CellTracker Tutorial

Table of Contents

1. Intruduction *2*

2. Environment Setup *2*

3. Software Architecture *3*

3.1 Main Window 3

3.2 View panel 4

3.3 Function panel 4

3.4 Cell management panel 5

4. Analysis workflow *6*

4.1 Start up CellTracker 6

4.2 Load input images and define project folder 7

4.3 Normalization 7

4.4 Segmentation and Tracking 8

4.5 Result output 9

5. Annotation tools *9*

6. Training module of Deep Neural network *10*

7. Demos *12*

7.1 demo 1: HeLa cells segmentation and tracking 12

7.2 demo 2: E.coli cells segmentation and tracking 16

1. Intruduction

CellTracker is a highly integrated graph user interface software developed for the cell segmentation and tracking of time-lapse microscopy image. CellTracker covers essential steps for full analyzing procedure of microscopy images with the following features:

- ✓ A GUI for cell image normalization
- ✓ A GUI for cell image segmentation
- ✓ A GUI for cell image tracking
- ✓ A GUI for manual correction
- ✓ A GUI for cell property profiling and intensity quantification
- ✓ A GUI for training data annotation
- ✓ A GUI for deep CNN model training
- ✓ A Python API for deep CNN model training

CellTracker is an open-source software under the GPL-3.0 license. It is implemented in pure Python. The source code, instruction manual, and demos can be found at *https://github.com/WangLabTHU/CellTracker.git*

2. Environment Setup

CellTracker supports popular operating systems including Windows, Linux, and MacOS. To eliminate the unexpected environment dependency errors caused by the operating system or customized environment of users, the python virtual environment is strongly recommended for users. The environment setup and CellTracker installation procedure from scratch are also available on the project homepage in GitHub.

CellTracker software runs on Python 3.5 or higher version and requires some dependency packages as shown in Table 1. In principle, all these packages can have dependencies that must be fulfilled.

package	version
numpy	1.17.2

pandas	1.0.4
matplotlib	3.1.2
opencv-python	4.1.1.26
PyQt5	5.14.1
PyQt5-sip	12.7.1
pyqtgraph	0.10.0
scipy	1.4.1
six	1.13.0
sklearn	0.22.1
Jinja2	2.11.1
torch	1.4.0
torchvision	0.5.0
imageio	2.6.1

Table 1: The dependency packages and version of CellTracker.

3. Software Architecture

CellTracker is designed for cell segmentation and tracking analysis of time-lapse microscopy image. The overall architecture of CellTracker is decoupled as four separate modules according to its function, including file explorer, algorithm analysis pipeline (normalization, segmentation, tracking, and visualization output), annotation tools, and retraining module for deep CNN model.

3.1 Main Window

The users can start CellTracker by running the file main.py in the terminal after that CellTracker will launch the initial GUI main window as shown in Figure 1. The main window of CellTracker is divided into four parts, which consist of a function panel, main view panel, cell management panel, and image preview panel.

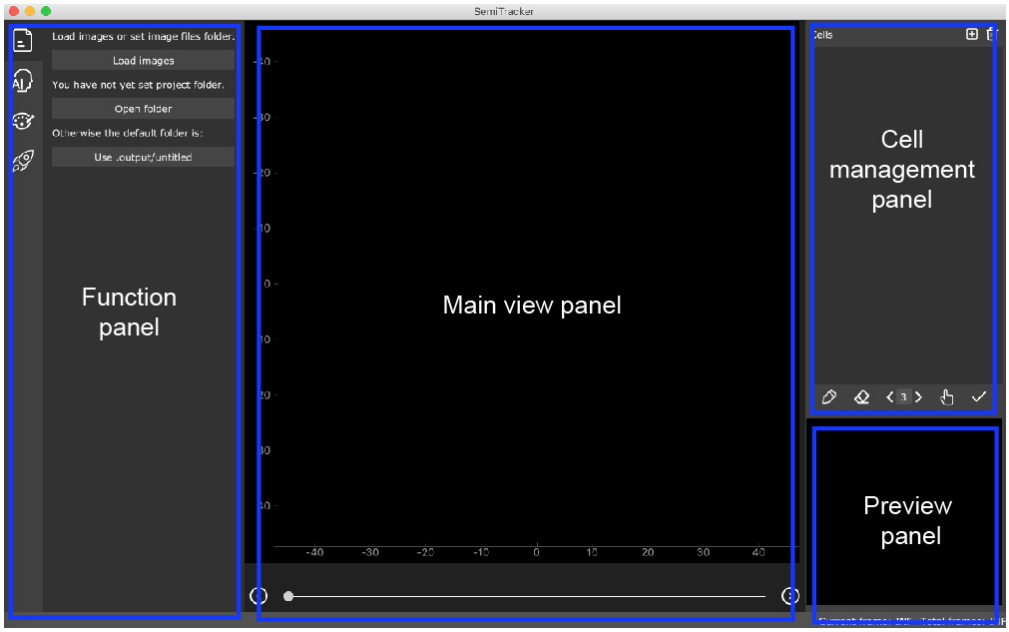

Figure 1: Main window of CellTracker with its four image subpanels, function panel (left blue rectangle), main view panel (central blue rectangle), cell management panel (top right blue rectangle), image preview panel (bottom right blue rectangle).

3.2 View panel

In this section, we simply describe the two view panel of CellTracker including the main view panel and the preview panel. The main view panel is used to show the resulting image of each process step such as normalization, segmentation, tracking and so on. The slider at the bottom of the main view is the indicator of the frame index of the image sequence. Browsing through the time series can be achieved by moving the slider, which leads to synchronous updating of the images in the main view. The main view panel supports the zoom function. Users can scroll up or down the mouse wheel to adjust the focal zone of the image. Meanwhile, the coordinate axis in the main view provides the absolute location information of the focal zone. The preview panel is a small window to display the origin image.

3.3 Function panel

The function panel provides access to the file explorer, algorithm analysis pipeline (normalization, segmentation, tracking, and visualization output), annotation tools, and retraining module as shown in Figure 2. Users can expand each subpanel by clicking the corresponding icon.

The algorithm analysis pipeline defines the workflow for the general task of microscopy image analysis, which is composed of four modules, normalization, segmentation, track, and output. The third function panel is annotation tools, which are used to generate supervised labels for customized datasets. This tool also can be used to correct the results of automatic algorithms at pixel resolution. The last function panel, retraining module, provides a GUI for deep CNN model training.

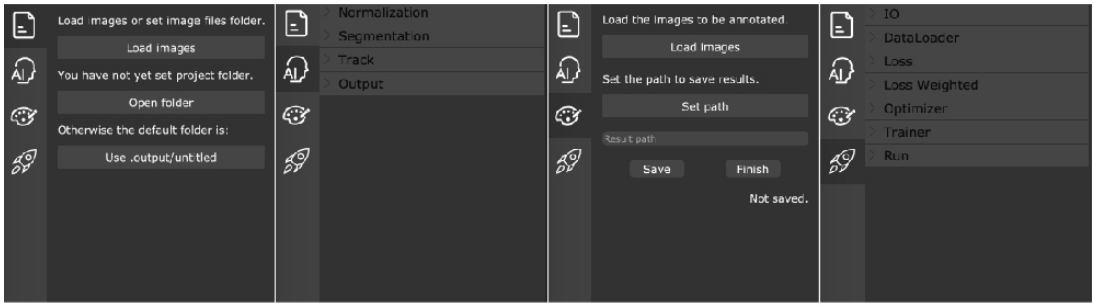

Figure 2: Function panel of CellTracker with its four subpanels, From left to right are file explorer, algorithm analysis pipeline, annotation tools, and re-training module.

3.4 Cell management panel

The cell management panel has two functions: one is to manage all the cell instances such as add and delete, and the other one is to edit the cell segmentation mask.

Firstly, the cell management panel organizes all the segmented cells of the corresponding image displayed in the main view. Double-clicking the cell name will bring up a menu showing the cell properties, such as cell size, cell intensity, cell centroid coordinates and other attribution as shown in Figure 3.

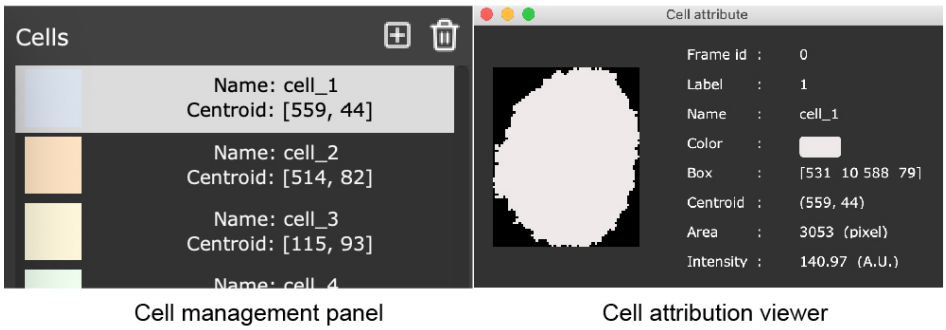

Figure 3: The cell management panel. Double-clicking the cell name pop up the cell attributions.

Second, as shown in Figure 4, the cell edit toolbar at the bottom of the cell management panel is used to edit the cell segmentation mask manually. The cell edit toolbar is composed of five sub-buttons, a brush button for adding a new mask, an eraser button to delete history mask, a drag button to move the image in the main view, a size modify button to change the size of brush or eraser, and an update buttons to ensure and save user changes. By leaving the mouse arrow on buttons for a while, a help text will appear with a short description.

To edit the existing cells, the user should select the cell to be edited firstly, and then choose the edit buttons in the edit toolbar and modify the mask. Finally click the Update button to put the changes into effect.

as shown in Figure 5, to add a new cell, the user should click the ‘Add’ button at the top of cell management panel, then set the name and color to the new cell. Next, select the brush buttons in the edit toolbar to add the mask as needed. Also, the user can modify the mask by the Eraser button. Finally click the Update button to make it effective.

Figure 4: The cell edit toolbar. The cell mask can be edit and delete as you want.

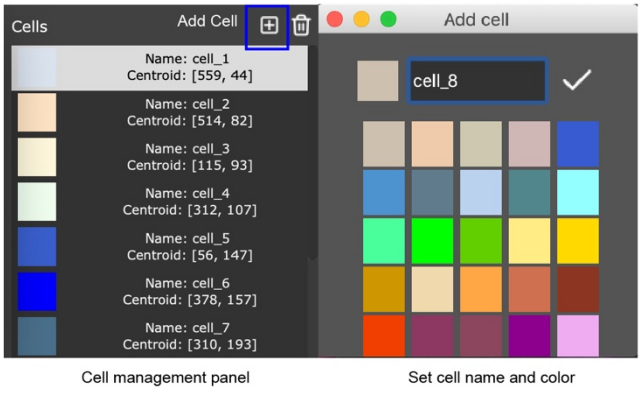

Figure 5: Add new cell, then set the name and color to the new cell.

4. Analysis workflow

4.1 Start up CellTracker

Create a new folder and drag inside all the files of the GUI. To run CellTracker, open a terminal

(terminal in macOS or Linux, or cmd on Windows), and create a new virtual environment as *https://github.com/WangLabTHU/CellTracker.git*. Move into the directory where the files are located and run: python main.py and then press Enter. The GUI of CellTracker will pop-up. It runs on Linux, Windows, and Mac.

4.2 Load input images and define project folder

Loaded images are required before the analysis process. As shown in Figure 6, the user should define the project folder which is used to save the output results of the analysis, and the default path is ‘untitled’ in the output folder. The image size of a given microscope must be the same. If the user loads a set of images, the displayed images will follow the order of the file names. After the time-series image is loaded, you can browse the time series by moving the slider at the bottom of the main view panel.

The supported formats for input image including static image format 'png', 'jpg', 'jpeg', 'tif' and video format 'mp4', 'avi', 'mpg'.

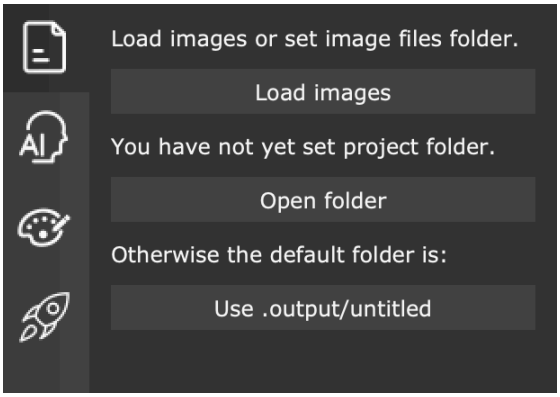

Figure 6: Loaded images are required before the analysis process.

4.3 Normalization

After loading images, the algorithm analysis pipeline is divided into four major steps, which consist of normalization, segmentation, tracking, and results output. Users can click on the options to choose the corresponding function.

CellTracker provides four optional normalization methods, including Histogram Equalize normalization [1], Min-Max normalization, Retinex algorithm MSRCP [2], and Retinex algorithm MSRCR [3] as shown in Figure 7. Normalization allows user to increase the amount of visible detail

by enhancing contrast. Users can select “Remove Norm” to recover the original image.

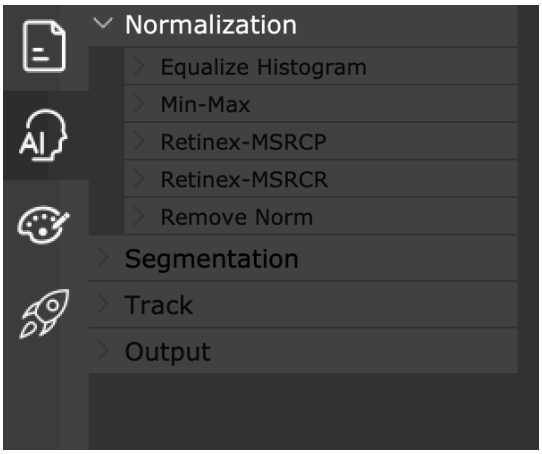

Figure 7: The four optional normalization methods of CellTracker.

4.4 Segmentation and Tracking

To improve the processing capability of the multi-modal image in real-world datasets, four candidate segmentation algorithms are provided for selection in CellTracker. One of them, GrabCut [4], is interactive algorithms. The remaining three methods are the baseline algorithm Otsu threshold [5], Watershed algorithm [6], and the deep model-based convolution network [7] as shown in Figure 8. After segmenting, the tracking algorithm is used to track each cells across frames in temporal dimension [8]. For each algorithm, the default parameters are recommended for the common case. When needed, CellTracker allows users to modify the hyper-parameters. Note that all algorithms are automatic, but users can correct the result of segmentation and tracking with the cell management panel manually. Please see the details in Section 3.3.

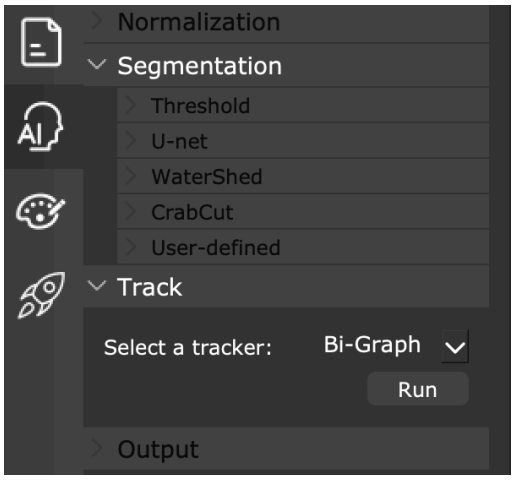

Figure 8: The segmentation and tracking algorithms of CellTracker.

4.5 Result output

For user convenience, CellTracker provides a comprehensive analysis reports with HTML and CSV, which includes tables of cell property profiling and figures of long-term single-cell fluorescence intensity quantification. Moreover, exporting images and videos for the visualization of cell segmentation and tracking are also supported. By select the attribution name with the checkmark, users can define the output attribution in the visualization image including color mask, bounding box, cell edge, trajectories, and cell label as shown in Figure 9. Finally, click on the push button “Output” to output the results, and the output file will be saved in the project folder.

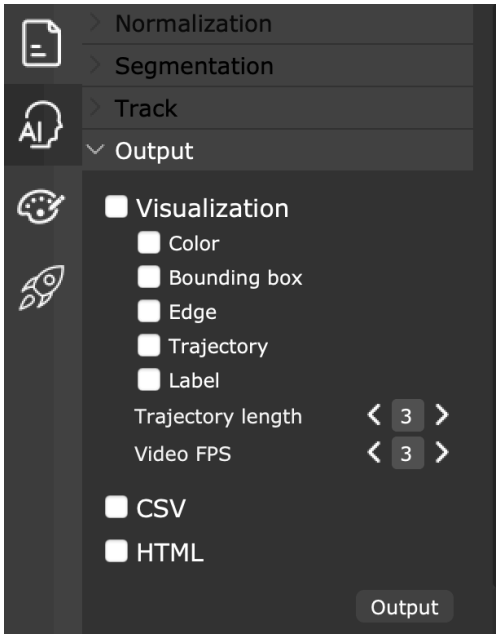

Figure 9: The optional output attribution of output module.

5. Annotation tools

Before annotation, the users should load the initial set of original images and define the output folder of annotation images. CellTracker can label images based on automated segmentation results to simplify the annotation process. Hence this hand-drawing work which consumes most of the time in annotation could be mitigated. Besides, CellTracker supports annotating the image with interruption resuming capability to prevent annotation file missing caused by possible software

corrupting as shown in Figure 10.

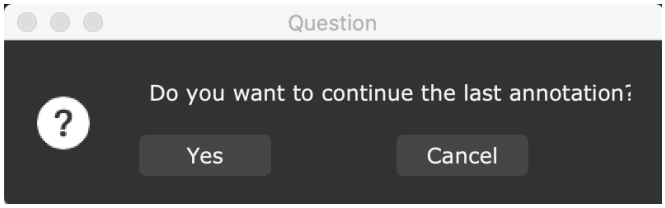

Figure 10: CellTracker supports annotating the image with interruption resuming capability. Users can continue the last annotation based on the annotation cache.

6. Training module of Deep Neural network

CellTracker provides a GUI and Python API for training the deep U-Net model of customized datasets. Before training U-Net, the users must set the following parameters in the GUI (Figure 11) or python scripts (the demo code is released in GitHub).

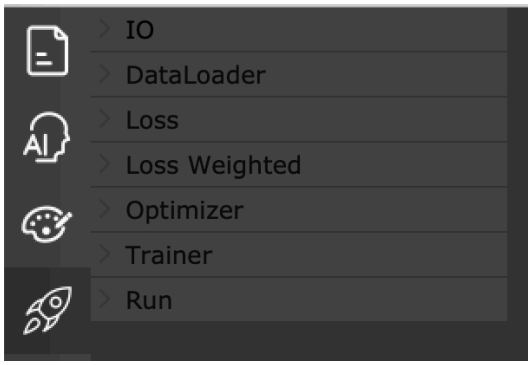

Figure 11: The GUI panel for defining the training parameters of U-Net.

- ***Set source folder [source_img_root in Python API, default= None]***

  The folder path of source image used to train the model. The supported formats for source image including static image format 'png', 'jpg', 'jpeg', 'tif'.

- ***Set label folder [label_img_root in Python API, default= None]***

  The folder path of label image used to train model. The supported formats for label image including 'png' and 'tif'. The numbers of source image and label image should be equal, and the order of the source image and label image should be consistent.

- ***Set log folder [log_root in Python API, default= None]***

  The log file saved path for training.

- ***Batch size [batch_size in Python API, default= 4]***

  The batch size for one forward and backward iteration in training.

- ***Validation ratio [validation_ratio in Python API, default= 0.3]***

  The random split ratio of validation set.

- ***Scale image [scale_img in Python API, default= 1]***

  The scale ratio of the training image compared to the original image. If the size of the original image is too large or too small, this parameter can be used to reduce computing costs or enhance image quality.

- ***Parallel works [workers in Python API, default= 0]***

  The number of parallel threads in the training stage. It can be used to lower the training time.

- ***Augmentation [aug_list in Python API, default= None]***

  The optional augmentation methods for training, including Flip, Rotate, GaussianNoise, and GaussianBlur. Parameter aug_list should be list type in Python API.

- ***Loss [loss_type in Python API, default= DiceLoss]***

  The loss function used to train U-Net, the valid option includes DiceLoss, WeightedSoftDiceLoss, BCELoss, WBCELoss, MSELoss, and MAELoss.

- ***Weight [weighted_type in Python API, default= None]***

  The weight option used to balance the positive and negative samples in training. Can be edge_weighted, sample_balance and None.

- ***Weight decay [weight_decay in Python API, default= 0.0005]***

  The weight decay for Adam optimizer.

- ***Learning rate [lr in Python API, default= 0.001]***

  The initial learning rate for Adam optimizer.

- ***epoch [epochs in Python API, default= 100]***

  The epoch number of training.

- ***GPU number [gpu_num in Python API, default= 0]***

  The GPU number in the training stage.

- ***Load model [resume in Python API, default= None]***

  The string of model path. If there is a valid pre-train model, the users can finetune U-Net based on the pre-train model.

7. Demos

In this section, we provide two demos to make it easier to use CellTracker.

7.1 demo 1: HeLa cells segmentation and tracking

In this section, we conducted a demonstration experiment to compare the performance of different segmentation algorithms based on HeLa cells. HeLa cell is a simulation data set derived from Cell Tracking Challenge [9]. Following the rules in this challenge, we use Jaccard coefficients to evaluate the performance of the algorithm.

After installing CellTracker, move to the directory where the file is located and run: python main.py, then press Enter. The GUI of CellTracker will pop up, and the initial panel is Explorer. The Explorer is used to load images and set the project folder to save the output results.

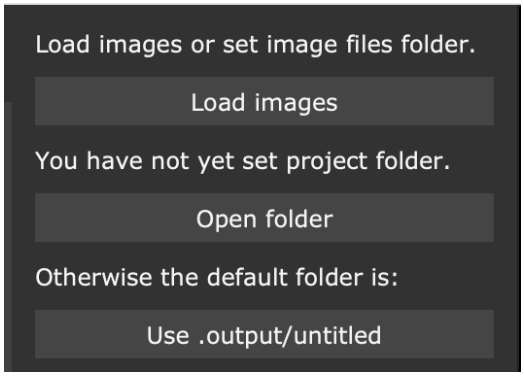

Figure 12: The GUI of Explorer.

For the HeLa cells, the test image is located in the folder datasets/demo-dataset1. And we set the default folder output/untitled as the project folder. After setting, users can use CellTracker to browse project folders:

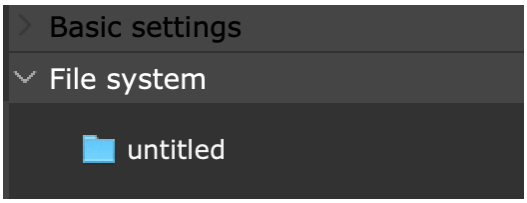

Figure 13: The GUI for browsing project folders.

Then, switch to the *Algorithm tools window*. Choose '*Equalize Histogram*' in the *Normalization tab* and click the *run* button to normalize the original images, and the normalized images are displayed

in the *Main Window*.

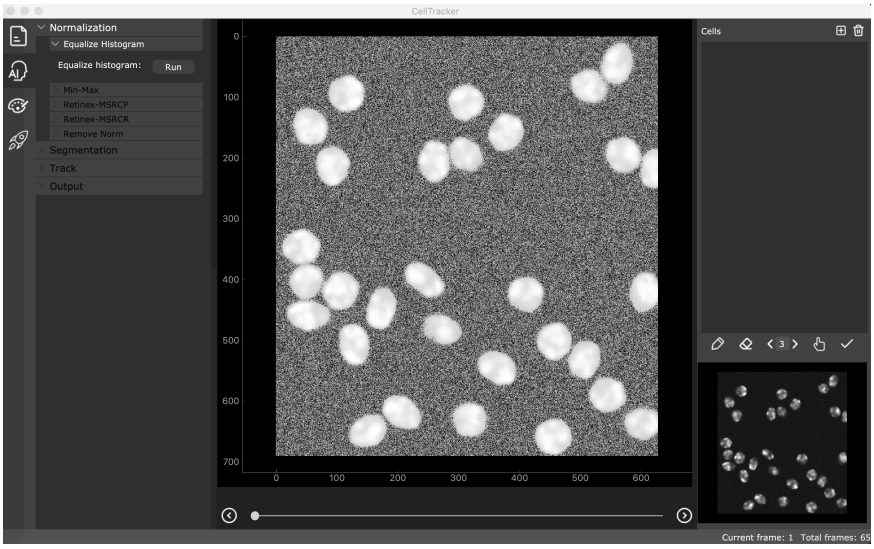

Figure 14: The visualization results of normalization.

Open the *Segmentation tab*, and then select the *WaterShed algorithm* to segment cells. We set the *Minimal size* as 50 to filter the small segmented object, and the other parameters are set as default values.

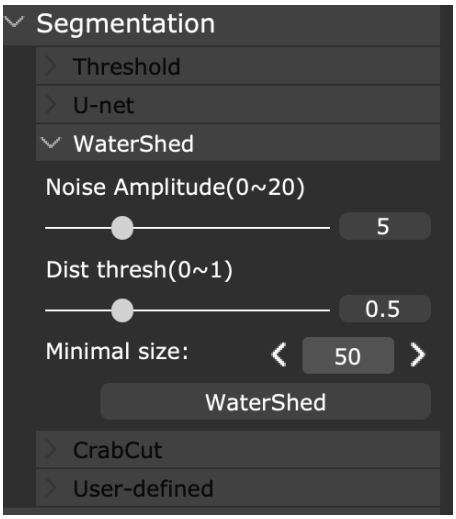

Figure 15: The setting panel of segmentation algorithms.

After setting the parameters, run the algorithm by clicking the WaterShed button. Then CellTracker will segment each cell and give it a unique color to represent the cell ID. Users can browse the results of time-series images by moving the slider at the bottom of *Main Window*.

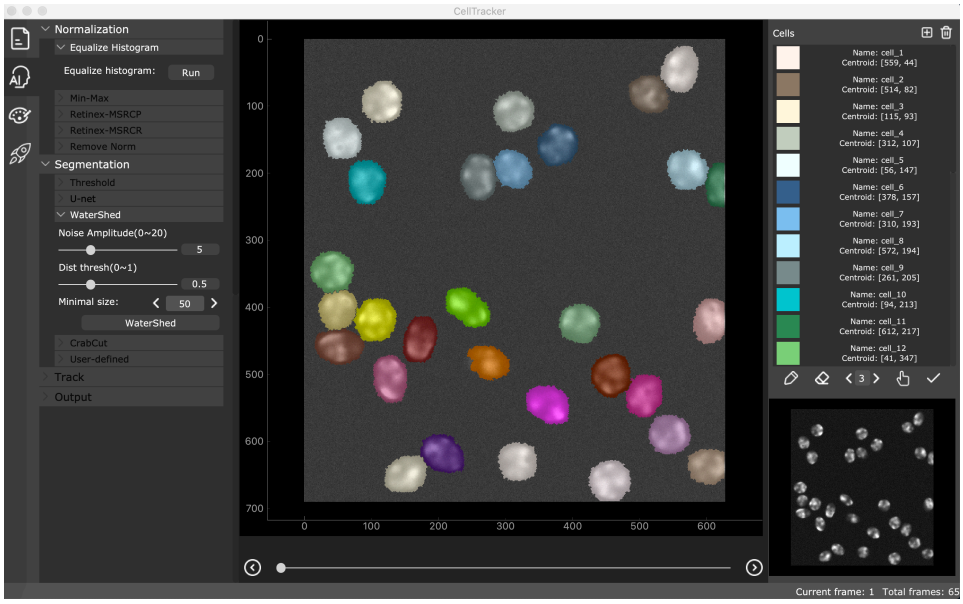

Figure 16: The visualization results of segmentation.

Double-click the cell name in the *Cell management panel* to view cell properties, such as cell size, cell intensity, cell centroid coordinates, and other properties. In addition, users can edit the result of segmentation with the *cell edit toolbar*. Please refer to Section 3.3 for details.

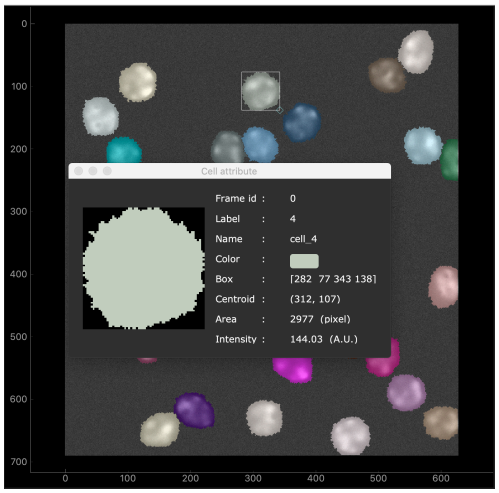

Figure 17: The viewer panel of cell properties.

After segmenting, open the *Track tab,* and then click the *Run* button to associate the cells in time dimension. After tracking, the colors belong to the same cell in consecutive frames will set be the same.

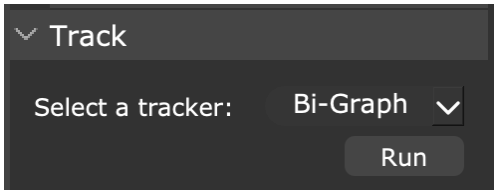

Figure 18: The setting panel of tracker algorithm.

Finally, set the output options in the *Output tab* as needed, and then click the *Output* button. The output results will be saved in the previously defined project folder.

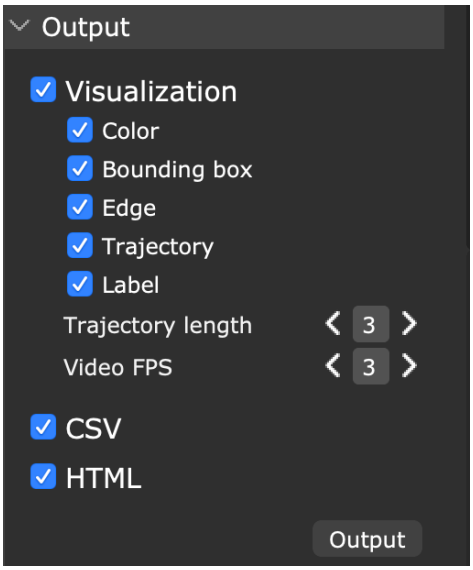

Figure 19: The setting panel of output.

Moreover, we have trained the U-Net model for HeLa cells, the training datasets, including the source images and label images, are provided in the *cell-tracker/training_demo1* folder, and the training scripts are released in cell-tracker/training_demo1/retraining.py. For details of the procedure of the U-Net training and inference, please refer to the next sub-section of *demo2*.

The performance of Ostu Threshold, WaterShed, and U-net in validation sets are summarized as the flowing Table. The higher SEG represents better performance.

Table 1: The performance of segmentation algorithm.

Algorithm	Ostu Threshold	WaterShed	U-net
SEG	0.545	0.789	0.801

7.2 demo 2: E.coli cells segmentation and tracking

The section provides a demonstration to help users develop the complete process of deep learning applications from datasets annotation to model training.

Datasets Annotation

Before annotating, the users should switch to the *Annotation tools window*, load the initial set of original images and define the output folder for the annotated images. For example, we load the E.coli cell images of the mother machine system in the *datasets/ demo-dataset1/source_img* folder.

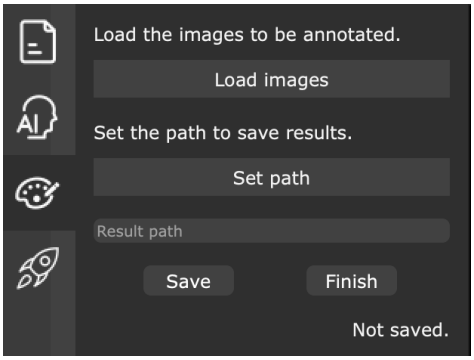

Figure 20: The IO setting of annotation tools.

Click the ‘Add’ button at the top of the *cell management panel*, then set the name and color to the new cell. Next, select the brush buttons in the *edit toolbar* to add the mask as needed. Also, the user can modify the mask by the Eraser button. Finally, click the Update button to make it effective. Also, CellTracker can label images based on automated segmentation results to simplify the annotation process.

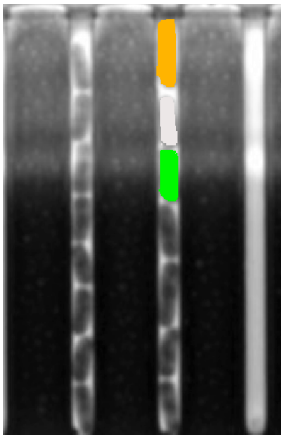

Figure 21: Annotation example.

Model Training

After annotating, users can train a deep U-Net model of a custom data set through the CellTracker GUI or Python API. For GUI training, switch to the *Training window* firstly. hen follow the instructions in **Section 6** to set the training parameters. For Python API training, two demo scripts for HeLa and E.coli cells are provided in *datasets/ demo-dataset1/ retraining.py* and *datasets/ demo-dataset2/ retraining.py* respectively. Considering the better PyTorch support, it is recommended to use Python API for training in the Linux environment.

Figure 22: The setting panel of training module.

References

[1] R. Hummel, "Image enhancement by histogram transformation," *ieht,* 1975.

[2] A. B. C. S. a. J. M. M. Petro, "Multiscale Retinex, Image Processing On Line," *IPOL,* vol. 4, pp. 71-88, 2014.

[3] J. D. J, R. Z and W. G. A, "A multiscale retinex for bridging the gap between color images and the human observation of scenes," *IEEE Transactions on Image processing,* pp. 6(7): 965-976, 1997.

[4] K. V. B. A. Rother C, "" GrabCut" interactive foreground extraction using iterated graph cuts," *ACM transactions on graphics (TOG),* vol. 23(3), pp. 309-314, 2004.

[5] O. N, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics,* vol. 9(1), pp. 62-66, 1979.

[6] M. F. Beucher S, "The morphological approach to segmentation: the watershed transformation," *Mathematical morphology in image processing,* vol. 34, pp. 433-481, 1993.

[7] M. D. B. R. e. a. Falk T, "U-Net: deep learning for cell counting, detection, and morphometry," *Nature methods,* vol. 16(1), pp. 67-70, 2019.

[8] A. e. a. Bewley, "Simple online and realtime tracking," *2016 IEEE International Conference on Image Processing (ICIP),* Vols. IEEE, 2016, pp. 3464-3468, 2016.

[9] M. e. a. Maška, "A benchmark for comparison of cell tracking algorithms," *Bioinformatics,* vol. 30.11, pp. 1609-1617, 2014.