bio-protocol

Published: Feb 20, 2022

# Tracking Moving Cells in 3D Time Lapse Images Using 3DeeCellTracker

**Chentao Wen[1, *] and Koutarou D. Kimura[1, 2, 3]**

[1]Graduate School of Science, Nagoya City University, Nagoya, Japan
[2]Department of Biological Sciences, Graduate School of Science, Osaka University, Toyonaka, Japan
[3]RIKEN center for Advanced Intelligence Project, Tokyo, Japan
*For correspondence: chentao-wen@umin.ac.jp

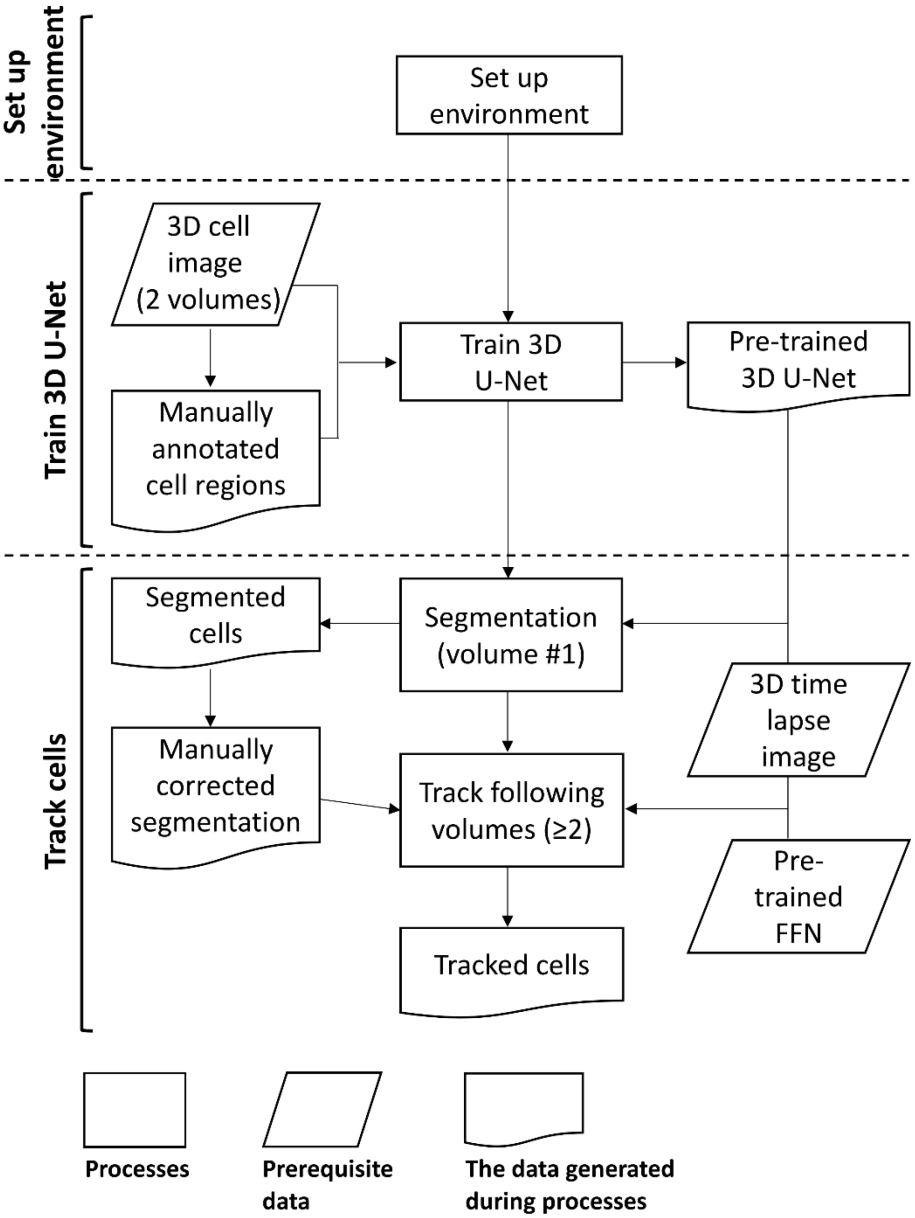
## Abstract

Recent advancements in 3D microscopy have enabled scientists to monitor signals of multiple cells in various animals/organs. However, segmenting and tracking the moving cells in three-dimensional time-lapse images (3D + T images), to extract their dynamic positions and activities, remains a considerable bottleneck in the field. We developed a deep learning-based software pipeline called 3DeeCellTracker, which precisely tracks cells with large movements in 3D + T images, obtained from different animals or organs, using highly divergent optical systems. In this protocol, we explain how to set up the computational environment, the required data, and the procedures to segment and track cells with 3DeeCellTracker. Our protocol will help scientists to analyze cell activities/movements in 3D + T image datasets that have been difficult to analyze.

**Keywords:** 3D microscopy, Time lapse images, Cell tracking, Cell segmentation, Deep learning

**This protocol was validated in:** eLife (2021), DOI: 10.7554/eLife.59187

## Graphic abstract:




**The flowchart illustrating how to use 3DeeCellTracker.**
See the Equipment and Procedure sections for detailed explanations.

## Background

Three-dimensional time-lapse (3D + T) microscope techniques allow scientists to monitor the dynamics of multiple cellular signals in living organs over time, such that the mechanism underlying various physiological phenomena can be investigated. However, it is difficult to extract signals from cells in 3D + T images. Specifically, large cell movements, insufficient image quality, and the unbalanced resolution between the *x*–*y* plane and *z*–axis made 3D cell segmentation and tracking challenging. Deep learning techniques, such as cell segmentation, have proven effective for image processing (Ronneberger *et al*., 2015; Van Valen *et al*., 2016). However, they have not been used

for cell tracking because of the difficulty in preparing a sufficient amount of annotated tracking data (Moen *et al*., 2019). Recently, we solved this problem by simulating cell movements in a 3D space and developed 3DeeCellTracker, a deep learning-based software that can flexibly track cell movements in 3D time-lapse images, under various imaging conditions (Wen *et al*., 2021). By applying deep learning to cell tracking, our method substantially improves the flexibility and accuracy of cell tracking. Our software can be applied readily to track up to ~1,000 cells on a desktop computer equipped with a CUDA-enabled graphics processing unit (GPU), within a relatively short runtime. 3DeeCellTracker can be used for tracking cells with large movements in 3D + T images. On the other hand, it cannot track dividing cells because our algorithm assumes that the number of cells does not change. Moreover, it is not suitable for tracking images where individual cells move independently. Herein, we explain the process to install and use our software to process cell images. For more information, readers can refer to our original paper, together with the README.md file and video tutorials from our GitHub repository (https://github.com/WenChentao/3DeeCellTracker/).

## Software

1. Software list
   a. 3DeeCellTracker v0.4.1 (Chentao Wen, https://github.com/WenChentao/3DeeCellTracker/)
   b. ImageJ (Wayne Rasband, https://imagej.nih.gov/ij/)
   c. ITK-SNAP (Paul Yushkevich and Guido Gerig, http://www. itksnap.org)
2. Datasets for demonstration
   The demo data and the pretrained models used in this protocol can be downloaded from https://osf.io/dt76c/.
   See Procedure section Steps A1, A3c-i, B1, B2c, B2e-i for where to store these data and models.

## Equipment

**Computational Requirements**

The users need to prepare a desktop computer with a CUDA-enabled GPU required for the deep learning. A large RAM (at least 16GB) and sufficient hard disk space to store the image data and the segmentation/tracking results are also strongly recommended. In this protocol, we tested our software under Ubuntu 16.04 using a desktop computer with the GPU NVIDIA GeForce GTX 1080. For other OS and GPU, the following procedures for installation may need some modifications.

**Install the computational environment**

1. Ubuntu
   The installer and the installation guide can be found here: https://ubuntu.com/download/desktop.
2. GPU driver
   a. Procedures for installation
      To install nvidia-418.56, which supports the GPU NVIDIA GeForce GTX 1080 in our PC, run the following commands in the Ubuntu terminal (pressing Ctrl + Alt + T to open it):

```
$ sudo add-apt-repository ppa:graphics-drivers
$ sudo apt-get update
$ sudo apt-get install nvidia-418
```

      After restarting the computer, the driver should have been correctly installed. The users can check the installation by a command:

```
$ nvidia-smi
```

If the driver has been installed correctly, the driver version will be displayed.

b. Important information
For different GPU models and/or OS, the users need to install the proper GPU drivers according to the information below:
i. Install a proper NVIDIA GPU driver from the official website: https://www.geforce.com/drivers.
ii. Or choose a proper driver in Ubuntu: https://phoenixnap.com/kb/install-nvidia-drivers-ubuntu.

3. Prerequisite packages used by 3DeeCellTracker
This section is for old NVIDIA GPU models, such as GTX 1080. For the latest GPU models such as RTX 30 series, see section 4.
a. Install Anaconda
We recommend that users install Anaconda to manage the computational environments and python packages conveniently.
Download the Anaconda installer here: https://www.anaconda.com/products/individual, and install it by following the command (replace the path and filename with the ones of the downloaded installer):

```
$ bash ~/Downloads/Anaconda_xxx.sh
```

Then re-start the terminal.
b. Install prerequisite packages (see the video tutorial 1 in our GitHub repository)
The users can install the prerequisite packages listed in the 3DCT.yml file prepared by us.
To do this, download our GitHub repository by clicking the "Code/Download ZIP" in https://github.com/WenChentao/3DeeCellTracker, and extract the downloaded file to a local directory. Then, open the terminal and change the current working directory to the extracted directory by the "cd" command. Finally, run the following command to create a new environment named 3DCT:

```
$ conda env create -f 3DCT.yml
```

The created environment 3DCT can be activated by the following command:

```
$ conda activate 3DCT
```

4. Prerequisite packages used by 3DeeCellTracker with latest GPU models
Some of the latest GPU models require the latest versions of CUDA, cuDNN, and TensorFlow, which currently cannot be simply installed using conda commands. Thus, the users need to manually install them. For these latest GPU models, such as NVIDIA GeForce RTX 30 series, we recommend that the users install the latest TensorFlow (≥2.4.0), after installing CUDA 11.x and cuDNN 8.x compatible with the TensorFlow version. The guides for these installations are listed below:
a. Choose the appropriate versions of TensorFlow and CUDA, cuDNN compatible with each other:
https://www.tensorflow.org/install/source#gpu
b. The guide for installing CUDA in Linux:
https://docs.nvidia.com/cuda/archive/11.2.0/cuda-installation-guide-linux/index.html
c. The guide for installing cuDNN in Linux:
https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#install-linux
d. To install libraries, including tensorflow, opencv-python, scikit-image, scikit-learn, matplotlib, and 3DeeCellTracker, the users should first install Anaconda (see section 3a). After that, the users can create a conda environment, activate it, and install the pip tool in the terminal with the following commands:

```
$conda create -name 3DCT
$conda activate 3DCT
$conda install pip
```

Then, the users can install the libraries mentioned above (and other libraries, if 3DeeCellTracker reports errors that they are not installed automatically) with the pip commands (replace the package_name with each name mentioned above):

```
$ pip install package_name
```

## Procedure

The detailed procedures of how to train the 3D U-Net and how to track cells are listed below. All programs (Jupyter notebooks), demo data, and the expected running results related to the following procedures can be found in the README.md file in our GitHub repository.

### A. **Train a 3D U-Net model** (see the video tutorial 1 in our GitHub repository)

1. Prepare the data for training 3D U-Net
   Before training the 3D U-Net, the users need to prepare two volumes of 3D images, together with the annotations of the cell's regions. The annotations can be made using ITK-SNAP. See **the video tutorial 3 in our GitHub repository, for how to annotate cell regions.** The first volume of image/annotation is used to optimize the weights of the 3D U-Net (training data), and the second volume of image/annotation is used to evaluate the accuracy of the 3D U-Net during training (validation data).
   For demonstration, we have prepared the training data and validation data (unet_training.zip), which can be downloaded from https://osf.io/dt76c/. The users can check their contents in ImageJ after extracting them to a local directory.
2. Launch the training program
   The training program "3D_U_Net_training-clear.ipynb" notebook is stored in the "3DeeCellTracker-masters/Examples/" folder in the downloaded GitHub repository (see Equipment: Install the computational environment—step 3b). After activating the newly created environment "3DCT", run following command to launch Jupyter Notebook:
   $ jupyter notebook
   Then find the notebook "3D_U_Net_training-clear.ipynb" and open it.
3. Run the training program
   In the opened notebook, the users can click the "Run" button or press "Shift + Enter" to run the multiline python code in a cell (To avoid confusion, we will use "code" below when mentioning a code cell). To use our program, modify parameters and run each code according to the following procedures (the alphabet numbers below are corresponding to the ones in the headings in the notebook).
   a. Import packages
      Run this code to import packages used in this notebook.
   b. **Initialize the parameters for training**
      After modifying the following three parameters, run the code:
      i. **noise_level:** As a starting point, set the value to be the intensity of the background pixels in the raw image. If the two images (train and validation) have different background intensities, set it to an intermediate value. A higher value will only include bright regions, while a lower value will enhance those regions with weak intensities. A proper value should enhance the intensities of weak cell regions, but not the background regions.
      ii. **folder_path:** This path is used to create a directory (if one does not exist) to save the training/validation data and the trained model. We recommend the users to set it as "./folder_name" (replace "folder_name" with a custom name), to create a directory with the custom name under the directory containing the current notebook file. For example, in this

demonstration, we set it as "./unet_01", which created the directory "unet_01" under "3DeeCellTracker-masters/Examples/".

iii. **model:** This should be a predefined 3D U-Net model. The simplest way is to use the default value unet3_a(). Advanced users can select other predefined models, such as unet3_b(), unet3_c() [described in Figure 2—figure supplement 1 in our eLife paper (Wen *et al*., 2021)], or a custom model defined by the users (you need to import the model in the "Import packages" code before modifying this parameter).

c. **Load the train/validation datasets**

i. In Step A3b, the program has automatically generated a working directory "unet_01" with the default parameters. The users should move the prepared image and annotation of the training data to the subfolders of "unet_01": "train_image" and "train_label", respectively, and move image and annotation of the validation data to the subfolders "valid_image" and "valid_label", respectively.

ii. Run the code to load and show the images/annotations of the training and validation datasets by max-projection.

d. **Pre-process the datasets**

i. **d1. Run the code and confirm the normalized images (Left).** If the normalization result is not satisfactory (*e.g*., cells are too weak, or background is too bright), go back to step B, modify "noise_level", and run the codes from there again.

ii. **d2.** Run the code to divide the images into smaller sub-images (to fit the input size of the 3D U-Net) and show a part of these sub-images.

e. **Train the 3D U-Net**

Run the code to start training. By default, the training will last for 100 epochs, which usually takes ~2 h on our desktop computer. The training time also depends on the image size of the training data (here 512 × 1024 × 21) and the architecture of the 3D U-Net used (here "unet3_a()"). After each epoch, the program will evaluate the accuracy of the trained 3D U-Net in validation data. If the accuracy is improved, the prediction of cell regions will be shown.

f. **Select the best weights and save the model**

i. After training, select the step number of the last figure, which had generated the most accurate prediction in the validation data. Or the users can visually inspect the figures to select the best step.

ii. Set the parameter "step" and run the code. The program will save the 3D U-Net model with the selected weights in the "models" folder with the name "unet3_pretrained.h5".

4. Close Jupyter Notebook

## B. Track the cells (single mode) (see the video tutorial 2 in our GitHub repository)

1. Prepare the data and models for tracking

Before starting the tracking, the users need to prepare the following data and pre-trained models: 1. The data of 3D time lapse images; 2. The 3D U-Net model pre-trained by the user's own image data obtained under the same conditions; 3. The FFN model pre-trained by the authors.

We have supplied the images and the pre-trained 3D U-Net for this demonstration, and the pre-trained FFN suitable for all conditions. The users can download them from https://osf.io/dt76c/.

2. Run the tracking program

a. Import packages

After launching the notebook "single_mode_worm1-clear.ipynb" in the folder "3DeeCellTracker-masters/Examples/" in the 3DCT environment, run this code to import the necessary packages.

b. Initialize the parameters for tracking

Run this code after modifying the following four groups of parameters:

i. Image parameters

1) **volume:** Number of the volumes (*i.e*., time points) of the 3D time lapse images to be tracked

2) **siz_xyz:** Sizes of each 3D image. Shape: (height, width, depth). Unit: voxels.

3) **z_xy_ratio:** The resolution (length/voxels) ratio between the z axis (depth) and the x-y plane.
4) **Z_scaling:** The scaling factor for interpolation along the z axis. Should be an integer greater than or equal to 1. If equal to 1, no interpolation will be applied. A higher value will lead to more accurate segmentation/tracking results, but also increase the runtime. We recommend the users set it to 5 or 10 if "z_xy_ratio" is greater than 1, or set it to 1 if z_xy_ratio=1.

ii. Segmentation parameters
1) **noise_level:** The program uses this value to enhance the cell regions with weak intensities. Adjust this value to enhance cell regions but not background regions. As a starting point, set it to the mean intensities of the background regions in the raw images.
2) **min_size:** Adjust this value so that the program will remove the regions whose sizes (unit: voxels) are smaller than this value, which may be non-cell artifacts.

*Note: To avoid re-calculation, our program saves the cached segmentations in the "unet_cache" folder. So, if the users need to change the segmentation parameters later, run "tracker.set_segmentation (noise_level=xx, min_size=xx)" instead of modifying it here (see descriptions below for the code d1), which will delete the cached segmentations before re-segmentation.*

iii. Tracking parameters
1) **beta_tk:** Related to the coherency of the predicted cell movements. A higher value will make more coherent predictions, while a lower value will make more independent predictions of cell movements. As a starting point, the user could set it to 300.
2) **lambda_tk:** Also related to the coherency of the predicted cell movements (higher: more coherent; lower: more independent). As a starting point, the user could set it to 0.1.
3) **maxiter_tk:** The number of iterations of "FFN+PR-GLS". A higher value will lead to more accurate predictions, but will also increase the runtime. As a starting point, the user could set it to 20.

iv. Paths
1) **folder_path:** The path of the directory to be created to save the data, model, and results. Set it as "./folder_name" (replace "folder_name" with a custom name) to create a directory with the specified name under the directory containing the current notebook.
2) **image_name:** The filenames of the images to be tracked.
Set it according to the format of the filenames, where the time index should come before the layer index. For example, for images with names like: "image_t0002_z011.tif", "image_t1502_z101.tif", *etc*., the user should set image_name = "image_t%04i_z%03i.tif", whereby "%04i" and "%03i" indicate a 4-digit integer and a 3-digit integer, respectively.
3) **unet_model_file:** The filename of the pre-trained 3D U-Net model described in Step B1.
4) **ffn_model_file:** The filename of the pre-trained FFN model described in Step B1.

c. Prepare images to be tracked, and the pre-trained U-Net and FFN models.
In the Step B2b, the program has automatically generated a working directory "worm1" (with the default parameter); we used this name because the demonstration data used here are worm's neurons. The users should move the prepared 3D time-lapse images to the folder "worm1/data", and move the pre-trained 3D U-Net and FFN model files to the folder "worm1/models".

d. Optimize segmentation parameters and segment the image at volume 1.
i. **d1. Modify the segmentation parameters**
This step can be skipped the first time. If the segmentation result below is poor, return here, modify these parameters, and run this code again.
ii. **d2. Segment cells at volume 1**
Run this code to segment the cells in volume 1.
iii. **d3. Draw the results of segmentation (Max projection)**

Run this code to show the segmentation result including the raw image (top left), the cell-like regions (top right), and the individual cells (bottom left). All images are shown with max projection.

iv. **d4. Show segmentation in each layer**

Run this code to show the segmentation result in each layer.

If the results in d3 and/or d4 are not satisfactory (*e.g.*, Some cells are not detected, or too many artifacts are included), go back to d1 to modify the parameters and run the codes from there again.

e. **Manually correct the segmentation at volume 1 and load it.**

i. **e1. Manual correction**

The automatically generated segmentation in volume 1 has been saved in the folder "auto_vol1". The users should correct mistakes in it using another software (We used ITK-SNAP. See our video tutorial 4 for how to use it, whose links can be found in our GitHub repository). In this step, the users should focus on correcting the artifacts, the missed cells, and the incorrect separation of cell regions. On the other hand, the cell boundaries do not need to be accurately corrected because the program will smoothen the boundaries later (see e4). In our experience, to correct 100-200 cells, 2-3 h will be enough. After correction, save the results as image sequence into the folder "manual_vol1".

ii. **e2. Load the manually corrected segmentation**

Run the code to load the manually corrected segmentation.

iii. **e3. Re-train the U-Net using the manual segmentation (optional)**

This step can be skipped if the segmentation is good enough (*i.e.*, all cells of interest have been detected, and most of them are correctly separated).

In e3, there are two code cells. To retrain the 3D U-Net, run the first code. By default, the training will last for 10 epochs (steps).

After training, select the step within the last figure, or visually inspect the figures to select the most accurate step, and then run the second code. If the prediction is not improved, set step = 0 to restore the initial model.

iv. **e4. Interpolate cells to make more accurate/smooth cell boundary**

Run the code to interpolate/smoothen the segmentation.

v. **e5. Initialize variables required for tracking**

Run the code to initialize the variables required for tracking.

f. Optimize tracking parameters.

i. **f1. Modify tracking parameters if the test result is not satisfactory (optional)**

This step could be skipped the first time. If the result of the tracking test below is poor, return here, modify these parameters, and run this code.

ii. **f2. Test a matching between volume 1 and a target volume, and show the FFN + PR-GLS process by an animation (five iterations)**

Set the target_volume and run the code to match the cells in volume 1 with the cells in the target volume. As a starting point, the user can set target volume = 2. Other target volumes can also be tried in order to test the performance under difficult conditions.

After the tracking is finished, an animation will show the matching results within all five iterations. If the parameters have been set properly, the cells in volume 1 should move to the corresponding positions in the target volume after 1-5 iterations.

iii. **f3. Show the accurate correction after the FFN + PR-GLS transformation**

Run the code to show the tiny correction of the cell positions.

iv. **f4. Show the superimposed cells + labels before/after tracking**

Run the code to show the final prediction for cell positions. If the tracking test is successful, the predicted cell positions (shown with colours) and the cell-like regions (gray) detected by 3D U-Net will overlap in the bottom figures. In addition, the spatial patterns of the cell should also be maintained between volume 1 and the target volume.

If the tracking test includes obvious mistakes, go back to f1 to modify the parameters and run the codes following from there again.

g. Track following volumes.
   i. **Track and show the processes**
   Run the code to track all the following volumes. After each volume is tracked, the tracking results between the last volume and this volume will be shown in a figure.
   The tracking results (image sequence of moving labels) in each volume will be saved in the folder "track_results_SingleMode" for further analyses.
   ii. **Show the processes as an animation (for diagnosis)**
   After all volumes have been tracked, run this code to confirm the tracking results in each volume.

### C. Track the cells (ensemble mode)

Launching the notebook "ensemble_mode_worm4-clear.ipynb" in the folder "3DeeCellTracker-masters/Examples/" in 3DCT environment, and run it to track the cells in ensemble mode.
The procedures for using the ensemble mode are basically the same as in the single mode, except that the user needs to add a parameter "ensemble = 20" (or other integers greater than 1). Then the program will predict cell positions as the average of 20 predictions from different reference volumes.

## Acknowledgments

We thank Yuto Endo, Ryoga Suzuki and the other Kimura laboratory members for providing suggestions and comments about this manuscript.
Funding sources: This work was supported by Japan Society for the Promotion of Science (KAKENHI JP16H06545, JP20H05700 to K.D.K.), Grant-in-Aid for Research in Nagoya City University (48, 1912011, 1921102), the Joint Research by National Institutes of Natural Sciences (01112002), and RIKEN Center for Advanced Intelligence Project (to K.D.K).
Original research paper from which this protocol was derived: Wen *et al.* (2021).

## Competing interests

The authors declare that no competing interests exist.

## References

Moen, E., Bannon, D., Kudo, T., Graf, W., Covert, M. and Van Valen, D. (2019). Deep learning for cellular image analysis. *Nat Methods* 16(12): 1233-1246.
Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention—MICCAI*. Springer.
Van Valen, D. A., Kudo, T., Lane, K. M., Macklin, D. N., Quach, N. T., DeFelice, M. M., Maayan, I., Tanouchi, Y., Ashley, E. A. and Covert, M. W. (2016). Deep Learning Automates the Quantitative Analysis of Individual Cells in Live-Cell Imaging Experiments. *PLoS Comput Biol* 12(11): e1005177.
Wen, C., Miura, T., Voleti, V., Yamaguchi, K., Tsutsumi, M., Yamamoto, K., Otomo, K., Fujie, Y., Teramoto, T., Ishihara, T., *et al*. (2021). 3DeeCellTracker, a deep learning-based pipeline for segmenting and tracking cells in 3D time lapse images. *eLife* 10: e59187.