close()
447
448             count = count + 1
449             pore_extract.counter   = pore_extract.counter   + 1
450
451         except Exception as e:
452             print(pore_sample.area, e)
453
454             continue
455
456
457 pore_extract.counter = 0
458
459 def find_intersection(boundary_images, centroid, polar_images = None,
    polar_image = None, boundary =None, real_image = None, prop = None, ts_list =
    None, rs_list = None, plot = False):
460     '''
461     Returns: For the point on the boundary nearest a given pore, returns the
    boundary distance, standard deviation of boundary distance,
462     angle to boundary and normalized distance from the pore to the center of
    the part.
463
464
465     Parameters
466     ----------
467
468     boundary_images: list of boundary images, converted to polar co-
    ordinates.
469     centroid: Co-ordinates of the center of the pore used for analysis
470     polar_images: list of segmented images, containing both pore and boundary
    material, converted to polar co-ordinates
471     boundary: list of boundary images, kept in cartesian coordinate frame
472     real_image: list of segmented images, containing both pore and boundary
    material, in original cartesian co-ordinates
473     ts_list: Angle discretization of polar co-ordinate conversion
474     rs_list: Radial discretization of polar co-ordinate conversion.
475
476
477
478     Returns
```

```
479     -------
480     boundary_radius: Distance from center, to closest point on the surface
    from the given pore.
481     std_dev: Standard deviation of the area on the surface nearest the pore,
    represents the local roughness
482     angle: Angle from center to closest point from the given pore to the
    surface.
483     pore_distance: Distance from center to given pore.
484     '''
485     z_idx = int(centroid[2])
486     polar_boundary = boundary_images[z_idx]
487     ts = ts_list[0]
488     rs = rs_list[0]
489     polar_images = polar_images[z_idx]
490     center_x = real_image.shape[0]//2
491     center_y = real_image.shape[1]//2
492     slope = (centroid[1] - center_y)/(centroid[0]  - center_x + 1e-6)
493     x_trial = np.linspace(0, real_image.shape[0]-1,
    int(real_image.shape[0]*1.5))
494     y = slope*x_trial -slope*center_x + center_y
495
496     y_int = np.array(y, dtype = 'int')
497     y_int[y_int > real_image.shape[1] - 1 ] = real_image.shape[1] - 1
498     y_int[y_int < 0 ] =0
499     x_int = np.array(x_trial, dtype = 'int')
500     collision = np.where(boundary[x_int,y_int])[0]
501     if len(collision) == 0:
502         x_trial = np.linspace(0, real_image.shape[0]-1,
    int(real_image.shape[0]*15))
503         y = slope*x_trial -slope*center_x + center_y
504         y_int = np.array(y, dtype = 'int')
505         y_int[y_int > real_image.shape[1] - 1 ] = real_image.shape[1] - 1
506         y_int[y_int < 0 ] =0
507         x_int = np.array(x_trial, dtype = 'int')
508         collision = np.where(boundary[x_int,y_int])[0]
509     pixel = map_pixel(int(centroid[0]), int(centroid[1]), real_image)
510     angle = (2*np.pi+ np.arctan2(centroid[0]-center_x, centroid[1] -
    center_y))%(2*np.pi)
511     idx_angle = np.argmin(np.abs(ts - angle))
512     angle_pore = ts[idx_angle]
513     radii = []
514     radii_center = []
515     try:
516         nearest_idx_angle  = np.where(polar_boundary)
    [1][np.argmin(np.abs(np.where(polar_boundary)[1] - idx_angle))]
517         radius_range = np.where(polar_boundary[:,nearest_idx_angle])[0]
518     except:
519         breakpoint()
520     if len(radius_range) == 0:
521         radius_range = np.where(polar_boundary[:, np.max([0, idx_angle -
    15]):idx_angle+15])[0]
522     elif len(radius_range) == 0:
523         radius_range = [np.mean(np.where(polar_boundary[:, np.max([0,
    idx_angle - 50]):idx_angle+50])[0])]
524
```

```
525     radii.extend((pixel[0]- radius_range)/(radius_range))
526 
527     test = np.array(np.copy(polar_boundary) > 0, dtype = 'int')
528     test[:,idx_angle-50:idx_angle+50] = test[:,idx_angle-50:idx_angle+50]*500
529 
530 
531     left_bound = np.max([0, idx_angle-25])
532     right_bound = np.min([len(ts) - 1, idx_angle+25])
533     extent_axial = int(np.round((ts[right_bound] -
    ts[left_bound])*np.mean(radius_range)))
534     left_bound_z = np.max([0,z_idx - extent_axial//2])
535     right_bound_z = np.min([z_idx + extent_axial//2, len(boundary_images)-1])
536 
537     patch =np.argmax(np.array(boundary_images)[left_bound_z:right_bound_z,:,
    left_bound:right_bound],axis = 1)#[:, :,idx_angle-50:idx_angle+50], axis =
    0))
538     patch = patch[patch > 0]
539     patch = patch[patch < boundary_images[0].shape[0]-2]
540     std_dev = np.std(patch)
541     boundary_radius = np.min(radii)
542     pore_distance = np.min(pixel[0]/radius_range)
543     return boundary_radius, std_dev, angle, pore_distance
544     # breakpoint()
545 
546 def process_images(k, dict_properties = None, save = True, save_pores =
    False,data_info = None, frame_window = 200, shift = 100):
547     '''
548     Returns: Updated dictionary of pore properties, list of pores found in
    dataset.
549 
550     Parameters
551     ----------
552 
553     k: Index of original folder used for analysis.
554     dict_properties: Pore properties stored in a dictionary, from a previous
    iteration
555     save: whether to save the probability matrices calculated during analysis
556     save_pores: whether to save the pores extracted during analysis
557     data_info: Allows for the specification of a specific part to be used
558     frame_window: Specifies the length of the processing window used for
    analysis.
559     shift: Specifies the shift of the processing window between iterations of
    analysis.
560 
561 
562 
563 
564     Returns
565     -------
566     dict_properties: Pore properties stored in a dictionary,
567     pores_total: A list of regionprops objecccccts, representing every pore
    found in the segment used for analysis.
568     '''
569 
570     os.makedirs('analyze_pore_samples/results', exist_ok = True)
```

```
571
572
573     index = 0
574     if dict_properties == None:
575         total_anisotropies = []
576         total_x = []
577         total_y = []
578         total_orientations = []
579         total_z = []
580         total_surf_dist  = []
581         total_maj = []
582         total_min = []
583         total_vols = []
584         anisotropy_zs = []
585         orientations_zs = []
586         vols_zs = []
587         total_phis = []
588         surf_dist = []
589         surf_angles = []
590         rough  = []
591         x_locs_zs = []
592         y_locs_zs = []
593         total_surf_angles = []
594
595
596     else:
597         total_anisotropies = dict_properties['anisotropies']
598         total_x =  dict_properties['x_locs']
599         total_y =  dict_properties['y_locs']
600         total_orientations =  dict_properties['orientations']
601         total_z =  dict_properties['z_locs']
602         total_maj =  dict_properties['maj_axis_l']
603         total_vols =  dict_properties['vols']
604         total_phis = dict_properties['phis']
605         total_surf_dist = dict_properties['surf_dist']
606         total_surf_angles = dict_properties['surf_angles']
607         total_surf_angles = dict_properties['surf_angles']
608
609     all_stats = [total_x, total_y,total_z,total_anisotropies, total_phis,
    total_orientations, total_vols, total_surf_dist, total_surf_angles,
    surf_dist]
610     strings = ['x centroid', 'y centroid', 'z centroid', 'Anisotropy',
    'Orientation', 'Volume', 'surf_dist', 'surf_angles', 'phis']
611     pore_count = 0
612
613     maj_axis_l_zs = []
614     min_axis_l_zs = []
615     if data_info is None:
616         limit = len(os.listdir(subfolders[k]))
617         voxelsize, imstack_all, _, _  = load_data(k, num = limit, start =
    index)
618     else:
619         voxelsize, imstack_all = data_info
620         limit =  data_info[1].shape[2]
621
```

```
622     pores_total =[]
623     while index < limit:
624         print("Processing pores, index = " + str(index) + " out of " +
    str(limit) + " ..." )
625         # print(index)
626         num = np.min([frame_window, limit - index])
627 
628         imstack = np.copy(imstack_all[:,:, index:index + frame_window])
629 
630         if imstack.shape[2] > 0:
631             props, im, im_orig = extract_pores(imstack)
632         else:
633             print("finished with part sample")
634             return dict_properties
635         polar_images_boundary = []
636         polar_images = []
637         polar_ts = []
638         polar_rs = []
639         # for sample in range()
640 
641         for i in range(index, np.min([limit, index+frame_window])):
642             polar_boundary, rs, ts, o, r, out_h, out_w =
    linear_polar(np.array(im_orig[:,:,i-index] > 0)*1000, verbose = 1)
643             polar_images_boundary.append(np.array(polar_boundary, 'uint8'))
644             if len(polar_ts) > 0:
645                 if np.sum(ts-polar_ts[0])!=0:
646                     breakpoint()
647             polar_ts = [ts]
648             polar_rs = [rs]
649 
650             polar_image = linear_polar(np.array(im[:,:,i-index]> 0)*1000)
651             polar_images.append(np.array(polar_image, dtype = 'uint8'))
652         endframe = frame_window -1 # right boundary of section
653         startframe = 0 # left boundary of section
654         pores_keep = []
655         windowboundary = frame_window - shift - 1
656         windowboundarypores = [prop for prop in props if windowboundary in
    range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that were
    cut off before
657         alreadycountedpores = [prop for prop in props if windowboundary not
    in range(prop.slice[2].start, prop.slice[2].stop) and prop.centroid[2] <
    windowboundary] # remove pores that were counted before, and were not cut off
658         pores = [prop for prop in props if endframe not in
    range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that are not
    cut off
659         boundaries = [prop for prop in props if endframe  in
    range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that are cut
    off
660 
661 
662         if index > 0: # some of the pores have been already counted
663 
664             for prop_idx in range(len(pores)):
665 
666                 alreadycount_bool  = False
```

```
667                 boundary_bool = False
668                 alreadycount_bool =  windowboundary not in
    range(pores[prop_idx].slice[2].start, pores[prop_idx].slice[2].stop) and
    (pores[prop_idx].centroid[2] < windowboundary)
669                 boundary_bool = endframe  in
    range(pores[prop_idx].slice[2].start, pores[prop_idx].slice[2].stop)
670
671                 if not alreadycount_bool and not boundary_bool:
672                     zlength = pores[prop_idx].slice[2].stop -
    pores[prop_idx].slice[2].start
673                     xlength = pores[prop_idx].slice[1].stop -
    pores[prop_idx].slice[1].start
674                     ylength = pores[prop_idx].slice[0].stop -
    pores[prop_idx].slice[0].start
675
676                     if xlength > 1 and ylength > 1 and zlength > 1:
677                         pores_keep.append(pores[prop_idx])
678                         # radius, rness  = find_intersection(boundary_image =
    im_orig[:,:,int(pores[prop_idx].centroid[2])], centroid =
    pores[prop_idx].centroid,  real_image = im[:,:,
    int(pores[prop_idx].centroid[2])], prop = pores[prop_idx])
679                         radius, rness, angle, min_rad  =
    find_intersection(boundary_images = polar_images_boundary, boundary =
    im_orig[:,:,  int(pores[prop_idx].centroid[2])], polar_images = polar_images,
    centroid = pores[prop_idx].centroid,  real_image = im[:,:,
    int(pores[prop_idx].centroid[2])], prop = pores[prop_idx], ts_list =
    polar_ts, rs_list = polar_rs)
680
681                         surf_dist.append(min_rad)
682                         rough.append(rness)
683                         surf_angles.append(angle)
684         elif index == 0:
685             for prop_idx in range(len(pores)):
686
687                 start_boundary_bool = False
688                 start_boundary_bool = 0 in
    range(pores[prop_idx].slice[2].start, pores[prop_idx].slice[2].stop)
689                 if not start_boundary_bool:
690                     zlength = pores[prop_idx].slice[2].stop -
    pores[prop_idx].slice[2].start
691                     xlength = pores[prop_idx].slice[1].stop -
    pores[prop_idx].slice[1].start
692                     ylength = pores[prop_idx].slice[0].stop -
    pores[prop_idx].slice[0].start
693
694                     if xlength > 1 and ylength > 1 and zlength > 1:
695                         pores_keep.append(pores[prop_idx])
696
697                         radius, rness, angle, min_rad  =
    find_intersection(boundary_images = polar_images_boundary,  boundary =
    im_orig[:,:,  int(pores[prop_idx].centroid[2])], polar_images = polar_images,
    centroid = pores[prop_idx].centroid,  real_image = im[:,:,
    int(pores[prop_idx].centroid[2])], prop = pores[prop_idx], ts_list =
    polar_ts, rs_list = polar_rs)
698
```

```
                        rough.append(rness)
                        surf_dist.append(min_rad)
                        surf_angles.append(angle)
        if save_pores:
            if index == 0:
                pore_list_attr = []
            pore_list_attr = create_data_channels(im, pores_keep, k =k,
pore_list_attr=pore_list_attr, save = True )
        pore_count = pore_count + 1
        pores_total.extend(pores_keep)
        print("Added " + str(len(pores_keep)) + " pores, total is now " +
str(len(pores_total)) + " pores" )

        stats = compute_statistics(pores_keep, voxelsize, imstack)
        anisotropy_zs.append(np.mean(stats['anisotropies']))
        orientations_zs.append(np.mean(stats['orientations']))
        vols_zs.append(np.mean(stats['vols']))

        x_locs_zs.append(np.mean(stats['x_locs']))
        y_locs_zs.append(np.mean(stats['y_locs']))
        total_x.extend(stats['x_locs'])
        total_y.extend(stats['y_locs'])
        total_surf_dist.extend(surf_dist)
        total_surf_angles.extend(surf_angles)

        total_maj.extend(stats['maj_axis_l'])
        total_vols.extend(stats['vols'])
        total_phis.extend(stats['phis'])
        total_anisotropies.extend(stats['anisotropies'])
        total_orientations.extend(stats['orientations'])
        total_z.extend(np.array(stats['z_locs'])+voxelsize*index)
        radii, angles = calc_polar(boundary = im_orig, xs = stats['x_locs'],
ys = stats['y_locs'], zs = stats['z_locs'], voxelsize =voxelsize, imstack=im)
        n_bins = 30
        dict_voxel_bins_vols = {}
        dict_polar_bins_vols = {}

        dict_voxel_bins_anis = {}
        dict_polar_bins_anis = {}
        dict_voxel_bins_phis = {}
        dict_polar_bins_phis = {}
        dict_voxel_bins_theta = {}
        dict_polar_bins_theta = {}

        dict_voxel_bins_num = {}
        dict_polar_bins_num = {}
        dict_voxel_bins_orientations = {}
        dict_polar_bins_orientations = {}

        dict_voxel_bins_vols_anis = {}
        x_y_bins = np.zeros((n_bins, n_bins))

        for idx1 in range(n_bins+1):
            dict_voxel_bins_vols_anis[str(idx1)] = []
            for idx2 in range(n_bins+1):
```

```
751                 dict_voxel_bins_vols[str(idx1) + '_' + str(idx2)] = []
752                 dict_polar_bins_vols[str(idx1) + '_' + str(idx2)] = []
753                 dict_voxel_bins_anis[str(idx1) + '_' + str(idx2)] = []
754                 dict_polar_bins_anis[str(idx1) + '_' + str(idx2)] = []
755 
756                 dict_voxel_bins_orientations[str(idx1) + '_' + str(idx2)] =
    []
757                 dict_polar_bins_orientations[str(idx1) + '_' + str(idx2)] =
    []
758                 dict_polar_bins_phis[str(idx1) + '_' + str(idx2)] = []
759 
760                 dict_voxel_bins_phis[str(idx1) + '_' + str(idx2)] = []
761 
762 
763         ybins = np.linspace(0,im.shape[1],num=n_bins,dtype= 'int')
764         xbins = np.linspace(0,im.shape[0],num=n_bins,dtype= 'int')
765         surf_dist_bins = np.linspace(0,1.1,num=n_bins,dtype= 'float')
766         surf_angles_bins = np.linspace(0,2*np.pi,num=n_bins,dtype= 'float')
767 
768 
769         # Volume
770         prob_matrix_volume = np.zeros((n_bins+1, n_bins+1, n_bins))
771         prob_matrix_volume_polar = np.zeros((n_bins+1, n_bins+1, n_bins))
772 
773         for dict_idx, vol in enumerate((total_vols)):
774             x_coord = total_x[dict_idx]
775             y_coord = total_y[dict_idx]
776             z_coord = total_z[dict_idx]
777 
778 
779             x_bin = np.digitize(x_coord/voxelsize, xbins)
780             y_bin = np.digitize(y_coord/voxelsize, ybins)
781             dict_voxel_bins_vols[str(x_bin) + '_' + str(y_bin)].append(vol)
782 
783             radius_coord = total_surf_dist[dict_idx]
784             angle_coord = total_surf_angles[dict_idx]
785             rad_bin = np.digitize(radius_coord, surf_dist_bins)
786             angles_bin = np.digitize(angle_coord, surf_angles_bins)
787             dict_polar_bins_vols[str(rad_bin) + '_' +
    str(angles_bin)].append(vol)
788 
789         bin_edges_vols = np.linspace((np.min(total_vols)**(1/3))/voxelsize,
    (np.max(total_vols)**(1/3))/voxelsize, n_bins+1)
790         polar_edges_vols = np.linspace((np.min(total_vols)**(1/3))/voxelsize,
    (np.max(total_vols)**(1/3))/voxelsize, n_bins+1)
791         for item in dict_voxel_bins_vols.keys():
792             histogram,edges =
    np.histogram((np.array(dict_voxel_bins_vols[item])**(1/3))/voxelsize, bins =
    bin_edges_vols)
793             first_idx = int(item.split('_')[0])
794             second_idx = int(item.split('_')[1])
795             prob_matrix_volume[first_idx,second_idx,:] = histogram
796         for item in dict_polar_bins_vols.keys():
797             histogram,edges =
    np.histogram((np.array(dict_polar_bins_vols[item])**(1/3))/voxelsize, bins =
```

```
    polar_edges_vols)
798             first_idx = int(item.split('_')[0])
799             second_idx = int(item.split('_')[1])
800             prob_matrix_volume_polar[first_idx,second_idx,:] = histogram
801 
802         prob_matrix_phi = np.zeros((n_bins+1, n_bins+1, n_bins))
803         prob_matrix_phi_polar = np.zeros((n_bins+1, n_bins+1, n_bins))
804 
805         for dict_idx, phi in enumerate((total_phis)):
806             x_coord = total_x[dict_idx]
807             y_coord = total_y[dict_idx]
808             z_coord = total_z[dict_idx]
809             x_bin = np.digitize(x_coord/voxelsize, xbins)
810             y_bin = np.digitize(y_coord/voxelsize, ybins)
811             dict_voxel_bins_phis[str(x_bin) + '_' + str(y_bin)].append(phi)
812 
813             radius_coord = total_surf_dist[dict_idx]
814             angle_coord = total_surf_angles[dict_idx]
815             rad_bin = np.digitize(radius_coord, surf_dist_bins)
816             angles_bin = np.digitize(angle_coord, surf_angles_bins)
817             dict_polar_bins_phis[str(rad_bin) + '_' +
    str(angles_bin)].append(phi)
818 
819 
820         bin_edges_phis = np.linspace(0,np.pi, n_bins+1)
821         bin_edges_phis_polar = np.linspace(0,np.pi, n_bins+1)
822 
823 
824         for item in dict_voxel_bins_phis.keys():
825             histogram,edges = np.histogram((dict_voxel_bins_phis[item]), bins
    = bin_edges_phis)
826             first_idx = int(item.split('_')[0])
827             second_idx = int(item.split('_')[1])
828             prob_matrix_phi[first_idx,second_idx,:] = histogram
829         for item in dict_polar_bins_phis.keys():
830             histogram,edges = np.histogram(dict_polar_bins_phis[item], bins =
    bin_edges_phis_polar)
831             first_idx = int(item.split('_')[0])
832             second_idx = int(item.split('_')[1])
833             prob_matrix_phi_polar[first_idx,second_idx,:] = histogram
834 
835         # Anisotropy
836         prob_matrix_anis = np.zeros((n_bins+1, n_bins+1, n_bins))
837         prob_matrix_anis_polar = np.zeros((n_bins+1, n_bins+1, n_bins))
838 
839         for dict_idx, anis in enumerate((total_anisotropies)):
840             x_coord = total_x[dict_idx]
841             y_coord = total_y[dict_idx]
842             z_coord = total_z[dict_idx]
843             x_bin = np.digitize(x_coord/voxelsize, xbins)
844             y_bin = np.digitize(y_coord/voxelsize, ybins)
845             dict_voxel_bins_anis[str(x_bin) + '_' + str(y_bin)].append(anis)
846 
847 
848             radius_coord = total_surf_dist[dict_idx]
```

```
849             angle_coord = total_surf_angles[dict_idx]
850             rad_bin = np.digitize(radius_coord, surf_dist_bins)
851             angles_bin = np.digitize(angle_coord, surf_angles_bins)
852             dict_polar_bins_anis[str(rad_bin) + '_' +
    str(angles_bin)].append(anis)
853
854
855
856         bin_edges_anis = np.linspace(0,1, n_bins+1)
857         bin_edges_anis_polar = np.linspace(0,1, n_bins+1)
858
859         for item in dict_voxel_bins_orientations.keys():
860             histogram,edges = np.histogram((dict_voxel_bins_anis[item]), bins
    = bin_edges_anis)
861             first_idx = int(item.split('_')[0])
862             second_idx = int(item.split('_')[1])
863             prob_matrix_anis[first_idx,second_idx,:] = histogram
864         for item in dict_polar_bins_orientations.keys():
865             histogram,edges = np.histogram((dict_polar_bins_anis[item]), bins
    = bin_edges_anis_polar)
866             first_idx = int(item.split('_')[0])
867             second_idx = int(item.split('_')[1])
868             prob_matrix_anis_polar[first_idx,second_idx,:] = histogram
869
870         prob_matrix_orientation = np.zeros((n_bins+1, n_bins+1, n_bins))
871         prob_matrix_orientation_polar = np.zeros((n_bins+1, n_bins+1,
    n_bins))
872
873         for dict_idx, angles in enumerate((total_orientations)):
874             x_coord = total_x[dict_idx]
875             y_coord = total_y[dict_idx]
876             z_coord = total_z[dict_idx]
877             x_bin = np.digitize(x_coord/voxelsize, xbins)
878             y_bin = np.digitize(y_coord/voxelsize, ybins)
879             dict_voxel_bins_orientations[str(x_bin) + '_' +
    str(y_bin)].append(angles)
880
881             radius_coord = total_surf_dist[dict_idx]
882             angle_coord = total_surf_angles[dict_idx]
883             rad_bin = np.digitize(radius_coord, surf_dist_bins)
884             angles_bin = np.digitize(angle_coord, surf_angles_bins)
885             dict_polar_bins_orientations[str(rad_bin) + '_' +
    str(angles_bin)].append(angles)
886
887
888
889
890         bin_edges_orientations = np.linspace(0,np.pi, n_bins+1)
891         bin_edges_orientations_polar = np.linspace(0,np.pi, n_bins+1)
892
893         for item in dict_voxel_bins_orientations.keys():
894             histogram,edges =
    np.histogram((dict_voxel_bins_orientations[item]), bins =
    bin_edges_orientations)
895             first_idx = int(item.split('_')[0])
```

```
896             second_idx = int(item.split('_')[1])
897             prob_matrix_orientation[first_idx,second_idx,:] = histogram
898 
899         for item in dict_polar_bins_orientations.keys():
900             histogram,edges =
    np.histogram((dict_polar_bins_orientations[item]), bins =
    bin_edges_orientations_polar)
901             first_idx = int(item.split('_')[0])
902             second_idx = int(item.split('_')[1])
903             prob_matrix_orientation_polar[first_idx,second_idx,:] = histogram
904         # Number of pores
905 
906         prob_matrix_num = np.zeros((n_bins, n_bins))
907         prob_matrix_num_polar = np.zeros((n_bins, n_bins))
908 
909         for dict_idx, anis in enumerate((total_x)):
910             x_coord = total_x[dict_idx]
911             y_coord = total_y[dict_idx]
912             z_coord = total_z[dict_idx]
913             x_bin = np.digitize(x_coord/voxelsize, xbins)
914             y_bin = np.digitize(y_coord/voxelsize, ybins)
915             prob_matrix_num[x_bin, y_bin] += 1
916             radius_coord = total_surf_dist[dict_idx]
917             angle_coord = total_surf_angles[dict_idx]
918             rad_bin = np.digitize(radius_coord, surf_dist_bins)
919             angles_bin = np.digitize(angle_coord, surf_angles_bins)
920             prob_matrix_num_polar[rad_bin, angles_bin] +=1
921         # breakpoint()
922         volbin = np.zeros((n_bins+1, n_bins))
923         volbin_anis = np.linspace((np.min(total_vols)**(1/3))/voxelsize,
    (np.max(total_vols)**(1/3))/voxelsize, n_bins)
924         for dict_idx, anis in enumerate((total_anisotropies)):
925             x_coord = total_x[dict_idx]
926             y_coord = total_y[dict_idx]
927             z_coord = total_z[dict_idx]
928             vol = (total_vols[dict_idx])**(1/3)/voxelsize
929             vol_bin_idx = np.digitize(vol, volbin_anis)
930             y_bin = np.digitize(y_coord/voxelsize, ybins)
931             dict_voxel_bins_vols_anis[str(vol_bin_idx)].append(vol)
932         bin_edges_anis_vols = np.linspace((np.min(total_vols)**(1/3))
    /voxelsize, (np.max(total_vols)**(1/3))/voxelsize, n_bins+1)
933         for item in dict_voxel_bins_vols_anis.keys():
934             histogram,edges = np.histogram((dict_voxel_bins_vols_anis[item]),
    bins = bin_edges_anis_vols)
935             first_idx = int(item)
936 
937             volbin[first_idx, :] = histogram
938 
939         # breakpoint()
940 
941         if index+shift >= limit:
942             case = 'full'
943         else:
944             case = ''
945         index = index + shift
```

```
946         # index < limit
947         saving_dir ='./analyze_pore_samples/results/pore_properties'
948         os.makedirs(saving_dir, exist_ok=True)
949 
950         assert np.sum(prob_matrix_num) == len(total_x)
951 
952 
953 
954         if save:
955             prob_matrix_dir = saving_dir + '/probability_matrices/'
956             # if not os.path.isdir(saving_dir + '/probability_matrices/'):
957             os.makedirs(prob_matrix_dir, exist_ok = True)
958             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allprob_matrix_volume.npy', prob_matrix_volume)
959             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allbin_edges_vols.npy', bin_edges_vols)
960             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allprob_matrix_num.npy', prob_matrix_num/((index+shift)/(shift)))
961             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allbin_edges_anis.npy', bin_edges_anis)
962             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allprob_matrix_anis.npy', prob_matrix_anis)
963             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allbin_edges_orientations.npy', bin_edges_orientations)
964             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allprob_matrix_orientations.npy', prob_matrix_orientation)
965             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allbin_edges_phis.npy', bin_edges_phis)
966             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'allprob_matrix_phis.npy', prob_matrix_phi)
967             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarprob_matrix_volume.npy', prob_matrix_volume_polar)
968             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarbin_edges_vols.npy', polar_edges_vols)
969             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarprob_matrix_num.npy', prob_matrix_num_polar/((index+shift)/(shift)))
970             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarbin_edges_anis.npy', bin_edges_anis_polar)
971             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarprob_matrix_anis.npy', prob_matrix_anis_polar)
972             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarbin_edges_orientations.npy', bin_edges_orientations_polar)
973             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarprob_matrix_orientations.npy', prob_matrix_orientation_polar)
974             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarbin_edges_phis.npy', bin_edges_phis_polar)
975             np.save(prob_matrix_dir + str(n_bins) + '_'+ str(k) + case + 
    'polarprob_matrix_phis.npy', prob_matrix_phi_polar)
976 
977 
978         dict_properties = {'x_locs': total_x, 'y_locs': total_y, 
    'maj_axis_l': total_maj, 'phis': total_phis, 'vols': total_vols, 
    'anisotropies':total_anisotropies,'orientations': total_orientations, 
    'z_locs': total_z, 'surf_dist': surf_dist, 'rough':rough, 'surf_angles': 
    surf_angles }
```

```
        locations = [pore.centroid for pore in pores_keep]
        df_properties = pd.DataFrame(dict_properties)
        df_properties.to_csv(saving_dir+'/dict_properties_partial' + str(k)+
    '.csv')
    del polar_images
    return dict_properties, pores_total

def compute_statistics(pore_dataset, voxelsize, imstack = None):
    anisotropies = []
    orientations = []
    vols = []
    sphericity = []
    x_locs = []
    y_locs = []
    z_locs = []
    locations = [pore.centroid for pore in pore_dataset]
    maj_axis_l = []
    min_axis_l = []
    phis = []
    for i in range(len(pore_dataset)):
        pore = pore_dataset[i]
        vols.append(pore_dataset[i]['area']*voxelsize*voxelsize*voxelsize)
        y_locs.append(pore_dataset[i]['centroid'][1]*voxelsize)
        x_locs.append(pore_dataset[i]['centroid'][0]*voxelsize)
        z_locs.append(pore_dataset[i]['centroid'][2]*voxelsize)
        maj_axis_l.append((pore_dataset[i].major_axis_length)*voxelsize)
        min_axis_l.append((pore_dataset[i].minor_axis_length)*voxelsize)
        thresh = 0.3

        inertia_eigval = pore.inertia_tensor_eigvals
        inertia = pore.inertia_tensor
        maxeig = np.argmax(inertia_eigval)
        eigvec = np.linalg.eig(pore.inertia_tensor)[1]
        eigvals = np.linalg.eig(pore.inertia_tensor)[0]
        anis = 1 - np.min(eigvals)/np.max(eigvals)
        anisotropies.append(anis)
        maxvector = eigvec[:, maxeig]
        orientation = angle_between(maxvector, np.array([0,0,1]))
        orientations.append(orientation)
        phi = angle_between(maxvector, np.array([0,1,0]))
        phis.append(phi)


    stats = {
        'anisotropies': anisotropies,
        'orientations' : orientations,
        'vols' : vols,
        'x_locs' : x_locs,
        'y_locs' : y_locs,
        'z_locs' : z_locs,
        'locations' : locations,
        'maj_axis_l' : maj_axis_l,
        'min_axis_l' : min_axis_l,
        'phis' : phis
    }
```

```
1033     return stats
1034
1035 def plot_pore_fn(idxs,imstack,global_idx = 0, save = False, name = None,
     pore_list = None, voxelsize = 3.49, quantity = 0):
1036
1037         figdir= './NN_analysis/'
1038         if pore_list is None:
1039             pore_list = pores_total
1040         idxs = np.squeeze(np.array([idxs]))
1041         fig = plt.figure()
1042         ax = fig.gca(projection='3d')
1043         maxlimits= []
1044         minlimits = []
1045
1046         if idxs.size == 1:
1047             idx = idxs[0]
1048             start = pore_list[idx].slice[2].start
1049             stop = pore_list[idx].slice[2].stop
1050
1051
1052             meshlist = [np.arange(pore_list[idx].image.shape[0]+1)*voxelsize,
     np.arange(pore_list[idx].image.shape[1]+1)*voxelsize,
     np.arange(pore_list[idx].image.shape[2]+1)*voxelsize]
1053             meshgrid = np.meshgrid(meshlist[0], meshlist[1], meshlist[2],
     indexing = 'ij')
1054             ax.voxels(meshgrid[0]+pore_list[idx].centroid[0]*voxelsize,
     meshgrid[1]+pore_list[idx].centroid[1]*voxelsize,
     meshgrid[2]+pore_list[idx].centroid[2]*voxelsize, pore_list[idx].image,
     edgecolor='k')
1055             ax.set_xlabel(r'x $[\mu m]$')
1056             ax.set_ylabel(r'y $[\mu m]$')
1057             ax.set_zlabel(r'z $[\mu m]$')
1058             plt.title('Pore volume:
     {:.3f}'.format(voxelsize*voxelsize*voxelsize*pore_list[idx].area) + r' [$\mu
     m^3$]' + " Voxel count: " + str(pore_list[idx].area))
1059             plt.title('Aspect Ratio:
     {:3.f}'.format(pore_dataset[i].minor_axis_length/pore_dataset[i].major_axis_l
     ength))
1060             xbound = (pore_list[idx].image.shape[0]+1)*voxelsize
1061             ybound = (pore_list[idx].image.shape[1]+1)*voxelsize
1062             zbound =(pore_list[idx].image.shape[2]+1)*voxelsize
1063
1064
1065             maxlimit = np.max(([xbound, ybound, zbound]))
1066             ax.set_xlim(np.min(minlimits), np.max(maxlimits))
1067             ax.set_ylim(np.min(minlimits), np.max(maxlimits))
1068             ax.set_zlim(np.min(minlimits), np.max(maxlimits))
1069             plt.show()
1070
1071         else:
1072
1073             print("HERE")
1074
1075             xlimits = []
1076             ylimits = []
```

```
1077            zlimits = []
1078            title = ''
1079
1080            start_all = []
1081            stop_all = []
1082            for index in idxs:
1083                start_all.append(pore_list[index].slice[2].start)
1084                stop_all.append(pore_list[index].slice[2].stop)
1085            start = np.min(start_all)
1086            stop = np.max(stop_all)
1087
1088            padded_start = np.max((0, start-3))
1089            padded_stop  = np.min((stop+3, imstack.shape[2]))
1090            for index in range(padded_start, padded_stop):
1091                pore_found = 0
1092                fig_2 = plt.figure()
1093                ax_2 = fig_2.gca()
1094                for idx in idxs:
1095
1096                    ax_2.imshow(imstack[:, :, index])
1097
1098                    if index in range(pore_list[idx].slice[2].start,
    pore_list[idx].slice[2].stop):
1099                        pore_found += 1
1100                        ax_2.set_title('Slice: ' + str(index-start) +
1101                                       " Pore: " + str(idx) + ' Frame:' +
    str(global_idx+index))
1102                        if pore_found % 2 == 0:
1103                            ax_2.annotate(str(idx),
    (pore_list[idx].centroid[1], pore_list[idx].centroid[0]), (
1104                                pore_list[idx].centroid[1] - 30,
    pore_list[idx].centroid[0] - 30), arrowprops=dict(edgecolor='r',
    arrowstyle='-'), color='r')
1105
1106                        else:
1107                            ax_2.annotate(str(idx),
    (pore_list[idx].centroid[1], pore_list[idx].centroid[0]), (
1108                                pore_list[idx].centroid[1] + 30,
    pore_list[idx].centroid[0] + 30), arrowprops=dict(edgecolor='r',
    arrowstyle='-'), color='r')
1109
1110
1111                    else:
1112                        print(index, idx)
1113
1114                if not pore_found:
1115                    ax_2.set_title('Slice: ' + str(index-start) + " Pore:
    none" + ' Frame:' + str(global_idx+index))
1116                print(pore_found)
1117                fig_2.savefig(name + '/frame' + str(global_idx+index)+
    'slice' + str(index-start) + '.png')
1118                pore_found = 0
1119
1120            for idx in idxs:
1121
```

```
1122                 meshlist =
     [np.arange(pore_list[idx].image.shape[0]+1)*voxelsize, np.arange(
1123                     pore_list[idx].image.shape[1]+1)*voxelsize,
     np.arange(global_idx,global_idx+ pore_list[idx].image.shape[2]+1)*voxelsize]
1124                 meshgrid = np.meshgrid(
1125                     meshlist[0], meshlist[1], meshlist[2], indexing='ij')
1126                 ax.voxels(meshgrid[0]+pore_list[idx].centroid[0]*voxelsize,
     meshgrid[1]+pore_list[idx].centroid[1]
1127                               * voxelsize,
     meshgrid[2]+pore_list[idx].centroid[2]*voxelsize, pore_list[idx].image,
     edgecolor='k', label = 'Pore id: ' + str(idx))
1128                 ax.set_xlabel(r'x $[\mu m]$')
1129                 ax.set_ylabel(r'y $[\mu m]$')
1130                 ax.set_zlabel(r'z $[\mu m]$')
1131
1132                 title = 'Anisotropy: {:.3f}'.format(quantity)
1133                 xbound = (pore_list[idx].image.shape[0]+1)*voxelsize
1134                 ybound = (pore_list[idx].image.shape[1]+1)*voxelsize
1135                 zbound = (global_idx+
     pore_list[idx].image.shape[2]+1)*voxelsize
1136                 xlimits.append(pore_list[idx].centroid[0]*voxelsize)
1137                 xlimits.append(
1138                     xbound + pore_list[idx].centroid[0]*voxelsize)
1139                 ylimits.append(pore_list[idx].centroid[1]*voxelsize)
1140                 ylimits.append(
1141                     ybound + pore_list[idx].centroid[1]*voxelsize)
1142                 zlimits.append(global_idx*voxelsize +
     pore_list[idx].centroid[2]*voxelsize)
1143                 zlimits.append(
1144                     zbound + pore_list[idx].centroid[2]*voxelsize)
1145
1146             ax.set_title(title, fontsize = 6)
1147             ax.set_xlim(np.min(xlimits), np.max(xlimits))
1148             ax.set_ylim(np.min(ylimits), np.max(ylimits))
1149             ax.set_zlim(np.min(zlimits), np.max(zlimits))
1150             set_axes_equal(ax)
1151             if save:
1152                 fig.savefig(name + '/3dpore.png')
1153                 plt.clf()
1154             else:
1155                 breakpoint()
1156                 plt.show()
1157                 plt.clf()
1158
1159 def plot_statistics(pore_dataset, voxelsize, k = 0):
1160
1161
1162     thresh = 1
1163
1164     maxvects = []
1165     for thresh in [1.0, 1.5, 2.0, 3.0]:
1166         orientations = []
1167         anisotropies = []
1168         for pore in pore_dataset:
1169
```

```
1170            try:
1171                if pore.major_axis_length/pore.minor_axis_length > thresh:
1172                    print(thresh, 'THRESH')
1173                    inertia_eigval = pore.inertia_tensor_eigvals
1174                    inertia = pore.inertia_tensor
1175                    maxeig = np.argmax(inertia_eigval)
1176                    eigvec = np.linalg.eig(pore.inertia_tensor)[1]
1177                    eigvals = np.linalg.eig(pore.inertia_tensor)[0]
1178                    anis = 1 - np.min(eigvals)/np.max(eigvals)
1179                    anisotropies.append(anis)
1180                    maxvector = eigvec[:, maxeig]
1181                    orientation = angle_between(maxvector, np.array([0,0,1]))
1182                    orientations.append(orientation)
1183
1184            except Exception as e:
1185                print(e)
1186                breakpoint()
1187
1188        histogram = plt.hist((np.array(orientations)/np.pi)*180, density =
    True, bins=30, edgecolor = 'k')
1189        plt.title("Orientation, Pore Sample: " + str(k + 1) + " , threshold =
    " + str(thresh))
1190        plt.xlabel(r"Angle [Degrees]")
1191        plt.ylabel("Probability")
1192        plt.ylim(0, 1.01*np.max(histogram[0]))
1193        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1194        plt.tight_layout()
1195        plt.savefig("orientation" + str(k+ 100) + "_" + str(thresh) + ".png")
1196        plt.clf()
1197        print('done')
1198
1199
1200    try:
1201        locations = [pore.centroid for pore in pore_dataset]
1202        biglocations = [pore.centroid for pore in pore_dataset if pore.area >
    1000]
1203        loc_pores = np.array(locations)
1204        kdtree = spatial.KDTree(locations)
1205        dd, ii = kdtree.query(locations, len(locations))
1206        nearest_neighbor = dd[:,1:]*voxelsize
1207        neighbors = dd[:,1]*voxelsize
1208        edges, hist = np.histogram(dd[:,1:]*voxelsize, bins =
    np.arange(0,800,25)*voxelsize)
1209        plt.plot(hist[:-1],edges, linewidth = 2.0)
1210        plt.title("Nearest Neighbor distance, Pore sample: " + str(k + 1))
1211
1212        plt.xlabel(r"Distance [$\mu m$]")
1213        plt.ylabel("Number of Pores")
1214        plt.tight_layout()
1215        plt.savefig("rdf" + str(k+ 100) + ".png")
1216    except:
1217        breakpoint()
1218
1219
```

```
1220     vols = []
1221     sphericity = []
1222     x_locs = []
1223     y_locs = []
1224     z_locs = []
1225     locations = [pore.centroid for pore in pore_dataset]
1226     maj_axis_l = []
1227     min_axis_l = []
1228     for i in range(len(pore_dataset)):
1229         vols.append(pore_dataset[i]['area']*voxelsize*voxelsize*voxelsize)
1230 
1231         y_locs.append(pore_dataset[i]['centroid'][1]*voxelsize)
1232         x_locs.append(pore_dataset[i]['centroid'][0]*voxelsize)
1233         z_locs.append(pore_dataset[i]['centroid'][2]*voxelsize)
1234         maj_axis_l.append((pore_dataset[i].major_axis_length)*voxelsize)
1235         min_axis_l.append((pore_dataset[i].minor_axis_length)*voxelsize)
1236 
1237     plt.clf()
1238     plt.plot(np.array(vols), np.array(maj_axis_l)/np.array(min_axis_l), '.')
1239     plt.title('Volume correlations with shape')
1240     plt.xlabel('Volume')
1241     plt.xscale('log')
1242     plt.ylabel(r'ratio: Major Axis Length/Minor Axis Length')
1243     plt.tight_layout()
1244     plt.savefig('vols_correlation' + str(k+100)+".png")
1245     plt.clf()
1246 
1247     histogram = plt.hist(neighbors, density = True, bins=100, edgecolor =
     'k')
1248     plt.title("Nearest Neighbor distance, Pore sample: " + str(k + 1))
1249 
1250     plt.xlabel(r"Distance [$\mu m$]")
1251     plt.text(300, 0.01, 'Min distance : {:.2f}'.format(np.min(neighbors)) +
     r'$\mu m$')
1252     plt.ylabel("Probability")
1253     plt.ylim(0, 1.01*np.max(histogram[0]))
1254     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1255     plt.tight_layout()
1256     plt.savefig("nearest" + str(k+ 100) + ".png")
1257     plt.clf()
1258     plt.clf()
1259 
1260     histogram = plt.hist(np.array(maj_axis_l)/np.array(min_axis_l), density =
     True, bins=100, edgecolor = 'k')
1261     plt.title("Eccentricity, Pore sample: " + str(k + 1))
1262 
1263     plt.xlabel(r"Ratio")
1264     plt.ylabel("Probability")
1265     plt.ylim(0, 1.01*np.max(histogram[0]))
1266     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1267     plt.tight_layout()
1268     plt.savefig("ratiominmax" + str(k+ 100) + ".png")
1269     plt.clf()
1270     plt.clf()
1271 
```

```
1272     histogram = plt.hist(min_axis_l, density = True,
     bins=np.logspace(np.log10(10e0),np.log10(10e4)), edgecolor = 'k')
1273     plt.title("Minor Axis Length, Pore sample: " + str(k + 1))
1274     plt.xscale('log')
1275     plt.xlabel(r"Length [$\mu m$]")
1276     plt.ylabel("Probability")
1277     plt.ylim(0, 1.01*np.max(histogram[0]))
1278     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1279     plt.tight_layout()
1280     plt.savefig("minor" + str(k+ 100) + ".png")
1281     plt.clf()
1282     plt.clf()
1283
1284
1285     # # prepare some coordinates
1286
1287     histogram = plt.hist(vols, density = True, alpha =
     0.5,bins=np.logspace(np.log10(10e1),np.log10(10e5)), edgecolor = 'k')
1288     plt.title("Volume, Pore sample: " + str(k + 1))
1289     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1290     redline_x_64 = np.ones(50)*64*voxelsize*voxelsize*voxelsize
1291     plt.plot(redline_x_64, redline_y, 'r', label = '64 voxel count
     threshold')
1292
1293     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1294     redline_x_32 = np.ones(50)*32*voxelsize*voxelsize*voxelsize
1295     plt.plot(redline_x_32, redline_y, 'r--', label = '32 voxel count
     threshold')
1296
1297     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1298     redline_x_16 = np.ones(50)*10*voxelsize*voxelsize*voxelsize
1299     plt.plot(redline_x_16, redline_y, 'r:', label = '10 voxel count
     threshold')
1300
1301
1302     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1303     redline_x_16 = np.ones(50)*4*voxelsize*voxelsize*voxelsize
1304     plt.plot(redline_x_16, redline_y, 'g', label = '4 voxel count threshold')
1305     plt.xscale('log')
1306     plt.xlabel(r"Volume [$\mu m^3$]")
1307     plt.ylabel("Probability")
1308     plt.ylim(0, 1.01*np.max(histogram[0]))
1309     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1310     plt.tight_layout()
1311     plt.savefig("vols_new" + str(k+ 100) + ".png")
1312     plt.clf()
1313
1314
1315
1316
1317
1318 #@profile
1319 def process_folder(k):
1320     print(k)
1321     frame_window = 200 # how many frames to analyze
```

```
1322     rgp_window = 200 # size of regionprops window
1323     shift = 200 # overlap of regionprops window (should be larger than all
     pores)
1324     pores_total = []
1325     problem_snow = 0
1326     neighbors = []
1327     pore_zs = []
1328     def plot_pore(idxs,imstack,global_idx = 0, save = False, name = None,
     pore_list = None):
1329 
1330         figdir= './NN_analysis/'
1331         if pore_list is None:
1332             pore_list = pores_total
1333         idxs = np.squeeze(np.array([idxs]))
1334         fig = plt.figure()
1335         ax = fig.gca(projection='3d')
1336         maxlimits= []
1337         minlimits = []
1338 
1339         if idxs.size == 1:
1340             idx = idxs[0]
1341             start = pore_list[idx].slice[2].start
1342             stop = pore_list[idx].slice[2].stop
1343 
1344 
1345             meshlist = [np.arange(pore_list[idx].image.shape[0]+1)*voxelsize,
     np.arange(pore_list[idx].image.shape[1]+1)*voxelsize,
     np.arange(pore_list[idx].image.shape[2]+1)*voxelsize]
1346             meshgrid = np.meshgrid(meshlist[0], meshlist[1], meshlist[2],
     indexing = 'ij')
1347             ax.voxels(meshgrid[0]+pore_list[idx].centroid[0]*voxelsize,
     meshgrid[1]+pore_list[idx].centroid[1]*voxelsize,
     meshgrid[2]+pore_list[idx].centroid[2]*voxelsize, pore_list[idx].image,
     edgecolor='k')
1348             ax.set_xlabel(r'x $[\mu m]$')
1349             ax.set_ylabel(r'y $[\mu m]$')
1350             ax.set_zlabel(r'z $[\mu m]$')
1351 
1352             plt.title('Pore volume:
     {:.3f}'.format(voxelsize*voxelsize*voxelsize*pore_list[idx].area) + r' [$\mu
     m^3$]' + " Voxel count: " + str(pore_list[idx].area))
1353 
1354             xbound = (pore_list[idx].image.shape[0]+1)*voxelsize
1355             ybound = (pore_list[idx].image.shape[1]+1)*voxelsize
1356             zbound =(pore_list[idx].image.shape[2]+1)*voxelsize
1357 
1358 
1359             maxlimit = np.max(([xbound, ybound, zbound]))
1360             ax.set_xlim(np.min(minlimits), np.max(maxlimits))
1361             ax.set_ylim(np.min(minlimits), np.max(maxlimits))
1362             ax.set_zlim(np.min(minlimits), np.max(maxlimits))
1363             plt.show()
1364 
1365         else:
1366 
```

```
1367            print("HERE")
1368
1369            xlimits = []
1370            ylimits = []
1371            zlimits = []
1372            title = ''
1373
1374            start_all = []
1375            stop_all = []
1376            for index in idxs:
1377                start_all.append(pore_list[index].slice[2].start)
1378                stop_all.append(pore_list[index].slice[2].stop)
1379            start = np.min(start_all)
1380            stop = np.max(stop_all)
1381
1382            padded_start = np.max((0, start-3))
1383            padded_stop  = np.min((stop+3, imstack.shape[2]))
1384            for index in range(padded_start, padded_stop):
1385                pore_found = 0
1386                #slice_idx = index - start_all
1387                fig_2 = plt.figure()
1388                ax_2 = fig_2.gca()
1389                for idx in idxs:
1390
1391                    ax_2.imshow(imstack[:, :, index])
1392                    # ax_2.set_title('Slice: ' + str(index) + " Pore: none" +
    ' Frame:' + str(index))
1393                    if index in range(pore_list[idx].slice[2].start,
    pore_list[idx].slice[2].stop):
1394                        # breakpoint()
1395                        pore_found += 1
1396                        ax_2.set_title('Slice: ' + str(index-start) +
1397                                       " Pore: " + str(idx) + ' Frame:' +
    str(global_idx+index))
1398                        if pore_found % 2 == 0:
1399                            ax_2.annotate(str(idx),
    (pore_list[idx].centroid[1], pore_list[idx].centroid[0]), (
1400                                      pore_list[idx].centroid[1] - 30,
    pore_list[idx].centroid[0] - 30), arrowprops=dict(edgecolor='r',
    arrowstyle='-'), color='r')
1401
1402                        else:
1403                            ax_2.annotate(str(idx),
    (pore_list[idx].centroid[1], pore_list[idx].centroid[0]), (
1404                                      pore_list[idx].centroid[1] + 30,
    pore_list[idx].centroid[0] + 30), arrowprops=dict(edgecolor='r',
    arrowstyle='-'), color='r')
1405
1406
1407
1408                    else:
1409                        print(index, idx)
1410                        # breakpoint()
1411
1412                if not pore_found:
```

```
1413                     ax_2.set_title('Slice: ' + str(index-start) + " Pore:
     none" + ' Frame:' + str(global_idx+index))
1414                 print(pore_found)
1415                 fig_2.savefig(name + '/frame' + str(global_idx+index)+
     'slice' + str(index-start) + '.png')
1416                 pore_found = 0
1417 
1418             for idx in idxs:
1419 
1420                 meshlist =
     [np.arange(pore_list[idx].image.shape[0]+1)*voxelsize, np.arange(
1421                     pore_list[idx].image.shape[1]+1)*voxelsize,
     np.arange(global_idx,global_idx+ pore_list[idx].image.shape[2]+1)*voxelsize]
1422                 meshgrid = np.meshgrid(
1423                     meshlist[0], meshlist[1], meshlist[2], indexing='ij')
1424                 ax.voxels(meshgrid[0]+pore_list[idx].centroid[0]*voxelsize,
     meshgrid[1]+pore_list[idx].centroid[1]
1425                               * voxelsize,
     meshgrid[2]+pore_list[idx].centroid[2]*voxelsize, pore_list[idx].image,
     edgecolor='k', label = 'Pore id: ' + str(idx))
1426                 ax.set_xlabel(r'x $[\mu m]$')
1427                 ax.set_ylabel(r'y $[\mu m]$')
1428                 ax.set_zlabel(r'z $[\mu m]$')
1429             #  breakpoint()
1430                 title += 'Pore volume: {:.3f}'.format(
1431                     voxelsize*voxelsize*voxelsize*pore_list[idx].area) + r'
     [$\mu m^3$]' + " Voxel count: " + str(pore_list[idx].area) + ', '
1432 
1433             #    breakpoint()
1434                 xbound = (pore_list[idx].image.shape[0]+1)*voxelsize
1435                 ybound = (pore_list[idx].image.shape[1]+1)*voxelsize
1436                 zbound = (global_idx+
     pore_list[idx].image.shape[2]+1)*voxelsize
1437                 #breakpoint()
1438                 xlimits.append(pore_list[idx].centroid[0]*voxelsize)
1439                 xlimits.append(
1440                     xbound + pore_list[idx].centroid[0]*voxelsize)
1441                 ylimits.append(pore_list[idx].centroid[1]*voxelsize)
1442                 ylimits.append(
1443                     ybound + pore_list[idx].centroid[1]*voxelsize)
1444                 zlimits.append(global_idx*voxelsize +
     pore_list[idx].centroid[2]*voxelsize)
1445                 zlimits.append(
1446                     zbound + pore_list[idx].centroid[2]*voxelsize)
1447 
1448             ax.set_aspect('equal', adjustable='box')
1449 
1450             ax.set_title(title, fontsize = 6)
1451             ax.set_xlim(np.min(xlimits), np.max(xlimits))
1452             ax.set_ylim(np.min(ylimits), np.max(ylimits))
1453             ax.set_zlim(np.min(zlimits), np.max(zlimits))
1454             set_axes_equal(ax)
1455             if save:
1456                 fig.savefig(name + '/3dpore.png')
1457 
```

```
1458                plt.clf()
1459            else:
1460                breakpoint()
1461                plt.show()
1462                plt.clf()
1463    totlistnum = np.zeros(len(np.arange(0,2000,50)))
1464
1465
1466
1467    for j in range(0, len(os.listdir(subfolders[k])), frame_window ):
1468        voxelsize, imstack,_,_ = load_data(k, num = frame_window, start = j)
1469        imstack[imstack  == 255] = 0
1470        imstack[imstack == 159] = 1
1471        imstack = np.array(imstack, dtype = 'int32')
1472
1473        try:
1474            im = measure.label(imstack[:,:,:])
1475
1476        except Exception as e:
1477            problem_snow += 1
1478            print(e)
1479            breakpoint()
1480
1481        boundaries = []
1482        for i in range(0, frame_window, shift ):
1483            if i+ shift> frame_window:
1484
1485                break
1486            print(im[:,:,i:i+shift].shape)
1487            print("REGIONPROPS")
1488            props = skimage.measure.regionprops(im[:,:,i:i+rgp_window])
1489            im_channels = np.copy(im)
1490            pores_cross = np.unique(im_channels[:,:,0]) + 1
1491            test = np.zeros((im_channels.shape[:2]))
1492            test2 = np.zeros((im_channels.shape[:2]))
1493            test3 = np.zeros((im_channels.shape[:2]))
1494            for p_idx in np.arange(len(props)):
1495                test[im_channels[:,:,0] == props[p_idx].label] =
    props[p_idx].area
1496                test2[im_channels[:,:,0] == props[p_idx].label] =
    props[p_idx].centroid[2]
1497
1498                inertia_eigval =  props[p_idx].inertia_tensor_eigvals
1499                inertia =  props[p_idx].inertia_tensor
1500                maxeig = np.argmax(inertia_eigval)
1501                eigvec = np.linalg.eig(props[p_idx].inertia_tensor)[1]
1502                eigvals = np.linalg.eig(props[p_idx].inertia_tensor)[0]
1503                anis = 1 - np.min(eigvals)/np.max(eigvals)
1504
1505                maxvector = eigvec[:, maxeig]
1506                test3[im_channels[:,:,0] == props[p_idx].label] = anis
1507
1508            print("REGIONPROPS DONE")
1509
1510            endframe = i+shift - 1
```

```
1511            pores_keep = []
1512            windowboundary = rgp_window - shift - 1
1513            windowboundarypores = [prop for prop in props if windowboundary
    in range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that were
    cut off before
1514            alreadycountedpores = [prop for prop in props if windowboundary
    not in range(prop.slice[2].start, prop.slice[2].stop) and prop.centroid[2] <
    windowboundary] # remove pores that were counted before, and were not cut off
1515            pores = [prop for prop in props if endframe not in
    range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that are not
    cut off
1516            boundaries = [prop for prop in props if endframe  in
    range(prop.slice[2].start, prop.slice[2].stop)] # identify pores that are cut
    off
1517
1518
1519            if i > 0:
1520                pores_keep = []
1521                for prop_idx in range(len(props)):
1522                    alreadycount_bool  = False
1523                    boundary_bool = False
1524                    alreadycount_bool =  windowboundary not in
    range(pores[prop_idx].slice[2].start, pores[prop_idx].slice[2].stop) and
    (pores[prop_idx].centroid[2] < windowboundary)
1525                    boundary_bool = endframe  in
    range(pores[prop_idx].slice[2].start, pores[prop_idx].slice[2].stop)
1526
1527                    if not alreadycount_bool and not boundary_bool:
1528
1529                        zlength = pores[prop_idx].slice[2].stop -
    pores[prop_idx].slice[2].start
1530                        xlength = pores[prop_idx].slice[1].stop -
    pores[prop_idx].slice[1].start
1531                        ylength = pores[prop_idx].slice[0].stop -
    pores[prop_idx].slice[0].start
1532
1533                        if xlength > 3 and ylength > 3 and zlength > 3:
1534                            pores_keep.append(pores[ prop_idx])
1535
1536            elif i == 0:
1537                for prop_idx in range(len(pores)):
1538                    zlength = pores[prop_idx].slice[2].stop -
    pores[prop_idx].slice[2].start
1539                    xlength = pores[prop_idx].slice[1].stop -
    pores[prop_idx].slice[1].start
1540                    ylength = pores[prop_idx].slice[0].stop -
    pores[prop_idx].slice[0].start
1541
1542                    if xlength > 3 and ylength > 3 and zlength > 3:
1543
1544                        pores_keep.append(pores[prop_idx])
1545
1546            curr_vols = [pore.area for pore in pores_keep]
1547            smallest  = np.argmin(curr_vols)
1548
```

```
1549            parentdir = 'test_smallest_radius'
1550            if not os.path.isdir(parentdir):
1551                os.mkdir(parentdir)
1552
1553            figdir = parentdir + '/smallest' + str(smallest)
1554            if not os.path.isdir(figdir):
1555                os.mkdir(figdir)
1556
1557            curr_zs = np.array([pore.centroid[2] for pore in pores_keep]) + j
1558            pore_zs.extend(curr_zs)
1559            pores_total.extend(pores_keep)
1560            print(len(pores_total), "length of pores_total")
1561
1562            print(i, i + rgp_window)
1563
1564        print("saving")
1565
1566        pore_dataset = pores_keep
1567
1568        thresh = 1
1569
1570        maxvects = []
1571        for thresh in [1.0, 1.5, 2.0, 3.0]:
1572            orientations = []
1573            anisotropies = []
1574            for pore in pore_dataset:
1575
1576                try:
1577                    if pore.major_axis_length/pore.minor_axis_length >
    thresh:
1578                        print(thresh, 'THRESH')
1579                        inertia_eigval = pore.inertia_tensor_eigvals
1580                        inertia = pore.inertia_tensor
1581                        maxeig = np.argmax(inertia_eigval)
1582                        eigvec = np.linalg.eig(pore.inertia_tensor)[1]
1583                        eigvals = np.linalg.eig(pore.inertia_tensor)[0]
1584                        anis = 1 - np.min(eigvals)/np.max(eigvals)
1585                        anisotropies.append(anis)
1586                        maxvector = eigvec[:, maxeig]
1587                        orientation = angle_between(maxvector,
    np.array([0,0,1]))
1588                        orientations.append(orientation)
1589
1590                except Exception as e:
1591                    print(e)
1592                    breakpoint()
1593            #breakpoint()
1594            histogram = plt.hist((np.array(orientations)/np.pi)*180, density
    = True, bins=30, edgecolor = 'k')
1595            plt.title("Orientation, Pore Sample: " + str(k + 1) + " ,
    threshold = " + str(thresh) + ", z = " + str(j*voxelsize))
1596            plt.xlabel(r"Angle [Degrees]")
1597            plt.ylabel("Probability")
1598            plt.ylim(0, 1.01*np.max(histogram[0]))
1599            plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
```

```
    borderaxespad=0)
1600            plt.tight_layout()
1601            plt.savefig(str(j) + "orientation" + str(k+ 100) + "_" +
    str(thresh) + ".png")
1602            plt.clf()
1603            print('done')
1604        #breakpoint()
1605
1606        try:
1607            locations = [pore.centroid for pore in pore_dataset]
1608            biglocations = [pore.centroid for pore in pore_dataset if
    pore.area > 1000]
1609            loc_pores = np.array(locations)
1610            kdtree = spatial.KDTree(locations)
1611            dd, ii = kdtree.query(locations, len(locations))
1612            nearest_neighbor = dd[:,1:]*voxelsize
1613            neighbors = dd[:,1]*voxelsize
1614            edges, hist = np.histogram(dd[:,1:]*voxelsize, bins =
    np.arange(0,800,25)*voxelsize)
1615            plt.plot(hist[:-1],edges, linewidth = 2.0)
1616            plt.title("Nearest Neighbor distance, Pore sample: " + str(k +
    1)+ ", z = " + str(j*voxelsize))
1617
1618            plt.xlabel(r"Distance [$\mu m$]")
1619            plt.ylabel("Number of Pores")
1620            plt.tight_layout()
1621            plt.savefig(str(j) +  "rdf" + str(k+ 100) + ".png")
1622        except:
1623            breakpoint()
1624
1625
1626        vols = []
1627        sphericity = []
1628        x_locs = []
1629        y_locs = []
1630        z_locs = []
1631        locations = [pore.centroid for pore in pore_dataset]
1632        maj_axis_l = []
1633        min_axis_l = []
1634        for i in range(len(pore_dataset)):
1635            vols.append(pore_dataset[i]
    ['area']*voxelsize*voxelsize*voxelsize)
1636            y_locs.append(pore_dataset[i]['centroid'][1]*voxelsize)
1637            x_locs.append(pore_dataset[i]['centroid'][0]*voxelsize)
1638            z_locs.append(pore_dataset[i]['centroid'][2]*voxelsize)
1639            maj_axis_l.append((pore_dataset[i].major_axis_length)*voxelsize)
1640            min_axis_l.append((pore_dataset[i].minor_axis_length)*voxelsize)
1641
1642        plt.clf()
1643        plt.plot(np.array(vols), np.array(maj_axis_l)/np.array(min_axis_l),
    '.')
1644        plt.title('Volume correlations with shape'+ ", z = " +
    str(j*voxelsize))
1645        plt.xlabel('Volume')
1646        plt.xscale('log')
```

```
1647
1648        plt.ylabel(r'ratio: Major Axis Length/Minor Axis Length')
1649        plt.tight_layout()
1650        plt.savefig(str(j) + 'vols_correlation' + str(k+100)+".png")
1651        plt.clf()
1652
1653        histogram = plt.hist(neighbors, density = True, bins=100, edgecolor =
    'k')
1654        plt.title("Nearest Neighbor distance, Pore sample: " + str(k + 1)+ ",
    z = " + str(j*voxelsize))
1655
1656        plt.xlabel(r"Distance [$\mu m$]")
1657        plt.text(300, 0.01, 'Min distance : {:.2f}'.format(np.min(neighbors))
    + r'$\mu m$')
1658        plt.ylabel("Probability")
1659        plt.ylim(0, 1.01*np.max(histogram[0]))
1660        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1661        plt.tight_layout()
1662        plt.savefig(str(j) + "nearest" + str(k+ 100) + ".png")
1663        plt.clf()
1664        plt.clf()
1665
1666        histogram = plt.hist(np.array(maj_axis_l)/np.array(min_axis_l),
    density = True, bins=100, edgecolor = 'k')
1667        plt.title("Eccentricity, Pore sample: " + str(k + 1)+ ", z = " +
    str(j*voxelsize))
1668        plt.xlabel(r"Ratio")
1669        plt.ylabel("Probability")
1670        plt.ylim(0, 1.01*np.max(histogram[0]))
1671        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1672        plt.tight_layout()
1673        plt.savefig(str(j) + "ratiominmax" + str(k+ 100) + ".png")
1674        plt.clf()
1675        plt.clf()
1676
1677        histogram = plt.hist(min_axis_l, density = True,
    bins=np.logspace(np.log10(10e0),np.log10(10e4)), edgecolor = 'k')
1678        plt.title("Minor Axis Length, Pore sample: " + str(k + 1)+ ", z = " +
    str(j*voxelsize))
1679        plt.xscale('log')
1680        plt.xlabel(r"Length [$\mu m$]")
1681        plt.ylabel("Probability")
1682        plt.ylim(0, 1.01*np.max(histogram[0]))
1683        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1684        plt.tight_layout()
1685        plt.savefig(str(j) + "minor" + str(k+ 100) + ".png")
1686        plt.clf()
1687        plt.clf()
1688
1689
1690        # # prepare some coordinates
1691
```

```
1692
1693        histogram = plt.hist(vols, density = True, alpha =
    0.5,bins=np.logspace(np.log10(10e1),np.log10(10e5)), edgecolor = 'k')
1694        plt.title("Volume, Pore sample: " + str(k + 1)+ ", z = " +
    str(j*voxelsize))
1695        redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1696        redline_x_64 = np.ones(50)*64*voxelsize*voxelsize*voxelsize
1697        plt.plot(redline_x_64, redline_y, 'r', label = '64 voxel count
    threshold')
1698
1699        redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1700        redline_x_32 = np.ones(50)*32*voxelsize*voxelsize*voxelsize
1701        plt.plot(redline_x_32, redline_y, 'r--', label = '32 voxel count
    threshold')
1702
1703        redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1704        redline_x_16 = np.ones(50)*10*voxelsize*voxelsize*voxelsize
1705        plt.plot(redline_x_16, redline_y, 'r:', label = '10 voxel count
    threshold')
1706
1707
1708        redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1709        redline_x_16 = np.ones(50)*4*voxelsize*voxelsize*voxelsize
1710        plt.plot(redline_x_16, redline_y, 'g', label = '4 voxel count
    threshold')
1711        plt.xscale('log')
1712        plt.xlabel(r"Volume [$\mu m^3$]")
1713        plt.ylabel("Probability")
1714        plt.ylim(0, 1.01*np.max(histogram[0]))
1715        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1716        plt.tight_layout()
1717        plt.savefig(str(j) + "vols_new" + str(k+ 100) + ".png")
1718        plt.clf()
1719
1720
1721
1722    pore_dataset = pores_total
1723    thresh = 1
1724
1725    maxvects = []
1726    for thresh in [1.0, 1.5, 2.0, 3.0]:
1727        orientations = []
1728        anisotropies = []
1729        for pore in pore_dataset:
1730
1731            try:
1732                if pore.major_axis_length/pore.minor_axis_length > thresh:
1733                    print(thresh, 'THRESH')
1734                    inertia_eigval = pore.inertia_tensor_eigvals
1735                    inertia = pore.inertia_tensor
1736                    maxeig = np.argmax(inertia_eigval)
1737                    eigvec = np.linalg.eig(pore.inertia_tensor)[1]
1738                    eigvals = np.linalg.eig(pore.inertia_tensor)[0]
1739                    anis = 1 - np.min(eigvals)/np.max(eigvals)
```

```
1740                    anisotropies.append(anis)
1741                    maxvector = eigvec[:, maxeig]
1742                    orientation = angle_between(maxvector, np.array([0,0,1]))
1743                    orientations.append(orientation)
1744
1745            except Exception as e:
1746                print(e)
1747                breakpoint()
1748        #breakpoint()
1749        histogram = plt.hist((np.array(orientations)/np.pi)*180, density =
    True, bins=30, edgecolor = 'k')
1750        plt.title("Orientation, Pore Sample: " + str(k + 1) + " , threshold =
    " + str(thresh) + ", z = " + str(j*voxelsize))
1751        plt.xlabel(r"Angle [Degrees]")
1752        plt.ylabel("Probability")
1753        plt.ylim(0, 1.01*np.max(histogram[0]))
1754        plt.legend(bbox_to_anchor=(1.04,0), loc="lower left",
    borderaxespad=0)
1755        plt.tight_layout()
1756        plt.savefig(str(j) + "orientation" + str(k+ 100) + "_" + str(thresh)
    + ".png")
1757        plt.clf()
1758        print('done')
1759    #breakpoint()
1760
1761    try:
1762        locations = [pore.centroid for pore in pore_dataset]
1763        biglocations = [pore.centroid for pore in pore_dataset if pore.area >
    1000]
1764        loc_pores = np.array(locations)
1765        kdtree = spatial.KDTree(locations)
1766        dd, ii = kdtree.query(locations, len(locations))
1767        nearest_neighbor = dd[:,1:]*voxelsize
1768        neighbors = dd[:,1]*voxelsize
1769        edges, hist = np.histogram(dd[:,1:]*voxelsize, bins =
    np.arange(0,800,25)*voxelsize)
1770        plt.plot(hist[:-1],edges, linewidth = 2.0)
1771        plt.title("Nearest Neighbor distance, Pore sample: " + str(k + 1)+ ",
    z = " + str(j*voxelsize))
1772
1773        plt.xlabel(r"Distance [$\mu m$]")
1774        plt.ylabel("Number of Pores")
1775        plt.tight_layout()
1776        plt.savefig(str(j) +  "rdf" + str(k+ 100) + ".png")
1777    except:
1778        breakpoint()
1779
1780
1781    vols = []
1782    sphericity = []
1783    x_locs = []
1784    y_locs = []
1785    z_locs = []
1786    locations = [pore.centroid for pore in pore_dataset]
1787    maj_axis_l = []
```

```
1788     min_axis_l = []
1789     for i in range(len(pore_dataset)):
1790         vols.append(pore_dataset[i]['area']*voxelsize*voxelsize*voxelsize)
1791     #  sphericity.append(props[i]['sphericity'])
1792         y_locs.append(pore_dataset[i]['centroid'][1]*voxelsize)
1793         x_locs.append(pore_dataset[i]['centroid'][0]*voxelsize)
1794         z_locs.append(pore_dataset[i]['centroid'][2]*voxelsize)
1795         maj_axis_l.append((pore_dataset[i].major_axis_length)*voxelsize)
1796         min_axis_l.append((pore_dataset[i].minor_axis_length)*voxelsize)
1797 
1798     plt.clf()
1799     plt.plot(np.array(vols), np.array(maj_axis_l)/np.array(min_axis_l), '.')
1800     plt.title('Volume correlations with shape'+ ", z = " + str(j*voxelsize))
1801     plt.xlabel('Volume')
1802     plt.xscale('log')
1803 # breakpoint()
1804     plt.ylabel(r'ratio: Major Axis Length/Minor Axis Length')
1805     plt.tight_layout()
1806     plt.savefig(str(j) + 'vols_correlation' + str(k+100)+".png")
1807     plt.clf()
1808 
1809     histogram = plt.hist(neighbors, density = True, bins=100, edgecolor =
     'k')
1810     plt.title("Nearest Neighbor distance, Pore sample: " + str(k + 1)+ ", z =
     " + str(j*voxelsize))
1811 
1812     plt.xlabel(r"Distance [$\mu m$]")
1813     plt.text(300, 0.01, 'Min distance : {:.2f}'.format(np.min(neighbors)) +
     r'$\mu m$')
1814     plt.ylabel("Probability")
1815     plt.ylim(0, 1.01*np.max(histogram[0]))
1816     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1817     plt.tight_layout()
1818     plt.savefig(str(j) + "nearest" + str(k+ 100) + ".png")
1819     plt.clf()
1820     plt.clf()
1821 #    breakpoint()
1822 
1823 
1824 
1825     histogram = plt.hist(np.array(maj_axis_l)/np.array(min_axis_l), density =
     True, bins=100, edgecolor = 'k')
1826     plt.title("Eccentricity, Pore sample: " + str(k + 1)+ ", z = " +
     str(j*voxelsize))
1827     plt.xlabel(r"Ratio")
1828     plt.ylabel("Probability")
1829     plt.ylim(0, 1.01*np.max(histogram[0]))
1830     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1831     plt.tight_layout()
1832     plt.savefig(str(j) + "ratiominmax" + str(k+ 100) + ".png")
1833     plt.clf()
1834     plt.clf()
1835 
1836 
1837 
```

```
1838
1839     histogram = plt.hist(min_axis_l, density = True,
     bins=np.logspace(np.log10(10e0),np.log10(10e4)), edgecolor = 'k')
1840     plt.title("Minor Axis Length, Pore sample: " + str(k + 1)+ ", z = " +
     str(j*voxelsize))
1841     plt.xscale('log')
1842     plt.xlabel(r"Length [$\mu m$]")
1843     plt.ylabel("Probability")
1844     plt.ylim(0, 1.01*np.max(histogram[0]))
1845     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1846     plt.tight_layout()
1847     plt.savefig(str(j) + "minor" + str(k+ 100) + ".png")
1848     plt.clf()
1849     plt.clf()
1850
1851
1852     histogram = plt.hist(vols, density = True, alpha =
     0.5,bins=np.logspace(np.log10(10e1),np.log10(10e5)), edgecolor = 'k')
1853     plt.title("Volume, Pore sample: " + str(k + 1)+ ", z = " +
     str(j*voxelsize))
1854     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1855     redline_x_64 = np.ones(50)*64*voxelsize*voxelsize*voxelsize
1856     plt.plot(redline_x_64, redline_y, 'r', label = '64 voxel count
     threshold')
1857
1858     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1859     redline_x_32 = np.ones(50)*32*voxelsize*voxelsize*voxelsize
1860     plt.plot(redline_x_32, redline_y, 'r--', label = '32 voxel count
     threshold')
1861
1862     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1863     redline_x_16 = np.ones(50)*10*voxelsize*voxelsize*voxelsize
1864     plt.plot(redline_x_16, redline_y, 'r:', label = '10 voxel count
     threshold')
1865
1866
1867     redline_y =  np.linspace(0,2*np.max(histogram[0]), 50)
1868     redline_x_16 = np.ones(50)*4*voxelsize*voxelsize*voxelsize
1869     plt.plot(redline_x_16, redline_y, 'g', label = '4 voxel count threshold')
1870     plt.xscale('log')
1871     plt.xlabel(r"Volume [$\mu m^3$]")
1872     plt.ylabel("Probability")
1873     plt.ylim(0, 1.01*np.max(histogram[0]))
1874     plt.legend(bbox_to_anchor=(1.04,0), loc="lower left", borderaxespad=0)
1875     plt.tight_layout()
1876     plt.savefig(str(j) + "vols_new" + str(k+ 100) + ".png")
1877     plt.clf()
1878
1879 def save_parts(k):
1880     os.makedirs('./analyze_pore_samples/results/pore_examples', exist_ok=
     True)
1881     limit = 2000
1882     index = 0
1883     voxelsize, imstack_test, _, _  = load_data(k, num = limit, start = index)
1884     os.makedirs('./analyze_pore_samples/results/pore_examples
```

```
    /part{}'.format(k), exist_ok = True)
1885    for case in range(100):
1886        plt.imsave('./analyze_pore_samples/results/pore_examples
    /part{}'.format(k) + '/frame{}.png'.format(1000+case), imstack_test[:,:,
    case])
1887        plt.clf()
1888    os.system('convert ./analyze_pore_samples/results/pore_examples
    /part{}/*png ./analyze_pore_samples/results/pore_examples
    /part{}.gif'.format(k,k))
1889    os.system('rm ./analyze_pore_samples/results/pore_examples/part{}
    /*png'.format(k))
1890
1891    return
1892
1893 def main():
1894
1895    parser = argparse.ArgumentParser()
1896
1897    parser.add_argument("-folder", "--folder", dest = 'folder', default =
    "1", help="which folder to load")
1898    args = parser.parse_args()
1899    k = int(args.folder)
1900    if 'analyze_porosity_clean.py' in os.listdir():
1901        os.chdir('..')
1902    for k in range(2,12):
1903        analyze_boundaries(k, 2000)
1904        process_images(k = 0,  frame_window = 200)
1905
1906
1907
1908 if __name__ == "__main__":
1909    main()
1910
```