cellpose

Release 3.0.7-19-g0ce3653

Carsen Stringer & Marius Pachitariu

Apr 09, 2024

BASICS:

1 Installation **3**
1.1 Built-in model directory . . . 3
1.2 M1 Mac installation . . . 3
1.3 AMD GPU ROCm installation . . . 4
1.4 Common issues . . . 4
1.5 Dependencies . . . 5

2 GUI **7**
2.1 Starting the GUI . . . 7
2.2 Using the GUI . . . 8
2.3 Drawing masks . . . 9
2.4 Bulk Mask Deletion . . . 9
2.5 Segmentation options . . . 9
2.6 Training your own cellpose model . . . 10
2.7 Contributing training data . . . 10
2.8 Keyboard shortcuts . . . 11

3 Inputs **13**
3.1 3D segmentation . . . 13

4 Settings **15**
4.1 Channels . . . 15
4.2 Diameter . . . 16
4.3 Resample . . . 16
4.4 Flow threshold . . . 16
4.5 Cellprob threshold . . . 17
4.6 Number of iterations niter . . . 17
4.7 3D settings . . . 17

5 Outputs **19**
5.1 in a notebook . . . 19
5.2 _seg.npy output . . . 19
5.3 PNG output . . . 20
5.4 Native ImageJ ROI archive output . . . 20
5.5 (Legacy ImageJ Interface) ROI manager compatible output for ImageJ . . . 21
5.6 Plotting functions . . . 21

6 Models **23**
6.1 Full built-in models . . . 23
6.1.1 Cytoplasm model (`'cyto3'`, `'cyto2'`, `'cyto'`) . . . 23
6.1.2 Nucleus model (‘*nuclei*’) . . . 24

6.2 Other built-in models 24
6.3 User-trained models 24

7 Image Restoration 25
7.1 DenoiseModel 25
7.2 CellposeDenoiseModel 26
7.3 Command line usage 26

8 Training 27

9 OpenVINO 31

10 FAQ 33

11 In a notebook 35

12 Command line 37
12.1 Command Line Usage 37

13 Cellpose API Guide 39
13.1 Cellpose class 39
13.2 CellposeModel 41
13.3 CellposeDenoiseModel 44
13.4 DenoiseModel 46
13.5 SizeModel 48
13.6 Training 50
13.7 Metrics 53
13.8 Flows to masks 54
13.9 Image transforms 60
13.10 Plot functions 66
13.11 I/O functions 68
13.12 Utils functions 72
13.13 Network classes 77
13.14 Core functions 80
13.15 All models functions 82

14 Cellpose CLI 91
14.1 Command Line Usage 91
14.1.1 Named Arguments 92
14.1.2 Hardware Arguments 92
14.1.3 Input Image Arguments 92
14.1.4 Model Arguments 93
14.1.5 Algorithm Arguments 93
14.1.6 Output Arguments 94
14.1.7 Training Arguments 95

Python Module Index 97

Index 99

cellpose is an anatomical segmentation algorithm written in Python 3 by Carsen Stringer and Marius Pachitariu. For support, please open an issue.

We make pip installable releases of cellpose, here is the pypi. You can install it as `pip install cellpose[gui]`.

You can try it out without installing at cellpose.org. Also check out these resources:

Cellpose3: one-click image restoration for improved cellular segmentation

- paper on biorxiv
- thread

Cellpose 2.0: how to train your own model

- paper on biorxiv
- talk
- twitter thread
- human-in-the-loop training protocol video

Cellpose: a generalist algorithm for cellular segmentation

- paper on biorxiv (see figure 1 below) and in nature methods
- twitter thread
- Marius's talk

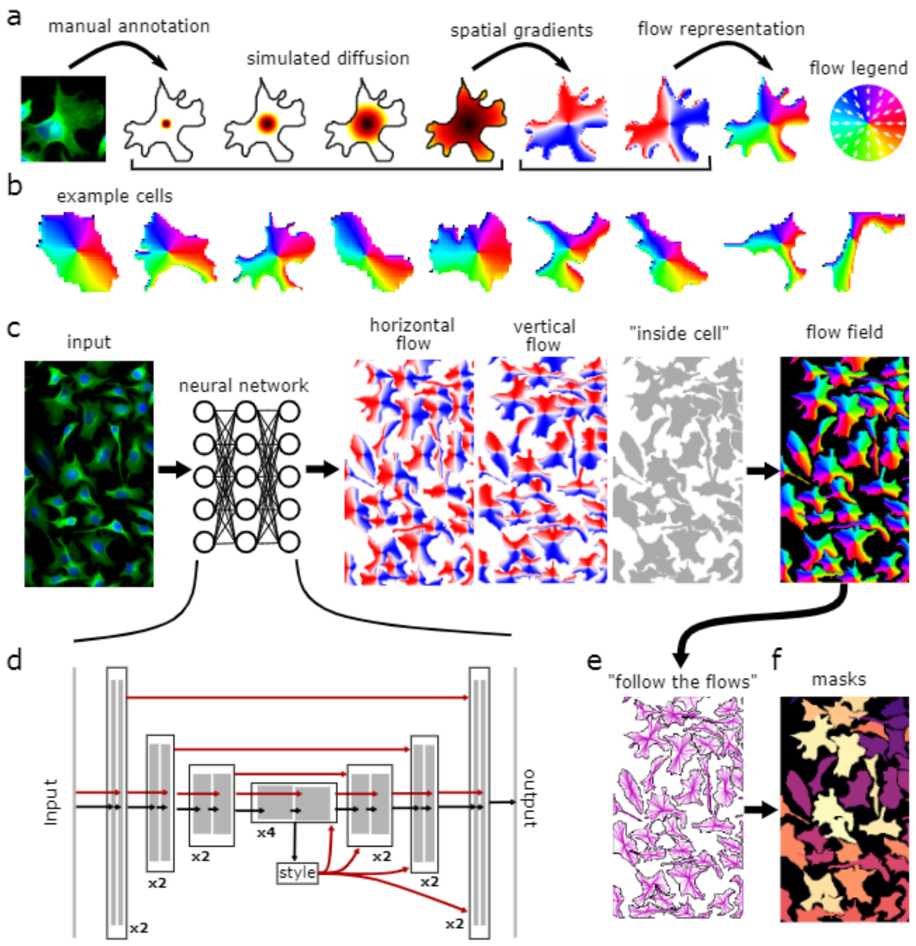


Figure 1: Model architecture. a, Procedure for transforming manually annotated masks into a vector flow representation that can be predicted by a neural network. A simulated diffusion process started at the center of the mask is used to derive spatial gradients that point towards the center of the cell, potentially indirectly around corners. The X and Y gradients are combined into a single normalized direction from 0° to 360°. **b,** Example spatial flows for cells from the training dataset. **cd,** A neural network is trained to predict the horizontal and vertical flows, as well as whether a pixel belongs to any cell. The three predicted maps are combined into a flow field. **d** shows the details of the neural network which contains a standard backbone neural network that downsamples and then upsamples the feature maps, contains skip connections between layers of the same size, and global skip connections from the image styles, computed at the lowest resolution, to all the successive computations. **e,** At test time, the predicted flow fields are used to construct a dynamical system with fixed points whose basins of attraction represent the predicted masks. Informally, every pixel "follows the flows" along the predicted flow fields towards their eventual fixed point. **f,** All the pixels that converge to the same fixed point are assigned to the same mask.

CHAPTER

ONE

INSTALLATION

For basic install instructions, look up the main github readme.

1.1 Built-in model directory

By default, the pretrained cellpose models are downloaded to `$HOME/.cellpose/models/`. This path on linux would look like `/home/USERNAME/.cellpose/`, and on Windows, `C:/Users/USERNAME/.cellpose/models/`. These models are downloaded the first time you try to use them, either on the command line, in the GUI or in a notebook.

If you'd like to download the models to a different directory, and are using the command line or the GUI, before you run `python -m cellpose ...`, you will need to always set the environment variable `CELLPOSE_LOCAL_MODELS_PATH` (thanks Chris Roat for implementing this!).

To set the environment variable in the command line/Anaconda prompt on windows run the following command modified for your path: `set CELLPOSE_LOCAL_MODELS_PATH=C:/PATH_FOR_MODELS/`. To set the environment variable in the command line on linux, run `export CELLPOSE_LOCAL_MODELS_PATH=/PATH_FOR_MODELS/`.

To set this environment variable when running cellpose in a jupyter notebook, run this code at the beginning of your notebook before you import cellpose:

```
import os
os.environ["CELLPOSE_LOCAL_MODELS_PATH"] = "/PATH_FOR_MODELS/"
```

1.2 M1 Mac installation

Please use the instructions provided on *image.sc <https://forum.image.sc/t/cellpose-on-macos-m1-pro-apple-silicon-arm64/68018/4>* by Peter Sobolewski. From the command line you can choose the Mac device with

```
python -m cellpose --dir path --gpu_device mps --use_gpu
```

1.3 AMD GPU ROCm installation

As an alternative to the CUDA acceleration for NVIDIA GPUs, you can use the ROCm acceleration for AMD GPUs. This is not yet supported on Windows, but is supported on Linux. Installation instructions are available here. Just like the NVIDIA CUDA installation, you will need to install the ROCm drivers first and then install Cellpose. Be warned that the ROCm project is significantly less mature than CUDA, and you may run into issues.

Warning: The ROCm acceleration is not yet supported on Windows, and is only supported on Linux. If you are on Windows, you will need to use CUDA acceleration.

Warning: ROCm is significantly less mature than the CUDA acceleration, and you may run into issues.

1.4 Common issues

If you are having issues with CUDA on Windows, or want to use Cuda Toolkit 10, please follow these instructions:

```
conda create -n cellpose pytorch=1.8.2 cudatoolkit=10.2 -c pytorch-lts
conda activate cellpose
pip install cellpose
```

If you receive the error: `No module named PyQt5.sip`, then try uninstalling and reinstalling pyqt5

```
pip uninstall pyqt5 pyqt5-tools
pip install pyqt5 pyqt5-tools pyqt5.sip
```

If you are having other issues with the graphical interface and QT, see some advice here .

If you have errors related to OpenMP and libiomp5, then try

::

    conda install nomkl

If you receive an error associated with **matplotlib**, try upgrading it:

```
pip install matplotlib --upgrade
```

If you receive the error: `ImportError:  _arpack DLL load failed`, then try uninstalling and reinstalling scipy

```
pip uninstall scipy
pip install scipy
```

If you are on Yosemite Mac OS or earlier, PyQt doesn't work and you won't be able to use the graphical interface for cellpose. More recent versions of Mac OS are fine. The software has been heavily tested on Windows 10 and Ubuntu 18.04, and less well tested on Mac OS. Please post an issue if you have installation problems.

1.5 Dependencies

cellpose relies on the following excellent packages (which are automatically installed with pip if missing):

- pytorch
- pyqtgraph
- PyQt5 or pyside or PyQt6
- numpy (>=1.16.0)
- numba
- scipy
- tifffile
- natsort
- fastremap
- roifile
- superqt

CHAPTER

TWO

GUI

2.1 Starting the GUI

The quickest way to start is to open the GUI from a command line terminal. You might need to open an anaconda prompt if you did not add anaconda to the path:

```
python -m cellpose
```

The first time cellpose runs it downloads the latest available trained model weights from the website.

You can **drag and drop** images (.tif, .png, .jpg, .gif) into the GUI and run Cellpose, and/or manually segment them. When the GUI is processing, you will see the progress bar fill up and during this time you cannot click on anything in the GUI. For more information about what the GUI is doing you can look at the terminal/prompt you opened the GUI with. For example data, see cellpose website. For best accuracy and runtime performance, resize images so cells are less than 100 pixels across.

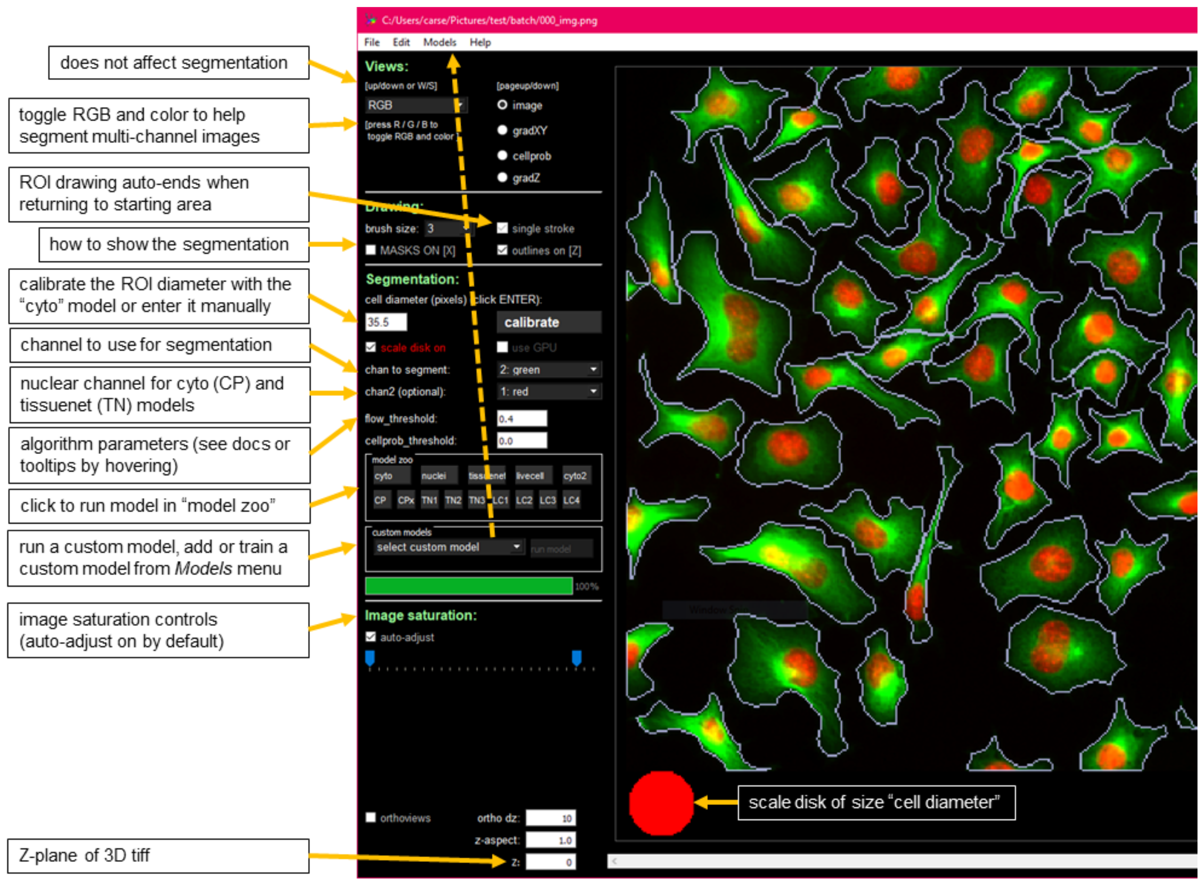
For multi-channel, multi-Z tiff's, the expected format is Z x channels x Ly x Lx.

For multi-Z 3D data, please use the 3D version of the GUI:

```
python -m cellpose --Zstack
```

Note: The output file with the masks is in the same folder as the loaded image with `_seg.npy` appended. The GUI automatically saves after you draw an ROI but NOT after running a model for segmentation and NOT after 3D mask drawing (too slow). Save in the file menu or with Ctrl+S.

Note: Since the output file is in the same folder as the loaded image with `_seg.npy` appended, make sure you have WRITE access in the folder, otherwise the file will not save.


does not affect segmentation
toggle RGB and color to help segment multi-channel images
ROI drawing auto-ends when returning to starting area
how to show the segmentation
calibrate the ROI diameter with the "cyto" model or enter it manually
channel to use for segmentation
nuclear channel for cyto (CP) and tissuenet (TN) models
algorithm parameters (see docs or tooltips by hovering)
click to run model in "model zoo"
run a custom model, add or train a custom model from *Models* menu
image saturation controls (auto-adjust on by default)
Z-plane of 3D tiff
scale disk of size "cell diameter"
C:/Users/carse/Pictures/test/batch/000_img.png
File Edit Models Help
Views:
[up/down or W/S]
[pageup/down]
RGB
image
gradXY
cellprob
gradZ
[press R / G / B to toggle RGB and color]
Drawing:
brush size: 3
single stroke
MASKS ON [X]
outlines on [Z]
Segmentation:
cell diameter (pixels) (click ENTER):
35.5
calibrate
scale disk on
use GPU
chan to segment:
2: green
chan2 (optional):
1: red
flow_threshold:
0.4
cellprob_threshold:
0.0
model zoo
cyto
nuclei
tissuenet
livecell
cyto2
CP CPx TN1 TN2 TN3 LC1 LC2 LC3 LC4
custom models
select custom model
run model
100%
Image saturation:
auto-adjust
orthoviews
ortho dz: 10
z-aspect: 1.0
z: 0

2.2 Using the GUI

The GUI serves two main functions:

1. Running the segmentation algorithm.
2. Manually labelling data.
3. (NEW) Fine-tuning a pretrained cellpose model on your own data.

Main GUI mouse controls (works in all views):

- Pan = left-click + drag
- Zoom = scroll wheel (or +/= and - buttons)
- Full view = double left-click
- Select mask = left-click on mask
- Delete mask = Ctrl (or Command on Mac) + left-click
- Merge masks = Alt + left-click (will merge last two)
- Start draw mask = right-click
- End draw mask = right-click, or return to circle at beginning

2.3 Drawing masks

Masks are started with right-click, then hover your mouse (do not hold it down), and return it to the red circle to complete the mask. The mask should now be completed.

Overlaps in masks are NOT allowed. If you draw a mask on top of another mask, it is cropped so that it doesn't overlap with the old mask. Masks in 2D should be single strokes (if *single_stroke* is checked).

If you want to draw masks in 3D, then you can turn *single_stroke* option off and draw a stroke on each plane with the cell and then press ENTER. You can also draw multiple strokes on the same plane for complex cell shapes, but do not do this in 2D if you plan to train a cellpose model (the cell flows will not work correctly).

Note: 3D labelling will fill in unlabelled z-planes so that you do not have to densely label, for example you can skip some planes, and the cell will be interpolated between planes.

After each mask is drawn in 2D, it is saved to the `_seg.npy`. If this is slow (for large images), this "autosave" option can be turned off in the "File" menu ("Disable autosave _seg.npy file"). In 3D, the mask is never auto-saved, instead save masks by clicking CTRL+S, or "Save" in the "File" menu.

2.4 Bulk Mask Deletion

Clicking the 'delete multiple' button will allow you to select and delete multiple masks at once. Masks can be deselected by clicking on them again. Once you have selected all the masks you want to delete, click the 'done' button to delete them.

Alternatively, you can create a rectangular region to delete a regions of masks by clicking the 'delete multiple' button, and then moving and/or resizing the region to select the masks you want to delete. Once you have selected the masks you want to delete, click the 'done' button to delete them.

At any point in the process, you can click the 'cancel' button to cancel the bulk deletion.

2.5 Segmentation options

SIZE: you can manually enter the approximate diameter for your cells, or press "calibrate" to let the model estimate it. The size is represented by a disk at the bottom of the view window (can turn this disk off by unchecking "scale disk on").

use GPU: if you have installed the cuda version of mxnet, then you can activate this, but it won't give huge speedups when running single images in the GUI.

MODEL: there is a *cytoplasm* model and a *nuclei* model, choose what you want to segment

CHAN TO SEG: this is the channel in which the cytoplasm or nuclei exist

CHAN2 (OPT): if *cytoplasm* model is chosen, then choose the nuclear channel for this option

2.6 Training your own cellpose model

Check out this video to learn the process.

1. Drag and drop an image from a folder of images with a similar style (like similar cell types).
2. Run the built-in models on one of the images using the "model zoo" and find the one that works best for your data. Make sure that if you have a nuclear channel you have selected it for CHAN2.
3. Fix the labelling by drawing new ROIs (right-click) and deleting incorrect ones (CTRL+click). The GUI autosaves any manual changes (but does not autosave after running the model, for that click CTRL+S). The segmentation is saved in a `_seg.npy` file.
4. Go to the "Models" menu in the File bar at the top and click "Train new model..." or use shortcut CTRL+T.
5. Choose the pretrained model to start the training from (the model you used in #2), and type in the model name that you want to use. The other parameters should work well in general for most data types. Then click OK.
6. The model will train (much faster if you have a GPU) and then auto-run on the next image in the folder. Next you can repeat #3-#5 as many times as is necessary.
7. The trained model is available to use in the future in the GUI in the "custom model" section and is saved in your image folder.

If you have **3D** data, please save random XY, YZ and XZ slices through your 3D data, ideally sufficiently spaced from each other so the information each slice has is distinct. Then put these slices into a folder and start the human-in-the-loop training. You can then use the new custom model on new 3D data.

Note: You can only start training with one of the built-in Cellpose models or from scratch. When you start training from a built-in model or from scratch each time, then you are training the network on all the previously labelled images in the folder and weighting them equally in your training set.

If you restart from a previous retraining, you are biasing the network towards the earlier images it has already been trained on. Conversely, if you have created a custom model with different images, and you retrain that model, then you are downweighting the images that you have already trained on and excluded from your new training set. Therefore, we recommend having all images that you want to be trained for the same model in the same folder so they are all used.

See the Models doc for info on the new model zoo and suggestion mode.

2.7 Contributing training data

We are very excited about receiving community contributions to the training data and re-training the cytoplasm model to make it better. Please follow these guidelines:

1. Run cellpose on your data to see how well it does. Try varying the diameter, which can change results a little.
2. If there are relatively few mistakes, it won't help much to contribute labelled data.
3. If there are consistent mistakes, your data is likely very different from anything in the training set, and you should expect major improvements from contributing even just a few manually segmented images.
4. For images that you contribute, the cells should be at least 10 pixels in diameter, and there should be **at least** several dozens of cells per image, ideally ~100. If your images are too small, consider combining multiple images into a single big one and then manually segmenting that. If they are too big, consider splitting them into smaller crops.

5. For the manual segmentation, please try to outline the boundaries of the cell, so that everything (membrane, cytoplasm, nucleus) is inside the boundaries. Do not just outline the cytoplasm and exclude the membrane, because that would be inconsistent with our own labelling and we wouldn't be able to use that.
6. Do not use the results of the algorithm in any way to do contributed manual segmentations. This can reinforce a vicious circle of mistakes, and compromise the dataset for further algorithm development.

If you are having problems with the nucleus model, please open an issue before contributing data. Nucleus images are generally much less diverse, and we think the current training dataset already covers a very large set of modalities. Additionally, you can run a non-nuclear model on nuclear data such as cyto.

2.8 Keyboard shortcuts

Keyboard shortcuts	Description
CTRL+H	help
=/+ // -	zoom in // zoom out
CTRL+Z	undo previously drawn mask/stroke
CTRL+0	clear all masks
CTRL+L	load image (can alternatively drag and drop image)
CTRL+S	SAVE MASKS IN IMAGE to `_seg.npy` file
CTRL+T	start model training using `_seg.npy` files
CTRL+P	load `_seg.npy` file (note: it will load automatically with image if it exists)
CTRL+M	load masks file (must be same size as image with 0 for NO mask, and 1,2,3… for masks)
CTRL+N	save masks as PNG
CTRL+R	save ROIs to native ImageJ ROI format
CTRL+F	save flows to image file
A/D or LEFT/RIGHT	cycle through images in current directory
W/S or UP/DOWN	change color (RGB/gray/red/green/blue)
R / G / B	press to toggle RGB and Red or Green or Blue
PAGE-UP / PAGE-DOWN	change to flows and cell prob views (if segmentation computed)
X	turn masks ON or OFF
Z	toggle outlines ON or OFF
, / .	increase / decrease brush size for drawing

CHAPTER

THREE

INPUTS

You can use tiffs or PNGs or JPEGs. We use the image loader from scikit-image. Single plane images can read into data as nY x nX x channels or channels x nY x nX. Then the channels settings will take care of reshaping the input appropriately for the network. Note the model also rescales the input for each channel so that 0 = 1st percentile of image values and 1 = 99th percentile.

If you want to run multiple images in a directory, use the command line or a jupyter notebook to run cellpose.

3.1 3D segmentation

Tiffs with multiple planes and multiple channels are supported in the GUI (can drag-and-drop tiffs) and supported when running in a notebook. Multiplane images should be of shape nplanes x channels x nY x nX or as nplanes x nY x nX. You can test this by running in python

```
import tifffile
data = tifffile.imread('img.tif')
print(data.shape)
```

If drag-and-drop of the tiff into the GUI does not work correctly, then it's likely that the shape of the tiff is incorrect. If drag-and-drop works (you can see a tiff with multiple planes), then the GUI will automatically run 3D segmentation and display it in the GUI. Watch the command line for progress. It is recommended to use a GPU to speed up processing.

When running cellpose in a notebook, set `do_3D=True` to enable 3D processing. You can give a list of 3D inputs, or a single 3D/4D stack. When running on the command line, add the flag `--do_3D` (it will run all tiffs in the folder as 3D tiffs if possible).

If the 3D segmentation is not working well and there is inhomogeneity in Z, try stitching masks in Z instead of running `do_3D=True`. See details for this option here: stitch_threshold.

If drag-and-drop doesn't work because of the shape of your tiff, you need to transpose the tiff and resave to use the GUI, or use the napari plugin for cellpose, or run CLI/notebook and specify the `channel_axis` and/or `z_axis` parameters:

> `channel_axis` and `z_axis` can be used to specify the axis (0-based) of the image which corresponds to the image channels and to the z axis. For example an image with 2 channels of shape (1024,1024,2,105,1) can be specified with `channel_axis=2` and `z_axis=3`. If `channel_axis=None` cellpose will try to automatically determine the channel axis by choosing the dimension with the minimal size after squeezing. If `z_axis=None` cellpose will automatically select the first non-channel axis of the image to be the Z axis. These parameters can be specified using the command line with `--channel_axis` or `--z_axis` or as inputs to `model.eval` for the `Cellpose` or `CellposeModel` model.

CHAPTER

FOUR

SETTINGS

The important settings are described on this page. See the *Cellpose class* for all run options.

Here is an example of calling the Cellpose class and running a list of images for reference: ::code-block:

```
from cellpose import models
from cellpose.io import imread

# model_type='cyto' or model_type='nuclei'
model = models.Cellpose(gpu=False, model_type='cyto')

files = ['img0.tif', 'img1.tif']
imgs = [imread(f) for f in files]
masks, flows, styles, diams = model.eval(imgs, diameter=None, channels=[0,0],
                                         flow_threshold=0.4, do_3D=False)
```

You can make lists of channels/diameter for each image, or set the same channels/diameter for all images as shown in the example above.

4.1 Channels

There are two channels inputs. The first channel is the channel you want to segment. The second channel is an optional channel that is helpful in models trained with images with a nucleus channel. See more details in the models page.

1. 0=grayscale, 1=red, 2=green, 3=blue
2. 0=None (will set to zero), 1=red, 2=green, 3=blue

Set channels to a list with each of these elements, e.g. `channels = [0,0]` if you want to segment cells in grayscale or for single channel images, or `channels = [2,3]` if you green cells with blue nuclei.

On the command line the above would be `--chan 0 --chan2 0` or `--chan 2 --chan2 3`.

Note, if you set the first channel input to use grayscale `0`, then no nuclear channel will be used (the second channel will be filled with zeros).

4.2 Diameter

The cellpose models have been trained on images which were rescaled to all have the same diameter (30 pixels in the case of the *cyto* model and 17 pixels in the case of the *nuclei* model). Therefore, cellpose needs a user-defined cell diameter (in pixels) as input, or to estimate the object size of an image-by-image basis.

The automated estimation of the diameter is a two-step process using the *style* vector from the network, a 64-dimensional summary of the input image. We trained a linear regression model to predict the size of objects from these style vectors on the training data. On a new image the procedure is as follows.

1. Run the image through the cellpose network and obtain the style vector. Predict the size using the linear regression model from the style vector.
2. Resize the image based on the predicted size and run cellpose again, and produce ROIs. Take the final estimated size as the median diameter of the predicted ROIs.

For automated estimation set `diameter = None` or `diameter = 0`. However, if this estimate is incorrect please set the diameter by hand.

Changing the diameter will change the results that the algorithm outputs. When the diameter is set smaller than the true size then cellpose may over-split cells. Similarly, if the diameter is set too big then cellpose may over-merge cells.

4.3 Resample

The cellpose network is run on your rescaled image – where the rescaling factor is determined by the diameter you input (or determined automatically as above). For instance, if you have an image with 60 pixel diameter cells, the rescaling factor is 30./60. = 0.5. After determining the flows (dX, dY, cellprob), the model runs the dynamics. The dynamics can be run at the rescaled size (`resample=False`), or the dynamics can be run on the resampled, interpolated flows at the true image size (`resample=True`). `resample=True` will create smoother ROIs when the cells are large but will be slower in case; `resample=False` will find more ROIs when the cells are small but will be slower in this case. By default in versions >=1.0 `resample=True`.

The nuclear model in cellpose is trained on two-channel images, where the first channel is the channel to segment, and the second channel is always set to an array of zeros. Therefore set the first channel as 0=grayscale, 1=red, 2=green, 3=blue; and set the second channel to zero, e.g. `channels = [0,0]` if you want to segment nuclei in grayscale or for single channel images, or `channels = [3,0]` if you want to segment blue nuclei.

If the nuclear model isn't working well, try the cytoplasmic model.

4.4 Flow threshold

Note there is nothing keeping the neural network from predicting horizontal and vertical flows that do not correspond to any real shapes at all. In practice, most predicted flows are consistent with real shapes, because the network was only trained on image flows that are consistent with real shapes, but sometimes when the network is uncertain it may output inconsistent flows. To check that the recovered shapes after the flow dynamics step are consistent with real ROIs, we recompute the flow gradients for these putative predicted ROIs, and compute the mean squared error between them and the flows predicted by the network.

The `flow_threshold` parameter is the maximum allowed error of the flows for each mask. The default is `flow_threshold=0.4`. Increase this threshold if cellpose is not returning as many ROIs as you'd expect. Similarly, decrease this threshold if cellpose is returning too many ill-shaped ROIs.

4.5 Cellprob threshold

The network predicts 3 outputs: flows in X, flows in Y, and cell "probability". The predictions the network makes of the probability are the inputs to a sigmoid centered at zero (1 / (1 + e^-x)), so they vary from around -6 to +6. The pixels greater than the `cellprob_threshold` are used to run dynamics and determine ROIs. The default is `cellprob_threshold=0.0`. Decrease this threshold if cellpose is not returning as many ROIs as you'd expect. Similarly, increase this threshold if cellpose is returning too ROIs particularly from dim areas.

4.6 Number of iterations niter

The flows from the network are used to simulate a dynamical system governing the movements of the pixels. We simulate the dynamics for `niter` iterations. The pixels that converge to the same position make up a single ROI. The default `niter=None` or `niter=0` sets the number of iterations to be proportional to the ROI diameter. For longer ROIs, more iterations might be needed, for example `niter=2000`, for convergence.

4.7 3D settings

Volumetric stacks do not always have the same sampling in XY as they do in Z. Therefore you can set an `anisotropy` parameter to allow for differences in sampling, e.g. set to 2.0 if Z is sampled half as dense as X or Y.

There may be additional differences in YZ and XZ slices that make them unable to be used for 3D segmentation. I'd recommend viewing the volume in those dimensions if the segmentation is failing. In those instances, you may want to turn off 3D segmentation (`do_3D=False`) and run instead with `stitch_threshold>0`. Cellpose will create ROIs in 2D on each XY slice and then stitch them across slices if the IoU between the mask on the current slice and the next slice is greater than or equal to the `stitch_threshold`.

3D segmentation ignores the `flow_threshold` because we did not find that it helped to filter out false positives in our test 3D cell volume. Instead, we found that setting `min_size` is a good way to remove false positives.

CHAPTER

FIVE

OUTPUTS

5.1 in a notebook

when you run

```
from cellpose import io, models
img = io.imread("img.tif")
masks, flows, styles = models.CellposeModel(model_type='tissuenet_cp3').eval(img,
                                diameter=25, channels=[1,2])
```

Internally, the network predicts 3 (or 4) outputs: (flows in Z), flows in Y, flows in X, and cellprob. The predictions the network makes of cellprob are used as inputs to a sigmoid centered at zero (1 / (1 + e^-x)) in the loss function (binary cross-entropy loss), so they vary from around -10 to +10. These are output from the *eval* function as the second variable flows. The Y flows and X flows are used to simulate a dynamical system on the pixels, which is run on only pixels with a `cellprob > cellprob_threshold`. All pixels which converge to the same point are assigned the same label in the *masks* output, of size (Lz x) Ly x Lx (0 = NO ROI; 1,2,… = ROI labels). The styles are the sum over pixels of the output of the last downsampling layer of the network.

Cellpose also produces various outputs from the command line and the GUI, which are described below:

5.2 _seg.npy output

`*_seg.npy` files have the following fields:

- *filename* : filename of image
- *masks* : each pixel in the image is assigned to an ROI (0 = NO ROI; 1,2,… = ROI labels)
- *outlines* : outlines of ROIs (0 = NO outline; 1,2,… = outline labels)
- *chan_choose* : channels that you chose in GUI (0=gray/none, 1=red, 2=green, 3=blue)
- *ismanual* : element *k* = whether or not mask *k* was manually drawn or computed by the cellpose algorithm
- ***flows* :**
  - flows[0] is XY flow in RGB
  - flows[1] is the cell probability in range 0-255 instead of -10.0 to 10.0
  - flows[2] is Z flow in range 0-255 (if it exists, otherwise zeros),
  - flows[3] is [dY, dX, cellprob] (or [dZ, dY, dX, cellprob] for 3D), flows[4] is pixel destinations (for internal use)
- *est_diam* : estimated diameter / diameter used

- *zdraw* : for each mask, which planes were manually labelled (planes in between manually drawn have interpolated ROIs)

Note: the 'img' is no longer saved in the `*_seg.npy` file to save time.

Here is an example of loading in a `*_seg.npy` file and plotting masks and outlines

```
import numpy as np
from cellpose import plot, utils, io
dat = np.load('_seg.npy', allow_pickle=True).item()
img = io.imread('img.tif')

# plot image with masks overlaid
mask_RGB = plot.mask_overlay(img, dat['masks'],
                        colors=np.array(dat['colors']))

# plot image with outlines overlaid in red
outlines = utils.outlines_list(dat['masks'])
plt.imshow(img)
for o in outlines:
    plt.plot(o[:,0], o[:,1], color='r')
```

If you run in a notebook and want to save to a **_seg.npy* file, run

```
from cellpose import io
io.masks_flows_to_seg(images, masks, flows, file_name, channels=channels, diams=diams)
```

where each of these inputs is a list (as the output of *model.eval* is)

5.3 PNG output

You can save masks to PNG in the GUI.

To save masks (and other plots in PNG) using the command line, add the flag `--save_png`.

Or use the function below if running in a notebook

```
from cellpose import io
io.save_masks(images, masks, flows, image_names, png=True)
```

5.4 Native ImageJ ROI archive output

You can save the outlines of the ROIs in a ImageJ-native ROI archive file. Rather than using the legacy solution below, you can use this function to create an ROI archive file that can be opened in directly in ImageJ. Recent versions of ImageJ can autodetect the file format. Open in ImageJ using File > Open… and select the file. The ROIs will appear in the ROI manager.

To save the outlines using the CLI use the flag `--save_rois`.

To save the outlines using the API use the `save_rois` function in `io.py`:

This function is also available in the GUI.

```
from cellpose import io, utils

# image_name is file name of image
# masks is numpy array of masks for image
io.save_rois(masks, '<your_filename_string>')

# the file will be saved as '<your_filename_string>_rois.zip'
```

5.5 (Legacy ImageJ Interface) ROI manager compatible output for ImageJ

You can save the outlines of ROIs in a text file that's compatible with ImageJ ROI Manager in the GUI File menu.

To save using the command line, add the flag `--save_outlines`.

Or use the function below if running in a notebook

```
from cellpose import io, utils

# image_name is file name of image
# masks is numpy array of masks for image
base = os.path.splitext(image_name)[0]
outlines = utils.outlines_list(masks)
io.outlines_to_text(base, outlines)
```

To load this `_cp_outlines.txt` file into ImageJ, use the python script provided in cellpose: `imagej_roi_converter.py`. Run this as a macro after opening your image file. It will ask you to input the path to the `_cp_outlines.txt` file. Input that and the ROIs will appear in the ROI manager.

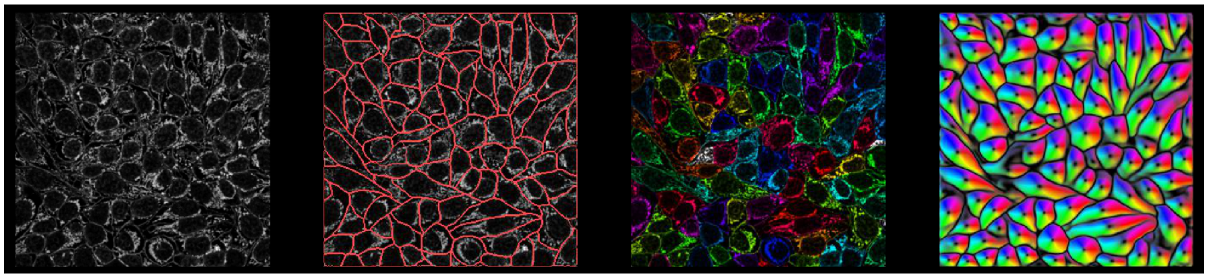
5.6 Plotting functions

In `plot.py` there are functions, like `show_segmentation`:

```
from cellpose import plot

nimg = len(imgs)
for idx in range(nimg):
    maski = masks[idx]
    flowi = flows[idx][0]

    fig = plt.figure(figsize=(12,5))
    plot.show_segmentation(fig, imgs[idx], maski, flowi, channels=channels[idx])
    plt.tight_layout()
    plt.show()
```



CHAPTER

SIX

MODELS

```
from cellpose import models
```

Each model will be downloaded automatically to your `models.MODELS_DIR` (see Installation instructions for more details on MODELS_DIR). You can also directly download a model by going to the URL, e.g.:

```
https://www.cellpose.org/models/MODEL_NAME
```

All built-in models were trained with the ROIs resized to a diameter of 30.0 (`diam_mean = 30`), except the *'nuclei'* model which was trained with a diameter of 17.0 (`diam_mean = 17`). User-trained models will be trained with the same `diam_mean` as the model they are initalized with. The models will internally take care of rescaling the images given a user-provided diameter (or with the diameter from auto-diameter estimation in full models).

6.1 Full built-in models

These models have Cellpose model weights and a size model. This means you can run with `diameter=0` or `--diameter 0` and the model can estimate the ROI size. However, we recommend that you set the diameter for your ROIs rather than having Cellpose guess the diameter.

These models can be loaded and used in the notebook with `models.Cellpose(model_type='cyto3')` or in the command line with `python -m cellpose --pretrained_model cyto3`.

We have a `nuclei` model and a super-generalist `cyto3` model. There are also two older models, `cyto`, which is trained on only the Cellpose training set, and `cyto2`, which is also trained on user-submitted images.

FYI we are no longer using the 4 different versions and `--net_avg` is deprecated.

6.1.1 Cytoplasm model (`'cyto3'`, `'cyto2'`, `'cyto'`)

The cytoplasm models in cellpose are trained on two-channel images, where the first channel is the channel to segment, and the second channel is an optional nuclear channel. Here are the options for each: 1. 0=grayscale, 1=red, 2=green, 3=blue 2. 0=None (will set to zero), 1=red, 2=green, 3=blue

Set channels to a list with each of these elements, e.g. `channels = [0,0]` if you want to segment cells in grayscale or for single channel images, or `channels = [2,3]` if you green cells with blue nuclei.

The *'cyto3'* model is trained on 9 datasets, see the Cellpose3 paper for more details.

6.1.2 Nucleus model (*'nuclei'*)

The nuclear model in cellpose is trained on two-channel images, where the first channel is the channel to segment, and the second channel is always set to an array of zeros. Therefore set the first channel as 0=grayscale, 1=red, 2=green, 3=blue; and set the second channel to zero, e.g. `channels = [0,0]` if you want to segment nuclei in grayscale or for single channel images, or `channels = [3,0]` if you want to segment blue nuclei.

6.2 Other built-in models

The main built-in models are dataset-specific models trained on one of the 9 datasets in the Cellpose3 paper. These models do not have a size model. If the diameter is set to 0.0, then the model uses the default `diam_mean` for the diameter (`30.0`).

These models can be loaded and used in the notebook with e.g. `models.CellposeModel(model_type='tissuenet_cp3')` or `models.CellposeModel(model_type='livecell_cp3')`, or in the command line with `python -m cellpose --pretrained_model tissuenet_cp3`.

The dataset-specific models were trained on the training images provided in the following datasets:

- `tissuenet_cp3`: tissuenet dataset.
- `livecell_cp3`: livecell dataset
- `yeast_PhC_cp3`: YEAZ dataset
- `yeast_BF_cp3`: YEAZ dataset
- `bact_phase_cp3`: omnipose dataset
- `bact_fluor_cp3`: omnipose dataset
- `deepbacs_cp3`: deepbacs dataset
- `cyto2_cp3`: cellpose dataset

6.3 User-trained models

By default, models are trained with the ROIs resized to a diameter of 30.0 (`diam_mean = 30`) – this is necessary if you want to start from a pretrained cellpose model. If you want to use a different diameter and use pretraining, we recommend performing training yourself on the cellpose dataset with that diameter so the model learns objects at that size. All user-trained models will save the `diam_mean` so it will be loaded automatically along with the model weights.

Each model also saves the `diam_labels` which is the mean diameter of the ROIs in the training images. This value is auto-loaded into the GUI for use with the model, or will be used if the diameter is 0 (`diameter=0` or `--diameter 0`).

These models can be loaded and used in the notebook with e.g. `models.CellposeModel(model_type='name_in_gui')` or with the full path `models.CellposeModel(pretrained_model='/full/path/to/model')` . If you trained in the GUI, you can automatically use the `model_type` argument. If you trained in the command line, you need to first add the model to the cellpose path either in the GUI in the Models menu, or using the command line: `python -m cellpose --add_model /full/path/to/model`.

Or these models can be used in the command line with `python -m cellpose --pretrained_model name_in_gui` or `python -m cellpose --pretrained_model /full/path/to/model` .

CHAPTER

SEVEN

IMAGE RESTORATION

The image restoration module denoise provides functions for restoring degraded images. There are two main classes, DenoiseModel for image restoration only, and CellposeDenoiseModel for image restoration and then segmentation. There are three types of image restoration provided, denoising, deblurring, and upsampling, and for each of these there are two models, one trained on the full cyto3 training set and one trained on the nuclei training set: 'denoise_cyto3', 'deblur_cyto3', 'upsample_cyto3', 'denoise_nuclei', 'deblur_nuclei', 'upsample_nuclei'.

7.1 DenoiseModel

Initialize a DenoiseModel with the model_type:

```
from cellpose import denoise
dn = denoise.DenoiseModel(model_type="denoise_cyto3", gpu=True)
```

Now you can apply this denoising model to specified channels in your images, using the Cellpose channel format (e.g. channels=[1,2]), or leave channels=None to apply the model to all channels. Make sure to set the diameter to the size of the objects in your image.

```
imgs_dn = dn.eval(imgs, channels=None, diameter=50.)
```

If you have two channels, and the second is a nuclear channel, you can specify to use the nuclei restoration models on the second channel, with chan2=True:

```
from cellpose import denoise
dn = denoise.DenoiseModel(model_type="denoise_cyto3", gpu=True, chan2=True)
imgs_dn = dn.eval(imgs, channels=[1,2], diameter=50.)
```

The upsampling model 'upsample_cyto3' enables upsampling to diameter of 30., and the upsampling model 'upsample_nuclei' enables upsampling to diameter of 17. If you have images, for example, in which the objects are of diameter 10, specify that in the function call, and then the model will upsample the image to 30 or 17:

```
from cellpose import denoise
dn = denoise.DenoiseModel(model_type="upsample_cyto3", gpu=True, chan2=True)
imgs_up = dn.eval(imgs, channels=[1,2], diameter=10.)
```

For more details refer to the API section.

7.2 CellposeDenoiseModel

The `CellposeDenoiseModel` wraps the CellposeModel and DenoiseModel into one class to ensure the channels and diameters are handled properly. See example:

```
from cellpose import denoise
model = denoise.CellposeDenoiseModel(gpu=True, model_type="cyto3",
             restore_type="denoise_cyto3", chan2_restore=True)
masks, flows, styles, imgs_dn = model.eval(imgs, channels=[1,2], diameter=50.)
```

For more details refer to the API section.

7.3 Command line usage

These models can be used on the command line with input `--restore_type` and flag `--chan2_restore`.

CHAPTER

EIGHT

TRAINING

At the beginning of training, cellpose computes the flow field representation for each mask image (`dynamics.labels_to_flows`).

The cellpose pretrained models are trained using resized images so that the cells have the same median diameter across all images. If you choose to use a pretrained model, then this fixed median diameter is used.

If you choose to train from scratch, you can set the median diameter you want to use for rescaling with the `--diam_mean` flag. We trained all model zoo models with a diameter of 30.0 pixels, except the *nuclei* model which used a diameter of 17 pixels, so if you want to start with a pretrained model, it will default to those values.

The models will be saved in the image directory (`--dir`) in a folder called `models/`.

The same channel settings apply for training models.

Note Cellpose expects the labelled masks (0=no mask, 1,2…=masks) in a separate file, e.g:

```
wells_000.tif
wells_000_masks.tif
```

You can use a different ending from `_masks` with the `--mask_filter` option, e.g. `--mask_filter _masks_2022`.

Also, you can train a model using the labels from the GUI (`_seg.npy`) by using the following option `--mask_filter _seg.npy`.

If you use the –img_filter option (`--img_filter _img` in this case):

```
wells_000_img.tif
wells_000_masks.tif
```

Warning: The path given to `--dir` and `--test_dir` should be an absolute path.

To train on cytoplasmic images (green cyto and red nuclei) starting with a pretrained model from cellpose (one of the model zoo models), we also have included the recommended training parameters in the command below:

```
python -m cellpose --train --dir ~/images_cyto/train/ --test_dir ~/images_cyto/test/ --
↪pretrained_model cyto --chan 2 --chan2 1 --learning_rate 0.1 --weight_decay 0.0001 --n_
↪epochs 100
```

You can train from scratch as well:

```
python -m cellpose --train --dir ~/images_nuclei/train/ --pretrained_model None
```

To train the cyto model from scratch using the same parameters we did, download the dataset and run

```
python -m cellpose --train --train_size --use_gpu --dir ~/cellpose_dataset/train/ --test_
↪dir ~/cellpose_dataset/test/ --img_filter _img --pretrained_model None --chan 2 --
↪chan2 1
```

You can also specify the full path to a pretrained model to use:

```
python -m cellpose --dir ~/images_cyto/test/ --pretrained_model ~/images_cyto/test/model/
↪cellpose_35_0 --save_png
```

In a notebook, you can train with the *train_seg* function:

```
from cellpose import io, models, train
io.logger_setup()

output = io.load_train_test_data(train_dir, test_dir, image_filter="_img",
                                mask_filter="_masks", look_one_level_down=False)
images, labels, image_names, test_images, test_labels, image_names_test = output

# e.g. retrain a Cellpose model
model = models.CellposeModel(model_type="cyto3")

model_path = train.train_seg(model.net, train_data=images, train_labels=labels,
                            channels=[1,2], normalize=True,
                            test_data=test_images, test_labels=test_labels,
                            weight_decay=1e-4, SGD=True, learning_rate=0.1,
                            n_epochs=100, model_name="my_new_model")
```

Training arguments on the CLI

```
--train               train network using images in dir
--train_size          train size network at end of training
--test_dir TEST_DIR   folder containing test data (optional)
--mask_filter MASK_FILTER
                        end string for masks to run on. use '_seg.npy' for
                        manual annotations from the GUI. Default: _masks
--diam_mean DIAM_MEAN
                        mean diameter to resize cells to during training -- if
                        starting from pretrained models it cannot be changed
                        from 30.0
--learning_rate LEARNING_RATE
                        learning rate. Default: 0.2
--weight_decay WEIGHT_DECAY
                        weight decay. Default: 1e-05
--n_epochs N_EPOCHS   number of epochs. Default: 500
--batch_size BATCH_SIZE
                        batch size. Default: 8
--min_train_masks MIN_TRAIN_MASKS
                        minimum number of masks a training image must have to
                        be used. Default: 5
--SGD SGD             use SGD
--save_every SAVE_EVERY
                        number of epochs to skip between saves. Default: 100
--model_name_out MODEL_NAME_OUT
```

(continues on next page)

(continued from previous page)

```
                        Name of model to save as, defaults to name describing
                        model architecture. Model is saved in the folder
                        specified by --dir in models subfolder.
```

CHAPTER

NINE

OPENVINO

OpenVINO is an optional backend for Cellpose which optimizes deep learning inference for Intel Architectures.

It should be installed in the same environment with Cellpose by the following command :

```
pip install --no-deps openvino
```

Using openvino_utils.to_openvino, convert PyTorch model to OpenVINO one:

```
from cellpose.contrib import openvino_utils

model = models.CellposeModel(...)

model = openvino_utils.to_openvino(model)
```

CHAPTER

TEN

FAQ

Q: What should I set the `--flow_threshold`/`--cell_prob`/`--diam` parameter to?

These parameters should be set experimentally by running Cellpose, viewing the results, and tuning the parameters to get the best results. The default parameters are set to work well for most images, but may not be optimal for your images. See *Settings* for more information.

Q: What accuracy is good enough? Is there a quantitative threshold that should be met before implementing a model?

Generally speaking you want to meet or exceed the accuracy of a human. You can estimate human accuracy by labeling the same image twice and evaluating accuracy metrics. In practice human accuracy is often lower than you would expect. You can see our results from this analysis in our Cellpose 2 paper.

Some additional information on precision and accuracy can be found here.

Q: How do I download the pretrained models?

The models will be downloaded automatically from the website when you first run a pretrained model in cellpose. If you are having issues with the downloads, you can download them from this google drive zip file, unzip the file and put the models in your home directory under the path `.cellpose/models/`, e.g. on Windows this would be `C:/Users/YOUR_USERNAME/.cellpose/models/` or on Linux this would be `/home/YOUR_USERNAME/.cellpose/models/`, so `/home/YOUR_USERNAME/.cellpose/models/cyto_0` is the full path to one model for example. If you cannot access google drive, the models are also available on baidu: https://pan.baidu.com/s/1CARpRGCBHIYaz7KeyoX-fg thanks to @qixinbo!

Q: How can I use cellpose to recognize different types of cells in the same image?

Cellpose does not natively support recognizing different types of cells (aka 'multiclass segmentation'). However, you can train individual models that are capable of recognizing only a given cell type at a time and run Cellpose multiple times on the same image. With sufficient training, the result will be two sets of outputs that could be combined in post-processing to identify the different cell types.

Q: Why does the PNG mask file look dim at the top and light at the bottom? I can't see the cell masks.

This is expected and intended behavior, although it is dependent on the image viewer used to view the mask file. The mask file is saved with each pixel as background (represented by a 0), or as a cell label (represented by the cell label number). The gradient is produced because each cell label is unique and monotonically increasing from top to bottom.

You can use different look up tables (LUTs) in ImageJ to view the resulting masks or threshold everything above zero to get everything that cellpose detects. Image post processing is outside the scope of cellpose, but you can find additional help at https://forum.image.sc/tag/cellpose.

Q: The Cellpose GUI is unresponsive/frozen. Is it broken?

Cellpose is likely not broken; it is just busy. Currently, the GUI cannot receive input while computing segmentation. Cellpose is a fairly computationally intensive program and may take a long time to run, depending on computer hardware specifications. Cellpose will take a long time to run on large images. Using hardware with a faster CPU and with more available memory will speed up the process. Using a GPU will also speed up the process, especially if you are training with a large dataset.

CHAPTER

ELEVEN

IN A NOTEBOOK

See *Settings* for more information on run settings.

```
import numpy as np
import matplotlib.pyplot as plt
from cellpose import models, io
from cellpose.io import imread

io.logger_setup()

# model_type='cyto' or 'nuclei' or 'cyto2' or 'cyto3'
model = models.Cellpose(model_type='cyto3')

# list of files
# PUT PATH TO YOUR FILES HERE!
files = ['/media/carsen/DATA1/TIFFS/onechan.tif']

imgs = [imread(f) for f in files]
nimg = len(imgs)

# define CHANNELS to run segementation on
# grayscale=0, R=1, G=2, B=3
# channels = [cytoplasm, nucleus]
# if NUCLEUS channel does not exist, set the second channel to 0
channels = [[0,0]]
# IF ALL YOUR IMAGES ARE THE SAME TYPE, you can give a list with 2 elements
# channels = [0,0] # IF YOU HAVE GRAYSCALE
# channels = [2,3] # IF YOU HAVE G=cytoplasm and B=nucleus
# channels = [2,1] # IF YOU HAVE G=cytoplasm and R=nucleus

# if diameter is set to None, the size of the cells is estimated on a per image basis
# you can set the average cell `diameter` in pixels yourself (recommended)
# diameter can be a list or a single number for all images

masks, flows, styles, diams = model.eval(imgs, diameter=None, channels=channels)


### or to run one of the other models, or a custom model, specify a CellposeModel
model = models.CellposeModel(model_type='livecell_cp3')

masks, flows, styles = model.eval(imgs, diameter=30, channels=[0,0])
```

See example notebook at run_cellpose.ipynb.

CHAPTER

TWELVE

COMMAND LINE

The full list of options and what they do can be found on the Command Line Interface (CLI) documentation page: *Cellpose CLI*. A description of the most important settings can be found on the *Settings* page.

12.1 Command Line Usage

Run `python -m cellpose` and specify parameters as below. For instance to run on a folder with images where cytoplasm is green and nucleus is blue and save the output as a png (using default diameter 30):

```
python -m cellpose --dir ~/images_cyto/test/ --pretrained_model cyto --chan 2 --chan2 3 -
↪-save_png
```

You can specify the diameter for all the images or set to 0 if you want the algorithm to estimate it on an image by image basis. Here is how to run on nuclear data (grayscale) where the diameter is automatically estimated:

```
python -m cellpose --dir ~/images_nuclei/test/ --pretrained_model nuclei --diameter 0. --
↪save_png
```

Warning: The path given to `--dir` must be an absolute path.

CHAPTER

THIRTEEN

CELLPOSE API GUIDE

13.1 Cellpose class

class cellpose.models.**Cellpose**(*gpu=False*, *model_type='cyto3'*, *nchan=2*, *device=None*, *backbone='default'*)

Main model which combines SizeModel and CellposeModel.

Parameters

- **gpu** (*bool, optional*) – Whether or not to use GPU, will check if GPU available. Defaults to False.
- **model_type** (*str, optional*) – Model type. "cyto"=cytoplasm model; "nuclei"=nucleus model; "cyto2"=cytoplasm model with additional user images; "cyto3"=super-generalist model; Defaults to "cyto3".
- **device** (*torch device, optional*) – Device used for model running / training. Overrides gpu input. Recommended if you want to use a specific GPU (e.g. torch.device("cuda:1")). Defaults to None.

device

Device used for model running / training.

Type

torch device

gpu

Flag indicating if GPU is used.

Type

bool

diam_mean

Mean diameter for cytoplasm model.

Type

float

cp

CellposeModel instance.

Type

CellposeModel

`pretrained_size`

Pretrained size model path.

Type

str

`sz`

SizeModel instance.

Type

SizeModel

`eval`(*x*, *batch_size=8*, *channels=[0, 0]*, *channel_axis=None*, *invert=False*, *normalize=True*, *diameter=30.0*, *do_3D=False*, *find_masks=True*, ***kwargs*)

Run cellpose size model and mask model and get masks.

Parameters

- **x** (*list or array*) – List or array of images. Can be list of 2D/3D images, or array of 2D/3D images, or 4D image array.
- **batch_size** (*int, optional*) – Number of 224x224 patches to run simultaneously on the GPU. Can make smaller or bigger depending on GPU memory usage. Defaults to 8.
- **channels** (*list, optional*) – List of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to [0,0].
- **channel_axis** (*int, optional*) – If None, channels dimension is attempted to be automatically determined. Defaults to None.
- **invert** (*bool, optional*) – Invert image pixel intensity before running network (if True, image is also normalized). Defaults to False.
- **normalize** (*bool, optional*) – If True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (see CellposeModel for details). Defaults to True.
- **diameter** (*float, optional*) – If set to None, then diameter is automatically estimated if size model is loaded. Defaults to 30..
- **do_3D** (*bool, optional*) – Set to True to run 3D segmentation on 4D image input. Defaults to False.

Returns

tuple containing

- masks (list of 2D arrays or single 3D array): Labelled image, where 0=no masks; 1,2,…=mask labels.
- **flows (list of lists 2D arrays or list of 3D arrays):**
  - flows[k][0] = XY flow in HSV 0-255
  - flows[k][1] = XY flows at each pixel
  - flows[k][2] = cell probability (if > cellprob_threshold, pixel used for dynamics)
  - flows[k][3] = final pixel locations after Euler integration

- styles (list of 1D arrays of length 256 or single 1D array): Style vector summarizing each image, also used to estimate size of objects in image.
- diams (list of diameters or float): List of diameters or float (if do_3D=True).

13.2 CellposeModel

`class cellpose.models.`**`CellposeModel`**`(`*gpu=False*, *pretrained_model=False*, *model_type=None*, *diam_mean=30.0*, *device=None*, *nchan=2*, *backbone='default'*`)`

Class representing a Cellpose model.

`diam_mean`

Mean "diameter" value for the model.

Type
float

`builtin`

Whether the model is a built-in model or not.

Type
bool

`device`

Device used for model running / training.

Type
torch device

`mkldnn`

MKLDNN flag for the model.

Type
None or bool

`nchan`

Number of channels used as input to the network.

Type
int

`nclasses`

Number of classes in the model.

Type
int

`nbase`

List of base values for the model.

Type
list

`net`

Cellpose network.

Type
CPnet

`pretrained_model`

Full path to pretrained cellpose model(s).

Type

str or list of strings

`diam_labels`

Diameter labels of the model.

Type

numpy array

`net_type`

Type of the network.

Type

str

`__init__`(*self*, *gpu=False*, *pretrained_model=False*, *model_type=None*, *diam_mean=30.*, *device=None*, *nchan=2*)

Initialize the CellposeModel.

`eval`(*self*, *x*, *batch_size=8*, *resample=True*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *invert=False*, *rescale=None*, *diameter=None*, *flow_threshold=0.4*, *cellprob_threshold=0.0*, *do_3D=False*, *anisotropy=None*, *stitch_threshold=0.0*, *min_size=15*, *niter=None*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*, *interp=True*, *compute_masks=True*, *progress=None*)

Segment list of images x, or 4D array - Z x nchan x Y x X.

`eval`(*x*, *batch_size=8*, *resample=True*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *invert=False*, *rescale=None*, *diameter=None*, *flow_threshold=0.4*, *cellprob_threshold=0.0*, *do_3D=False*, *anisotropy=None*, *stitch_threshold=0.0*, *min_size=15*, *niter=None*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*, *interp=True*, *compute_masks=True*, *progress=None*)

segment list of images x, or 4D array - Z x nchan x Y x X

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **resample** (*bool, optional*) – run dynamics at original image size (will be slower but create more accurate boundaries). Defaults to True.
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.

- **z_axis** (*int, optional*) – z axis in element of list x, or of np.ndarray x. if None, z dimension is attempted to be automatically determined. Defaults to None.
- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - ”lowhigh”=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - ”sharpen”=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - ”normalize”=True ; run normalization (if False, all following parameters ignored)
  - ”percentile”=None : pass in percentiles to use as list [perc_low, perc_high]
  - ”tile_norm”=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - ”norm3D”=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.
- **invert** (*bool, optional*) – invert image pixel intensity before running network. Defaults to False.
- **rescale** (*float, optional*) – resize factor for each image, if None, set to 1.0; (only used if diameter is None). Defaults to None.
- **diameter** (*float, optional*) – diameter for each image, if diameter is None, set to diam_mean or diam_train if available. Defaults to None.
- **flow_threshold** (*float, optional*) – flow error threshold (all cells with errors below threshold are kept) (not used for 3D). Defaults to 0.4.
- **cellprob_threshold** (*float, optional*) – all pixels with value above threshold kept for masks, decrease to find more and larger masks. Defaults to 0.0.
- **do_3D** (*bool, optional*) – set to True to run 3D segmentation on 3D/4D image input. Defaults to False.
- **anisotropy** (*float, optional*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y). Defaults to None.
- **stitch_threshold** (*float, optional*) – if stitch_threshold>0.0 and not do_3D, masks are stitched in 3D to return volume segmentation. Defaults to 0.0.
- **min_size** (*int, optional*) – all ROIs below this size, in pixels, will be discarded. Defaults to 15.
- **niter** (*int, optional*) – number of iterations for dynamics computation. if None, it is set proportional to the diameter. Defaults to None.
- **augment** (*bool, optional*) – tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **tile_overlap** (*float, optional*) – fraction of overlap of tiles when computing flows. Defaults to 0.1.

- **bsize** (*int, optional*) – block size for tiles, recommended to keep at 224, like in training. Defaults to 224.
- **interp** (*bool, optional*) – interpolate during 2D dynamics (not available in 3D) . Defaults to True.
- **compute_masks** (*bool, optional*) – Whether or not to compute dynamics and return masks. This is set to False when retrieving the styles for the size model. Defaults to True.
- **progress** (*QProgressBar, optional*) – pyqt progress bar. Defaults to None.

Returns

- masks (list, np.ndarray): labelled image(s), where 0=no masks; 1,2,. . . =mask labels
- flows (list): list of lists: flows[k][0] = XY flow in HSV 0-255; flows[k][1] = XY(Z) flows at each pixel; flows[k][2] = cell probability (if > cellprob_threshold, pixel used for dynamics); flows[k][3] = final pixel locations after Euler integration
- styles (list, np.ndarray): style vector summarizing each image of size 256.

Return type

A tuple containing

13.3 CellposeDenoiseModel

class cellpose.denoise.**CellposeDenoiseModel**(*gpu=False*, *pretrained_model=False*, *model_type=None*, *restore_type='denoise_cyto3'*, *chan2_restore=False*, *device=None*)

model to run Cellpose and Image restoration

eval(*x*, *batch_size=8*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *rescale=None*, *diameter=None*, *tile=True*, *tile_overlap=0.1*, *augment=False*, *resample=True*, *invert=False*, *flow_threshold=0.4*, *cellprob_threshold=0.0*, *do_3D=False*, *anisotropy=None*, *stitch_threshold=0.0*, *min_size=15*, *niter=None*, *interp=True*)

Restore array or list of images using the image restoration model, and then segment.

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.

- **z_axis** (*int, optional*) – z axis in element of list x, or of np.ndarray x. if None, z dimension is attempted to be automatically determined. Defaults to None.
- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - ”lowhigh”=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - ”sharpen”=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - ”normalize”=True ; run normalization (if False, all following parameters ignored)
  - ”percentile”=None : pass in percentiles to use as list [perc_low, perc_high]
  - ”tile_norm”=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - ”norm3D”=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.
- **rescale** (*float, optional*) – resize factor for each image, if None, set to 1.0; (only used if diameter is None). Defaults to None.
- **diameter** (*float, optional*) – diameter for each image, if diameter is None, set to diam_mean or diam_train if available. Defaults to None.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **tile_overlap** (*float, optional*) – fraction of overlap of tiles when computing flows. Defaults to 0.1.
- **augment** (*bool, optional*) – augment tiles by flipping and averaging for segmentation. Defaults to False.
- **resample** (*bool, optional*) – run dynamics at original image size (will be slower but create more accurate boundaries). Defaults to True.
- **invert** (*bool, optional*) – invert image pixel intensity before running network. Defaults to False.
- **flow_threshold** (*float, optional*) – flow error threshold (all cells with errors below threshold are kept) (not used for 3D). Defaults to 0.4.
- **cellprob_threshold** (*float, optional*) – all pixels with value above threshold kept for masks, decrease to find more and larger masks. Defaults to 0.0.
- **do_3D** (*bool, optional*) – set to True to run 3D segmentation on 3D/4D image input. Defaults to False.
- **anisotropy** (*float, optional*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y). Defaults to None.
- **stitch_threshold** (*float, optional*) – if stitch_threshold>0.0 and not do_3D, masks are stitched in 3D to return volume segmentation. Defaults to 0.0.
- **min_size** (*int, optional*) – all ROIs below this size, in pixels, will be discarded. Defaults to 15.

- **niter** (*int, optional*) – number of iterations for dynamics computation. if None, it is set proportional to the diameter. Defaults to None.
- **interp** (*bool, optional*) – interpolate during 2D dynamics (not available in 3D) . Defaults to True.

Returns
labelled image(s), where 0=no masks; 1,2,…=mask labels flows (list): list of lists: flows[k][0] = XY flow in HSV 0-255; flows[k][1] = XY(Z) flows at each pixel; flows[k][2] = cell probability (if > cellprob_threshold, pixel used for dynamics); flows[k][3] = final pixel locations after Euler integration styles (list, np.ndarray): style vector summarizing each image of size 256. imgs (list of 2D/3D arrays): Restored images

Return type
masks (list, np.ndarray)

13.4 DenoiseModel

`class cellpose.denoise.DenoiseModel`(*gpu=False*, *pretrained_model=False*, *nchan=1*, *model_type=None*, *chan2=False*, *diam_mean=30.0*, *device=None*)

DenoiseModel class for denoising images using Cellpose denoising model.

Parameters

- **gpu** (*bool, optional*) – Whether to use GPU for computation. Defaults to False.
- **pretrained_model** (*bool or str or Path, optional*) – Pretrained model to use for denoising. Can be a string or path. Defaults to False.
- **nchan** (*int, optional*) – Number of channels in the input images, all Cellpose 3 models were trained with nchan=1. Defaults to 1.
- **model_type** (*str, optional*) – Type of pretrained model to use ("denoise_cyto3", "deblur_cyto3", "upsample_cyto3", …). Defaults to None.
- **chan2** (*bool, optional*) – Whether to use a separate model for the second channel. Defaults to False.
- **diam_mean** (*float, optional*) – Mean diameter of the objects in the images. Defaults to 30.0.
- **device** (*torch.device, optional*) – Device to use for computation. Defaults to None.

`nchan`

Number of channels in the input images.

Type
int

`diam_mean`

Mean diameter of the objects in the images.

Type
float

`net`

Cellpose network for denoising.

Type
CPnet

`pretrained_model`

Pretrained model path to use for denoising.

Type
bool or str or Path

`net_chan2`

Cellpose network for the second channel, if applicable.

Type
CPnet or None

`net_type`

Type of the denoising network.

Type
str

`eval(x, batch_size=8, channels=None, channel_axis=None, z_axis=None,`

normalize=True, rescale=None, diameter=None, tile=True, tile_overlap=0.1)

Denoise array or list of images using the denoising model.

`_eval(net, x, normalize=True, rescale=None, diameter=None, tile=True,`

tile_overlap=0.1)

Run denoising model on a single channel.

`eval`(*x*, *batch_size=8*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *rescale=None*, *diameter=None*, *tile=True*, *tile_overlap=0.1*)

Restore array or list of images using the image restoration model.

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.
- **z_axis** (*int, optional*) – z axis in element of list x, or of np.ndarray x. if None, z dimension is attempted to be automatically determined. Defaults to None.

- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - ”lowhigh”=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - ”sharpen”=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - ”normalize”=True ; run normalization (if False, all following parameters ignored)
  - ”percentile”=None : pass in percentiles to use as list [perc_low, perc_high]
  - ”tile_norm”=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - ”norm3D”=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.
- **rescale** (*float, optional*) – resize factor for each image, if None, set to 1.0; (only used if diameter is None). Defaults to None.
- **diameter** (*float, optional*) – diameter for each image, if diameter is None, set to diam_mean or diam_train if available. Defaults to None.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **tile_overlap** (*float, optional*) – fraction of overlap of tiles when computing flows. Defaults to 0.1.

Returns
Restored images

Return type
imgs (list of 2D/3D arrays)

13.5 SizeModel

`class cellpose.models.SizeModel(`*cp_model*, *device=None*, *pretrained_size=None*, ***kwargs*`)`

Linear regression model for determining the size of objects in image used to rescale before input to cp_model. Uses styles from cp_model.

pretrained_size

Path to pretrained size model.

Type
str

cp

Model from which to get styles.

Type
UnetModel or *CellposeModel*

`device`

Device used for model running / training (torch.device("cuda") or torch.device("cpu")), overrides gpu input, recommended if you want to use a specific GPU (e.g. torch.device("cuda:1")).

Type
torch device

`diam_mean`

Mean diameter of objects.

Type
float

`eval(self, x, channels=None, channel_axis=None, normalize=True, invert=False,`

augment=False, tile=True, batch_size=8, progress=None, interp=True):

Use images x to produce style or use style input to predict size of objects in image.

Raises
`ValueError` – If no pretrained cellpose model is specified, cannot compute size.

`eval`(*x*, *channels=None*, *channel_axis=None*, *normalize=True*, *invert=False*, *augment=False*, *tile=True*, *batch_size=8*, *progress=None*)

Use images x to produce style or use style input to predict size of objects in image.

Object size estimation is done in two steps: 1. Use a linear regression model to predict size from style in image. 2. Resize image to predicted size and run CellposeModel to get output masks.

Take the median object size of the predicted masks as the final predicted size.

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.
- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - "lowhigh"=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - "sharpen"=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - "normalize"=True ; run normalization (if False, all following parameters ignored)
  - "percentile"=None : pass in percentiles to use as list [perc_low, perc_high]

  - "tile_norm"=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - "norm3D"=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.

- **invert** (*bool, optional*) – Invert image pixel intensity before running network (if True, image is also normalized). Defaults to False.
- **augment** (*bool, optional*) – tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **progress** (*QProgressBar, optional*) – pyqt progress bar. Defaults to None.

Returns

- diam (np.ndarray): Final estimated diameters from images x or styles style after running both steps.
- diam_style (np.ndarray): Estimated diameters from style alone.

Return type

A tuple containing

13.6 Training

cellpose.train.**train_seg**(*net*, *train_data=None*, *train_labels=None*, *train_files=None*, *train_labels_files=None*, *train_probs=None*, *test_data=None*, *test_labels=None*, *test_files=None*, *test_labels_files=None*, *test_probs=None*, *load_files=True*, *batch_size=8*, *learning_rate=0.005*, *n_epochs=2000*, *weight_decay=1e-05*, *momentum=0.9*, *SGD=False*, *channels=None*, *channel_axis=None*, *rgb=False*, *normalize=True*, *compute_flows=False*, *save_path=None*, *save_every=100*, *nimg_per_epoch=None*, *nimg_test_per_epoch=None*, *rescale=True*, *scale_range=None*, *bsize=224*, *min_train_masks=5*, *model_name=None*)

Train the network with images for segmentation.

Parameters

- **net** (*object*) – The network model to train.
- **train_data** (*List[np.ndarray], optional*) – List of arrays (2D or 3D) - images for training. Defaults to None.
- **train_labels** (*List[np.ndarray], optional*) – List of arrays (2D or 3D) - labels for train_data, where 0=no masks; 1,2,…=mask labels. Defaults to None.
- **train_files** (*List[str], optional*) – List of strings - file names for images in train_data (to save flows for future runs). Defaults to None.
- **train_labels_files** (*list or None*) – List of training label file paths. Defaults to None.

- **train_probs** (*List[float], optional*) – List of floats - probabilities for each image to be selected during training. Defaults to None.
- **test_data** (*List[np.ndarray], optional*) – List of arrays (2D or 3D) - images for testing. Defaults to None.
- **test_labels** (*List[np.ndarray], optional*) – List of arrays (2D or 3D) - labels for test_data, where 0=no masks; 1,2,…=mask labels. Defaults to None.
- **test_files** (*List[str], optional*) – List of strings - file names for images in test_data (to save flows for future runs). Defaults to None.
- **test_labels_files** (*list or None*) – List of test label file paths. Defaults to None.
- **test_probs** (*List[float], optional*) – List of floats - probabilities for each image to be selected during testing. Defaults to None.
- **load_files** (*bool, optional*) – Boolean - whether to load images and labels from files. Defaults to True.
- **batch_size** (*int, optional*) – Integer - number of patches to run simultaneously on the GPU. Defaults to 8.
- **learning_rate** (*float or List[float], optional*) – Float or list/np.ndarray - learning rate for training. Defaults to 0.005.
- **n_epochs** (*int, optional*) – Integer - number of times to go through the whole training set during training. Defaults to 2000.
- **weight_decay** (*float, optional*) – Float - weight decay for the optimizer. Defaults to 1e-5.
- **momentum** (*float, optional*) – Float - momentum for the optimizer. Defaults to 0.9.
- **SGD** (*bool, optional*) – Boolean - whether to use SGD as optimization instead of RAdam. Defaults to False.
- **channels** (*List[int], optional*) – List of ints - channels to use for training. Defaults to None.
- **channel_axis** (*int, optional*) – Integer - axis of the channel dimension in the input data. Defaults to None.
- **normalize** (*bool or dict, optional*) – Boolean or dictionary - whether to normalize the data. Defaults to True.
- **compute_flows** (*bool, optional*) – Boolean - whether to compute flows during training. Defaults to False.
- **save_path** (*str, optional*) – String - where to save the trained model. Defaults to None.
- **save_every** (*int, optional*) – Integer - save the network every [save_every] epochs. Defaults to 100.
- **nimg_per_epoch** (*int, optional*) – Integer - minimum number of images to train on per epoch. Defaults to None.
- **nimg_test_per_epoch** (*int, optional*) – Integer - minimum number of images to test on per epoch. Defaults to None.
- **rescale** (*bool, optional*) – Boolean - whether or not to rescale images during training. Defaults to True.

- **min_train_masks** (*int, optional*) – Integer - minimum number of masks an image must have to use in the training set. Defaults to 5.
- **model_name** (*str, optional*) – String - name of the network. Defaults to None.

Returns
path to saved model weights

Return type
Path

`cellpose.train.`**`train_size`**(*net*, *pretrained_model*, *train_data=None*, *train_labels=None*, *train_files=None*, *train_labels_files=None*, *train_probs=None*, *test_data=None*, *test_labels=None*, *test_files=None*, *test_labels_files=None*, *test_probs=None*, *load_files=True*, *min_train_masks=5*, *channels=None*, *channel_axis=None*, *rgb=False*, *normalize=True*, *nimg_per_epoch=None*, *nimg_test_per_epoch=None*, *batch_size=64*, *scale_range=1.0*, *bsize=512*, *l2_regularization=1.0*, *n_epochs=10*)

Train the size model.

Parameters

- **net** (*object*) – The neural network model.
- **pretrained_model** (*str*) – The path to the pretrained model.
- **train_data** (*numpy.ndarray, optional*) – The training data. Defaults to None.
- **train_labels** (*numpy.ndarray, optional*) – The training labels. Defaults to None.
- **train_files** (*list, optional*) – The training file paths. Defaults to None.
- **train_labels_files** (*list, optional*) – The training label file paths. Defaults to None.
- **train_probs** (*numpy.ndarray, optional*) – The training probabilities. Defaults to None.
- **test_data** (*numpy.ndarray, optional*) – The test data. Defaults to None.
- **test_labels** (*numpy.ndarray, optional*) – The test labels. Defaults to None.
- **test_files** (*list, optional*) – The test file paths. Defaults to None.
- **test_labels_files** (*list, optional*) – The test label file paths. Defaults to None.
- **test_probs** (*numpy.ndarray, optional*) – The test probabilities. Defaults to None.
- **load_files** (*bool, optional*) – Whether to load files. Defaults to True.
- **min_train_masks** (*int, optional*) – The minimum number of training masks. Defaults to 5.
- **channels** (*list, optional*) – The channels. Defaults to None.
- **channel_axis** (*int, optional*) – The channel axis. Defaults to None.
- **normalize** (*bool or dict, optional*) – Whether to normalize the data. Defaults to True.
- **nimg_per_epoch** (*int, optional*) – The number of images per epoch. Defaults to None.
- **nimg_test_per_epoch** (*int, optional*) – The number of test images per epoch. Defaults to None.

- **batch_size** (*int, optional*) – The batch size. Defaults to 64.
- **l2_regularization** (*float, optional*) – The L2 regularization factor. Defaults to 1.0.
- **n_epochs** (*int, optional*) – The number of epochs. Defaults to 10.

Returns
The trained size model parameters.

Return type
dict

13.7 Metrics

cellpose.metrics.**aggregated_jaccard_index**(*masks_true*, *masks_pred*)

AJI = intersection of all matched masks / union of all masks

Parameters

- **masks_true** (*list of np.ndarrays (int) or np.ndarray (int)*) – where 0=NO masks; 1,2… are mask labels
- **masks_pred** (*list of np.ndarrays (int) or np.ndarray (int)*) – np.ndarray (int) where 0=NO masks; 1,2… are mask labels

Returns
aggregated jaccard index for each set of masks

Return type
aji (float)

cellpose.metrics.**average_precision**(*masks_true*, *masks_pred*, *threshold=[0.5, 0.75, 0.9]*)

Average precision estimation: AP = TP / (TP + FP + FN)

This function is based heavily on the *fast* stardist matching functions (https://github.com/mpicbg-csbd/stardist/blob/master/stardist/matching.py)

Parameters

- **masks_true** (*list of np.ndarrays (int) or np.ndarray (int)*) – where 0=NO masks; 1,2… are mask labels
- **masks_pred** (*list of np.ndarrays (int) or np.ndarray (int)*) – np.ndarray (int) where 0=NO masks; 1,2… are mask labels

Returns

average precision at thresholds tp (array [len(masks_true) x len(threshold)]):

number of true positives at thresholds

fp (array [len(masks_true) x len(threshold)]):
number of false positives at thresholds

fn (array [len(masks_true) x len(threshold)]):
number of false negatives at thresholds

Return type
ap (array [len(masks_true) x len(threshold)])

`cellpose.metrics.`**`boundary_scores`**(*masks_true*, *masks_pred*, *scales*)

Calculate boundary precision, recall, and F-score.

Parameters

- **`masks_true`** (*`list`*) – List of true masks.
- **`masks_pred`** (*`list`*) – List of predicted masks.
- **`scales`** (*`list`*) – List of scales.

Returns

A tuple containing precision, recall, and F-score arrays.

Return type

tuple

`cellpose.metrics.`**`flow_error`**(*maski*, *dP_net*, *device=None*)

Error in flows from predicted masks vs flows predicted by network run on image.

This function serves to benchmark the quality of masks. It works as follows: 1. The predicted masks are used to create a flow diagram. 2. The mask-flows are compared to the flows that the network predicted.

If there is a discrepancy between the flows, it suggests that the mask is incorrect. Masks with flow_errors greater than 0.4 are discarded by default. This setting can be changed in Cellpose.eval or CellposeModel.eval.

Parameters

- **`maski`** (*`np.ndarray, int`*) – Masks produced from running dynamics on dP_net, where 0=NO masks; 1,2… are mask labels.
- **`dP_net`** (*`np.ndarray, float`*) – ND flows where dP_net.shape[1:] = maski.shape.

Returns

Mean squared error between predicted flows and flows from masks. dP_masks (np.ndarray, float): ND flows produced from the predicted masks.

Return type

flow_errors (np.ndarray, float)

`cellpose.metrics.`**`mask_ious`**(*masks_true*, *masks_pred*)

Return best-matched masks.

13.8 Flows to masks

Copyright © 2023 Howard Hughes Medical Institute, Authored by Carsen Stringer and Marius Pachitariu.

`cellpose.dynamics.`**`compute_masks`**(*dP*, *cellprob*, *p=None*, *niter=200*, *cellprob_threshold=0.0*, *flow_threshold=0.4*, *interp=True*, *do_3D=False*, *min_size=15*, *device=None*)

Compute masks using dynamics from dP and cellprob.

Parameters

- **`dP`** (*`numpy.ndarray`*) – The dynamics flow field array.
- **`cellprob`** (*`numpy.ndarray`*) – The cell probability array.
- **`p`** (*`numpy.ndarray, optional`*) – The pixels on which to run dynamics. Defaults to None
- **`niter`** (*`int, optional`*) – The number of iterations for mask computation. Defaults to 200.

- **cellprob_threshold** (*float, optional*) – The threshold for cell probability. Defaults to 0.0.
- **flow_threshold** (*float, optional*) – The threshold for quality control metrics. Defaults to 0.4.
- **interp** (*bool, optional*) – Whether to interpolate during dynamics computation. Defaults to True.
- **do_3D** (*bool, optional*) – Whether to perform mask computation in 3D. Defaults to False.
- **min_size** (*int, optional*) – The minimum size of the masks. Defaults to 15.
- **device** (*str, optional*) – The torch device to use for computation. Defaults to None.

Returns
A tuple containing the computed masks and the final pixel locations.

Return type
tuple

`cellpose.dynamics.`**`follow_flows`**`(`*dP*, *mask=None*, *niter=200*, *interp=True*, *device=None*`)`

Run dynamics to recover masks in 2D or 3D.

Pixels are represented as a meshgrid. Only pixels with non-zero cell-probability are used (as defined by inds).

Parameters

- **dP** (*np.ndarray*) – Flows [axis x Ly x Lx] or [axis x Lz x Ly x Lx].
- **mask** (*np.ndarray, optional*) – Pixel mask to seed masks. Useful when flows have low magnitudes.
- **niter** (*int, optional*) – Number of iterations of dynamics to run. Default is 200.
- **interp** (*bool, optional*) – Interpolate during 2D dynamics (not available in 3D). Default is True.
- **use_gpu** (*bool, optional*) – Use GPU to run interpolated dynamics (faster than CPU). Default is False.

Returns

- p (np.ndarray): Final locations of each pixel after dynamics; [axis x Ly x Lx] or [axis x Lz x Ly x Lx].
- inds (np.ndarray): Indices of pixels used for dynamics; [axis x Ly x Lx] or [axis x Lz x Ly x Lx].

Return type
tuple containing

`cellpose.dynamics.`**`get_centers`**`(`*masks*, *slices*`)`

Get the centers of the masks and their extents.

Parameters

- **masks** (*ndarray*) – The labeled masks.
- **slices** (*ndarray*) – The slices of the masks.

Returns

tuple containing

- centers (ndarray): The centers of the masks.

- ext (ndarray): The extents of the masks.

`cellpose.dynamics.`**`get_masks`**`(`*p*, *iscell=None*, *rpad=20*`)`

Create masks using pixel convergence after running dynamics.

Makes a histogram of final pixel locations p, initializes masks at peaks of histogram and extends the masks from the peaks so that they include all pixels with more than 2 final pixels p. Discards masks with flow errors greater than the threshold.

Parameters

- **p** (*float32, 3D or 4D array*) – Final locations of each pixel after dynamics, size [axis x Ly x Lx] or [axis x Lz x Ly x Lx].
- **iscell** (*bool, 2D or 3D array*) – If iscell is not None, set pixels that are iscell False to stay in their original location.
- **rpad** (*int, optional*) – Histogram edge padding. Default is 20.

Returns

Masks with inconsistent flow masks removed,
0=NO masks; 1,2,…=mask labels, size [Ly x Lx] or [Lz x Ly x Lx].

Return type

M0 (int, 2D or 3D array)

`cellpose.dynamics.`**`labels_to_flows`**`(`*labels*, *files=None*, *device=None*, *redo_flows=False*, *niter=None*, *return_flows=True*`)`

Converts labels (list of masks or flows) to flows for training model.

Parameters

- **labels** (*list of ND-arrays*) – The labels to convert. labels[k] can be 2D or 3D. If [3 x Ly x Lx], it is assumed that flows were precomputed. Otherwise, labels[k][0] or labels[k] (if 2D) is used to create flows and cell probabilities.
- **files** (*list of str, optional*) – The files to save the flows to. If provided, flows are saved to files to be reused. Defaults to None.
- **device** (*str, optional*) – The device to use for computation. Defaults to None.
- **redo_flows** (*bool, optional*) – Whether to recompute the flows. Defaults to False.
- **niter** (*int, optional*) – The number of iterations for computing flows. Defaults to None.

Returns

The flows for training the model. flows[k][0] is labels[k], flows[k][1] is cell distance transform, flows[k][2] is Y flow, flows[k][3] is X flow, and flows[k][4] is heat distribution.

Return type

list of [4 x Ly x Lx] arrays

`cellpose.dynamics.`**`map_coordinates`**`(`*I*, *yc*, *xc*, *Y*`)`

Bilinear interpolation of image "I" in-place with y-coordinates yc and x-coordinates xc to Y.

Parameters

- **I** (*numpy.ndarray*) – Input image of shape (C, Ly, Lx).
- **yc** (*numpy.ndarray*) – New y-coordinates.
- **xc** (*numpy.ndarray*) – New x-coordinates.
- **Y** (*numpy.ndarray*) – Output array of shape (C, ni).

Returns
None

`cellpose.dynamics.masks_to_flows`(*masks*, *device=None*, *niter=None*)

Convert masks to flows using diffusion from center pixel.

Center of masks where diffusion starts is defined to be the closest pixel to the mean of all pixels that is inside the mask. Result of diffusion is converted into flows by computing the gradients of the diffusion density map.

Parameters
masks (*int, 2D or 3D array*) – Labelled masks 0=NO masks; 1,2,…=mask labels

Returns

Flows in Y = mu[-2], flows in X = mu[-1].
If masks are 3D, flows in Z = mu[0].

Return type
mu (float, 3D or 4D array)

`cellpose.dynamics.masks_to_flows_cpu`(*masks*, *device=None*, *niter=None*)

Convert masks to flows using diffusion from center pixel.

Center of masks where diffusion starts is defined to be the closest pixel to the mean of all pixels that is inside the mask. Result of diffusion is converted into flows by computing the gradients of the diffusion density map.

Parameters
masks (*int, 2D or 3D array*) – Labelled masks 0=NO masks; 1,2,…=mask labels

Returns

tuple containing

- **mu (float, 3D or 4D array): Flows in Y = mu[-2], flows in X = mu[-1].**
  If masks are 3D, flows in Z = mu[0].
- meds (float, 2D or 3D array): cell centers

`cellpose.dynamics.masks_to_flows_gpu`(*masks*, *device=None*, *niter=None*)

Convert masks to flows using diffusion from center pixel.

Center of masks where diffusion starts is defined using COM.

Parameters
masks (*int, 2D or 3D array*) – Labelled masks. 0=NO masks; 1,2,…=mask labels.

Returns

tuple containing

- **mu (float, 3D or 4D array): Flows in Y = mu[-2], flows in X = mu[-1].**
  If masks are 3D, flows in Z = mu[0].
- meds_p (float, 2D or 3D array): cell centers

`cellpose.dynamics.masks_to_flows_gpu_3d`(*masks*, *device=None*)

Convert masks to flows using diffusion from center pixel.

Parameters
masks (*int, 2D or 3D array*) – Labelled masks. 0=NO masks; 1,2,…=mask labels.

Returns

tuple containing

- mu (float, 3D or 4D array): Flows in Y = mu[-2], flows in X = mu[-1]. If masks are 3D, flows in Z = mu[0].

- mu_c (float, 2D or 3D array): zeros

`cellpose.dynamics.`**`remove_bad_flow_masks`**(*masks*, *flows*, *threshold=0.4*, *device=None*)

Remove masks which have inconsistent flows.

Uses metrics.flow_error to compute flows from predicted masks and compare flows to predicted flows from the network. Discards masks with flow errors greater than the threshold.

Parameters

- **masks** (*int, 2D or 3D array*) – Labelled masks, 0=NO masks; 1,2,…=mask labels, size [Ly x Lx] or [Lz x Ly x Lx].
- **flows** (*float, 3D or 4D array*) – Flows [axis x Ly x Lx] or [axis x Lz x Ly x Lx].
- **threshold** (*float, optional*) – Masks with flow error greater than threshold are discarded. Default is 0.4.

Returns

Masks with inconsistent flow masks removed,
0=NO masks; 1,2,…=mask labels, size [Ly x Lx] or [Lz x Ly x Lx].

Return type

masks (int, 2D or 3D array)

`cellpose.dynamics.`**`resize_and_compute_masks`**(*dP*, *cellprob*, *p=None*, *niter=200*, *cellprob_threshold=0.0*, *flow_threshold=0.4*, *interp=True*, *do_3D=False*, *min_size=15*, *resize=None*, *device=None*)

Compute masks using dynamics from dP and cellprob, and resizes masks if resize is not None.

Parameters

- **dP** (*numpy.ndarray*) – The dynamics flow field array.
- **cellprob** (*numpy.ndarray*) – The cell probability array.
- **p** (*numpy.ndarray, optional*) – The pixels on which to run dynamics. Defaults to None
- **niter** (*int, optional*) – The number of iterations for mask computation. Defaults to 200.
- **cellprob_threshold** (*float, optional*) – The threshold for cell probability. Defaults to 0.0.
- **flow_threshold** (*float, optional*) – The threshold for quality control metrics. Defaults to 0.4.
- **interp** (*bool, optional*) – Whether to interpolate during dynamics computation. Defaults to True.
- **do_3D** (*bool, optional*) – Whether to perform mask computation in 3D. Defaults to False.
- **min_size** (*int, optional*) – The minimum size of the masks. Defaults to 15.
- **resize** (*tuple, optional*) – The desired size for resizing the masks. Defaults to None.
- **device** (*str, optional*) – The torch device to use for computation. Defaults to None.

Returns

A tuple containing the computed masks and the final pixel locations.

Return type
tuple

`cellpose.dynamics.`**`steps2D`**(*p*, *dP*, *inds*, *niter*)

Run dynamics of pixels to recover masks in 2D.

Euler integration of dynamics dP for niter steps.

Parameters

- **p** (*np.ndarray*) – Pixel locations [axis x Ly x Lx] (start at initial meshgrid).
- **dP** (*np.ndarray*) – Flows [axis x Ly x Lx].
- **inds** (*np.ndarray*) – Non-zero pixels to run dynamics on [npixels x 2].
- **niter** (*int*) – Number of iterations of dynamics to run.

Returns
Final locations of each pixel after dynamics.

Return type
np.ndarray

`cellpose.dynamics.`**`steps2D_interp`**(*p*, *dP*, *niter*, *device=None*)

Run dynamics of pixels to recover masks in 2D, with interpolation between pixel values.

Euler integration of dynamics dP for niter steps.

Parameters

- **p** (*numpy.ndarray*) – Array of shape (n_points, 2) representing the initial pixel locations.
- **dP** (*numpy.ndarray*) – Array of shape (2, Ly, Lx) representing the flow field.
- **niter** (*int*) – Number of iterations to perform.
- **device** (*torch.device, optional*) – Device to use for computation. Defaults to None.

Returns
Array of shape (n_points, 2) representing the final pixel locations.

Return type
numpy.ndarray

Raises
`None` –

`cellpose.dynamics.`**`steps3D`**(*p*, *dP*, *inds*, *niter*)

Run dynamics of pixels to recover masks in 3D.

Euler integration of dynamics dP for niter steps.

Parameters

- **p** (*np.ndarray*) – Pixel locations [axis x Lz x Ly x Lx] (start at initial meshgrid).
- **dP** (*np.ndarray*) – Flows [axis x Lz x Ly x Lx].
- **inds** (*np.ndarray*) – Non-zero pixels to run dynamics on [npixels x 3].
- **niter** (*int*) – Number of iterations of dynamics to run.

Returns
Final locations of each pixel after dynamics.

Return type
np.ndarray

13.9 Image transforms

`cellpose.transforms.`**`average_tiles`**`(`*y*, *ysub*, *xsub*, *Ly*, *Lx*`)`

Average the results of the network over tiles.

Parameters

- **y** (*float*) – Output of cellpose network for each tile. Shape: [ntiles x nclasses x bsize x bsize]
- **ysub** (*list*) – List of arrays with start and end of tiles in Y of length ntiles
- **xsub** (*list*) – List of arrays with start and end of tiles in X of length ntiles
- **Ly** (*int*) – Size of pre-tiled image in Y (may be larger than original image if image size is less than bsize)
- **Lx** (*int*) – Size of pre-tiled image in X (may be larger than original image if image size is less than bsize)

Returns
Network output averaged over tiles. Shape: [nclasses x Ly x Lx]

Return type
yf (float32)

`cellpose.transforms.`**`convert_image`**`(`*x*, *channels*, *channel_axis=None*, *z_axis=None*, *do_3D=False*, *nchan=2*`)`

Converts the image to have the z-axis first, channels last, and normalized intensities.

Parameters

- **x** (*numpy.ndarray or torch.Tensor*) – The input image.
- **channels** (*list or None*) – The list of channels to use (ones-based, 0=gray). If None, all channels are kept.
- **channel_axis** (*int or None*) – The axis of the channels in the input image. If None, the axis is determined automatically.
- **z_axis** (*int or None*) – The axis of the z-dimension in the input image. If None, the axis is determined automatically.
- **do_3D** (*bool*) – Whether to process the image in 3D mode. Defaults to False.
- **nchan** (*int*) – The number of channels to keep if the input image has more than nchan channels.

Returns
The converted image.

Return type
numpy.ndarray

Raises

- **ValueError** – If the input image has less than two channels and channels are not specified.

- `ValueError` – If the input image is 2D and do_3D is True.
- `ValueError` – If the input image is 4D and do_3D is False.

`cellpose.transforms.`**`gaussian_kernel`**(*sigma*, *Ly*, *Lx*, *device=torch.device*)

Generates a 2D Gaussian kernel.

Parameters

- **sigma** (*float*) – Standard deviation of the Gaussian distribution.
- **Ly** (*int*) – Number of pixels in the y-axis.
- **Lx** (*int*) – Number of pixels in the x-axis.
- **device** (*torch.device, optional*) – Device to store the kernel tensor. Defaults to torch.device("cpu").

Returns

2D Gaussian kernel tensor.

Return type

torch.Tensor

`cellpose.transforms.`**`make_tiles`**(*imgi*, *bsize=224*, *augment=False*, *tile_overlap=0.1*)

Make tiles of image to run at test-time.

Parameters

- **imgi** (*np.ndarray*) – Array of shape (nchan, Ly, Lx) representing the input image.
- **bsize** (*int, optional*) – Size of tiles. Defaults to 224.
- **augment** (*bool, optional*) – Whether to flip tiles and set tile_overlap=2. Defaults to False.
- **tile_overlap** (*float, optional*) – Fraction of overlap of tiles. Defaults to 0.1.

Returns

tuple containing

- IMG (np.ndarray): Array of shape (ntiles, nchan, bsize, bsize) representing the tiles.
- ysub (list): List of arrays with start and end of tiles in Y of length ntiles.
- xsub (list): List of arrays with start and end of tiles in X of length ntiles.
- Ly (int): Height of the input image.
- Lx (int): Width of the input image.

`cellpose.transforms.`**`move_axis`**(*img*, *m_axis=-1*, *first=True*)

move axis m_axis to first or last position

`cellpose.transforms.`**`move_min_dim`**(*img*, *force=False*)

Move the minimum dimension last as channels if it is less than 10 or force is True.

Parameters

- **img** (*ndarray*) – The input image.
- **force** (*bool, optional*) – If True, the minimum dimension will always be moved. Defaults to False.

Returns

The image with the minimum dimension moved to the last axis as channels.

Return type
ndarray

`cellpose.transforms.`**`normalize99`**`(`*Y*, *lower=1*, *upper=99*, *copy=True*`)`

Normalize the image so that 0.0 corresponds to the 1st percentile and 1.0 corresponds to the 99th percentile.

Parameters

- **`Y`** (*`ndarray`*) – The input image.
- **`lower`** (*`int, optional`*) – The lower percentile. Defaults to 1.
- **`upper`** (*`int, optional`*) – The upper percentile. Defaults to 99.
- **`copy`** (*`bool, optional`*) – Whether to create a copy of the input image. Defaults to True.

Returns
The normalized image.

Return type
ndarray

`cellpose.transforms.`**`normalize99_tile`**`(`*img*, *blocksize=100*, *lower=1.0*, *upper=99.0*, *tile_overlap=0.1*, *norm3D=False*, *smooth3D=1*, *is3D=False*`)`

Compute normalization like normalize99 function but in tiles.

Parameters

- **`img`** (*`numpy.ndarray`*) – Array of shape (Lz x) Ly x Lx (x nchan) containing the image.
- **`blocksize`** (*`float, optional`*) – Size of tiles. Defaults to 100.
- **`lower`** (*`float, optional`*) – Lower percentile for normalization. Defaults to 1.0.
- **`upper`** (*`float, optional`*) – Upper percentile for normalization. Defaults to 99.0.
- **`tile_overlap`** (*`float, optional`*) – Fraction of overlap of tiles. Defaults to 0.1.
- **`norm3D`** (*`bool, optional`*) – Use same tiled normalization for each z-plane. Defaults to False.
- **`smooth3D`** (*`int, optional`*) – Smoothing factor for 3D normalization. Defaults to 1.
- **`is3D`** (*`bool, optional`*) – Set to True if image is a 3D stack. Defaults to False.

Returns
Normalized image array of shape (Lz x) Ly x Lx (x nchan).

Return type
numpy.ndarray

`cellpose.transforms.`**`normalize_img`**`(`*img*, *normalize=True*, *norm3D=False*, *invert=False*, *lowhigh=None*, *percentile=None*, *sharpen_radius=0*, *smooth_radius=0*, *tile_norm_blocksize=0*, *tile_norm_smooth3D=1*, *axis=-1*`)`

Normalize each channel of the image.

Parameters

- **`img`** (*`ndarray`*) – The input image. It should have at least 3 dimensions. If it is 4-dimensional, it assumes the first non-channel axis is the Z dimension.
- **`normalize`** (*`bool, optional`*) – Whether to perform normalization. Defaults to True.
- **`norm3D`** (*`bool, optional`*) – Whether to normalize in 3D. Defaults to False.
- **`invert`** (*`bool, optional`*) – Whether to invert the image. Useful if cells are dark instead of bright. Defaults to False.

- **lowhigh** (*tuple, optional*) – The lower and upper bounds for normalization. If provided, it should be a tuple of two values. Defaults to None.
- **percentile** (*tuple, optional*) – The lower and upper percentiles for normalization. If provided, it should be a tuple of two values. Each value should be between 0 and 100. Defaults to None.
- **sharpen_radius** (*int, optional*) – The radius for sharpening the image. Defaults to 0.
- **smooth_radius** (*int, optional*) – The radius for smoothing the image. Defaults to 0.
- **tile_norm_blocksize** (*int, optional*) – The block size for tile-based normalization. Defaults to 0.
- **tile_norm_smooth3D** (*int, optional*) – The smoothness factor for tile-based normalization in 3D. Defaults to 1.
- **axis** (*int, optional*) – The channel axis to loop over for normalization. Defaults to -1.

Returns
The normalized image of the same size.

Return type
ndarray

Raises

- **ValueError** – If the image has less than 3 dimensions.
- **ValueError** – If the provided lowhigh or percentile values are invalid.
- **ValueError** – If the image is inverted without normalization.

cellpose.transforms.**pad_image_ND**(*img0*, *div=16*, *extra=1*, *min_size=None*)

Pad image for test-time so that its dimensions are a multiple of 16 (2D or 3D).

Parameters

- **img0** (*ndarray*) – Image of size [nchan (x Lz) x Ly x Lx].
- **div** (*int, optional*) – Divisor for padding. Defaults to 16.
- **extra** (*int, optional*) – Extra padding. Defaults to 1.
- **min_size** (*tuple, optional*) – Minimum size of the image. Defaults to None.

Returns

tuple containing

- I (ndarray): Padded image.
- ysub (ndarray): Y range of pixels in the padded image corresponding to img0.
- xsub (ndarray): X range of pixels in the padded image corresponding to img0.

cellpose.transforms.**random_rotate_and_resize**(*X*, *Y=None*, *scale_range=1.0*, *xy=(224, 224)*, *do_3D=False*, *do_flip=True*, *rotate=True*, *rescale=None*, *unet=False*, *random_per_image=True*)

Augmentation by random rotation and resizing.

Parameters

- **X** (*list of ND-arrays, float*) – List of image arrays of size [nchan x Ly x Lx] or [Ly x Lx].
- **Y** (*list of ND-arrays, float, optional*) – List of image labels of size [nlabels x Ly x Lx] or [Ly x Lx]. The 1st channel of Y is always nearest-neighbor interpolated (assumed to be masks or 0-1 representation). If Y.shape[0]==3 and not unet, then the labels are assumed to be [cell probability, Y flow, X flow]. If unet, second channel is dist_to_bound. Defaults to None.
- **scale_range** (*float, optional*) – Range of resizing of images for augmentation. Images are resized by (1-scale_range/2) + scale_range * np.random.rand(). Defaults to 1.0.
- **xy** (*tuple, int, optional*) – Size of transformed images to return. Defaults to (224,224).
- **do_flip** (*bool, optional*) – Whether or not to flip images horizontally. Defaults to True.
- **rotate** (*bool, optional*) – Whether or not to rotate images. Defaults to True.
- **rescale** (*array, float, optional*) – How much to resize images by before performing augmentations. Defaults to None.
- **unet** (*bool, optional*) – Whether or not to use unet. Defaults to False.
- **random_per_image** (*bool, optional*) – Different random rotate and resize per image. Defaults to True.

Returns

tuple containing

- imgi (ND-array, float): Transformed images in array [nimg x nchan x xy[0] x xy[1]].
- lbl (ND-array, float): Transformed labels in array [nimg x nchan x xy[0] x xy[1]].
- scale (array, float): Amount each image was resized by.

cellpose.transforms.**reshape**(*data*, *channels=[0, 0]*, *chan_first=False*)

Reshape data using channels.

Parameters

- **data** (*numpy.ndarray*) – The input data. It should have shape (Z x) Ly x Lx x nchan if data.ndim==8 and data.shape[0]<8, it is assumed to be nchan x Ly x Lx.
- **channels** (*list of int, optional*) – The channels to use for reshaping. The first element of the list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). The second element of the list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to train on grayscale images, input [0,0]. To train on images with cells in green and nuclei in blue, input [2,3]. Defaults to [0, 0].
- **chan_first** (*bool, optional*) – Whether to return the reshaped data with channel as the first dimension. Defaults to False.

Returns

The reshaped data with shape (Z x) Ly x Lx x nchan (if chan_first==False).

Return type

numpy.ndarray

`cellpose.transforms.`**`resize_image`**`(`*img0*, *Ly=None*, *Lx=None*, *rsz=None*, *interpolation=cv2.INTER_LINEAR*, *no_channels=False*`)`

Resize image for computing flows / unresize for computing dynamics.

Parameters

- **img0** (*ndarray*) – Image of size [Y x X x nchan] or [Lz x Y x X x nchan] or [Lz x Y x X].
- **Ly** (*int, optional*) – Desired height of the resized image. Defaults to None.
- **Lx** (*int, optional*) – Desired width of the resized image. Defaults to None.
- **rsz** (*float, optional*) – Resize coefficient(s) for the image. If Ly is None, rsz is used. Defaults to None.
- **interpolation** (*int, optional*) – OpenCV interpolation method. Defaults to cv2.INTER_LINEAR.
- **no_channels** (*bool, optional*) – Flag indicating whether to treat the third dimension as a channel. Defaults to False.

Returns

Resized image of size [Ly x Lx x nchan] or [Lz x Ly x Lx x nchan].

Return type

ndarray

Raises

ValueError – If Ly is None and rsz is None.

`cellpose.transforms.`**`smooth_sharpen_img`**`(`*img*, *smooth_radius=6*, *sharpen_radius=12*, *device=torch.device*, *is3D=False*`)`

Sharpen blurry images with surround subtraction and/or smooth noisy images.

Parameters

- **img** (*float32*) – Array that’s (Lz x) Ly x Lx (x nchan).
- **smooth_radius** (*float, optional*) – Size of gaussian smoothing filter, recommended to be 1/10-1/4 of cell diameter (if also sharpening, should be 2-3x smaller than sharpen_radius). Defaults to 6.
- **sharpen_radius** (*float, optional*) – Size of gaussian surround filter, recommended to be 1/8-1/2 of cell diameter (if also smoothing, should be 2-3x larger than smooth_radius). Defaults to 12.
- **device** (*torch.device, optional*) – Device on which to perform sharpening. Will be faster on GPU but need to ensure GPU has RAM for image. Defaults to torch.device(“cpu”).
- **is3D** (*bool, optional*) – If image is 3D stack (only necessary to set if img.ndim==3). Defaults to False.

Returns

Array that’s (Lz x) Ly x Lx (x nchan).

Return type

img_sharpen (float32)

`cellpose.transforms.`**`unaugment_tiles`**`(`*y*`)`

Reverse test-time augmentations for averaging (includes flipping of flowsY and flowsX).

Parameters
y (*float32*) – Array of shape (ntiles_y, ntiles_x, chan, Ly, Lx) where chan = (flowsY, flowsX, cell prob).

Returns
Array of shape (ntiles_y, ntiles_x, chan, Ly, Lx).

Return type
float32

`cellpose.transforms.update_axis(m_axis, to_squeeze, ndim)`

Squeeze the axis value based on the given parameters.

Parameters

- **m_axis** (*int*) – The current axis value.
- **to_squeeze** (*numpy.ndarray*) – An array of indices to squeeze.
- **ndim** (*int*) – The number of dimensions.

Returns
The updated axis value.

Return type
int or None

13.10 Plot functions

Copyright © 2023 Howard Hughes Medical Institute, Authored by Carsen Stringer and Marius Pachitariu.

`cellpose.plot.disk(med, r, Ly, Lx)`

Returns the pixels of a disk with a given radius and center.

Parameters

- **med** (*tuple*) – The center coordinates of the disk.
- **r** (*float*) – The radius of the disk.
- **Ly** (*int*) – The height of the image.
- **Lx** (*int*) – The width of the image.

Returns
A tuple containing the y and x coordinates of the pixels within the disk.

Return type
tuple

`cellpose.plot.dx_to_circ(dP, transparency=False, mask=None)`

Converts the optic flow representation to a circular color representation.

Parameters

- **dP** (*ndarray*) – Flow field components [dy, dx].
- **transparency** (*bool, optional*) – Controls the opacity based on the magnitude of flow. Defaults to False.
- **mask** (*ndarray, optional*) – Multiplies each RGB component to suppress noise.

Returns
The circular color representation of the optic flow.

Return type
ndarray

`cellpose.plot.`**`image_to_rgb`**(*img0*, *channels=[0, 0]*)

Converts image from 2 x Ly x Lx or Ly x Lx x 2 to RGB Ly x Lx x 3.

Parameters
`img0` (*`ndarray`*) – Input image of shape 2 x Ly x Lx or Ly x Lx x 2.

Returns
RGB image of shape Ly x Lx x 3.

Return type
ndarray

`cellpose.plot.`**`interesting_patch`**(*mask*, *bsize=130*)

Get patch of size bsize x bsize with most masks.

Parameters

- **`mask`** (*`ndarray`*) – Input mask.
- **`bsize`** (*`int`*) – Size of the patch.

Returns
Patch coordinates (y, x).

Return type
tuple

`cellpose.plot.`**`mask_overlay`**(*img*, *masks*, *colors=None*)

Overlay masks on image (set image to grayscale).

Parameters

- **`img`** (*`int or float, 2D or 3D array`*) – Image of size [Ly x Lx (x nchan)].
- **`masks`** (*`int, 2D array`*) – Masks where 0=NO masks; 1,2,…=mask labels.
- **`colors`** (*`int, 2D array, optional`*) – Size [nmasks x 3], each entry is a color in 0-255 range.

Returns
Array of masks overlaid on grayscale image.

Return type
RGB (uint8, 3D array)

`cellpose.plot.`**`mask_rgb`**(*masks*, *colors=None*)

Masks in random RGB colors.

Parameters

- **`masks`** (*`int, 2D array`*) – Masks where 0=NO masks; 1,2,…=mask labels.
- **`colors`** (*`int, 2D array, optional`*) – Size [nmasks x 3], each entry is a color in 0-255 range.

Returns
Array of masks overlaid on grayscale image.

Return type
RGB (uint8, 3D array)

`cellpose.plot.`**`outline_view`**(*img0*, *maski*, *color=[1, 0, 0]*, *mode='inner'*)

Generates a red outline overlay onto the image.

Parameters

- **img0** (*numpy.ndarray*) – The input image.
- **maski** (*numpy.ndarray*) – The mask representing the region of interest.
- **color** (*list, optional*) – The color of the outline overlay. Defaults to [1, 0, 0] (red).
- **mode** (*str, optional*) – The mode for generating the outline. Defaults to "inner".

Returns
The image with the red outline overlay.

Return type
numpy.ndarray

`cellpose.plot.`**`show_segmentation`**(*fig*, *img*, *maski*, *flowi*, *channels=[0, 0]*, *file_name=None*)

Plot segmentation results (like on website).

Can save each panel of figure with file_name option. Use channels option if img input is not an RGB image with 3 channels.

Parameters

- **fig** (*matplotlib.pyplot.figure*) – Figure in which to make plot.
- **img** (*ndarray*) – 2D or 3D array. Image input into cellpose.
- **maski** (*int, ndarray*) – For image k, masks[k] output from Cellpose.eval, where 0=NO masks; 1,2,…=mask labels.
- **flowi** (*int, ndarray*) – For image k, flows[k][0] output from Cellpose.eval (RGB of flows).
- **channels** (*list of int, optional*) – Channels used to run Cellpose, no need to use if image is RGB. Defaults to [0, 0].
- **file_name** (*str, optional*) – File name of image. If file_name is not None, figure panels are saved. Defaults to None.
- **seg_norm** (*bool, optional*) – Improve cell visibility under labels. Defaults to False.

13.11 I/O functions

`cellpose.io.`**`add_model`**(*filename*)

add model to .cellpose models folder to use with GUI or CLI

`cellpose.io.`**`get_image_files`**(*folder*, *mask_filter*, *imf=None*, *look_one_level_down=False*)

Finds all images in a folder and its subfolders (if specified) with the given file extensions.

Parameters

- **folder** (*str*) – The path to the folder to search for images.
- **mask_filter** (*str*) – The filter for mask files.
- **imf** (*str, optional*) – The additional filter for image files. Defaults to None.
- **look_one_level_down** (*bool, optional*) – Whether to search for images in subfolders. Defaults to False.

Returns
A list of image file paths.

Return type
list

Raises

- **ValueError** – If no files are found in the specified folder.
- **ValueError** – If no images are found in the specified folder with the supported file extensions.
- **ValueError** – If no images are found in the specified folder without the mask or flow file endings.

`cellpose.io.`**`get_label_files`**`(`*image_names*, *mask_filter*, *imf=None*`)`

Get the label files corresponding to the given image names and mask filter.

Parameters

- **image_names** (*list*) – List of image names.
- **mask_filter** (*str*) – Mask filter to be applied.
- **imf** (*str, optional*) – Image file extension. Defaults to None.

Returns
A tuple containing the label file names and flow file names (if present).

Return type
tuple

`cellpose.io.`**`imread`**`(`*filename*`)`

Read in an image file with tif or image file type supported by cv2.

Parameters
filename (*str*) – The path to the image file.

Returns
The image data as a NumPy array.

Return type
numpy.ndarray

Raises
None –

Raises an error if the image file format is not supported.

Examples

```
>>> img = imread("image.tif")
```

`cellpose.io.`**`imsave`**`(`*filename*, *arr*`)`

Saves an image array to a file.

Parameters

- **filename** (*str*) – The name of the file to save the image to.
- **arr** (*numpy.ndarray*) – The image array to be saved.

Returns
None

`cellpose.io.`**`load_images_labels`**(*tdir*, *mask_filter='_masks'*, *image_filter=None*, *look_one_level_down=False*)

Loads images and corresponding labels from a directory.

Parameters

- **`tdir`** (*`str`*) – The directory path.
- **`mask_filter`** (*`str, optional`*) – The filter for mask files. Defaults to "_masks".
- **`image_filter`** (*`str, optional`*) – The filter for image files. Defaults to None.
- **`look_one_level_down`** (*`bool, optional`*) – Whether to look for files one level down. Defaults to False.

Returns

A tuple containing a list of images, a list of labels, and a list of image names.

Return type

tuple

`cellpose.io.`**`load_train_test_data`**(*train_dir*, *test_dir=None*, *image_filter=None*, *mask_filter='_masks'*, *look_one_level_down=False*)

Loads training and testing data for a Cellpose model.

Parameters

- **`train_dir`** (*`str`*) – The directory path containing the training data.
- **`test_dir`** (*`str, optional`*) – The directory path containing the testing data. Defaults to None.
- **`image_filter`** (*`str, optional`*) – The filter for selecting image files. Defaults to None.
- **`mask_filter`** (*`str, optional`*) – The filter for selecting mask files. Defaults to "_masks".
- **`look_one_level_down`** (*`bool, optional`*) – Whether to look for data in subdirectories of train_dir and test_dir. Defaults to False.

Returns

A list of training images. labels (list): A list of labels corresponding to the training images. image_names (list): A list of names of the training images. test_images (list, optional): A list of testing images. None if test_dir is not provided. test_labels (list, optional): A list of labels corresponding to the testing images. None if test_dir is not provided. test_image_names (list, optional): A list of names of the testing images. None if test_dir is not provided.

Return type

images (list)

`cellpose.io.`**`masks_flows_to_seg`**(*images*, *masks*, *flows*, *file_names*, *diams=30.0*, *channels=None*, *imgs_restore=None*, *restore_type=None*, *ratio=1.0*)

Save output of model eval to be loaded in GUI.

Can be list output (run on multiple images) or single output (run on single image).

Saved to file_names[k]+"_seg.npy".

Parameters

- **`images`** (*`list`*) – Images input into cellpose.
- **`masks`** (*`list`*) – Masks output from Cellpose.eval, where 0=NO masks; 1,2,…=mask labels.

- **flows** (*list*) – Flows output from Cellpose.eval.
- **file_names** (*list, str*) – Names of files of images.
- **diams** (*float array*) – Diameters used to run Cellpose. Defaults to 30.
- **channels** (*list, int, optional*) – Channels used to run Cellpose. Defaults to None.

Returns
None

`cellpose.io.`**`remove_model`**(*filename*, *delete=False*)

remove model from .cellpose custom model list

`cellpose.io.`**`save_masks`**(*images*, *masks*, *flows*, *file_names*, *png=True*, *tif=False*, *channels=[0, 0]*, *suffix=''*, *save_flows=False*, *save_outlines=False*, *dir_above=False*, *in_folders=False*, *savedir=None*, *save_txt=False*, *save_mpl=False*)

Save masks + nicely plotted segmentation image to png and/or tiff.

Can save masks, flows to different directories, if in_folders is True.

If png, masks[k] for images[k] are saved to file_names[k]+"_cp_masks.png".

If tif, masks[k] for images[k] are saved to file_names[k]+"_cp_masks.tif".

If png and matplotlib installed, full segmentation figure is saved to file_names[k]+"_cp.png".

Only tif option works for 3D data, and only tif option works for empty masks.

Parameters

- **images** (*list*) – Images input into cellpose.
- **masks** (*list*) – Masks output from Cellpose.eval, where 0=NO masks; 1,2,. . . =mask labels.
- **flows** (*list*) – Flows output from Cellpose.eval.
- **file_names** (*list, str*) – Names of files of images.
- **png** (*bool, optional*) – Save masks to PNG. Defaults to True.
- **tif** (*bool, optional*) – Save masks to TIF. Defaults to False.
- **channels** (*list, int, optional*) – Channels used to run Cellpose. Defaults to [0,0].
- **suffix** (*str, optional*) – Add name to saved masks. Defaults to "".
- **save_flows** (*bool, optional*) – Save flows output from Cellpose.eval. Defaults to False.
- **save_outlines** (*bool, optional*) – Save outlines of masks. Defaults to False.
- **dir_above** (*bool, optional*) – Save masks/flows in directory above. Defaults to False.
- **in_folders** (*bool, optional*) – Save masks/flows in separate folders. Defaults to False.
- **savedir** (*str, optional*) – Absolute path where images will be saved. If None, saves to image directory. Defaults to None.
- **save_txt** (*bool, optional*) – Save masks as list of outlines for ImageJ. Defaults to False.

- `save_mpl` (*bool, optional*) – If True, saves a matplotlib figure of the original image/segmentation/flows. Does not work for 3D. This takes a long time for large images. Defaults to False.

Returns
None

`cellpose.io.`**`save_rois`**(*masks*, *file_name*)

save masks to .roi files in .zip archive for ImageJ/Fiji

Parameters

- **`masks`** (*np.ndarray*) – masks output from Cellpose.eval, where 0=NO masks; 1,2,. . . =mask labels
- **`file_name`** (*str*) – name to save the .zip file to

Returns
None

`cellpose.io.`**`save_to_png`**(*images*, *masks*, *flows*, *file_names*)

deprecated (runs io.save_masks with png=True)

does not work for 3D images

13.12 Utils functions

Copyright © 2023 Howard Hughes Medical Institute, Authored by Carsen Stringer and Marius Pachitariu.

`class cellpose.utils.`**`TqdmToLogger`**(*logger*, *level=None*)

Output stream for TQDM which will output to logger module instead of the StdOut.

`flush`()

Flush write buffers, if applicable.

This is not implemented for read-only and non-blocking streams.

`write`(*buf*)

Write string to file.

Returns the number of characters written, which is always equal to the length of the string.

`cellpose.utils.`**`circleMask`**(*d0*)

Creates an array with indices which are the radius of that x,y point.

Parameters
`d0` (*tuple*) – Patch of (-d0, d0+1) over which radius is computed.

Returns

A tuple containing:

- rs (ndarray): Array of radii with shape (2*d0[0]+1, 2*d0[1]+1).
- dx (ndarray): Indices of the patch along the x-axis.
- dy (ndarray): Indices of the patch along the y-axis.

Return type
tuple

`cellpose.utils.`**`diameters`**`(`*masks*`)`

Calculate the diameters of the objects in the given masks.

Parameters: masks (ndarray): masks (0=no cells, 1=first cell, 2=second cell,. . .)

Returns: tuple: A tuple containing the median diameter and an array of diameters for each object.

Examples: >>> masks = np.array([[0, 1, 1], [1, 0, 0], [1, 1, 0]]) >>> diameters(masks) (1.0, array([1.41421356, 1.0, 1.0]))

`cellpose.utils.`**`dilate_masks`**`(`*masks*, *n_iter=5*`)`

Dilate masks by n_iter pixels.

Parameters

- **`masks`** (*`ndarray`*) – Array of masks.
- **`n_iter`** (*`int, optional`*) – Number of pixels to dilate the masks. Defaults to 5.

Returns

Dilated masks.

Return type

ndarray

`cellpose.utils.`**`distance_to_boundary`**`(`*masks*`)`

Get the distance to the boundary of mask pixels.

Parameters

`masks` (*`int, 2D or 3D array`*) – The masks array. Size [Ly x Lx] or [Lz x Ly x Lx], where 0 represents no mask and 1, 2, . . . represent mask labels.

Returns

The distance to the boundary. Size [Ly x Lx] or [Lz x Ly x Lx].

Return type

dist_to_bound (2D or 3D array)

Raises

`ValueError` – If the masks array is not 2D or 3D.

`cellpose.utils.`**`download_url_to_file`**`(`*url*, *dst*, *progress=True*`)`

Download object at the given URL to a local path.

Thanks to torch, slightly modified

Parameters

- **`url`** (*`string`*) – URL of the object to download
- **`dst`** (*`string`*) – Full path where object will be saved, e.g. */tmp/temporary_file*
- **`progress`** (*`bool, optional`*) – whether or not to display a progress bar to stderr Default: True

`cellpose.utils.`**`fill_holes_and_remove_small_masks`**`(`*masks*, *min_size=15*`)`

Fills holes in masks (2D/3D) and discards masks smaller than min_size.

This function fills holes in each mask using scipy.ndimage.morphology.binary_fill_holes. It also removes masks that are smaller than the specified min_size.

Parameters: masks (ndarray): Int, 2D or 3D array of labelled masks.

0 represents no mask, while positive integers represent mask labels. The size can be [Ly x Lx] or [Lz x Ly x Lx].

min_size (int, optional): Minimum number of pixels per mask.
Masks smaller than min_size will be removed. Set to -1 to turn off this functionality. Default is 15.

Returns: ndarray: Int, 2D or 3D array of masks with holes filled and small masks removed.
0 represents no mask, while positive integers represent mask labels. The size is [Ly x Lx] or [Lz x Ly x Lx].

`cellpose.utils.`**`get_mask_compactness`**`(`*masks*`)`

Calculate the compactness of masks.

Parameters
`masks` (*`ndarray`*) – Binary masks representing objects.

Returns
Array of compactness values for each mask.

Return type
ndarray

`cellpose.utils.`**`get_mask_perimeters`**`(`*masks*`)`

Calculate the perimeters of the given masks.

Parameters
`masks` (*`numpy.ndarray`*) – Binary masks representing objects.

Returns
Array containing the perimeters of each mask.

Return type
numpy.ndarray

`cellpose.utils.`**`get_mask_stats`**`(`*masks_true*`)`

Calculate various statistics for the given binary masks.

Parameters
`masks_true` (*`ndarray`*) – masks (0=no cells, 1=first cell, 2=second cell,. . .)

Returns
Convexity values for each mask. solidity (ndarray): Solidity values for each mask. compactness (ndarray): Compactness values for each mask.

Return type
convexity (ndarray)

`cellpose.utils.`**`get_masks_unet`**`(`*output*, *cell_threshold=0*, *boundary_threshold=0*`)`

Create masks using cell probability and cell boundary.

Parameters

- **`output`** (*`ndarray`*) – The output array containing cell probability and cell boundary.
- **`cell_threshold`** (*`float, optional`*) – The threshold value for cell probability. Defaults to 0.
- **`boundary_threshold`** (*`float, optional`*) – The threshold value for cell boundary. Defaults to 0.

Returns
The masks representing the segmented cells.

Return type
ndarray

`cellpose.utils.`**`get_outline_multi`**`(`*args*`)`

Get the outline of a specific mask in a multi-mask image.

Parameters
: **args** (*tuple*) – A tuple containing the masks and the mask number.

Returns
: The outline of the specified mask as an array of coordinates.

Return type
: numpy.ndarray

`cellpose.utils.`**`get_perimeter`**`(`*points*`)`

Calculate the perimeter of a set of points.

Parameters
: **points** (*ndarray*) – An array of points with shape (npoints, ndim).

Returns
: The perimeter of the points.

Return type
: float

`cellpose.utils.`**`masks_to_edges`**`(`*masks*, *threshold=1.0*`)`

Get edges of masks as a 0-1 array.

Parameters

- **masks** (*int, 2D or 3D array*) – Size [Ly x Lx] or [Lz x Ly x Lx], where 0=NO masks and 1,2,…=mask labels.
- **threshold** (*float, optional*) – Threshold value for distance to boundary. Defaults to 1.0.

Returns
: Size [Ly x Lx] or [Lz x Ly x Lx], where True pixels are edge pixels.

Return type
: edges (2D or 3D array)

`cellpose.utils.`**`masks_to_outlines`**`(`*masks*`)`

Get outlines of masks as a 0-1 array.

Parameters
: **masks** (*int, 2D or 3D array*) – Size [Ly x Lx] or [Lz x Ly x Lx], where 0=NO masks and 1,2,…=mask labels.

Returns
: Size [Ly x Lx] or [Lz x Ly x Lx], where True pixels are outlines.

Return type
: outlines (2D or 3D array)

`cellpose.utils.`**`outlines_list`**`(`*masks*, *multiprocessing_threshold=1000*, *multiprocessing=None*`)`

Get outlines of masks as a list to loop over for plotting.

Parameters

- **masks** (*ndarray*) – Array of masks.
- **multiprocessing_threshold** (*int, optional*) – Threshold for enabling multiprocessing. Defaults to 1000.
- **multiprocessing** (*bool, optional*) – Flag to enable multiprocessing. Defaults to None.

Returns
: List of outlines.

Return type
list

Raises
`None` –

Notes

- This function is a wrapper for outlines_list_single and outlines_list_multi.
- Multiprocessing is disabled for Windows.

`cellpose.utils.`**`outlines_list_multi`**(*masks*, *num_processes=None*)

Get outlines of masks as a list to loop over for plotting.

Parameters
masks (*ndarray*) – masks (0=no cells, 1=first cell, 2=second cell,…)

Returns
List of outlines as pixel coordinates.

Return type
list

`cellpose.utils.`**`outlines_list_single`**(*masks*)

Get outlines of masks as a list to loop over for plotting.

Parameters
masks (*ndarray*) – masks (0=no cells, 1=first cell, 2=second cell,…)

Returns
List of outlines as pixel coordinates.

Return type
list

`cellpose.utils.`**`radius_distribution`**(*masks*, *bins*)

Calculate the radius distribution of masks.

Parameters

- **masks** (*ndarray*) – masks (0=no cells, 1=first cell, 2=second cell,…)
- **bins** (*int*) – Number of bins for the histogram.

Returns

A tuple containing:

- nb (ndarray): Normalized histogram of radii.
- md (float): Median radius.
- radii (ndarray): Array of radii.

Return type
tuple

`cellpose.utils.`**`remove_edge_masks`**(*masks*, *change_index=True*)

Removes masks with pixels on the edge of the image.

Parameters

- **masks** (*int, 2D or 3D array*) – The masks to be processed. Size [Ly x Lx] or [Lz x Ly x Lx], where 0 represents no mask and 1, 2, … represent mask labels.

- **change_index** (`bool, optional`) – If True, after removing masks, changes the indexing so that there are no missing label numbers. Defaults to True.

Returns
The processed masks. Size [Ly x Lx] or [Lz x Ly x Lx], where 0 represents no mask and 1, 2, … represent mask labels.

Return type
outlines (2D or 3D array)

`cellpose.utils.`**`size_distribution`**`(`*masks*`)`

Calculates the size distribution of masks.

Parameters
masks (`ndarray`) – masks (0=no cells, 1=first cell, 2=second cell,…)

Returns
The ratio of the 25th percentile of mask sizes to the 75th percentile of mask sizes.

Return type
float

`cellpose.utils.`**`stitch3D`**`(`*masks*, *stitch_threshold=0.25*`)`

Stitch 2D masks into a 3D volume using a stitch_threshold on IOU.

Parameters

- **masks** (`list or ndarray`) – List of 2D masks.
- **stitch_threshold** (`float, optional`) – Threshold value for stitching. Defaults to 0.25.

Returns
List of stitched 3D masks.

Return type
list

13.13 Network classes

Copyright © 2023 Howard Hughes Medical Institute, Authored by Carsen Stringer and Marius Pachitariu.

`class cellpose.resnet_torch.`**`CPnet`**`(`**args: Any*, ***kwargs: Any*`)`

CPnet is the Cellpose neural network model used for cell segmentation and image restoration.

Parameters

- **nbase** (`list`) – List of integers representing the number of channels in each layer of the downsample path.
- **nout** (`int`) – Number of output channels.
- **sz** (`int`) – Size of the input image.
- **mkldnn** (`bool, optional`) – Whether to use MKL-DNN acceleration. Defaults to False.
- **conv_3D** (`bool, optional`) – Whether to use 3D convolution. Defaults to False.
- **max_pool** (`bool, optional`) – Whether to use max pooling. Defaults to True.
- **diam_mean** (`float, optional`) – Mean diameter of the cells. Defaults to 30.0.

`nbase`
: List of integers representing the number of channels in each layer of the downsample path.

  Type
  list

`nout`
: Number of output channels.

  Type
  int

`sz`
: Size of the input image.

  Type
  int

`residual_on`
: Whether to use residual connections.

  Type
  bool

`style_on`
: Whether to use style transfer.

  Type
  bool

`concatenation`
: Whether to use concatenation.

  Type
  bool

`conv_3D`
: Whether to use 3D convolution.

  Type
  bool

`mkldnn`
: Whether to use MKL-DNN acceleration.

  Type
  bool

`downsample`
: Downsample blocks of the network.

  Type
  nn.Module

`upsample`
: Upsample blocks of the network.

  Type
  nn.Module

`make_style`
Style module, avgpool's over all spatial positions.

Type
nn.Module

`output`
Output module - batchconv layer.

Type
nn.Module

`diam_mean`
Parameter representing the mean diameter to which the cells are rescaled to during training.

Type
nn.Parameter

`diam_labels`
Parameter representing the mean diameter of the cells in the training set (before rescaling).

Type
nn.Parameter

`property device`
Get the device of the model.

Returns
The device of the model.

Return type
torch.device

`forward`(*data*)
Forward pass of the CPnet model.

Parameters
`data` (*torch.Tensor*) – Input data.

Returns
A tuple containing the output tensor, style tensor, and downsampled tensors.

Return type
tuple

`load_model`(*filename*, *device=None*)
Load the model from a file.

Parameters

- `filename` (*str*) – The path to the file where the model is saved.
- `device` (*torch.device, optional*) – The device to load the model on. Defaults to None.

`save_model`(*filename*)
Save the model to a file.

Parameters
`filename` (*str*) – The path to the file where the model will be saved.

`class cellpose.resnet_torch.batchconvstyle`(**args: Any*, ***kwargs: Any*) `class cellpose.resnet_torch.`**`downsample`**`(`**args: Any*, ***kwargs: Any*`)`

`class cellpose.resnet_torch.`**`make_style`**`(`**args: Any*, ***kwargs: Any*`)`

`class cellpose.resnet_torch.`**`resdown`**`(`**args: Any*, ***kwargs: Any*`)`

`class cellpose.resnet_torch.`**`resup`**`(`**args: Any*, ***kwargs: Any*`)`

`class cellpose.resnet_torch.`**`upsample`**`(`**args: Any*, ***kwargs: Any*`)`

13.14 Core functions

`cellpose.core.`**`assign_device`**`(`*use_torch=True*, *gpu=False*, *device=0*`)`

Assigns the device (CPU or GPU or mps) to be used for computation.

Parameters

- **use_torch** (*bool, optional*) – Whether to use torch for GPU detection. Defaults to True.
- **gpu** (*bool, optional*) – Whether to use GPU for computation. Defaults to False.
- **device** (*int or str, optional*) – The device index or name to be used. Defaults to 0.

Returns

The assigned device. bool: True if GPU is used, False otherwise.

Return type

torch.device

`cellpose.core.`**`check_mkl`**`(`*use_torch=True*`)`

Checks if MKL-DNN is enabled and working.

Parameters

use_torch (*bool, optional*) – Whether to use torch. Defaults to True.

Returns

True if MKL-DNN is enabled, False otherwise.

Return type

bool

`cellpose.core.`**`run_3D`**`(`*net*, *imgs*, *batch_size=8*, *rsz=1.0*, *anisotropy=None*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*, *progress=None*`)`

Run network on image z-stack.

(faster if augment is False)

Parameters

- **imgs** (*np.ndarray*) – The input image stack of size [Lz x Ly x Lx x nchan].
- **batch_size** (*int, optional*) – Number of tiles to run in a batch. Defaults to 8.
- **rsz** (*float, optional*) – Resize coefficient(s) for image. Defaults to 1.0.
- **anisotropy** (*float, optional*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y). Defaults to None.

- `augment` (*bool, optional*) – Tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- `tile` (*bool, optional*) – Tiles image to ensure GPU/CPU memory usage limited (recommended); cannot be turned off for 3D segmentation. Defaults to True.
- `tile_overlap` (*float, optional*) – Fraction of overlap of tiles when computing flows. Defaults to 0.1.
- `bsize` (*int, optional*) – Size of tiles to use in pixels [bsize x bsize]. Defaults to 224.
- `progress` (*QProgressBar, optional*) – pyqt progress bar. Defaults to None.

Returns

output of network, if tiled it is averaged in tile overlaps. Size of [Ly x Lx x 3] or [Lz x Ly x Lx x 3].
y[…,0] is Y flow; y[…,1] is X flow; y[…,2] is cell probability.

style (np.ndarray): 1D array of size 256 summarizing the style of the image, if tiled it is averaged over tiles.

Return type

y (np.ndarray)

`cellpose.core.run_net`(*net*, *imgs*, *batch_size=8*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*)

Run network on image or stack of images.

(faster if augment is False)

Parameters

- `net` (*class*) – cellpose network (model.net)
- `imgs` (*np.ndarray*) – The input image or stack of images of size [Ly x Lx x nchan] or [Lz x Ly x Lx x nchan].
- `batch_size` (*int, optional*) – Number of tiles to run in a batch. Defaults to 8.
- `rsz` (*float, optional*) – Resize coefficient(s) for image. Defaults to 1.0.
- `augment` (*bool, optional*) – Tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- `tile` (*bool, optional*) – Tiles image to ensure GPU/CPU memory usage limited (recommended); cannot be turned off for 3D segmentation. Defaults to True.
- `tile_overlap` (*float, optional*) – Fraction of overlap of tiles when computing flows. Defaults to 0.1.
- `bsize` (*int, optional*) – Size of tiles to use in pixels [bsize x bsize]. Defaults to 224.

Returns

output of network, if tiled it is averaged in tile overlaps. Size of [Ly x Lx x 3] or [Lz x Ly x Lx x 3].
y[…,0] is Y flow; y[…,1] is X flow; y[…,2] is cell probability.

style (np.ndarray): 1D array of size 256 summarizing the style of the image, if tiled it is averaged over tiles.

Return type

y (np.ndarray)

`cellpose.core.`**`use_gpu`**(*gpu_number=0*, *use_torch=True*)

Check if GPU is available for use.

Parameters

- **`gpu_number`** (*`int`*) – The index of the GPU to be used. Default is 0.
- **`use_torch`** (*`bool`*) – Whether to use PyTorch for GPU check. Default is True.

Returns

True if GPU is available, False otherwise.

Return type

bool

Raises

`ValueError` – If use_torch is False, as cellpose only runs with PyTorch now.

13.15 All models functions

Copyright © 2023 Howard Hughes Medical Institute, Authored by Carsen Stringer and Marius Pachitariu.

`class` `cellpose.models.`**`Cellpose`**(*gpu=False*, *model_type='cyto3'*, *nchan=2*, *device=None*, *backbone='default'*)

Main model which combines SizeModel and CellposeModel.

Parameters

- **`gpu`** (*`bool, optional`*) – Whether or not to use GPU, will check if GPU available. Defaults to False.
- **`model_type`** (*`str, optional`*) – Model type. "cyto"=cytoplasm model; "nuclei"=nucleus model; "cyto2"=cytoplasm model with additional user images; "cyto3"=super-generalist model; Defaults to "cyto3".
- **`device`** (*`torch device, optional`*) – Device used for model running / training. Overrides gpu input. Recommended if you want to use a specific GPU (e.g. torch.device("cuda:1")). Defaults to None.

`device`

Device used for model running / training.

Type

torch device

`gpu`

Flag indicating if GPU is used.

Type

bool

`diam_mean`

Mean diameter for cytoplasm model.

Type

float

`cp`

CellposeModel instance.

Type

CellposeModel

`pretrained_size`

Pretrained size model path.

Type

str

`sz`

SizeModel instance.

Type

SizeModel

`eval`(*x*, *batch_size=8*, *channels=[0, 0]*, *channel_axis=None*, *invert=False*, *normalize=True*, *diameter=30.0*, *do_3D=False*, *find_masks=True*, ***kwargs*)

Run cellpose size model and mask model and get masks.

Parameters

- **x** (*list or array*) – List or array of images. Can be list of 2D/3D images, or array of 2D/3D images, or 4D image array.
- **batch_size** (*int, optional*) – Number of 224x224 patches to run simultaneously on the GPU. Can make smaller or bigger depending on GPU memory usage. Defaults to 8.
- **channels** (*list, optional*) – List of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to [0,0].
- **channel_axis** (*int, optional*) – If None, channels dimension is attempted to be automatically determined. Defaults to None.
- **invert** (*bool, optional*) – Invert image pixel intensity before running network (if True, image is also normalized). Defaults to False.
- **normalize** (*bool, optional*) – If True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (see CellposeModel for details). Defaults to True.
- **diameter** (*float, optional*) – If set to None, then diameter is automatically estimated if size model is loaded. Defaults to 30..
- **do_3D** (*bool, optional*) – Set to True to run 3D segmentation on 4D image input. Defaults to False.

Returns

tuple containing

- masks (list of 2D arrays or single 3D array): Labelled image, where 0=no masks; 1,2,…=mask labels.
- **flows (list of lists 2D arrays or list of 3D arrays):**
  - flows[k][0] = XY flow in HSV 0-255
  - flows[k][1] = XY flows at each pixel

  - flows[k][2] = cell probability (if > cellprob_threshold, pixel used for dynamics)
  - flows[k][3] = final pixel locations after Euler integration
- styles (list of 1D arrays of length 256 or single 1D array): Style vector summarizing each image, also used to estimate size of objects in image.
- diams (list of diameters or float): List of diameters or float (if do_3D=True).

`class cellpose.models.CellposeModel`(*gpu=False*, *pretrained_model=False*, *model_type=None*, *diam_mean=30.0*, *device=None*, *nchan=2*, *backbone='default'*)

Class representing a Cellpose model.

`diam_mean`

Mean "diameter" value for the model.

Type
float

`builtin`

Whether the model is a built-in model or not.

Type
bool

`device`

Device used for model running / training.

Type
torch device

`mkldnn`

MKLDNN flag for the model.

Type
None or bool

`nchan`

Number of channels used as input to the network.

Type
int

`nclasses`

Number of classes in the model.

Type
int

`nbase`

List of base values for the model.

Type
list

`net`

Cellpose network.

Type
CPnet

`pretrained_model`

Full path to pretrained cellpose model(s).

Type

str or list of strings

`diam_labels`

Diameter labels of the model.

Type

numpy array

`net_type`

Type of the network.

Type

str

`__init__`(*self*, *gpu=False*, *pretrained_model=False*, *model_type=None*, *diam_mean=30.*, *device=None*, *nchan=2*)

Initialize the CellposeModel.

`eval`(*self*, *x*, *batch_size=8*, *resample=True*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *invert=False*, *rescale=None*, *diameter=None*, *flow_threshold=0.4*, *cellprob_threshold=0.0*, *do_3D=False*, *anisotropy=None*, *stitch_threshold=0.0*, *min_size=15*, *niter=None*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*, *interp=True*, *compute_masks=True*, *progress=None*)

Segment list of images x, or 4D array - Z x nchan x Y x X.

`eval`(*x*, *batch_size=8*, *resample=True*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *invert=False*, *rescale=None*, *diameter=None*, *flow_threshold=0.4*, *cellprob_threshold=0.0*, *do_3D=False*, *anisotropy=None*, *stitch_threshold=0.0*, *min_size=15*, *niter=None*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *bsize=224*, *interp=True*, *compute_masks=True*, *progress=None*)

segment list of images x, or 4D array - Z x nchan x Y x X

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **resample** (*bool, optional*) – run dynamics at original image size (will be slower but create more accurate boundaries). Defaults to True.
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.

- **z_axis** (*int, optional*) – z axis in element of list x, or of np.ndarray x. if None, z dimension is attempted to be automatically determined. Defaults to None.
- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - ”lowhigh”=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - ”sharpen”=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - ”normalize”=True ; run normalization (if False, all following parameters ignored)
  - ”percentile”=None : pass in percentiles to use as list [perc_low, perc_high]
  - ”tile_norm”=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - ”norm3D”=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.
- **invert** (*bool, optional*) – invert image pixel intensity before running network. Defaults to False.
- **rescale** (*float, optional*) – resize factor for each image, if None, set to 1.0; (only used if diameter is None). Defaults to None.
- **diameter** (*float, optional*) – diameter for each image, if diameter is None, set to diam_mean or diam_train if available. Defaults to None.
- **flow_threshold** (*float, optional*) – flow error threshold (all cells with errors below threshold are kept) (not used for 3D). Defaults to 0.4.
- **cellprob_threshold** (*float, optional*) – all pixels with value above threshold kept for masks, decrease to find more and larger masks. Defaults to 0.0.
- **do_3D** (*bool, optional*) – set to True to run 3D segmentation on 3D/4D image input. Defaults to False.
- **anisotropy** (*float, optional*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y). Defaults to None.
- **stitch_threshold** (*float, optional*) – if stitch_threshold>0.0 and not do_3D, masks are stitched in 3D to return volume segmentation. Defaults to 0.0.
- **min_size** (*int, optional*) – all ROIs below this size, in pixels, will be discarded. Defaults to 15.
- **niter** (*int, optional*) – number of iterations for dynamics computation. if None, it is set proportional to the diameter. Defaults to None.
- **augment** (*bool, optional*) – tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **tile_overlap** (*float, optional*) – fraction of overlap of tiles when computing flows. Defaults to 0.1.

- `bsize` (*int, optional*) – block size for tiles, recommended to keep at 224, like in training. Defaults to 224.
- `interp` (*bool, optional*) – interpolate during 2D dynamics (not available in 3D) . Defaults to True.
- `compute_masks` (*bool, optional*) – Whether or not to compute dynamics and return masks. This is set to False when retrieving the styles for the size model. Defaults to True.
- `progress` (*QProgressBar, optional*) – pyqt progress bar. Defaults to None.

Returns

- masks (list, np.ndarray): labelled image(s), where 0=no masks; 1,2,…=mask labels
- flows (list): list of lists: flows[k][0] = XY flow in HSV 0-255; flows[k][1] = XY(Z) flows at each pixel; flows[k][2] = cell probability (if > cellprob_threshold, pixel used for dynamics); flows[k][3] = final pixel locations after Euler integration
- styles (list, np.ndarray): style vector summarizing each image of size 256.

Return type
A tuple containing

`class cellpose.models.SizeModel`(*cp_model*, *device=None*, *pretrained_size=None*, ***kwargs*)

Linear regression model for determining the size of objects in image used to rescale before input to cp_model. Uses styles from cp_model.

`pretrained_size`

Path to pretrained size model.

Type
str

`cp`

Model from which to get styles.

Type
UnetModel or *CellposeModel*

`device`

Device used for model running / training (torch.device("cuda") or torch.device("cpu")), overrides gpu input, recommended if you want to use a specific GPU (e.g. torch.device("cuda:1")).

Type
torch device

`diam_mean`

Mean diameter of objects.

Type
float

`eval(self, x, channels=None, channel_axis=None, normalize=True, invert=False,`

augment=False, tile=True, batch_size=8, progress=None, interp=True):

Use images x to produce style or use style input to predict size of objects in image.

Raises
`ValueError` – If no pretrained cellpose model is specified, cannot compute size.

eval(*x*, *channels=None*, *channel_axis=None*, *normalize=True*, *invert=False*, *augment=False*, *tile=True*, *batch_size=8*, *progress=None*)

Use images x to produce style or use style input to predict size of objects in image.

Object size estimation is done in two steps: 1. Use a linear regression model to predict size from style in image. 2. Resize image to predicted size and run CellposeModel to get output masks.

Take the median object size of the predicted masks as the final predicted size.

Parameters

- **x** (*list, np.ndarry*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **channels** (*list, optional*) – list of channels, either of length 2 or of length number of images by 2. First element of list is the channel to segment (0=grayscale, 1=red, 2=green, 3=blue). Second element of list is the optional nuclear channel (0=none, 1=red, 2=green, 3=blue). For instance, to segment grayscale images, input [0,0]. To segment images with cells in green and nuclei in blue, input [2,3]. To segment one grayscale image and one image with cells in green and nuclei in blue, input [[0,0], [2,3]]. Defaults to None.
- **channel_axis** (*int, optional*) – channel axis in element of list x, or of np.ndarray x. if None, channels dimension is attempted to be automatically determined. Defaults to None.
- **normalize** (*bool, optional*) – if True, normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel; can also pass dictionary of parameters (all keys are optional, default values shown):
  - "lowhigh"=None : pass in normalization values for 0.0 and 1.0 as list [low, high] (if not None, all following parameters ignored)
  - "sharpen"=0 ; sharpen image with high pass filter, recommended to be 1/4-1/8 diameter of cells in pixels
  - "normalize"=True ; run normalization (if False, all following parameters ignored)
  - "percentile"=None : pass in percentiles to use as list [perc_low, perc_high]
  - "tile_norm"=0 ; compute normalization in tiles across image to brighten dark areas, to turn on set to window size in pixels (e.g. 100)
  - "norm3D"=False ; compute normalization across entire z-stack rather than plane-by-plane in stitching mode.

  Defaults to True.
- **invert** (*bool, optional*) – Invert image pixel intensity before running network (if True, image is also normalized). Defaults to False.
- **augment** (*bool, optional*) – tiles image with overlapping tiles and flips overlapped regions to augment. Defaults to False.
- **tile** (*bool, optional*) – tiles image to ensure GPU/CPU memory usage limited (recommended). Defaults to True.
- **batch_size** (*int, optional*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage). Defaults to 8.
- **progress** (*QProgressBar, optional*) – pyqt progress bar. Defaults to None.

Returns

- diam (np.ndarray): Final estimated diameters from images x or styles style after running both steps.
- diam_style (np.ndarray): Estimated diameters from style alone.

Return type

A tuple containing

CHAPTER

FOURTEEN

CELLPOSE CLI

See example usage at *CLI examples*. A description of the most important settings can be found on the *Settings* page.

14.1 Command Line Usage

Cellpose Command Line Parameters

```
usage: cellpose [-h] [--version] [--verbose] [--Zstack] [--use_gpu]
                [--gpu_device GPU_DEVICE] [--check_mkl] [--dir DIR]
                [--image_path IMAGE_PATH] [--look_one_level_down]
                [--img_filter IMG_FILTER] [--channel_axis CHANNEL_AXIS]
                [--z_axis Z_AXIS] [--chan CHAN] [--chan2 CHAN2] [--invert]
                [--all_channels] [--pretrained_model PRETRAINED_MODEL]
                [--restore_type RESTORE_TYPE] [--chan2_restore]
                [--add_model ADD_MODEL] [--transformer] [--no_resample]
                [--no_interp] [--no_norm] [--do_3D] [--diameter DIAMETER]
                [--stitch_threshold STITCH_THRESHOLD] [--min_size MIN_SIZE]
                [--flow_threshold FLOW_THRESHOLD]
                [--cellprob_threshold CELLPROB_THRESHOLD] [--niter NITER]
                [--anisotropy ANISOTROPY] [--exclude_on_edges] [--augment]
                [--save_png] [--save_tif] [--no_npy] [--savedir SAVEDIR]
                [--dir_above] [--in_folders] [--save_flows] [--save_outlines]
                [--save_rois] [--save_txt] [--save_mpl] [--train]
                [--train_size] [--test_dir TEST_DIR] [--file_list FILE_LIST]
                [--mask_filter MASK_FILTER] [--diam_mean DIAM_MEAN]
                [--learning_rate LEARNING_RATE] [--weight_decay WEIGHT_DECAY]
                [--n_epochs N_EPOCHS] [--batch_size BATCH_SIZE]
                [--nimg_per_epoch NIMG_PER_EPOCH]
                [--nimg_test_per_epoch NIMG_TEST_PER_EPOCH]
                [--min_train_masks MIN_TRAIN_MASKS] [--SGD SGD]
                [--save_every SAVE_EVERY] [--model_name_out MODEL_NAME_OUT]
```

14.1.1 Named Arguments

--version	show cellpose version info Default: False
--verbose	show information about running and settings and save to log Default: False
--Zstack	run GUI in 3D mode Default: False

14.1.2 Hardware Arguments

--use_gpu	use gpu if torch with cuda installed Default: False
--gpu_device	which gpu device to use, use an integer for torch, or mps for M1 Default: "0"
--check_mkl	check if mkl working Default: False

14.1.3 Input Image Arguments

--dir	folder containing data to run or train on. Default: []
--image_path	if given and –dir not given, run on single image instead of folder (cannot train with this option) Default: []
--look_one_level_down	run processing on all subdirectories of current folder Default: False
--img_filter	end string for images to run on Default: []
--channel_axis	axis of image which corresponds to image channels
--z_axis	axis of image which corresponds to Z dimension
--chan	channel to segment; 0: GRAY, 1: RED, 2: GREEN, 3: BLUE. Default: 0 Default: 0
--chan2	nuclear channel (if cyto, optional); 0: NONE, 1: RED, 2: GREEN, 3: BLUE. Default: 0 Default: 0
--invert	invert grayscale channel Default: False

--all_channels	use all channels in image if using own model and images with special channels Default: False

14.1.4 Model Arguments

--pretrained_model	model to use for running or starting training Default: "cyto"
--restore_type	model to use for image restoration
--chan2_restore	use nuclei restore model for second channel Default: False
--add_model	model path to copy model to hidden .cellpose folder for using in GUI/CLI
--transformer	use transformer backbone (pretrained_model from Cellpose3 is transformer_cp3) Default: False

14.1.5 Algorithm Arguments

--no_resample	disable dynamics on full image (makes algorithm faster for images with large diameters) Default: False
--no_interp	do not interpolate when running dynamics (was default) Default: False
--no_norm	do not normalize images (normalize=False) Default: False
--do_3D	process images as 3D stacks of images (nplanes x nchan x Ly x Lx Default: False
--diameter	cell diameter, if 0 will use the diameter of the training labels used in the model, or with built-in model will estimate diameter for each image Default: 30.0
--stitch_threshold	compute masks in 2D then stitch together masks with IoU>0.9 across planes Default: 0.0
--min_size	minimum number of pixels per mask, can turn off with -1 Default: 15
--flow_threshold	flow error threshold, 0 turns off this optional QC step. Default: 0.4 Default: 0.4
--cellprob_threshold	cellprob threshold, default is 0, decrease to find more and larger masks Default: 0
--niter	niter, number of iterations for dynamics for mask creation, default of 0 means it is proportional to diameter, set to a larger number like 2000 for very long ROIs Default: 0

--anisotropy	anisotropy of volume in 3D Default: 1.0
--exclude_on_edges	discard masks which touch edges of image Default: False
--augment	tiles image with overlapping tiles and flips overlapped regions to augment Default: False

14.1.6 Output Arguments

--save_png	save masks as png and outlines as text file for ImageJ Default: False
--save_tif	save masks as tif and outlines as text file for ImageJ Default: False
--no_npy	suppress saving of npy Default: False
--savedir	folder to which segmentation results will be saved (defaults to input image directory)
--dir_above	save output folders adjacent to image folder instead of inside it (off by default) Default: False
--in_folders	flag to save output in folders (off by default) Default: False
--save_flows	whether or not to save RGB images of flows when masks are saved (disabled by default) Default: False
--save_outlines	whether or not to save RGB outline images when masks are saved (disabled by default) Default: False
--save_rois	whether or not to save ImageJ compatible ROI archive (disabled by default) Default: False
--save_txt	flag to enable txt outlines for ImageJ (disabled by default) Default: False
--save_mpl	save a figure of image/mask/flows using matplotlib (disabled by default). This is slow, especially with large images. Default: False

14.1.7 Training Arguments

--train	train network using images in dir Default: False
--train_size	train size network at end of training Default: False
--test_dir	folder containing test data (optional) Default: []
--file_list	path to list of files for training and testing and probabilities for each image (optional) Default: []
--mask_filter	end string for masks to run on. use '_seg.npy' for manual annotations from the GUI. Default: "_masks" Default: "_masks"
--diam_mean	mean diameter to resize cells to during training – if starting from pretrained models it cannot be changed from 30.0 Default: 30.0
--learning_rate	learning rate. Default: 0.2 Default: 0.2
--weight_decay	weight decay. Default: 1e-05 Default: 1e-05
--n_epochs	number of epochs. Default: 500 Default: 500
--batch_size	batch size. Default: 8 Default: 8
--nimg_per_epoch	number of train images per epoch. Default is to use all train images.
--nimg_test_per_epoch	number of test images per epoch. Default is to use all test images.
--min_train_masks	minimum number of masks a training image must have to be used. Default: 5 Default: 5
--SGD	use SGD Default: 1
--save_every	number of epochs to skip between saves. Default: 100 Default: 100
--model_name_out	Name of model to save as, defaults to name describing model architecture. Model is saved in the folder specified by –dir in models subfolder.

PYTHON MODULE INDEX

C

cellpose.core, 80
cellpose.dynamics, 54
cellpose.io, 68
cellpose.metrics, 53
cellpose.models, 82
cellpose.plot, 66
cellpose.resnet_torch, 77
cellpose.train, 50
cellpose.transforms, 60
cellpose.utils, 72

INDEX

Symbols

`__init__()` (*cellpose.models.CellposeModel method*), 42, 85

A

`add_model()` (*in module cellpose.io*), 68
`aggregated_jaccard_index()` (*in module cellpose.metrics*), 53
`assign_device()` (*in module cellpose.core*), 80
`average_precision()` (*in module cellpose.metrics*), 53
`average_tiles()` (*in module cellpose.transforms*), 60

B

`batchconvstyle` (*class in cellpose.resnet_torch*), 79
`boundary_scores()` (*in module cellpose.metrics*), 53
`builtin` (*cellpose.models.CellposeModel attribute*), 41, 84

C

`Cellpose` (*class in cellpose.models*), 39, 82
`cellpose.core`
  `module`, 80
`cellpose.dynamics`
  `module`, 54
`cellpose.io`
  `module`, 68
`cellpose.metrics`
  `module`, 53
`cellpose.models`
  `module`, 82
`cellpose.plot`
  `module`, 66
`cellpose.resnet_torch`
  `module`, 77
`cellpose.train`
  `module`, 50
`cellpose.transforms`
  `module`, 60
`cellpose.utils`
  `module`, 72
`CellposeDenoiseModel` (*class in cellpose.denoise*), 44
`CellposeModel` (*class in cellpose.models*), 41, 84
`check_mkl()` (*in module cellpose.core*), 80
`circleMask()` (*in module cellpose.utils*), 72
`compute_masks()` (*in module cellpose.dynamics*), 54
`concatenation` (*cellpose.resnet_torch.CPnet attribute*), 78
`conv_3D` (*cellpose.resnet_torch.CPnet attribute*), 78
`convert_image()` (*in module cellpose.transforms*), 60
`cp` (*cellpose.models.Cellpose attribute*), 39, 82
`cp` (*cellpose.models.SizeModel attribute*), 48, 87
`CPnet` (*class in cellpose.resnet_torch*), 77

D

`DenoiseModel` (*class in cellpose.denoise*), 46
`device` (*cellpose.models.Cellpose attribute*), 39, 82
`device` (*cellpose.models.CellposeModel attribute*), 41, 84
`device` (*cellpose.models.SizeModel attribute*), 48, 87
`device` (*cellpose.resnet_torch.CPnet property*), 79
`diam_labels` (*cellpose.models.CellposeModel attribute*), 42, 85
`diam_labels` (*cellpose.resnet_torch.CPnet attribute*), 79
`diam_mean` (*cellpose.denoise.DenoiseModel attribute*), 46
`diam_mean` (*cellpose.models.Cellpose attribute*), 39, 82
`diam_mean` (*cellpose.models.CellposeModel attribute*), 41, 84
`diam_mean` (*cellpose.models.SizeModel attribute*), 49, 87
`diam_mean` (*cellpose.resnet_torch.CPnet attribute*), 79
`diameters()` (*in module cellpose.utils*), 72
`dilate_masks()` (*in module cellpose.utils*), 73
`disk()` (*in module cellpose.plot*), 66
`distance_to_boundary()` (*in module cellpose.utils*), 73
`download_url_to_file()` (*in module cellpose.utils*), 73
`downsample` (*cellpose.resnet_torch.CPnet attribute*), 78
`downsample` (*class in cellpose.resnet_torch*), 79
`dx_to_circ()` (*in module cellpose.plot*), 66

E

`eval()` (*cellpose.denoise.CellposeDenoiseModel method*), 44

eval() (*cellpose.denoise.DenoiseModel method*), 47
eval() (*cellpose.models.Cellpose method*), 40, 83
eval() (*cellpose.models.CellposeModel method*), 42, 85
eval() (*cellpose.models.SizeModel method*), 49, 87

F

fill_holes_and_remove_small_masks() (*in module cellpose.utils*), 73
flow_error() (*in module cellpose.metrics*), 54
flush() (*cellpose.utils.TqdmToLogger method*), 72
follow_flows() (*in module cellpose.dynamics*), 55
forward() (*cellpose.resnet_torch.CPnet method*), 79

G

gaussian_kernel() (*in module cellpose.transforms*), 61
get_centers() (*in module cellpose.dynamics*), 55
get_image_files() (*in module cellpose.io*), 68
get_label_files() (*in module cellpose.io*), 69
get_mask_compactness() (*in module cellpose.utils*), 74
get_mask_perimeters() (*in module cellpose.utils*), 74
get_mask_stats() (*in module cellpose.utils*), 74
get_masks() (*in module cellpose.dynamics*), 56
get_masks_unet() (*in module cellpose.utils*), 74
get_outline_multi() (*in module cellpose.utils*), 74
get_perimeter() (*in module cellpose.utils*), 75
gpu (*cellpose.models.Cellpose attribute*), 39, 82

I

image_to_rgb() (*in module cellpose.plot*), 67
imread() (*in module cellpose.io*), 69
imsave() (*in module cellpose.io*), 69
interesting_patch() (*in module cellpose.plot*), 67

L

labels_to_flows() (*in module cellpose.dynamics*), 56
load_images_labels() (*in module cellpose.io*), 69
load_model() (*cellpose.resnet_torch.CPnet method*), 79
load_train_test_data() (*in module cellpose.io*), 70

M

make_style (*cellpose.resnet_torch.CPnet attribute*), 78
make_style (*class in cellpose.resnet_torch*), 80
make_tiles() (*in module cellpose.transforms*), 61
map_coordinates() (*in module cellpose.dynamics*), 56
mask_ious() (*in module cellpose.metrics*), 54
mask_overlay() (*in module cellpose.plot*), 67
mask_rgb() (*in module cellpose.plot*), 67
masks_flows_to_seg() (*in module cellpose.io*), 70
masks_to_edges() (*in module cellpose.utils*), 75
masks_to_flows() (*in module cellpose.dynamics*), 57
masks_to_flows_cpu() (*in module cellpose.dynamics*), 57
masks_to_flows_gpu() (*in module cellpose.dynamics*), 57
masks_to_flows_gpu_3d() (*in module cellpose.dynamics*), 57
masks_to_outlines() (*in module cellpose.utils*), 75
mkldnn (*cellpose.models.CellposeModel attribute*), 41, 84
mkldnn (*cellpose.resnet_torch.CPnet attribute*), 78
module
    cellpose.core, 80
    cellpose.dynamics, 54
    cellpose.io, 68
    cellpose.metrics, 53
    cellpose.models, 82
    cellpose.plot, 66
    cellpose.resnet_torch, 77
    cellpose.train, 50
    cellpose.transforms, 60
    cellpose.utils, 72
move_axis() (*in module cellpose.transforms*), 61
move_min_dim() (*in module cellpose.transforms*), 61

N

nbase (*cellpose.models.CellposeModel attribute*), 41, 84
nbase (*cellpose.resnet_torch.CPnet attribute*), 77
nchan (*cellpose.denoise.DenoiseModel attribute*), 46
nchan (*cellpose.models.CellposeModel attribute*), 41, 84
nclasses (*cellpose.models.CellposeModel attribute*), 41, 84
net (*cellpose.denoise.DenoiseModel attribute*), 46
net (*cellpose.models.CellposeModel attribute*), 41, 84
net_chan2 (*cellpose.denoise.DenoiseModel attribute*), 47
net_type (*cellpose.denoise.DenoiseModel attribute*), 47
net_type (*cellpose.models.CellposeModel attribute*), 42, 85
normalize99() (*in module cellpose.transforms*), 62
normalize99_tile() (*in module cellpose.transforms*), 62
normalize_img() (*in module cellpose.transforms*), 62
nout (*cellpose.resnet_torch.CPnet attribute*), 78

O

outline_view() (*in module cellpose.plot*), 67
outlines_list() (*in module cellpose.utils*), 75
outlines_list_multi() (*in module cellpose.utils*), 76
outlines_list_single() (*in module cellpose.utils*), 76
output (*cellpose.resnet_torch.CPnet attribute*), 79

P

pad_image_ND() (*in module cellpose.transforms*), 63

pretrained_model (*cellpose.denoise.DenoiseModel attribute*), 47
pretrained_model (*cellpose.models.CellposeModel attribute*), 41, 84
pretrained_size (*cellpose.models.Cellpose attribute*), 39, 82
pretrained_size (*cellpose.models.SizeModel attribute*), 48, 87

R

radius_distribution() (*in module cellpose.utils*), 76
random_rotate_and_resize() (*in module cellpose.transforms*), 63
remove_bad_flow_masks() (*in module cellpose.dynamics*), 58
remove_edge_masks() (*in module cellpose.utils*), 76
remove_model() (*in module cellpose.io*), 71
resdown (*class in cellpose.resnet_torch*), 80
reshape() (*in module cellpose.transforms*), 64
residual_on (*cellpose.resnet_torch.CPnet attribute*), 78
resize_and_compute_masks() (*in module cellpose.dynamics*), 58
resize_image() (*in module cellpose.transforms*), 64
resup (*class in cellpose.resnet_torch*), 80
run_3D() (*in module cellpose.core*), 80
run_net() (*in module cellpose.core*), 81

S

save_masks() (*in module cellpose.io*), 71
save_model() (*cellpose.resnet_torch.CPnet method*), 79
save_rois() (*in module cellpose.io*), 72
save_to_png() (*in module cellpose.io*), 72
show_segmentation() (*in module cellpose.plot*), 68
size_distribution() (*in module cellpose.utils*), 77
SizeModel (*class in cellpose.models*), 48, 87
smooth_sharpen_img() (*in module cellpose.transforms*), 65
steps2D() (*in module cellpose.dynamics*), 59
steps2D_interp() (*in module cellpose.dynamics*), 59
steps3D() (*in module cellpose.dynamics*), 59
stitch3D() (*in module cellpose.utils*), 77
style_on (*cellpose.resnet_torch.CPnet attribute*), 78
sz (*cellpose.models.Cellpose attribute*), 40, 83
sz (*cellpose.resnet_torch.CPnet attribute*), 78

T

TqdmToLogger (*class in cellpose.utils*), 72
train_seg() (*in module cellpose.train*), 50
train_size() (*in module cellpose.train*), 52

U

unaugment_tiles() (*in module cellpose.transforms*), 65
update_axis() (*in module cellpose.transforms*), 66
upsample (*cellpose.resnet_torch.CPnet attribute*), 78
upsample (*class in cellpose.resnet_torch*), 80
use_gpu() (*in module cellpose.core*), 81

W

write() (*cellpose.utils.TqdmToLogger method*), 72