STANDARD OPERATING PROCEDURE

- CELL SEGMENTATION & SIGNAL ANALYSIS PIPELINE -

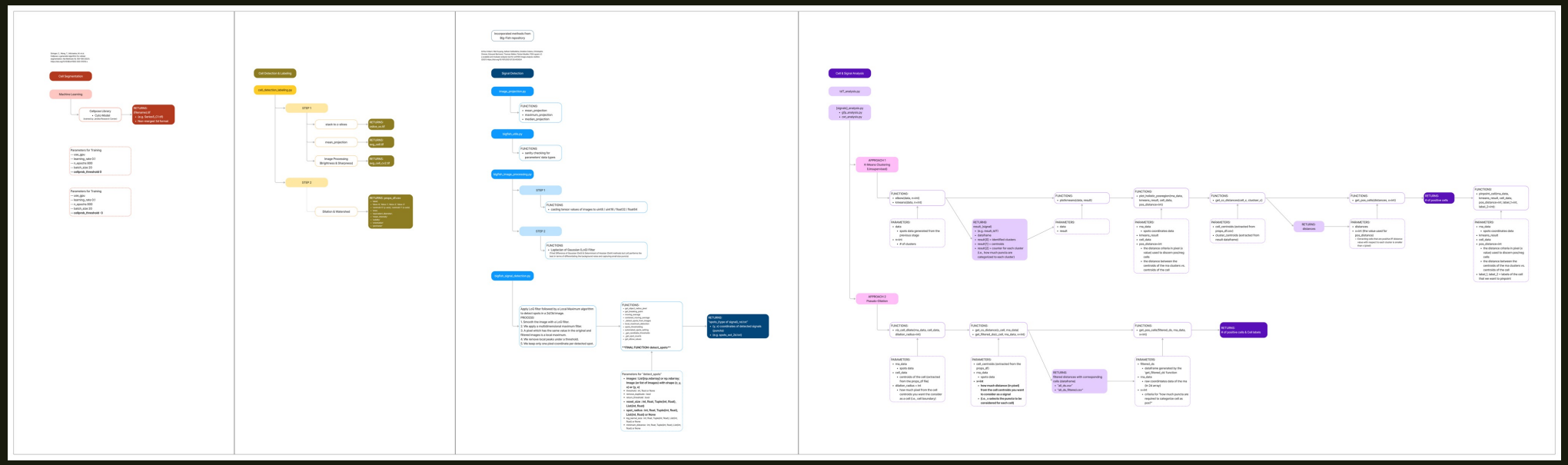
Developed from **Kozorovitskiy Lab**, Lauren Hyoseo Yoon

CONTENTS

FULL FRAMEWORK

The pipeline consists of 4 steps: (1) Cell Segmentation (Fig 1., red), (2) Cell Detection and Labeling (Fig 2., yellow), (3) Signal Detection (Fig 3., blue, and (4) Cell and Signal analysis (Fig 4., purple).

For closed-up images and instructions in each step, please refer to following slides.



(1) (2) (3) (4)

Fig 1. Stage 1

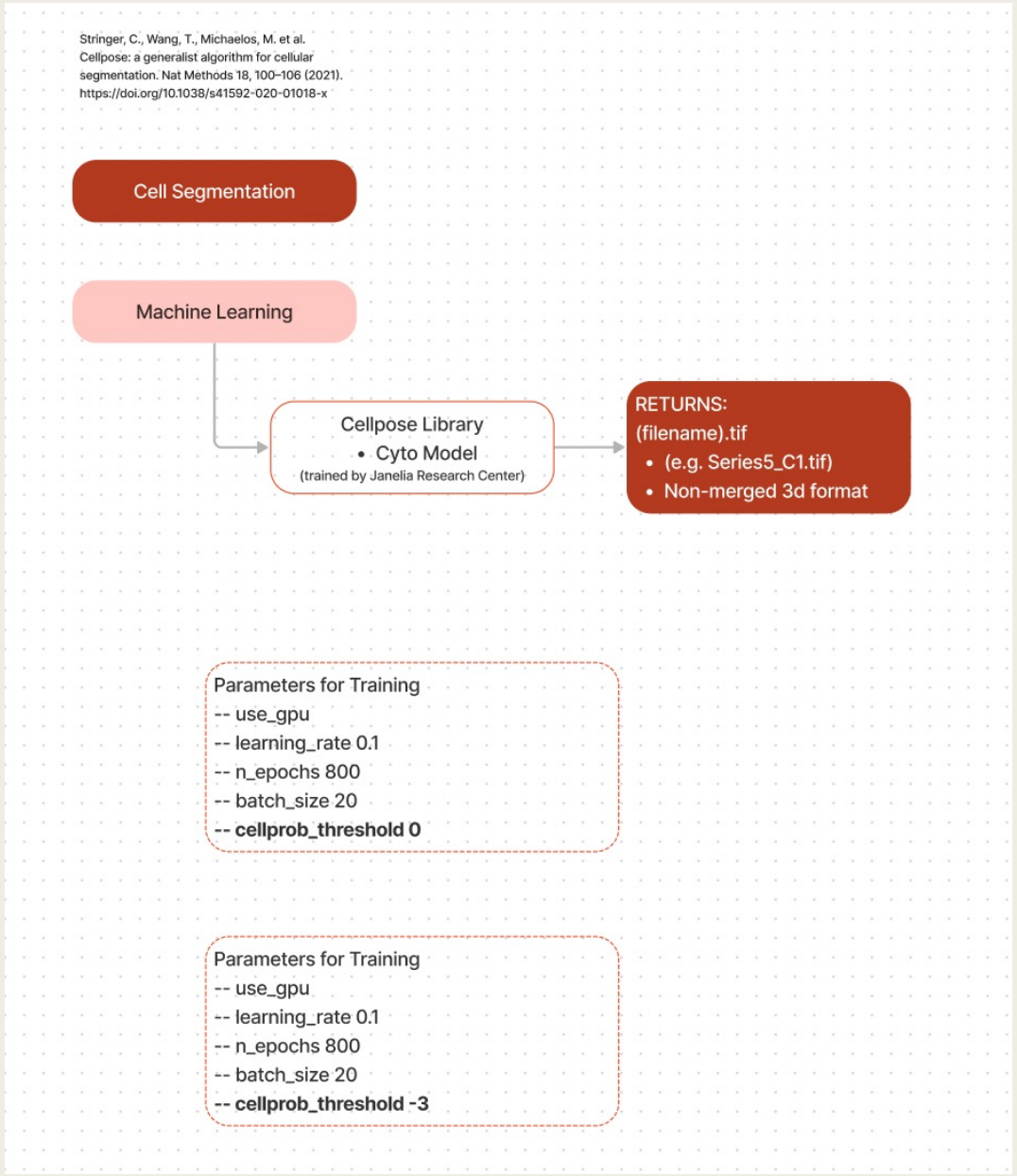

Stringer, C., Wang, T., Michaelos, M. et al.
Cellpose: a generalist algorithm for cellular
segmentation. Nat Methods 18, 100–106 (2021).
https://doi.org/10.1038/s41592-020-01018-x
Cell Segmentation
Machine Learning
Cellpose Library
• Cyto Model
(trained by Janelia Research Center)
RETURNS:
(filename).tif
• (e.g. Series5_C1.tif)
• Non-merged 3d format
Parameters for Training
-- use_gpu
-- learning_rate 0.1
-- n_epochs 800
-- batch_size 20
-- cellprob_threshold 0
Parameters for Training
-- use_gpu
-- learning_rate 0.1
-- n_epochs 800
-- batch_size 20
-- cellprob_threshold -3

Fig 2. Stage 2

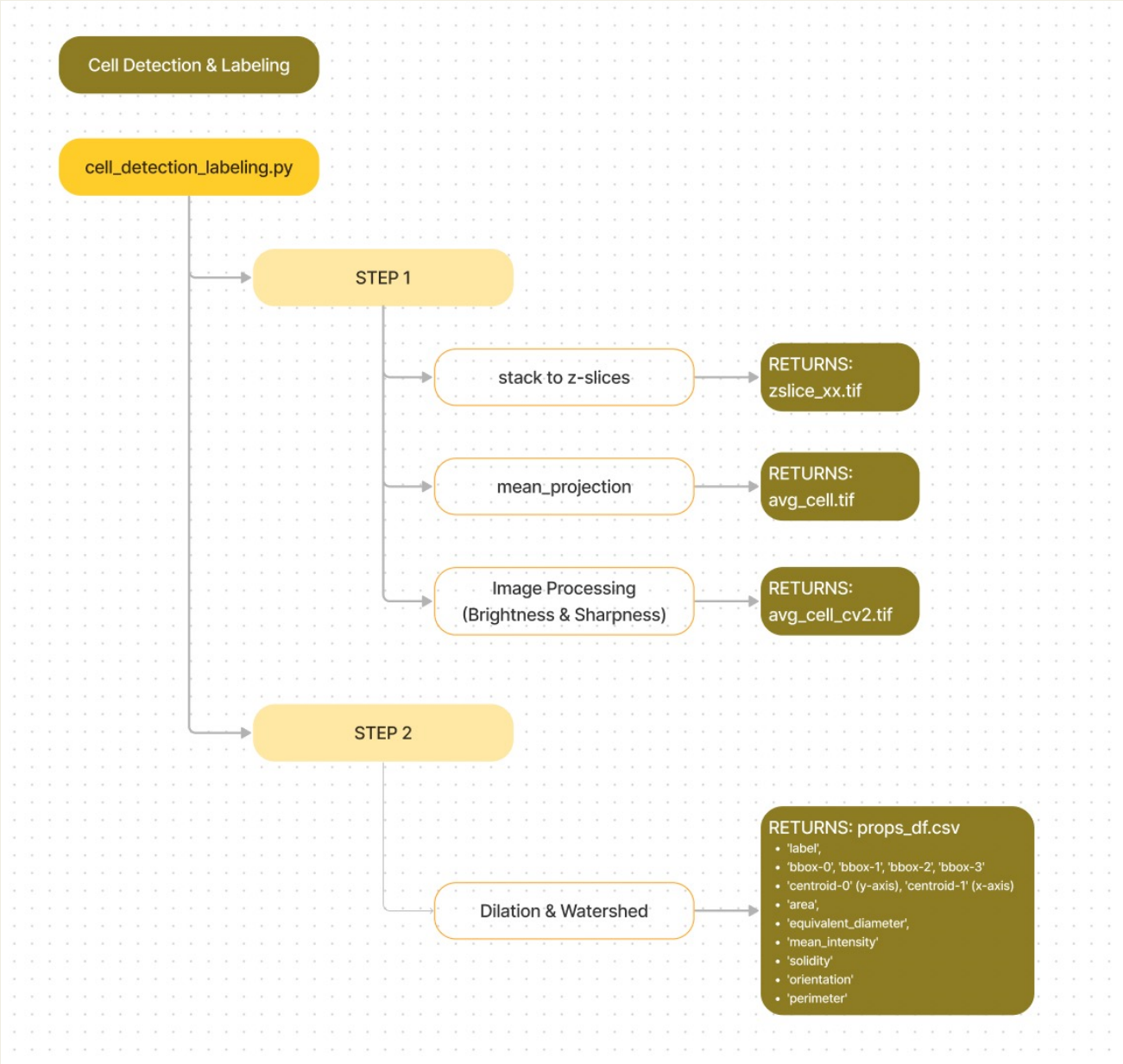

Cell Detection & Labeling
cell_detection_labeling.py
STEP 1
stack to z-slices
RETURNS:
zslice_xx.tif
mean_projection
RETURNS:
avg_cell.tif
Image Processing
(Brightness & Sharpness)
RETURNS:
avg_cell_cv2.tif
STEP 2
Dilation & Watershed
RETURNS: props_df.csv
'label',
'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3'
'centroid-0' (y-axis), 'centroid-1' (x-axis)
'area',
'equivalent_diameter',
'mean_intensity'
'solidity'
'orientation'
'perimeter'

Fig 3. Stage 3

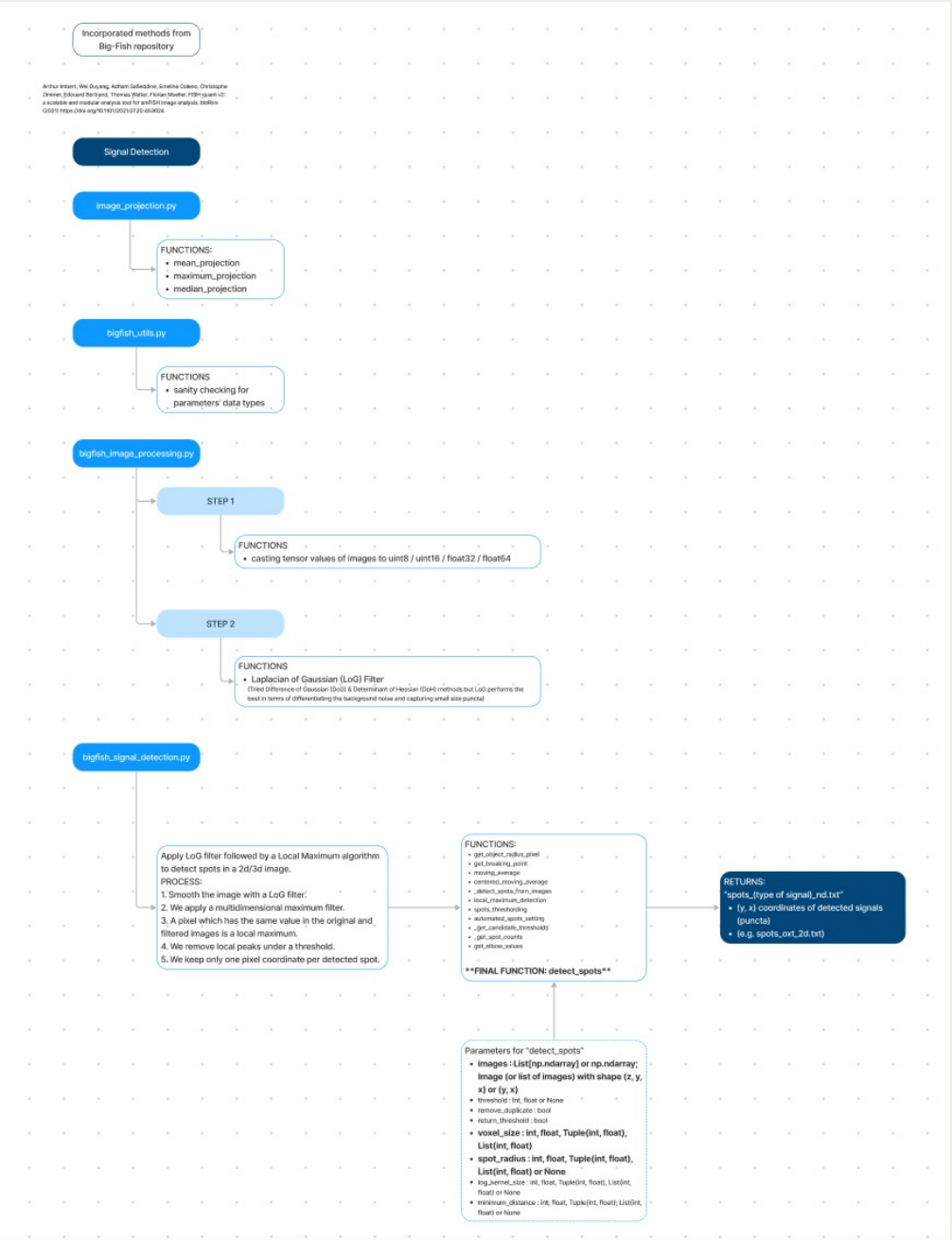
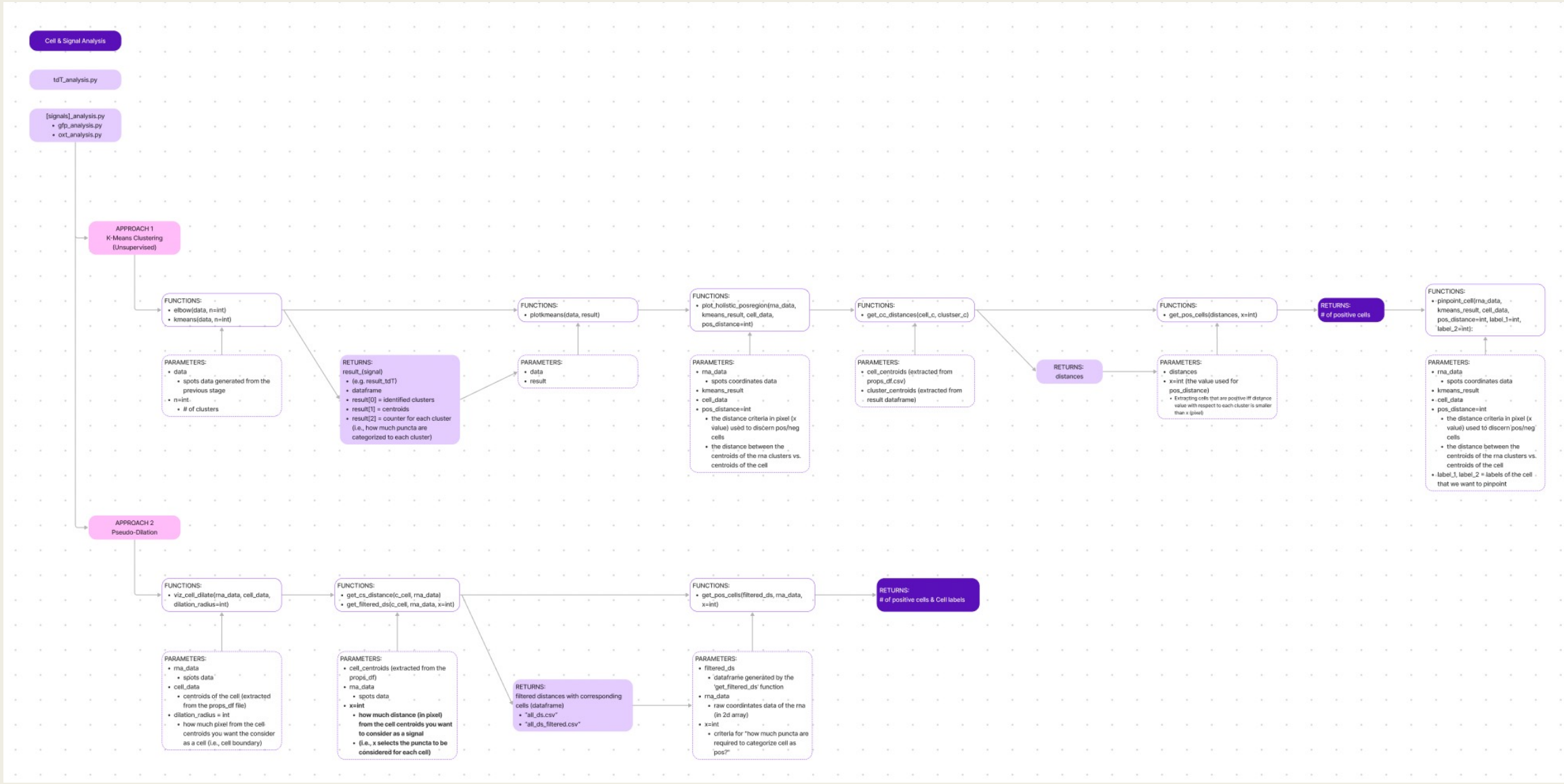

Incorporated methods from Big-Fish repository
Signal Detection
image_projection.py
FUNCTIONS:
• mean_projection
• maximum_projection
• median_projection
bigfish_utils.py
FUNCTIONS
• sanity checking for parameters' data types
bigfish_image_processing.py
STEP 1
FUNCTIONS
• casting tensor values of images to uint8 / uint16 / float32 / float64
STEP 2
FUNCTIONS
• Laplacian of Gaussian (LoG) Filter
bigfish_signal_detection.py
Apply LoG filter followed by a Local Maximum algorithm to detect spots in a 2d/3d image.
PROCESS:
1. Smooth the image with a LoG filter.
2. We apply a multidimensional maximum filter.
3. A pixel which has the same value in the original and filtered images is a local maximum.
4. We remove local peaks under a threshold.
5. We keep only one pixel coordinate per detected spot.
FUNCTIONS:
FINAL FUNCTION: detect_spots
RETURNS:
"spots_{type of signal}_nd.txt"
• (y, x) coordinates of detected signals (puncta)
• (e.g. spots_0xt_2d.txt)
Parameters for "detect_spots"
• images : List[np.ndarray] or np.ndarray; Image (or list of images) with shape (z, y, x) or (y, x)
• threshold : Int, float or None
• remove_duplicate : bool
• return_threshold : bool
• voxel_size : Int, float, Tuple(int, float), List(int, float)
• spot_radius : Int, float, Tuple(int, float), List(int, float) or None
• minimum_distance : int, float, Tuple(int, float); List(int, float) or None

Fig 4. Stage 4


Cell & Signal Analysis
tdT_analysis.py
[signals]_analysis.py
• gfp_analysis.py
• oxt_analysis.py
APPROACH 1
K-Means Clustering
(Unsupervised)
FUNCTIONS:
• elbow(data, n=int)
• kmeans(data, n=int)
PARAMETERS:
• data
• spots data generated from the previous stage
• n=int
• # of clusters
RETURNS:
result_(signal)
• (e.g. result_tdT)
• dataframe
• result[0] = identified clusters
• result[1] = centroids
• result[2] = counter for each cluster (i.e., how much puncta are categorized to each cluster)
FUNCTIONS:
• plotkmeans(data, result)
PARAMETERS:
• data
• result
FUNCTIONS:
• plot_holistic_posregion(rna_data, kmeans_result, cell_data, pos_distance=int)
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
FUNCTIONS:
• get_cc_distances(cell_c, clustser_c)
PARAMETERS:
• cell_centroids (extracted from props_df.csv)
• cluster_centroids (extracted from result dataframe)
RETURNS:
distances
FUNCTIONS:
• get_pos_cells(distances, x=int)
PARAMETERS:
• distances
• x=int (the value used for pos_distance)
• Extracting cells that are positive iff distance value with respect to each cluster is smaller than x (pixel)
RETURNS:
# of positive cells
FUNCTIONS:
• pinpoint_cell(rna_data, kmeans_result, cell_data, pos_distance=int, label_1=int, label_2=int):
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
• label_1, label_2 = labels of the cell that we want to pinpoint
APPROACH 2
Pseudo-Dilation
FUNCTIONS:
• viz_cell_dilate(rna_data, cell_data, dilation_radius=int)
PARAMETERS:
• rna_data
• spots data
• cell_data
• centroids of the cell (extracted from the props_df file)
• dilation_radius = int
• how much pixel from the cell centroids you want the consider as a cell (i.e., cell boundary)
FUNCTIONS:
• get_cs_distance(c_cell, rna_data)
• get_filtered_ds(c_cell, rna_data, x=int)
PARAMETERS:
• cell_centroids (extracted from the props_df)
• rna_data
• spots data
• x=int
• how much distance (in pixel) from the cell centroids you want to consider as a signal
• (i.e., x selects the puncta to be considered for each cell)
RETURNS:
filtered distances with corresponding cells (dataframe)
• "all_ds.csv"
• "all_ds_filtered.csv"
FUNCTIONS:
• get_pos_cells(filtered_ds, rna_data, x=int)
PARAMETERS:
• filtered_ds
• dataframe generated by the 'get_filtered_ds' function
• rna_data
• raw coordintates data of the rna (in 2d array)
• x=int
• criteria for "how much puncta are required to categorize cell as pos?"
RETURNS:
# of positive cells & Cell labels

STAGE 1: Cell Segmentation

Let’s start with the first step of the pipeline: cell segmentation.
Rather than taking the mean projection from the initial stage, I aimed to retain as much as information in the beginning.
So, using the machine learning model, we first segment the cells in 3d. The library that I used is the Cyto model from the cellpose library.
As I tried this model with multiple different parameters, the parameters shown in Fig 1., (refer to the image in the following slide) shows the best performance so far.
Here, I want to highlight the cellprob_threshold parameter which is the parameter that tunes how much dilation of the cells should be done.
The smaller the value, the larger the dilation would be.
I found the value between 0 and -3 be the optimum, but considering the further steps that I will explain, 0 always performs the best because what I aimed for this first step is how much granular and precise the cells are segmented.

1

Stringer, C., Wang, T., Michaelos, M. et al.
Cellpose: a generalist algorithm for cellular segmentation. Nat Methods 18, 100–106 (2021).
https://doi.org/10.1038/s41592-020-01018-x

Cell Segmentation

Machine Learning

Cellpose Library
- Cyto Model

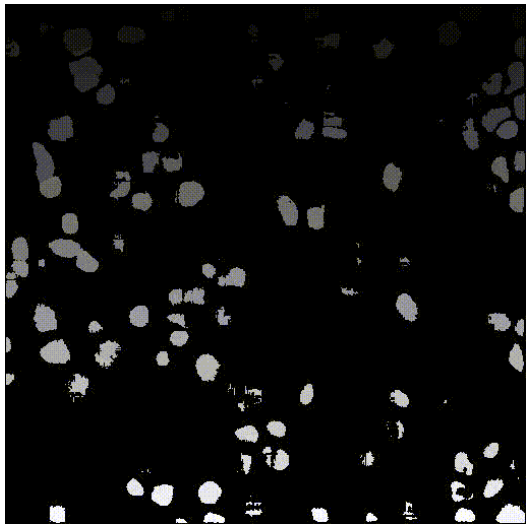
(trained by Janelia Research Center)

RETURNS:
(filename).tif
- (e.g. Series5_C1.tif)
- Non-merged 3d format

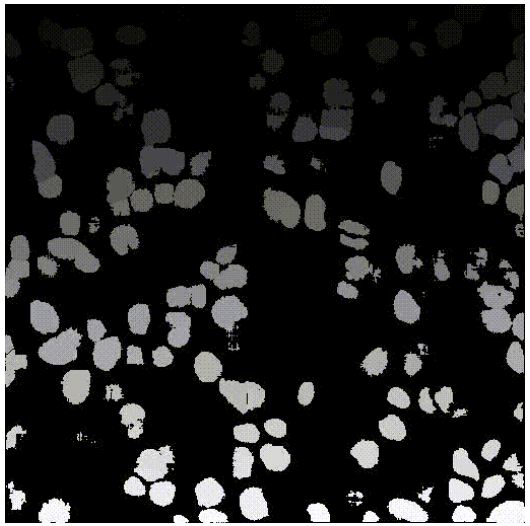
Parameters for Training
-- use_gpu
-- learning_rate 0.1
-- n_epochs 800
-- batch_size 20
-- cellprob_threshold 0

Parameters for Training
-- use_gpu
-- learning_rate 0.1
-- n_epochs 800
-- batch_size 20
-- cellprob_threshold -3

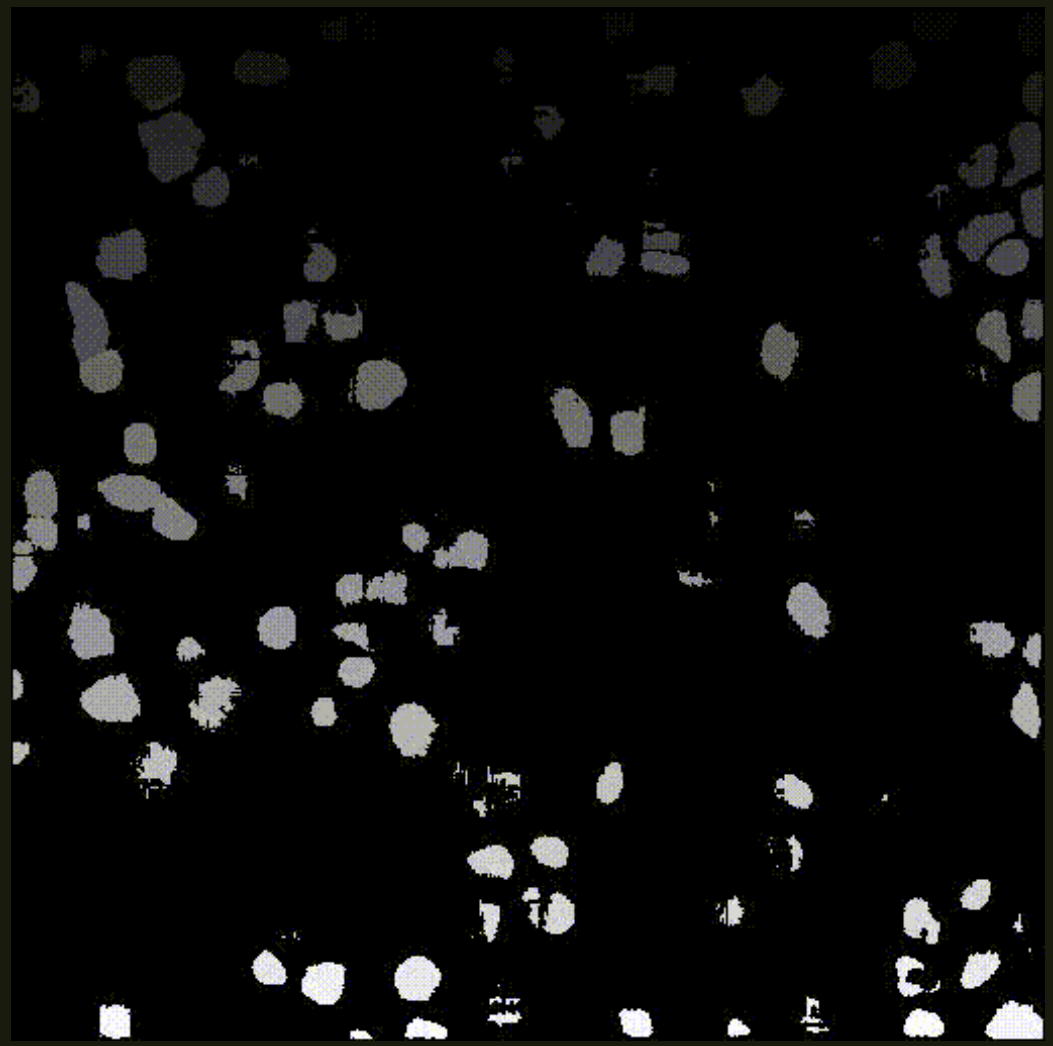
cellprob_threshold 0



cellprob_threshold -3



STAGE 1 OUTPUT



STAGE 2: Cell Detection and Labeling

Moving off to the second step, the main function of the cell_detection_labeling python file is divided into two aspects.

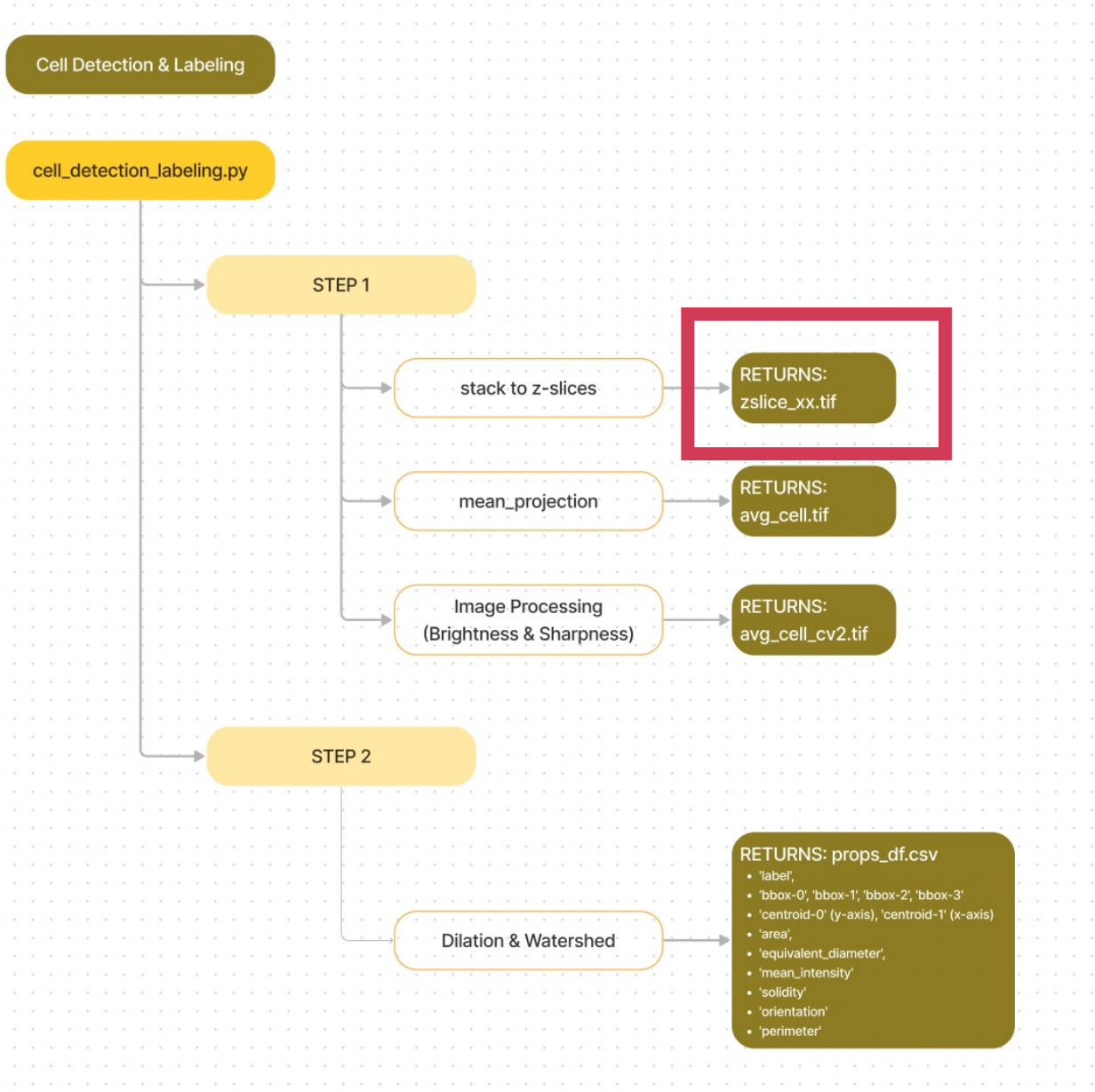
First is the image processing to better detect cells, and the second is the actual detection and labeling of the cells.

What's written in the RETURNS are the files that will automatically be stored in the local directory that you are running this code.

To briefly explain what each output is, the first output generates each z-slice from the segmented image. The second and third output is the mean projection of the z-slices and the image of the processed image which tunes the brightness and sharpness.

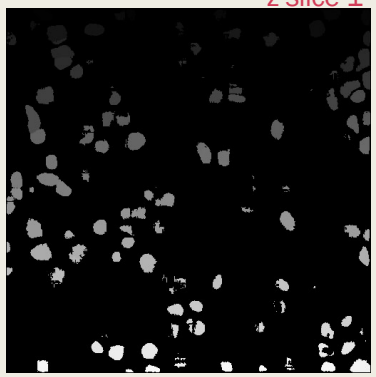
The last output of this second stage is the dataframe of labeled cells and respective information of each cell. How this information is generated is through the dilation and watershed method. This tethers to the reason why in the initial cell segmentation stage, I did not suggest to use smaller values for cellprob_threshold paramter; cells are being dilated in this step. In this cell properties dataframe, the main variables that we would use are centroids-0 and centroids-1 which corresponds to y-axis and x-axis values of the cells.

2

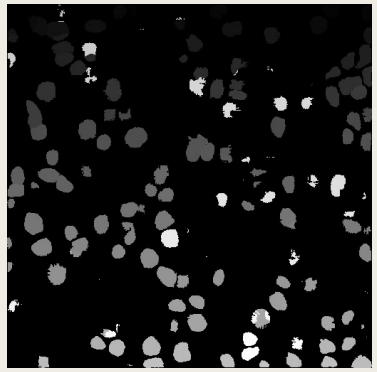

Cell Detection & Labeling
cell_detection_labeling.py
STEP 1
stack to z-slices
RETURNS:
zslice_xx.tif
mean_projection
RETURNS:
avg_cell.tif
Image Processing
(Brightness & Sharpness)
RETURNS:
avg_cell_cv2.tif
STEP 2
Dilation & Watershed
RETURNS: props_df.csv
'label',
'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3'
'centroid-0' (y-axis), 'centroid-1' (x-axis)
'area',
'equivalent_diameter',
'mean_intensity'
'solidity'
'orientation'
'perimeter'

STAGE 2 OUTPUT (1)

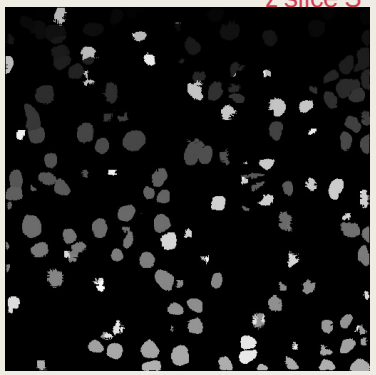
z slice 1



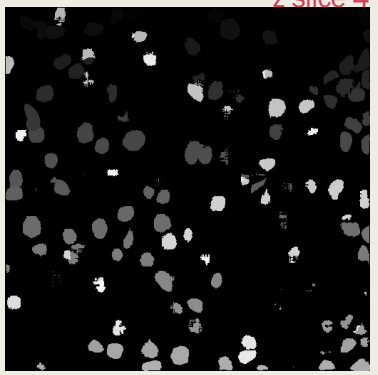
z slice 2



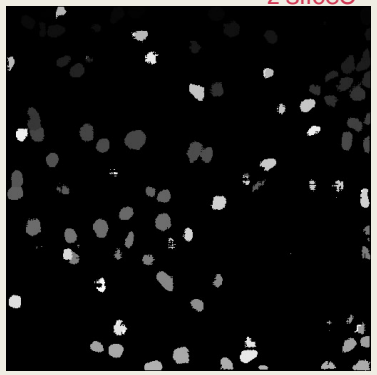
z slice 3



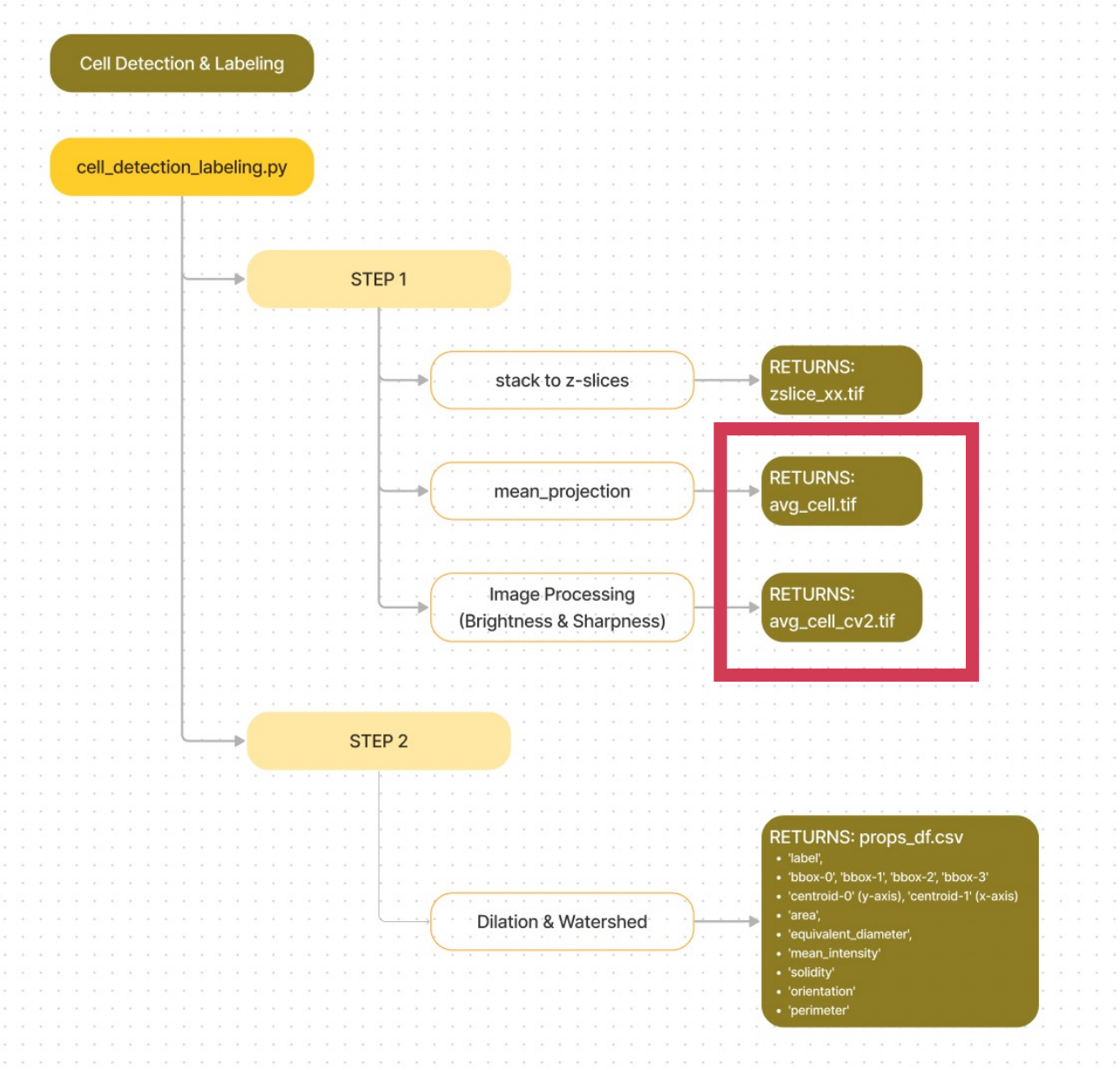
z slice 4



z slice5

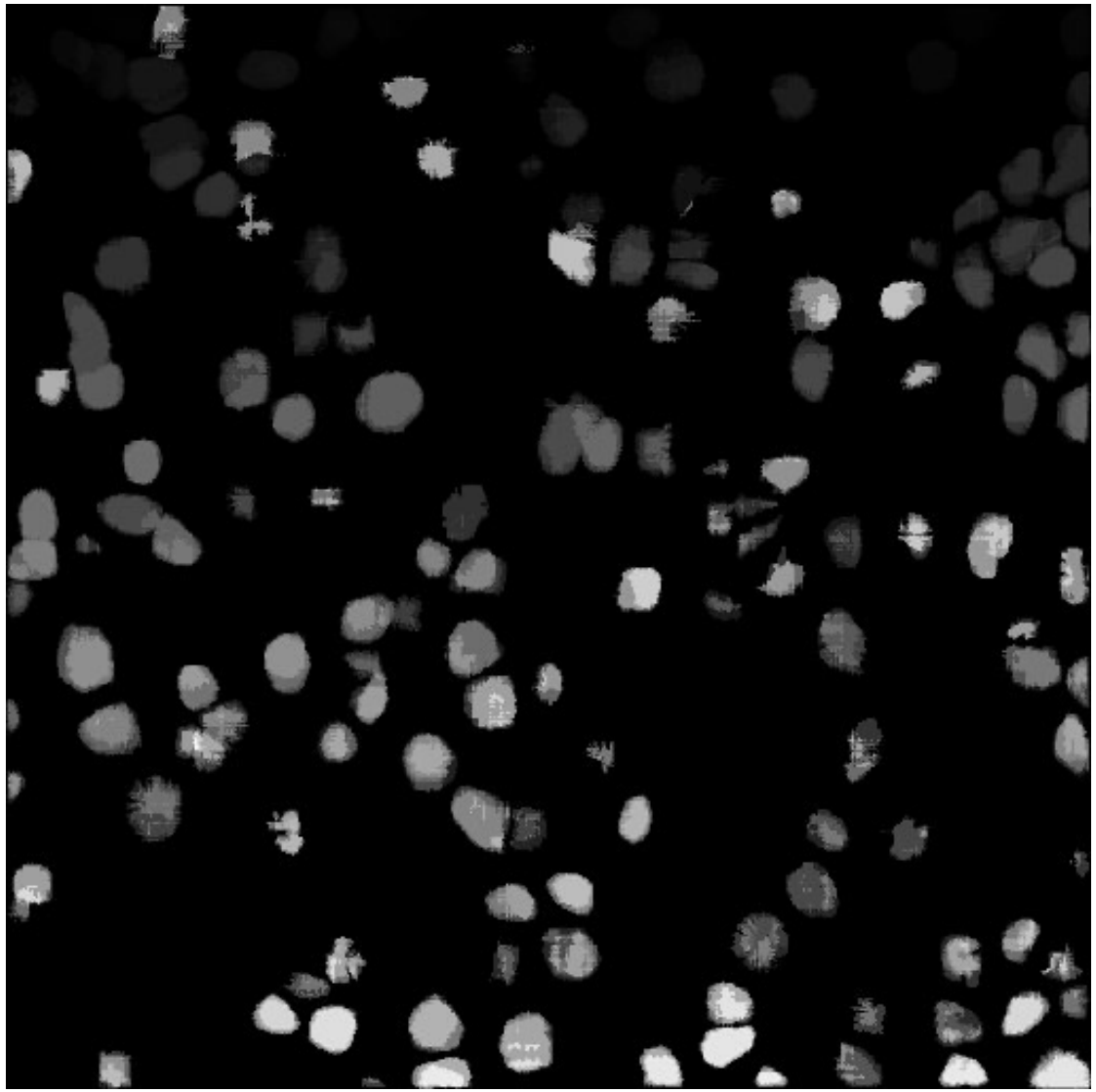


2

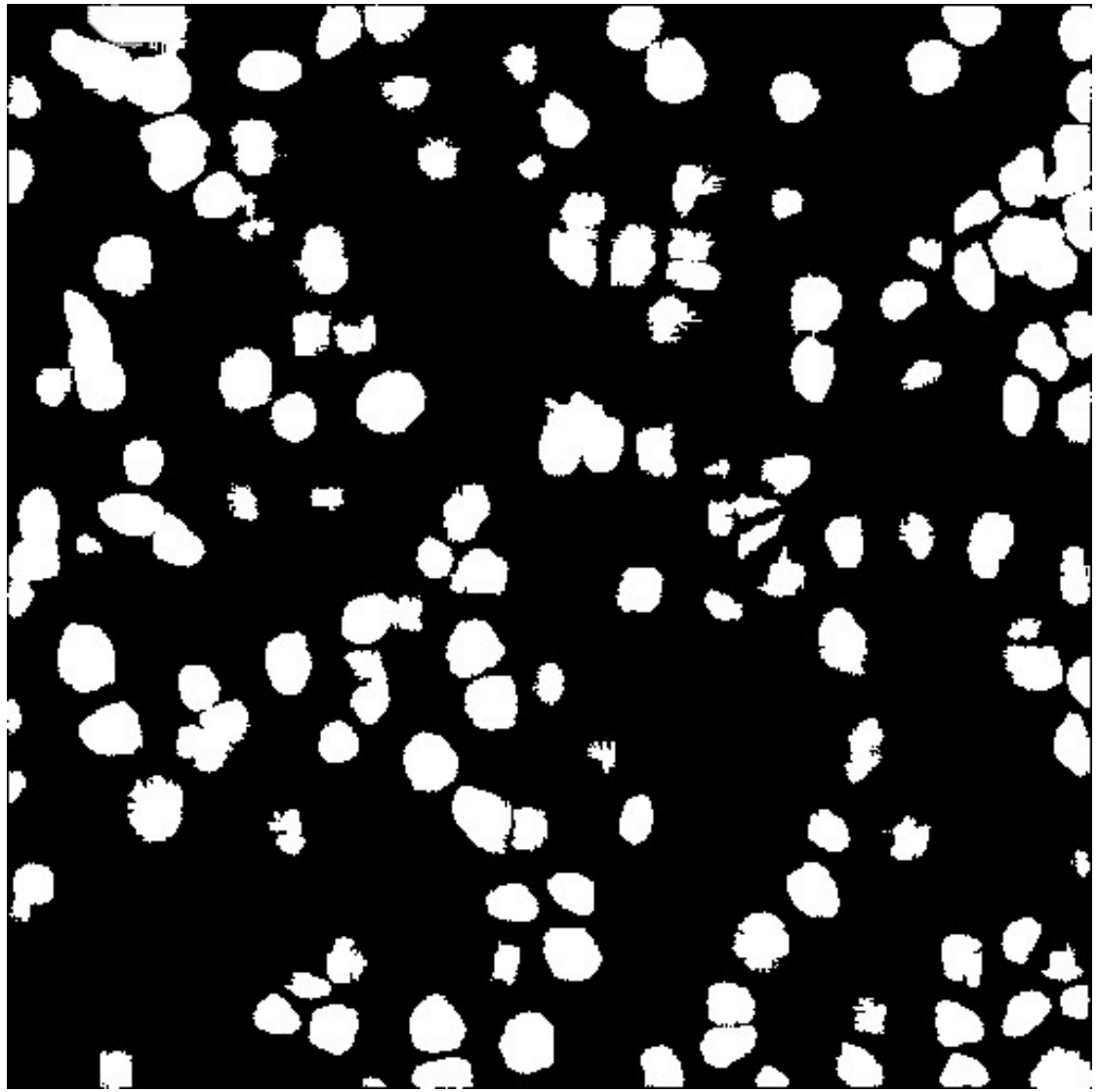

Cell Detection & Labeling
cell_detection_labeling.py
STEP 1
stack to z-slices
RETURNS:
zslice_xx.tif
mean_projection
RETURNS:
avg_cell.tif
Image Processing
(Brightness & Sharpness)
RETURNS:
avg_cell_cv2.tif
STEP 2
Dilation & Watershed
RETURNS: props_df.csv
'label',
'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3'
'centroid-0' (y-axis), 'centroid-1' (x-axis)
'area',
'equivalent_diameter',
'mean_intensity'
'solidity'
'orientation'
'perimeter'

STAGE 2 OUTPUT (2, 3)

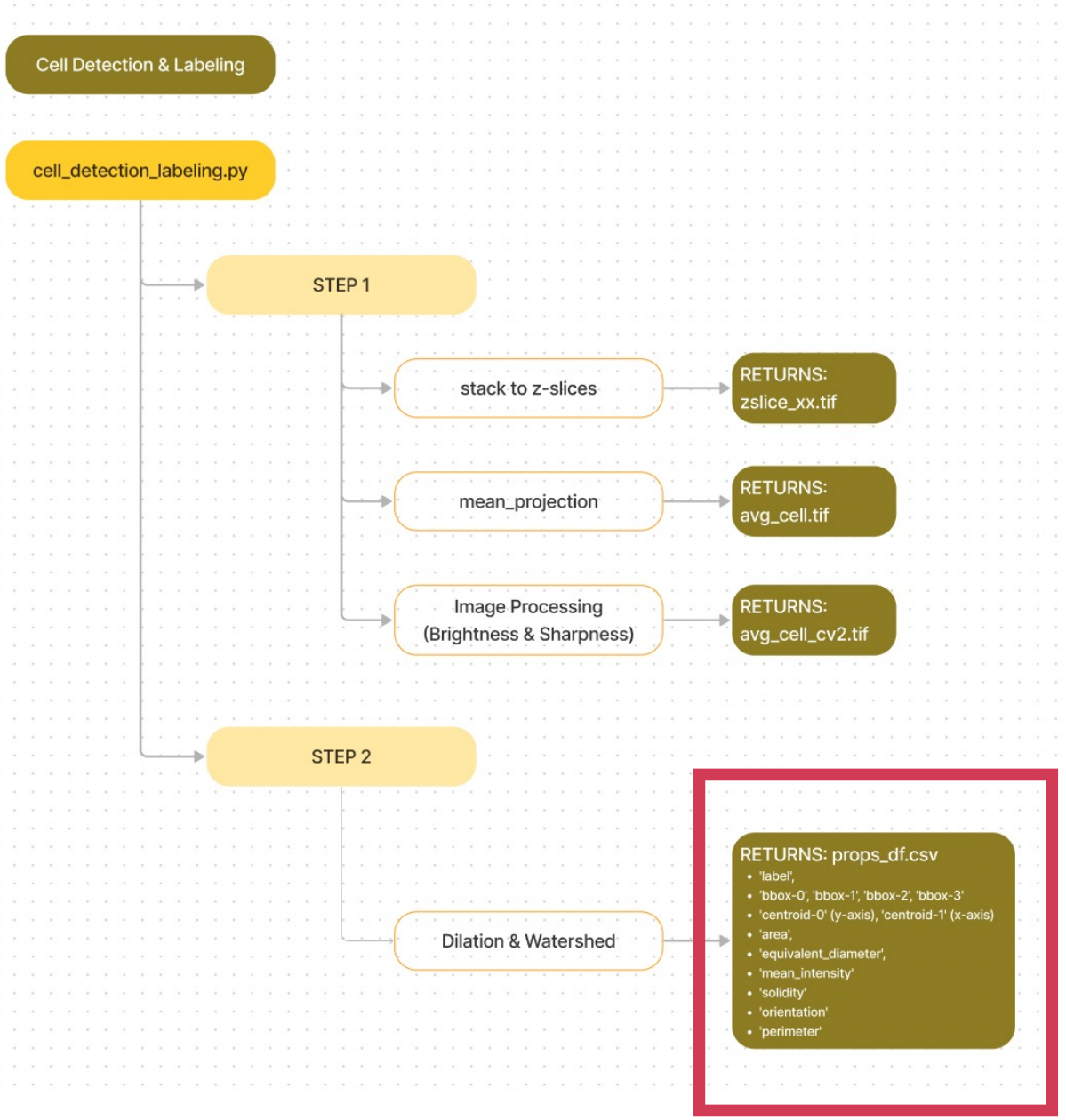
avg_cell.tif



avg_cell_cv2.tif



2


Cell Detection & Labeling
cell_detection_labeling.py
STEP 1
stack to z-slices
RETURNS:
zslice_xx.tif
mean_projection
RETURNS:
avg_cell.tif
Image Processing
(Brightness & Sharpness)
RETURNS:
avg_cell_cv2.tif
STEP 2
Dilation & Watershed
RETURNS: props_df.csv
'label',
'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3'
'centroid-0' (y-axis), 'centroid-1' (x-axis)
'area',
'equivalent_diameter',
'mean_intensity'
'solidity'
'orientation'
'perimeter'

STAGE 2 OUTPUT (4)

props_df

	label	bbox-0	bbox-1	bbox-2	bbox-3	centroid-0	centroid-1	area	equivalent_diameter	mean_intensity	solidity	orientation	perimeter
0	10	1	1	511	511	259.09011582239600	259.488239242513	210063	517.1658520104580	11.909784207594900	0.8076239907727800	-0.8136165963236350	9744.471206285090
1	11	1	38	25	86	10.33177570093460	62.92289719626170	856	33.01352829210020	232.28271028037400	0.9427312775330400	1.4720496106756700	128.3259018078050
2	12	1	146	26	168	11.861650485436900	156.23300970873800	412	22.903595622322900	246.33495145631100	0.9493087557603690	-0.6069368348304550	81.94112549695430
3	13	1	168	19	197	8.092731829573940	181.2531328320800	399	22.539356209735200	247.9699248120300	0.9545454545454550	-1.4745656426086600	80.5269119345812
4	14	1	283	22	310	10.35857461024500	295.6726057906460	449	23.90992587997900	252.16035634743900	0.9676724137931030	-1.3210662591737900	80.42640687119280
5	15	1	442	20	469	8.75794621026895	454.7799511002450	409	22.820056393372100	254.3765281173590	0.964622641509434	1.5375074853195000	76.87005768508880
6	16	12	20	53	87	31.880184331797200	55.00855826201450	1519	43.977845200199500	251.47465437788000	0.8533707865168540	1.236519332525580	190.65180361560900
7	17	16	301	47	331	31.33706293706290	314.6363636363640	715	30.172276587716100	253.93006993007000	0.9457671957671960	0.37868566518787900	104.56854249492400
8	18	17	424	43	450	29.124521072796900	436.53639846743300	522	25.780439141949400	254.51149425287400	0.93048128342246	-0.5148335264978210	88.84062043356590
9	19	22	109	41	139	30.851612903225800	123.39139784946200	465	24.33220886606580	255.0	0.9667359667359670	-1.5504955787736300	81.94112549695430
10	20	32	360	56	384	43.36893203883500	370.96844660194200	412	22.903595622322900	254.38106796116500	0.9342403628117910	0.34335949246460200	77.59797974644670
11	21	34	178	50	199	40.952	188.056	250	17.841241161527700	248.88	0.9328358208955220	-1.4847774921893400	61.35533905932740
12	22	42	251	68	275	55.148491879350300	262.1577726218100	431	23.42576026046660	253.8167053364270	0.9472527472527470	0.5400509712703200	79.01219330881980
13	23	51	63	89	96	68.78099652375430	78.93279258400930	863	33.14823867276280	253.5225955967560	0.9349945828819070	-0.03276190121106780	115.63961030678900
14	24	54	105	81	128	66.7991169977925	116.08830022075100	453	24.016192740836900	251.05960264900700	0.893491124260355	0.15021114914790400	89.49747468305830
15	25	63	194	83	212	73.03065134099620	202.77011494252900	261	18.229523339239500	252.06896551724100	0.9	0.5949543688459810	69.66904755831210
16	26	79	88	101	117	90.46501128668170	100.60722347629800	443	23.74963406702670	252.69751693002300	0.8842315369261480	1.5540056547030800	92.70458146424490
17	27	88	448	108	469	97.03831417624520	457.2068965517240	261	18.229523339239500	255.0	0.9157894736842110	-0.9579289623725040	65.35533905932740
18	28	89	261	109	283	97.97697368421050	271.7006578947370	304	19.67396303746370	249.12828947368400	0.8941176470588240	1.5196147175938800	71.80508652763320
19	29	99	463	135	505	116.21621621621600	484.3914259086670	1073	36.961953837707600	249.53401677539600	0.917094017094017	1.129067375469460	138.36753236814700
20	30	104	137	137	161	120.84283246977500	148.9516407599310	579	27.151532118863200	252.3575129533680	0.9293739967897270	0.09995583908814890	98.5685424949238
21	31	106	256	133	279	118.27593818984500	267.2626931567330	453	24.016192740836900	251.6225165562910	0.9282786885245900	0.5931083134815900	86.66904755831210
22	32	104	284	134	306	118.58266129032300	294.65120967741900	496	25.13019725725690	251.40120967741900	0.906764168190128	-0.2634070046617880	94.32590180780450
23	33	109	41	138	69	122.64469453376200	54.950160771704200	622	28.14169498849120	252.9501607717040	0.9452887537993920	-0.431398800131584000	94.42640687119290
24	34	112	446	145	467	128.43092783505200	456.49484536082500	485	24.849973424463700	247.11340206185600	0.9220532319391640	0.2060197379204260	88.42640687119280
25	35	128	369	157	394	141.19165085389000	380.73055028463	527	25.903614421069300	254.03225806451600	0.9181184668989550	-0.24288004881657300	90.63351365237940
26	36	130	412	150	434	138.81012658227800	422.2151898734180	316	20.0585068271871	255.0	0.9349112426035500	-1.018805117996890	67.94112549695430
27	37	136	27	192	56	165.79408960915200	40.18684461391800	1049	36.54624854109090	253.05529075309800	0.916958041958042	0.2595136669880190	150.46803743153500
28	38	137	302	159	323	147.4655172413790	311.56206896551700	290	19.21560480373170	243.56896551724100	0.8923076923076920	-0.43663467190558200	71.08326112068520
29	39	145	135	166	153	155.07467532467500	143.05194805194800	308	19.802974013476600	249.20454545454500	0.927710843373494	-0.26753682154703800	69.45584412271570
30	40	150	476	178	501	163.67549668874200	487.92273730684300	453	24.016192740836900	253.87417218543000	0.9359504132231410	0.6424338095536510	83.25483399593900
31	41	157	369	188	391	171.97308488612800	379.69565217391300	483	24.798683434954400	255.0	0.9415204678362570	0.0271524190225492 0	84.42640687119290
32	42	162	100	193	125	176.63439065108500	112.45075125208700	599	27.616489409343200	253.7228714524210	0.9523052464228940	-0.2244922077426940	93.42640687119280
33	43	171	14	190	31	179.48706896551700	21.719827586206900	232	17.186959427966200	250.60344827586200	0.90625	-0.6480370665526110	59.698484809835000
34	44	173	165	203	198	187.46164199192500	180.63257065948900	743	30.757389059187500	254.65679676985200	0.9599483204134370	-1.062773564846780	102.5685424949240
35	45	175	470	204	487	188.47869674185500	477.4736842105260	399	22.539356209735200	255.0	0.973170731707317	-0.014856780165621100	75.698484809835

STAGE 3: Signal Detection

Moving on to the third step, it's the signal detection stage.

In this stage, this pipeline incorporated some functions from the big-fish repository.
Going over what each python file does, image_projection file contains functions that allows you to take mean, maximum, or median projection of each channel. Bigfish_utils file consists of generic sanity checking functions which assures whether the data types of the parameters are consistent and are in the correct format.

Next, bigfish image processing file contains the function that casts tensor values of the images to correct data format that can be processed in skimage or opencv. It also contains one of the most integral functions for puncta/blob detection which is the Laplacian of Gaussian (aka LoG) filter. In convention, there are three main filters that are utilized for puncta detection which are LoG, Difference of Gaussian, and Determinant of Hessian methods.
Trying out all these methods, LoG filter always performs the best as it considers each kernel against all local regions of the image.

STAGE 3: Signal Detection

The last step in this third stage is now the actual signal detection.

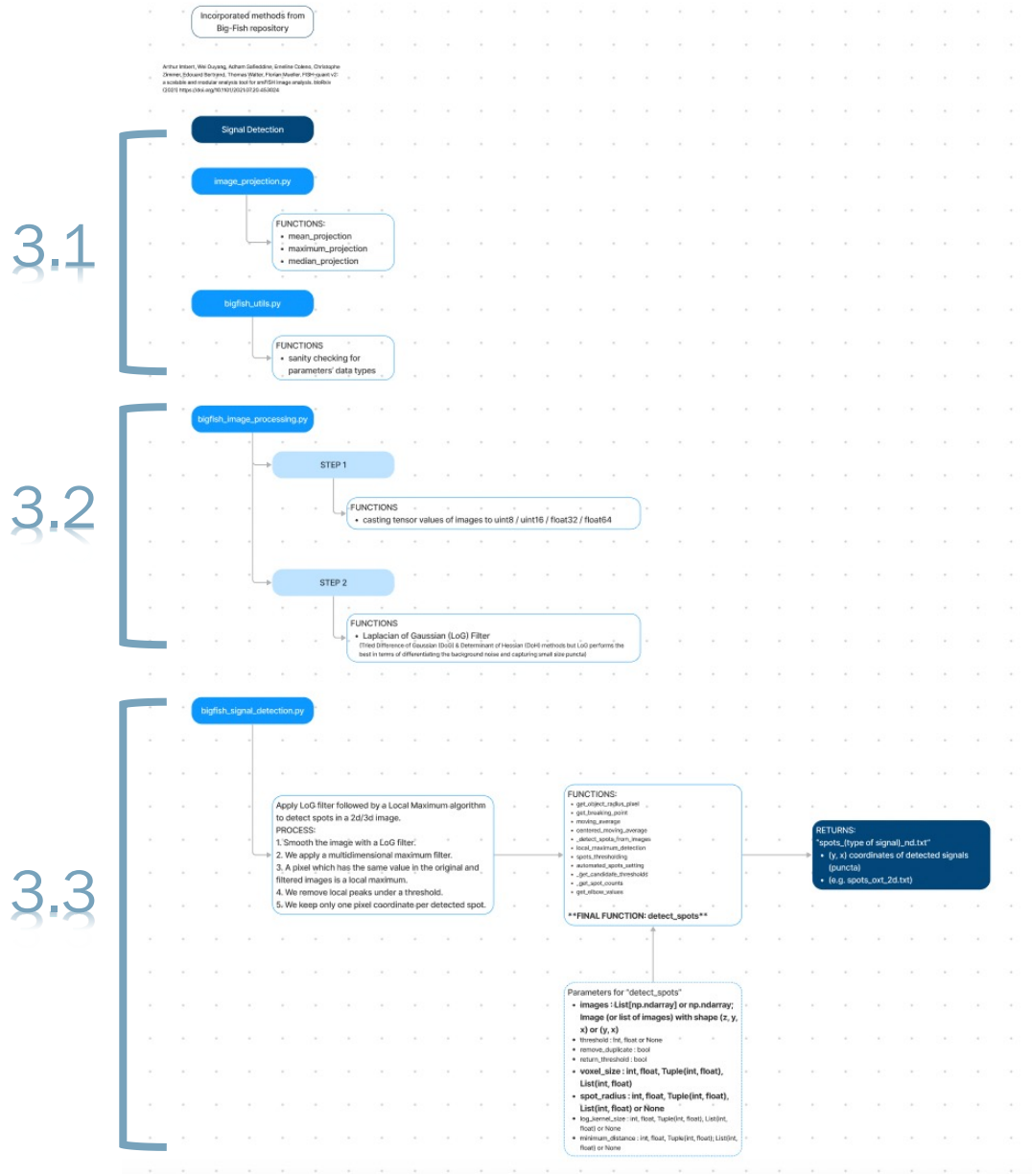
The generic scheme of this signal detection python file is:

1. *Applying the LoG filter*
2. *Applying a multidimensional maximum filter to brighten up the signals*
3. *Comparing the filtered image and the original image and picking up the pixel with a local maximum. -> Here it automatically generates the optimum threshold.*
4. *Removing local peaks under a threshold.*
5. *Last, keeping only one pixel coordinate per detected spot. -> This last step is important if we want to analyze in 3d which ensures that puncta are not detected in the overlapping region across z-slices.*

Given this scheme, there are multiple functions wrapped in this file, but the final function that the user would need to use if the “detect_spots”. The core parameters for this function is the image, voxel size, and the spot radius.

The final output of this stage is array with spots coordinates, and the information will be stored in the txt file in the local directory.

3


Incorporated methods from Big-Fish repository
Arthur Imbert, Wei Ouyang, Adham Safsafeddine, Emeline Culeux, Christophe Zimmer, Edouard Bertrand, Thomas Walter, Florian Mueller. FISH-quant v2: a scalable and modular analysis tool for smFISH image analysis. bioRxiv (2021) https://doi.org/10.1101/2021.07.20.453024
Signal Detection
3.1
image_projection.py
FUNCTIONS:
• mean_projection
• maximum_projection
• median_projection
bigfish_utils.py
FUNCTIONS
• sanity checking for parameters' data types
3.2
bigfish_image_processing.py
STEP 1
FUNCTIONS
• casting tensor values of images to uint8 / uint16 / float32 / float64
STEP 2
FUNCTIONS
• Laplacian of Gaussian (LoG) Filter
(Tried Difference of Gaussian (DoG) & Determinant of Hessian (DoH) methods but LoG performs the best in terms of differentiating the background noise and capturing small size puncta)
3.3
bigfish_signal_detection.py
Apply LoG filter followed by a Local Maximum algorithm to detect spots in a 2d/3d image.
PROCESS:
1. Smooth the image with a LoG filter.
2. We apply a multidimensional maximum filter.
3. A pixel which has the same value in the original and filtered images is a local maximum.
4. We remove local peaks under a threshold.
5. We keep only one pixel coordinate per detected spot.
FUNCTIONS:
• get_object_radius_pixel
• get_breaking_point
• moving_average
• centered_moving_average
• _detect_spots_from_images
• local_maximum_detection
• spots_thresholding
• automated_spots_setting
• _get_candidate_thresholds
• _get_spot_counts
• get_elbow_values
FINAL FUNCTION: detect_spots
RETURNS:
"spots_{type of signal}_nd.txt"
• (y, x) coordinates of detected signals (puncta)
• (e.g. spots_oxt_2d.txt)
Parameters for "detect_spots"
• images : List[np.ndarray] or np.ndarray; Image (or list of images) with shape (z, y, x) or (y, x)
• threshold : int, float or None
• remove_duplicate : bool
• return_threshold : bool
• voxel_size : int, float, Tuple(int, float), List(int, float)
• spot_radius : int, float, Tuple(int, float), List(int, float) or None
• log_kernel_size : int, float, Tuple(int, float), List(int, float) or None
• minimum_distance : int, float, Tuple(int, float); List(int, float) or None

3.1

Incorporated methods from Big-Fish repository

Arthur Imbert, Wei Ouyang, Adham Safieddine, Emeline Coleno, Christophe Zimmer, Edouard Bertrand, Thomas Walter, Florian Mueller. FISH-quant v2: a scalable and modular analysis tool for smFISH image analysis. bioRxiv (2021) https://doi.org/10.1101/2021.07.20.453024

Signal Detection

image_projection.py

FUNCTIONS:
- mean_projection
- maximum_projection
- median_projection

bigfish_utils.py

FUNCTIONS
- sanity checking for parameters' data types

3.2

bigfish_image_processing.py

STEP 1

FUNCTIONS
- casting tensor values of images to uint8 / uint16 / float32 / float64

STEP 2

FUNCTIONS
- Laplacian of Gaussian (LoG) Filter
  (Tried Difference of Gaussian (DoG) & Determinant of Hessian (DoH) methods but LoG performs the best in terms of differentiating the background noise and capturing small size puncta)

3.3

bigfish_signal_detection.py

Apply LoG filter followed by a Local Maximum algorithm to detect spots in a 2d/3d image.
PROCESS:
1. Smooth the image with a LoG filter.
2. We apply a multidimensional maximum filter.
3. A pixel which has the same value in the original and filtered images is a local maximum.
4. We remove local peaks under a threshold.
5. We keep only one pixel coordinate per detected spot.

FUNCTIONS:
- get_object_radius_pixel
- get_breaking_point
- moving_average
- centered_moving_average
- _detect_spots_from_images
- local_maximum_detection
- spots_thresholding
- automated_spots_setting
- _get_candidate_thresholds
- _get_spot_counts
- get_elbow_values

****FINAL FUNCTION: detect_spots****

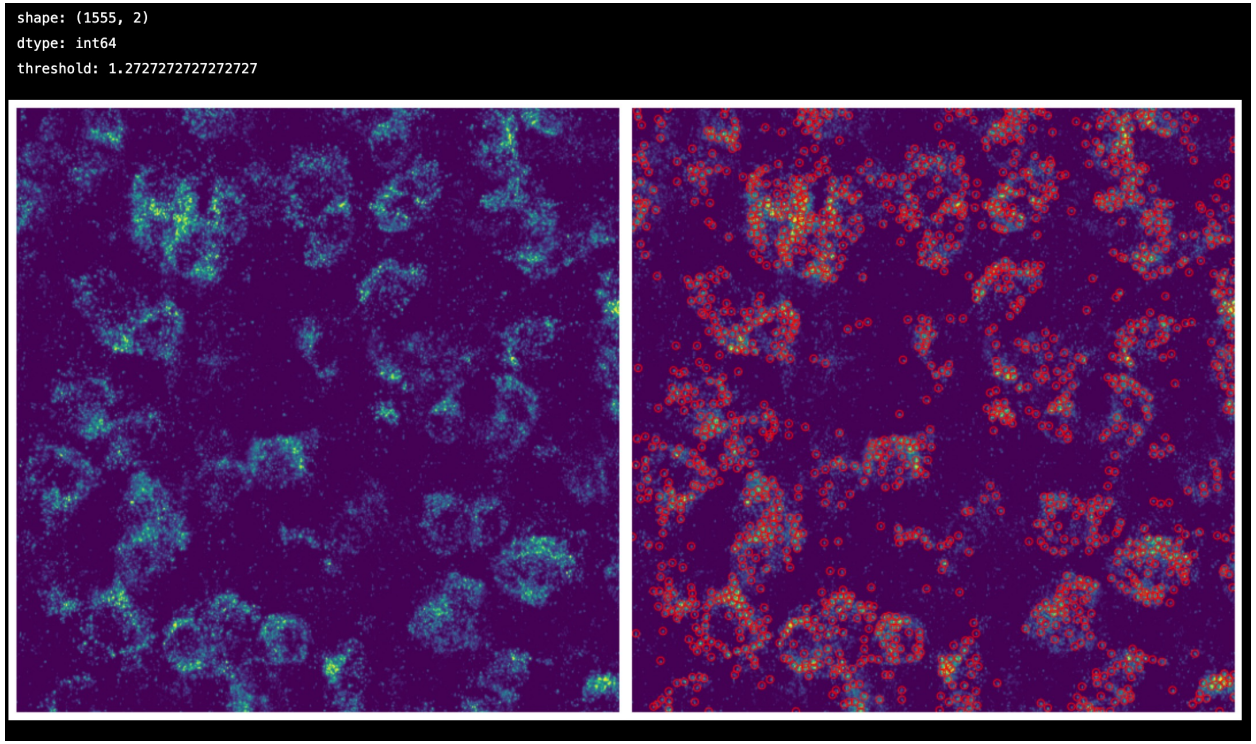
RETURNS:
"spots_(type of signal)_nd.txt"
- (y, x) coordinates of detected signals (puncta)
- (e.g. spots_oxt_2d.txt)

Parameters for "detect_spots"
- **images : List[np.ndarray] or np.ndarray; Image (or list of images) with shape (z, y, x) or (y, x)**
- threshold : int, float or None
- remove_duplicate : bool
- return_threshold : bool
- **voxel_size : int, float, Tuple(int, float), List(int, float)**
- **spot_radius : int, float, Tuple(int, float), List(int, float) or None**
- log_kernel_size : int, float, Tuple(int, float), List(int, float) or None
- minimum_distance : int, float, Tuple(int, float), List(int, float) or None

These are the sample images and file that will be generated. This is the sample data for tdT signals.
Two columns refer to the y, x coordinates of the signals, respectively.

spots_tdT_2d

	0	1
0	0.0	30.0
1	0.0	79.0
2	0.0	147.0
3	0.0	167.0
4	0.0	269.0
5	0.0	278.0
6	0.0	319.0
7	0.0	341.0
8	0.0	403.0
9	0.0	442.0
10	1.0	152.0
11	2.0	4.0
12	2.0	345.0
13	3.0	144.0
14	4.0	155.0
15	3.0	178.0
16	4.0	278.0
17	3.0	285.0
18	3.0	329.0
19	3.0	348.0
20	4.0	408.0
21	5.0	78.0
22	4.0	164.0
23	4.0	320.0
24	4.0	432.0
25	5.0	333.0
26	5.0	441.0
27	6.0	167.0
28	6.0	184.0
29	6.0	353.0
30	6.0	429.0
31	6.0	436.0



STAGE 4: Cell and Signal Analysis

Now moving on to the last stage, this is the stage where the information extracted from the previous stages is merged.

This stage has two approaches that users can choose from. The **first (4.1)** is the k-means clustering approach and the **second (4.2)** is the pseudo-dilation approach.

Keep in mind that the first step would be slightly a non-conventional approach than the second, and this inspiration of this idea is derived during the process of tackling issues with detecting tdT positive cells due to modality of tdT signals (i.e., signals being accumulated around the circular boundary of the cells). This pattern engenders a major issue when we aim to discern cells with positive/negative expression. As we aim to segment cells as accurate as possible, this would yield a cell segmentation result with smaller cell sizes. The smaller the cell sizes are, the less the tdT signals would be counted within the cell boundary. To rectify this issue, the first approach – k-means clustering approach – in Stage 4 is created.

STAGE 4: Cell and Signal Analysis – Approach 1

The first functions that you would see in the python file would be the *elbow* and *kmeans* function.
The parameters for both of these functions are (1) spots coordinates data in array that would automatically be stored in a txt file in your local directory from the previous third signal detection stage and (2) the number of clusters.

But for n in the elbow function, you may want to put bigger number since you want to know where the breaking point and get the optimum number of clusters. Now using the graph generated from the *elbow* function, you can put the *n* value with the breaking point that you can see in the graph for the *kmeans* function. But normally you would want to put larger value to detect more circular regions.

The *kmeans* function will generate the result file which contains the identified clusters, centroids, and the counter for each cluster which is how many puncta are categorized to each cluster.

You can plot this result using *plotkmeans* and *plot_holistic_posregion*. But *plot_holistic_posregion* has one more important parameter which is the *pos_distance*. This is the integer value that you simulates the distance in pixel used to discern whether the cells would be categorized as positive or negative. This distance value is in other words the distance between the centroids of the cluster and the centroids of the cell.

STAGE 4: Cell and Signal Analysis – Approach 1

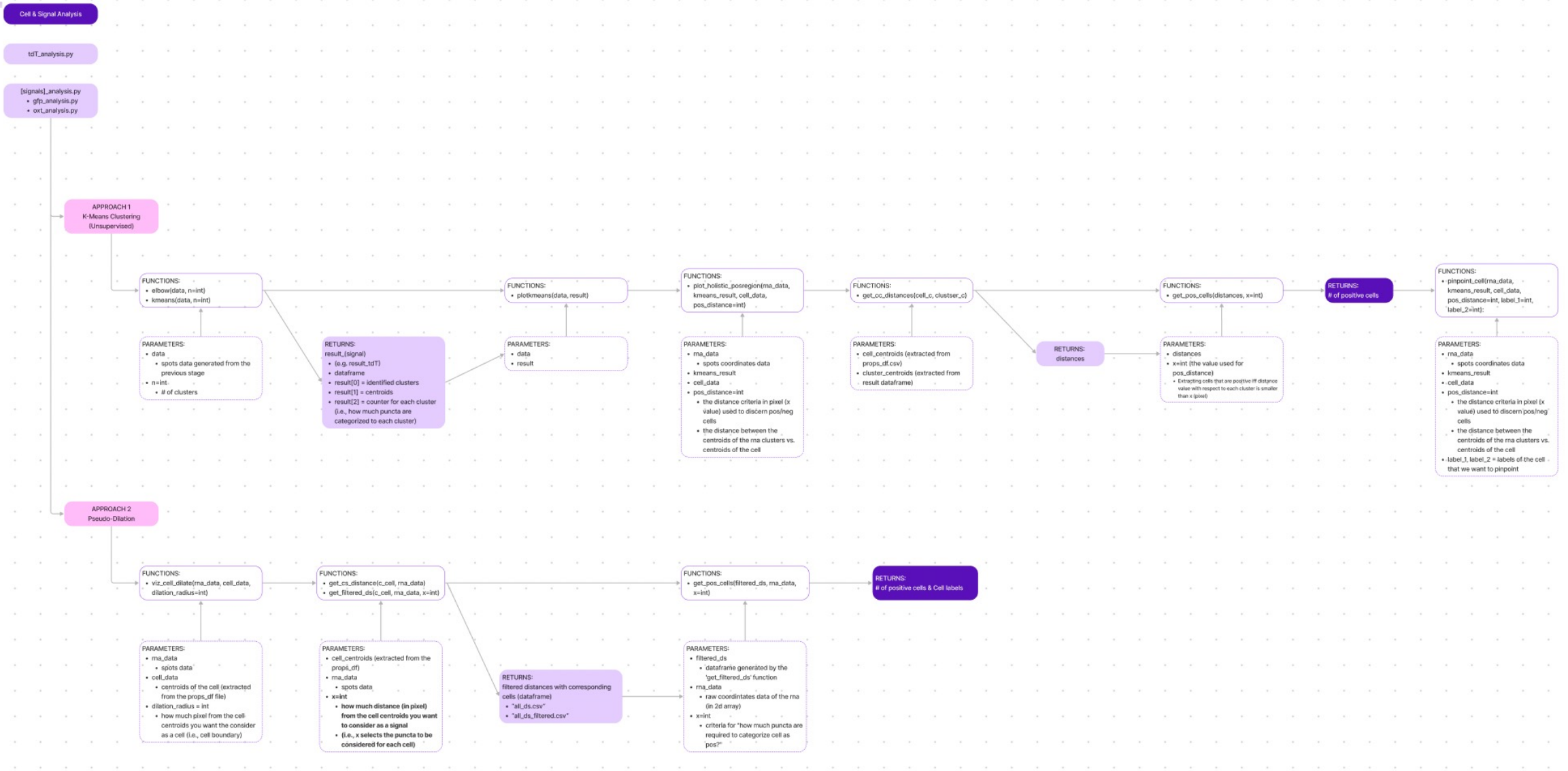
The last step of this first approach would now be getting how much positive cells would there be.

This would be generated by running *get_cc_distances* function and *get_pos_cells* function using the parameters shown here. The most important parameter here would be the *x* value in the *get_pos_cells* function which is the same as the *pos_distance* value that we used in *plot_holistic_posregion* function shown in the previous slide.

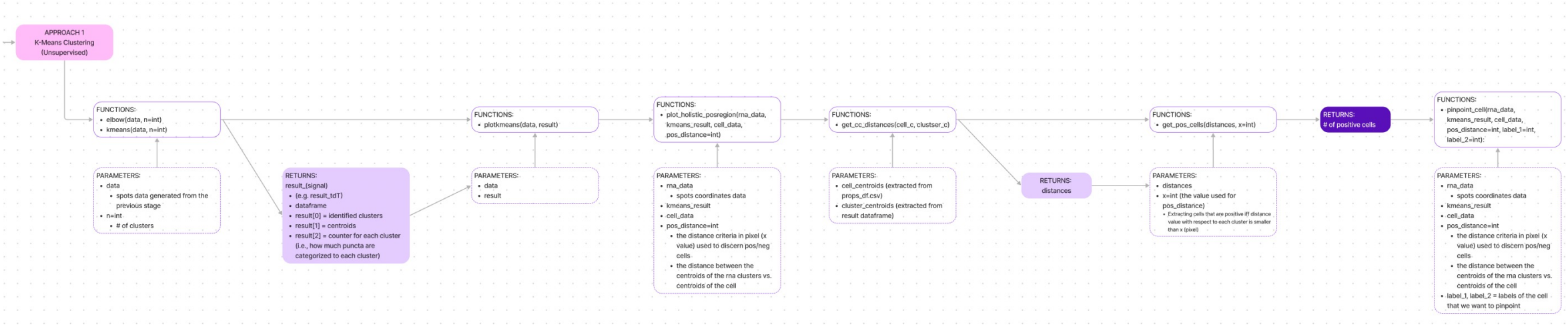
Lastly, the *pinpoint_cell* is just an auxiliary function that allows users to pinpoint the cell that they want to see by putting the label of each cell as a parameter.

This ends the first approach.

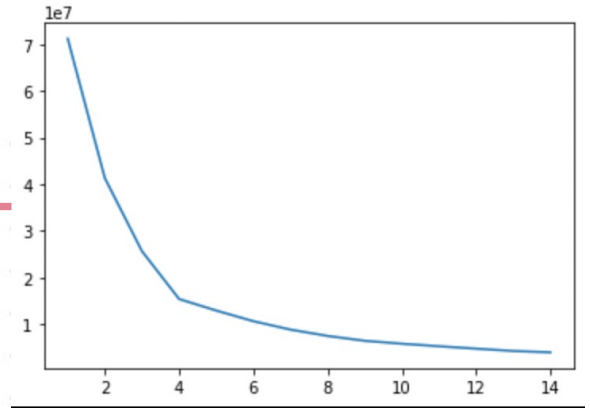
4


Cell & Signal Analysis
tdT_analysis.py
[signals]_analysis.py
• gfp_analysis.py
• oxt_analysis.py
APPROACH 1
K-Means Clustering
(Unsupervised)
FUNCTIONS:
• elbow(data, n=int)
• kmeans(data, n=int)
PARAMETERS:
• data
• spots data generated from the previous stage
• n=int
• # of clusters
RETURNS:
result_(signal)
• (e.g. result_tdT)
• dataframe
• result[0] = identified clusters
• result[1] = centroids
• result[2] = counter for each cluster (i.e., how much puncta are categorized to each cluster)
FUNCTIONS:
• plotkmeans(data, result)
PARAMETERS:
• data
• result
FUNCTIONS:
• plot_holistic_posregion(rna_data, kmeans_result, cell_data, pos_distance=int)
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
FUNCTIONS:
• get_cc_distances(cell_c, clustser_c)
PARAMETERS:
• cell_centroids (extracted from props_df.csv)
• cluster_centroids (extracted from result dataframe)
RETURNS:
distances
FUNCTIONS:
• get_pos_cells(distances, x=int)
PARAMETERS:
• distances
• x=int (the value used for pos_distance)
• Extracting cells that are positive iff distance value with respect to each cluster is smaller than x (pixel)
RETURNS:
# of positive cells
FUNCTIONS:
• pinpoint_cell(rna_data, kmeans_result, cell_data, pos_distance=int, label_1=int, label_2=int):
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
• label_1, label_2 = labels of the cell that we want to pinpoint
APPROACH 2
Pseudo-Dilation
FUNCTIONS:
• viz_cell_dilate(rna_data, cell_data, dilation_radius=int)
PARAMETERS:
• rna_data
• spots data
• cell_data
• centroids of the cell (extracted from the props_df file)
• dilation_radius = int
• how much pixel from the cell centroids you want the consider as a cell (i.e., cell boundary)
FUNCTIONS:
• get_cs_distance(c_cell, rna_data)
• get_filtered_ds(c_cell, rna_data, x=int)
PARAMETERS:
• cell_centroids (extracted from the props_df)
• rna_data
• spots data
• x=int
• how much distance (in pixel) from the cell centroids you want to consider as a signal
• (i.e., x selects the puncta to be considered for each cell)
RETURNS:
filtered distances with corresponding cells (dataframe)
• "all_ds.csv"
• "all_ds_filtered.csv"
FUNCTIONS:
• get_pos_cells(filtered_ds, rna_data, x=int)
PARAMETERS:
• filtered_ds
• dataframe generated by the 'get_filtered_ds' function
• rna_data
• raw coordintates data of the rna (in 2d array)
• x=int
• criteria for "how much puncta are required to categorize cell as pos?"
RETURNS:
# of positive cells & Cell labels

4.1


APPROACH 1
K-Means Clustering
(Unsupervised)
FUNCTIONS:
• elbow(data, n=int)
• kmeans(data, n=int)
PARAMETERS:
• data
• spots data generated from the previous stage
• n=int
• # of clusters
RETURNS:
result_(signal)
• (e.g. result_tdT)
• dataframe
• result[0] = identified clusters
• result[1] = centroids
• result[2] = counter for each cluster (i.e., how much puncta are categorized to each cluster)
FUNCTIONS:
• plotkmeans(data, result)
PARAMETERS:
• data
• result
FUNCTIONS:
• plot_holistic_posregion(rna_data, kmeans_result, cell_data, pos_distance=int)
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
FUNCTIONS:
• get_cc_distances(cell_c, clustser_c)
PARAMETERS:
• cell_centroids (extracted from props_df.csv)
• cluster_centroids (extracted from result dataframe)
RETURNS:
distances
FUNCTIONS:
• get_pos_cells(distances, x=int)
PARAMETERS:
• distances
• x=int (the value used for pos_distance)
• Extracting cells that are positive iff distance value with respect to each cluster is smaller than x (pixel)
RETURNS:
# of positive cells
FUNCTIONS:
• pinpoint_cell(rna_data, kmeans_result, cell_data, pos_distance=int, label_1=int, label_2=int):
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
• label_1, label_2 = labels of the cell that we want to pinpoint

4.1



APPROACH 1
K-Means Clustering
(Unsupervised)

FUNCTIONS:
- elbow(data, n=int)
- kmeans(data, n=int)

PARAMETERS:
- data
  - spots data generated from the previous stage
- n=int
  - # of clusters

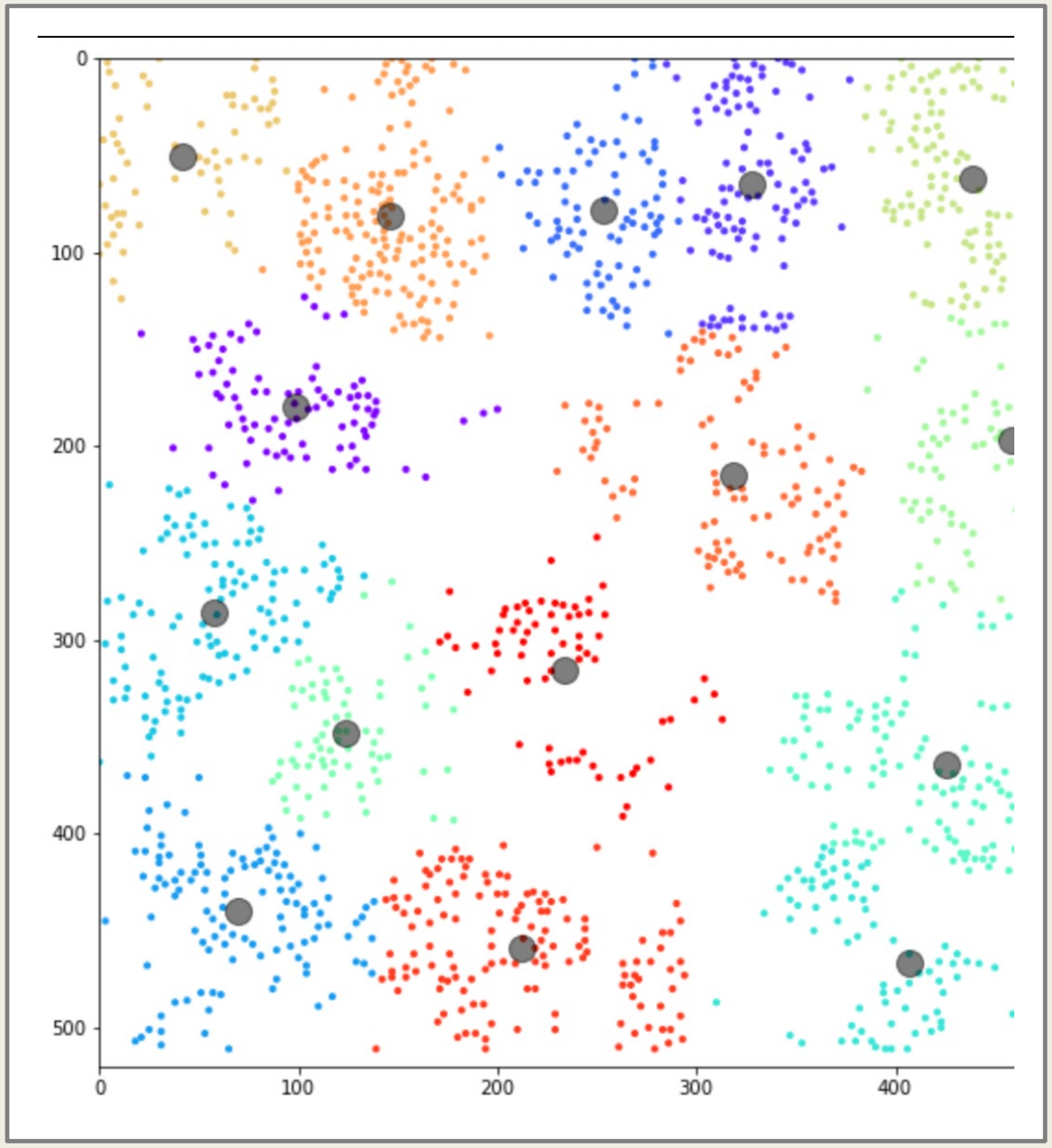
RETURNS:
result_(signal)
- (e.g. result_tdT)
- dataframe
- result[0] = identified clusters
- result[1] = centroids
- result[2] = counter for each cluster (i.e., how much puncta are categorized to each cluster)

FUNCTIONS:
- plotkmeans(data, result)

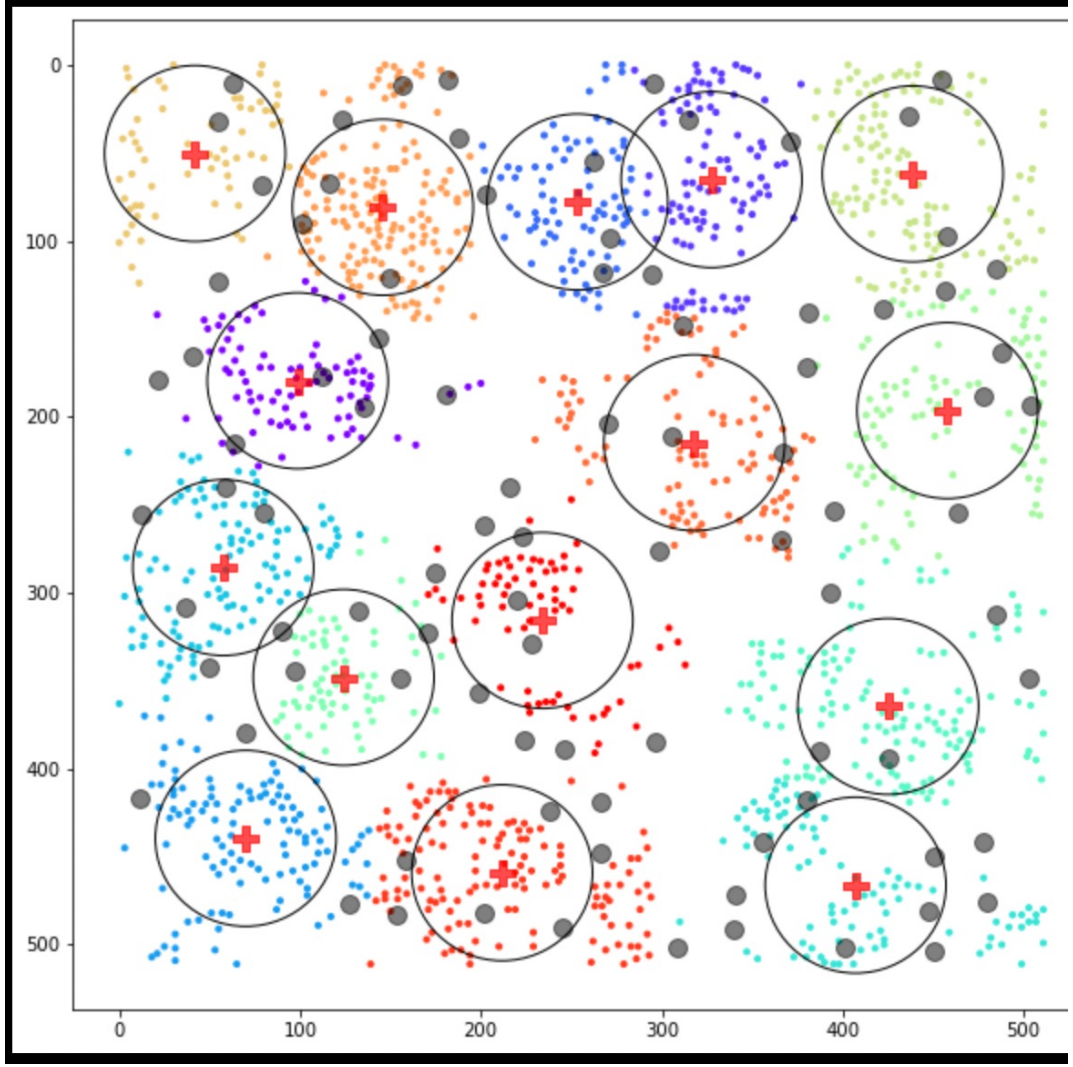
PARAMETERS:
- data
- result

FUNCTIONS:
- plot_holistic_posregion(rna_data, kmeans_result, cell_data, pos_distance=int)

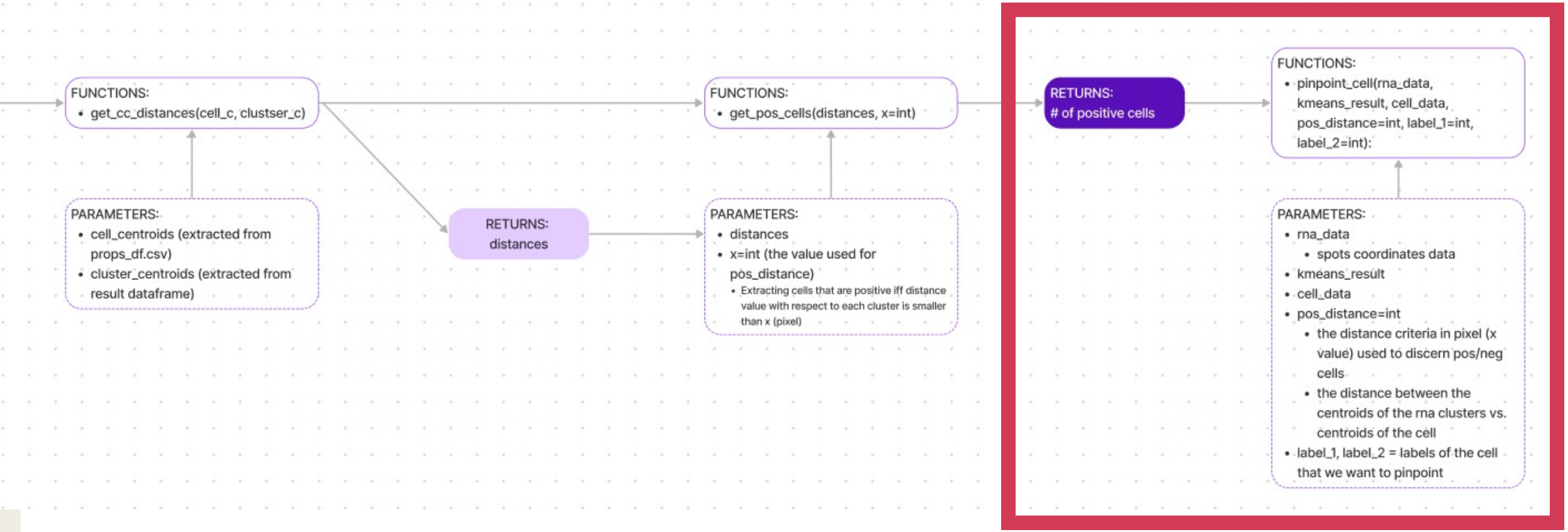
PARAMETERS:
- rna_data
  - spots coordinates data
- kmeans_result
- cell_data
- pos_distance=int
  - the distance criteria in pixel (x value) used to discern pos/neg cells
  - the distance between the centroids of the rna clusters vs. centroids of the cell



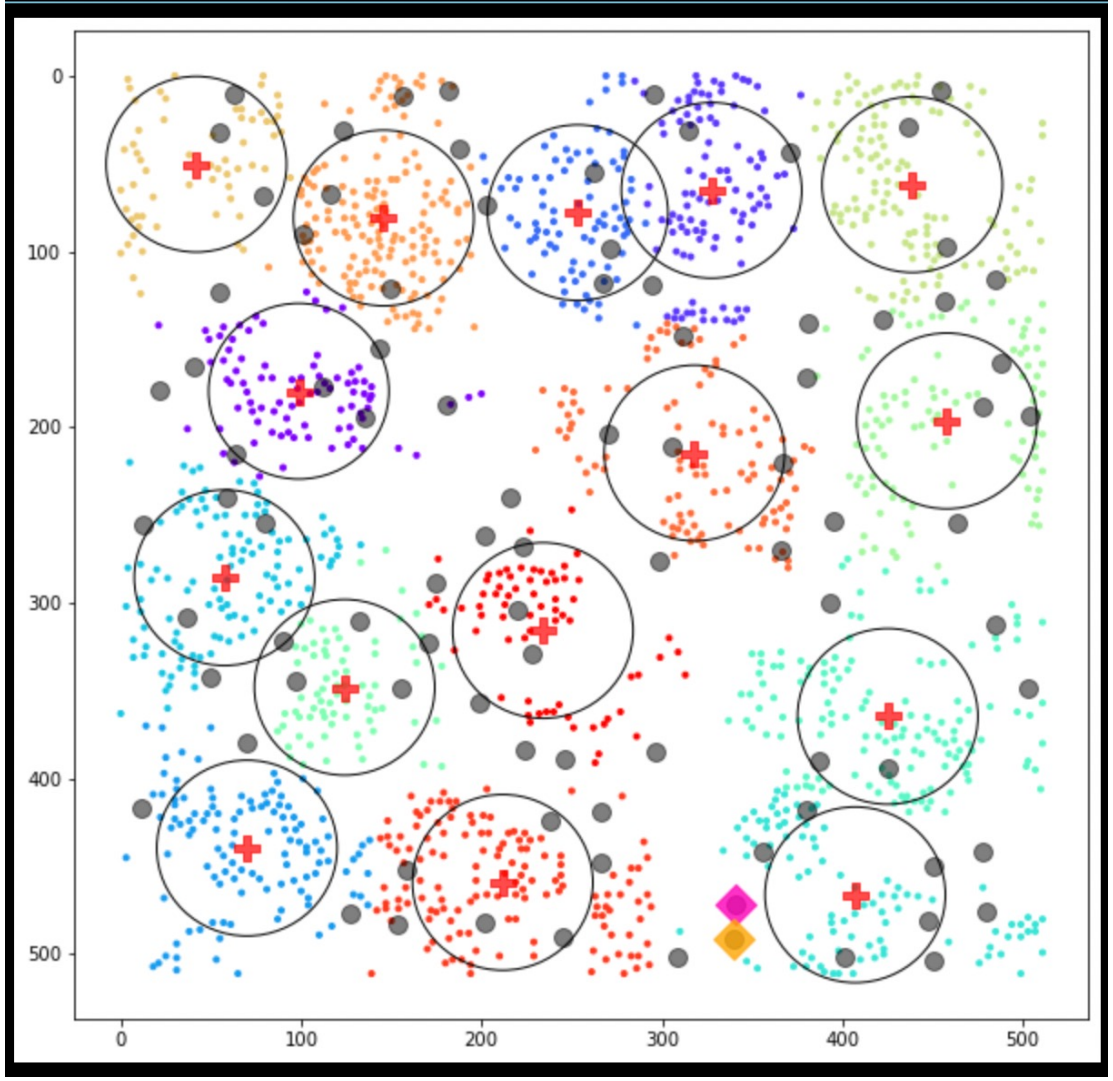

```
plot_holistic_posregion(data_tdT, result_tdT, centroids_cell, 50)
```



4.1


FUNCTIONS:
• get_cc_distances(cell_c, clustser_c)
PARAMETERS:
• cell_centroids (extracted from props_df.csv)
• cluster_centroids (extracted from result dataframe)
RETURNS:
distances
FUNCTIONS:
• get_pos_cells(distances, x=int)
PARAMETERS:
• distances
• x=int (the value used for pos_distance)
• Extracting cells that are positive iff distance value with respect to each cluster is smaller than x (pixel)
RETURNS:
# of positive cells
FUNCTIONS:
• pinpoint_cell(rna_data, kmeans_result, cell_data, pos_distance=int, label_1=int, label_2=int):
PARAMETERS:
• rna_data
• spots coordinates data
• kmeans_result
• cell_data
• pos_distance=int
• the distance criteria in pixel (x value) used to discern pos/neg cells
• the distance between the centroids of the rna clusters vs. centroids of the cell
• label_1, label_2 = labels of the cell that we want to pinpoint

```
pinpoint_cell(data_tdT, result_tdT, centroids_cell, 81, 88)
```




```
distances = get_distances(centroids_cell, centroids_cluster)
pos_cells(distances, 50)
```

```
40
```

STAGE 4: Cell and Signal Analysis – Approach 2

The second approach for this last step is the pseudo-dilation.

The reason why I named this method as pseudo-dilation is because we are not actually dilating the cells at the beginning. Rather using the centroids, we are drawing circular boundary across all cells and considering the puncta within the boundary.

To simulate what would be the best dilation radius for the cell, this first function called viz_cell_dilate would do justice. The parameters would be the spots data of the signals, the cell centroids data, and last and the most important, the dilation radius.

STAGE 4: Cell and Signal Analysis – Approach 2

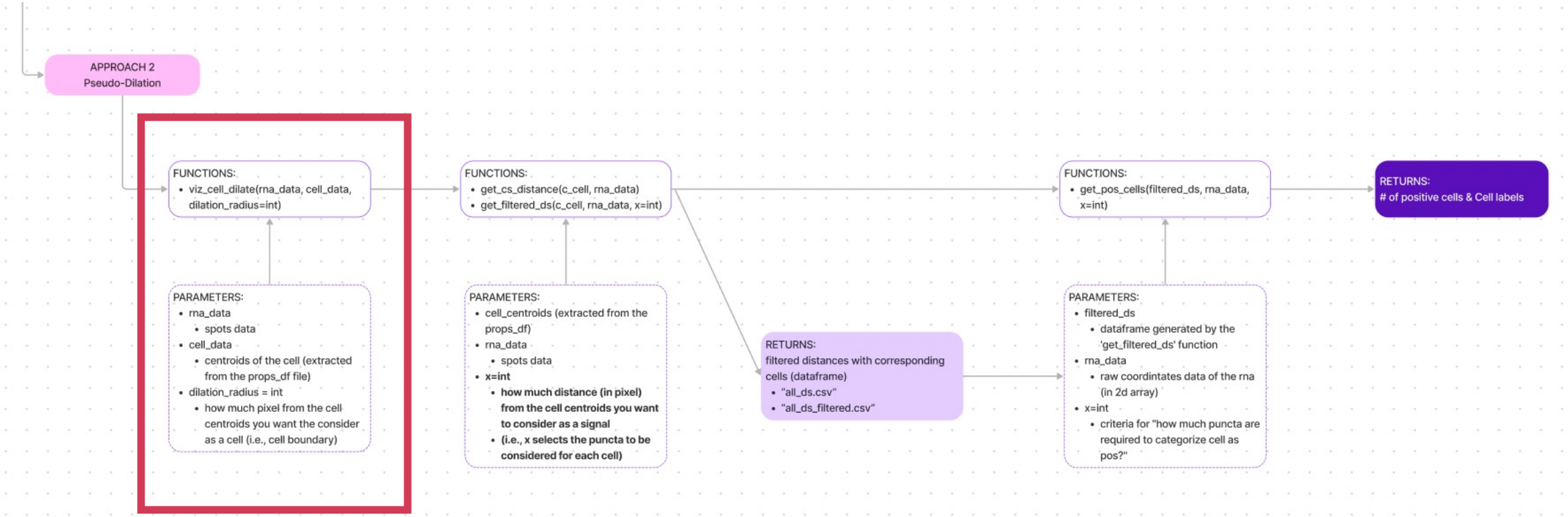
Now that we visualized the dilation radius, the user should have found the dilation radius that they want to use.

Using the user-defined dilation radius value, we would run *get_cs_distance* and *get_filtered_dis*. The parmater *x* in the *get_filtered_ds* would be the *dilation_radius* that the user chose.

This will spit out the puncta that are within the dilation radius from the cell centroid. And last, *get_pos_cells* would backtrack what the cells are and how many there are.

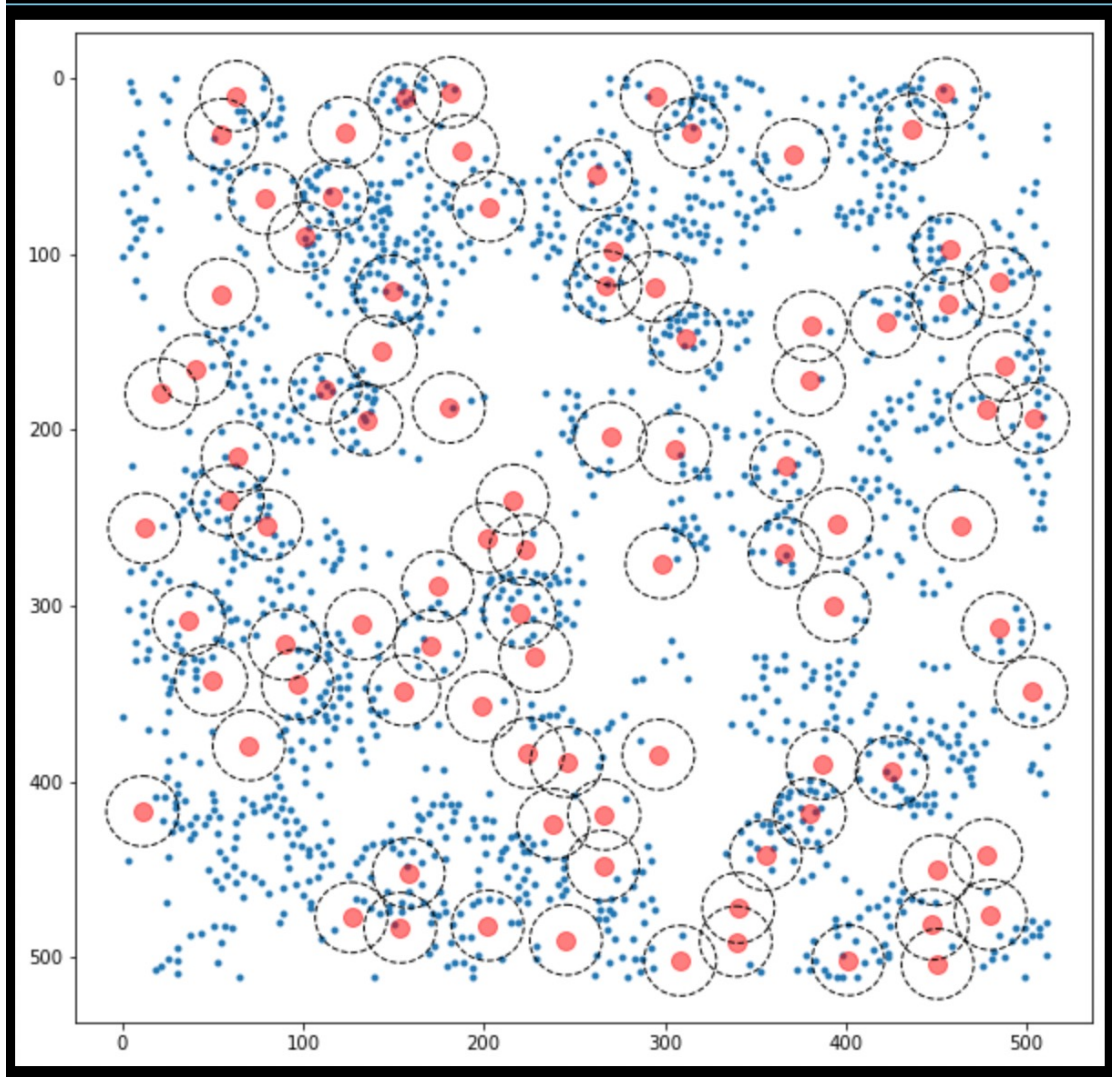
This ends the second approach of the last stage of the pipeline. Note that using the cell labels, we can also conduct colocalization analysis with other signals by matching the cell labels of the cell.

4.2

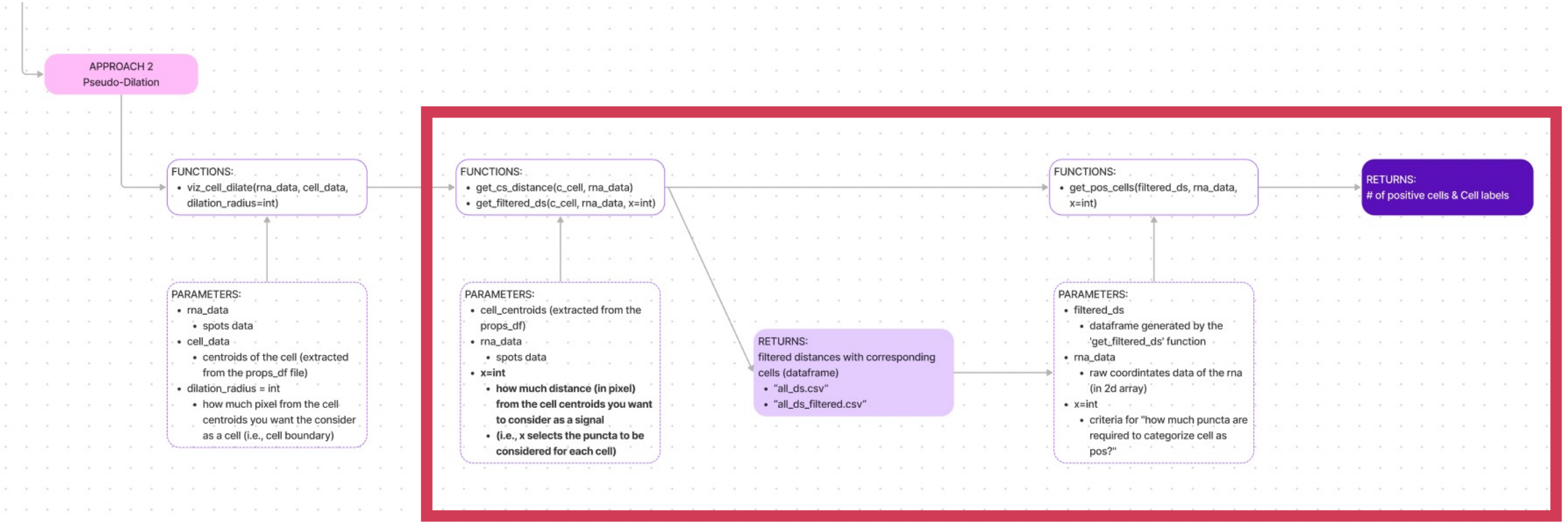

APPROACH 2
Pseudo-Dilation
FUNCTIONS:
• viz_cell_dilate(rna_data, cell_data, dilation_radius=int)
PARAMETERS:
• rna_data
• spots data
• cell_data
• centroids of the cell (extracted from the props_df file)
• dilation_radius = int
• how much pixel from the cell centroids you want the consider as a cell (i.e., cell boundary)
FUNCTIONS:
• get_cs_distance(c_cell, rna_data)
• get_filtered_ds(c_cell, rna_data, x=int)
PARAMETERS:
• cell_centroids (extracted from the props_df)
• rna_data
• spots data
• x=int
• how much distance (in pixel) from the cell centroids you want to consider as a signal
• (i.e., x selects the puncta to be considered for each cell)
RETURNS:
filtered distances with corresponding cells (dataframe)
• "all_ds.csv"
• "all_ds_filtered.csv"
FUNCTIONS:
• get_pos_cells(filtered_ds, rna_data, x=int)
PARAMETERS:
• filtered_ds
• dataframe generated by the 'get_filtered_ds' function
• rna_data
• raw coordintates data of the rna (in 2d array)
• x=int
• criteria for "how much puncta are required to categorize cell as pos?"
RETURNS:
# of positive cells & Cell labels

This image shows what this function would do. Here, the circular region is drawn 20 pixels away from the cell centroids.

```
viz_cell_dilate(data_tdT, centroids_cell, 20)
```



4.2


APPROACH 2
Pseudo-Dilation
FUNCTIONS:
• viz_cell_dilate(rna_data, cell_data, dilation_radius=int)
PARAMETERS:
• rna_data
• spots data
• cell_data
• centroids of the cell (extracted from the props_df file)
• dilation_radius = int
• how much pixel from the cell centroids you want the consider as a cell (i.e., cell boundary)
FUNCTIONS:
• get_cs_distance(c_cell, rna_data)
• get_filtered_ds(c_cell, rna_data, x=int)
PARAMETERS:
• cell_centroids (extracted from the props_df)
• rna_data
• spots data
• x=int
• how much distance (in pixel) from the cell centroids you want to consider as a signal
• (i.e., x selects the puncta to be considered for each cell)
RETURNS:
filtered distances with corresponding cells (dataframe)
• "all_ds.csv"
• "all_ds_filtered.csv"
FUNCTIONS:
• get_pos_cells(filtered_ds, rna_data, x=int)
PARAMETERS:
• filtered_ds
• dataframe generated by the 'get_filtered_ds' function
• rna_data
• raw coordintates data of the rna (in 2d array)
• x=int
• criteria for "how much puncta are required to categorize cell as pos?"
RETURNS:
# of positive cells & Cell labels

Cell Labels that are categorized as positive

```
Output exceeds th size limit. Open the full output data in a text editor
([2,
  4,
  5,
  6,
  7,
  8,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  24,
  26,
  28,
  29,
  30,
  32,
  36,
  37,
...
  83,
  84,
  86,
  52)
```

The number of positive cells

```
data_tdT_txt = np.loadtxt('spots_tdT_2d.txt')
all_ds_filtered = get_filtered_ds(centroids_cell, data_tdT_txt, 25) # Consider puncta which are within the distance of 25 pixel from the cell centroid
pos_cells = get_pos_cells(all_ds_filtered, data_tdT_txt, 10) # If over 10 puncta, categorize as 10.
pos_cells
```

```
utput exceeds the size limit. Open the full output data in a text editor
2
```