cellpose

Release 0.7.2

Carsen Stringer & Marius Pachitariu

Apr 05, 2022

BASICS:

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:

- twitter thread
- Marius's talk on cellpose
- paper on biorxiv (see figure 1 below)

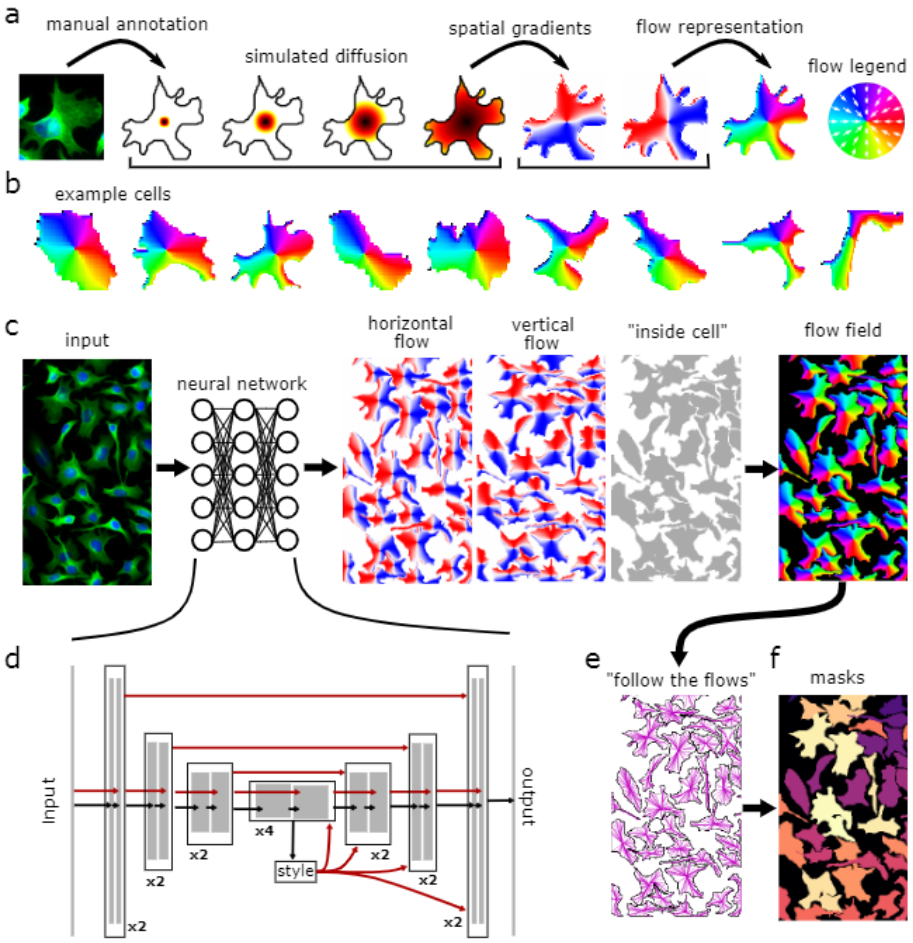


Figure 1: Model architecture. a, Procedure for transforming manually annotated masks into a vector flow representation that can be predicted by a neural network. A simulated diffusion process started at the center of the mask is used to derive spatial gradients that point towards the center of the cell, 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 Common issues

If you receive the error: `Illegal instruction (core dumped)`, then likely mxnet does not recognize your MKL version. Please uninstall and reinstall mxnet without mkl:

```
pip uninstall mxnet-mkl
pip uninstall mxnet
pip install mxnet==1.4.0
```

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 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 having issues with the graphical interface, make sure you have **python 3.7** and not python 3.8 installed.

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.

1.3 Dependencies

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

- mxnet_mkl
- pyqtgraph
- PyQt5
- numpy (>=1.16.0)
- numba
- scipy
- scikit-image
- natsort
- matplotlib

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 [website](http://www.cellpose.org). 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.

2.2 Using the GUI

The GUI serves two main functions:

1. Running the segmentation algorithm.
2. Manually labelling 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

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. 3D labelling will fill in unlabelled z-planes so that you do not have to as densely label.

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

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+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	load numpy stack (NOT WORKING ATM)
A/D or LEFT/RIGHT	cycle through images in current directory
W/S or UP/DOWN	change color (RGB/gray/red/green/blue)
PAGE-UP / PAGE-DOWN	change to flows and cell prob views (if segmentation computed)
, / .	increase / decrease brush size for drawing masks
X	turn masks ON or OFF
Z	toggle outlines ON or OFF
C	cycle through labels for image type (saved to `_seg.npy`)

2.3 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

2.4 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.

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 skimage.io
data = skimage.io.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:

```
from cellpose import models
import skimage.io

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

files = ['img0.tif', 'img1.tif']
imgs = [skimage.io.imread(f) for f in files]
masks, flows, styles, diams = model.eval(imgs, diameter=None, channels=[0,0],
                                         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

4.1.1 Cytoplasm model (*'cyto'*)

The cytoplasm model in cellpose is 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.

4.1.2 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.

4.1.3 Cytoplasm 2.0 model (*'cyto2'*)

The cytoplasm 2.0 model in cellpose is trained on two-channel images, where the first channel is the channel to segment, and the second channel is an optional nuclear channel, as the cytoplasm model.

In addition to the training data in our dataset, it was trained with user-submitted images.

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 masks. Take the final estimated size as the median diameter of the predicted masks.

For automated estimation set `diameter = None`. 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 masks when the cells are large but will be slower in case; `resample=False` will find more masks when the cells are small but will be slower in this case. By default in v0.5 `resample=False`, but in previous releases the default was `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 (aka model fit threshold in GUI)

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 masks, we recompute the flow gradients for these putative predicted masks, 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 masks as you'd expect. Similarly, decrease this threshold if cellpose is returning too many ill-shaped masks.

4.5 Mask 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 `mask_threshold` are used to run dynamics and determine masks. The default is `mask_threshold=0.0`. Decrease this threshold if cellpose is not returning as many masks as you'd expect. Similarly, increase this threshold if cellpose is returning too masks particularly from dim areas.

4.6 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 masks 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

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 the inputs to a sigmoid centered at zero (1 / (1 + e^-x)), so they vary from around -6 to +6.

5.1 _seg.npy output

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

- *filename* : filename of image
- *img* : image with chosen channels (nchan x Ly x Lx) (if not multiplane)
- *masks* : masks (0 = NO masks; 1,2,... = mask labels)
- *colors* : colors for masks
- *outlines* : outlines of masks (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 0.0 to 1.0, 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 (if run on command line)
- *zdraw* : for each mask, which planes were manually labelled (planes in between manually drawn have interpolated masks)

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

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

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

# plot image with outlines overlaid in red
outlines = plot.outlines_list(dat['masks'])
plt.imshow(dat['img'])
```

(continues on next page)

(continued from previous page)

```
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, diams, file_name, channels)
```

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

5.2 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_to_png(images, masks, flows, image_names)
```

5.3 ROI manager compatible output for ImageJ

You can save the outlines of masks 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_png`.

Or use the function below if running in a notebook

```
from cellpose import io, plot

# 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.

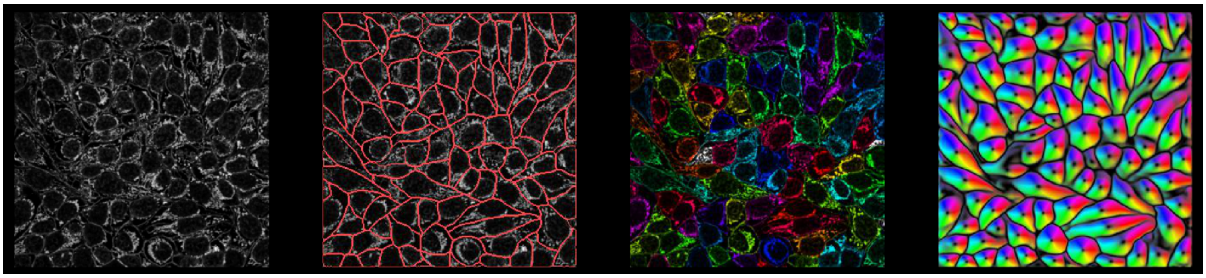
5.4 Plotting functions

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

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 `--diameter` flag, or set it to 0 to disable rescaling. We trained the *cyto* model with a diameter of 30 pixels and the *nuclei* model with a diameter of 17 pixels.

When you rescale everything to 30. pixel diameter, if you have images with varying diameters you may also want to learn a *SizeModel* that predicts the diameter from the styles that the network outputs. Add the flag `--train_size` and this model will be trained and saved as an `*.npy` file.

The same channel settings apply for training models (see all Command line options).

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
```

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` must be an absolute path.

Training-specific options

```
--test_dir TEST_DIR       folder containing test data (optional)
--n_epochs N_EPOCHS       number of epochs (default: 500)
```

To train on cytoplasmic images (green cyto and red nuclei) starting with a pretrained model from cellpose (cyto or nuclei):

```
python -m cellpose --train --dir ~/images_cyto/train/ --test_dir ~/images_cyto/test/ --
↪pretrained_model cyto --chan 2 --chan2 1
```

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
```

CHAPTER

SEVEN

IN A NOTEBOOK

See settings for more information on run settings.

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

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

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

imgs = [skimage.io.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)
```

See full notebook at run_cellpose.ipynb.

CHAPTER

EIGHT

COMMAND LINE

8.1 Input settings

- **dir: (string)** directory of images
- **img_filter: (string)** (optional) ending of filenames (excluding extension) for processing

8.2 Run settings

These are the same settings, but set up for the command line, e.g. *channels = [chan, chan2]*.

- **chan: (int)** 0 = grayscale; 1 = red; 2 = green; 3 = blue
- **chan2: (int)** (optional); 0 = None (will be set to zero); 1 = red; 2 = green; 3 = blue
- **pretrained_model: (string)** cyto = cellpose cytoplasm model; nuclei = cellpose nucleus model; can also specify absolute path to model file
- **diameter: (float)** average diameter of objects in image, if 0 cellpose will estimate for each image, default is 30
- **use_gpu: (bool)** run network on GPU
- **save_png: FLAG** save masks as png and outlines as text file for ImageJ
- **save_tif: FLAG** save masks as tif and outlines as text file for ImageJ
- **fast_mode: FLAG** make code run faster by turning off augmentations and 4 network averaging
- **all_channels: FLAG** run cellpose on all image channels (use for custom models ONLY)
- **no_npy: FLAG** turn off saving of _seg.npy file
- **batch_size: (int, optional 8)** batch size to run tiles of size 224 x 224

8.3 Command line examples

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.

8.4 Options

You can run the help string and see all the options:

::

usage: __main__.py [-h] [–use_gpu] [–check_mkl] [–mkldnn] [–dir DIR] [–look_one_level_down] [–mxnet]

[–img_filter IMG_FILTER] [–channel_axis CHANNEL_AXIS] [–z_axis Z_AXIS] [–chan CHAN] [–chan2 CHAN2] [–invert] [–all_channels] [–pretrained_model PRETRAINED_MODEL] [–unet UNET] [–nclasses NCLASSES] [–omni] [–cluster] [–fast_mode] [–resample] [–no_interp] [–do_3D] [–diameter DIAMETER] [–stitch_threshold STITCH_THRESHOLD] [–flow_threshold FLOW_THRESHOLD] [–mask_threshold MASK_THRESHOLD] [–anisotropy ANISOTROPY] [–diam_threshold DIAM_THRESHOLD] [–exclude_on_edges] [–save_png] [–save_tif] [–no_npy] [–savedir SAVEDIR] [–dir_above] [–in_folders] [–save_flows] [–save_outlines] [–save_ncolor] [–save_txt] [–train] [–train_size] [–mask_filter MASK_FILTER] [–test_dir TEST_DIR] [–learning_rate LEARNING_RATE] [–n_epochs N_EPOCHS] [–batch_size BATCH_SIZE] [–residual_on RESIDUAL_ON] [–style_on STYLE_ON] [–concatenation CONCATENATION] [–save_every SAVE_EVERY] [–save_each] [–verbose] [–testing]

cellpose parameters

optional arguments: -h, –help show this help message and exit –pretrained_model PRETRAINED_MODEL

model to use

--unet UNET run standard unet instead of cellpose flow output

--omni Omnipose algorithm (disabled by default)

--cluster DBSCAN clustering. Reduces oversegmentation of thin features (disabled by default).

--fast_mode make code run faster by turning off 4 network averaging

--resample run dynamics on full image (slower for images with large diameters)

--no_interp do not interpolate when running dynamics (was default)

--do_3D process images as 3D stacks of images (nplanes x nchan x Ly x Lx

--diameter DIAMETER cell diameter, if 0 cellpose will estimate for each image

--stitch_threshold STITCH_THRESHOLD compute masks in 2D then stitch together masks with IoU>0.9 across planes

--anisotropy ANISOTROPY anisotropy of volume in 3D

--diam_threshold DIAM_THRESHOLD cell diameter threshold for upscaling before mask rescontruction, default 12.

--exclude_on_edges discard masks which touch edges of image

--verbose flag to output extra information (e.g. diameter metrics) for debugging and fine-tuning parameters

--testing flag to suppress CLI user confirmation for saving output; for test scripts

hardware arguments: –use_gpu use gpu if torch or mxnet with cuda installed –check_mkl check if mkl working –mkldnn for mxnet, force MXNET_SUBGRAPH_BACKEND = "MKLDNN"

input image arguments: –dir DIR folder containing data to run or train on. –look_one_level_down run processing on all subdirectories of current folder –mxnet use mxnet –img_filter IMG_FILTER

end string for images to run on

--channel_axis CHANNEL_AXIS axis of image which corresponds to image channels

--z_axis Z_AXIS axis of image which corresponds to Z dimension

--chan CHAN channel to segment; 0: GRAY, 1: RED, 2: GREEN, 3: BLUE. Default: 0

--chan2 CHAN2 nuclear channel (if cyto, optional); 0: NONE, 1: RED, 2: GREEN, 3: BLUE. Default: 0

--invert invert grayscale channel

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

model arguments: –nclasses NCLASSES if running unet, choose 2 or 3; if training omni, choose 4; standard Cellpose uses 3

algorithm arguments: –flow_threshold FLOW_THRESHOLD

flow error threshold, 0 turns off this optional QC step. Default: 0.4

--mask_threshold MASK_THRESHOLD mask threshold, default is 0, decrease to find more and larger masks

output arguments: –save_png save masks as png and outlines as text file for ImageJ –save_tif save masks as tif and outlines as text file for ImageJ –no_npy suppress saving of npy –savedir 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) –in_folders flag to save output in folders (off by default) –save_flows whether or not to save RGB images of flows when masks are saved (disabled by default) –save_outlines whether or not to save RGB outline images when masks are saved (disabled by default) –save_ncolor whether or not to save minimal "n-color" masks (disabled by default –save_txt flag to enable txt outlines for ImageJ (disabled by default)

training arguments: –train train network using images in dir –train_size train size network at end of training –mask_filter MASK_FILTER

end string for masks to run on. Default: _masks

--test_dir TEST_DIR folder containing test data (optional)

--learning_rate LEARNING_RATE learning rate. Default: 0.2

--n_epochs N_EPOCHS number of epochs. Default: 500

--batch_size BATCH_SIZE batch size. Default: 8

--residual_on RESIDUAL_ON use residual connections

--style_on STYLE_ON use style vector

--concatenation CONCATENATION concatenate downsampled layers with upsampled layers (off by default which means they are added)

--save_every SAVE_EVERY number of epochs to skip between saves. Default: 100

--save_each save the model under a different filename per –save_every epoch for later comparsion

CHAPTER

NINE

CELLPOSE API GUIDE

9.1 Cellpose class

class cellpose.models.**Cellpose**(*gpu=False*, *model_type='cyto'*, *net_avg=True*, *device=None*, *torch=True*, *model_dir=None*, *omni=False*)

main model which combines SizeModel and CellposeModel

Parameters

- **gpu** (*bool (optional, default False)*) – whether or not to use GPU, will check if GPU available
- **model_type** (*str (optional, default 'cyto')*) – 'cyto'=cytoplasm model; 'nuclei'=nucleus model
- **net_avg** (*bool (optional, default True)*) – loads the 4 built-in networks and averages them if True, loads one network if False
- **device** (*gpu device (optional, default None)*) – where model is saved (e.g. mx.gpu() or mx.cpu()), overrides gpu input, recommended if you want to use a specific GPU (e.g. mx.gpu(4) or torch.cuda.device(4))
- **torch** (*bool (optional, default True)*) – run model using torch if available

eval(*x*, *batch_size=8*, *channels=None*, *channel_axis=None*, *z_axis=None*, *invert=False*, *normalize=True*, *diameter=30.0*, *do_3D=False*, *anisotropy=None*, *net_avg=True*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *resample=True*, *interp=True*, *cluster=False*, *flow_threshold=0.4*, *mask_threshold=0.0*, *cellprob_threshold=None*, *dist_threshold=None*, *diam_threshold=12.0*, *min_size=15*, *stitch_threshold=0.0*, *rescale=None*, *progress=None*, *omni=False*, *verbose=False*, *transparency=False*, *model_loaded=False*)

run cellpose and get masks

Parameters

- **x** (*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, default 8)*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage)
- **channels** (*list (optional, default None)*) – 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]].

- **channel_axis** (*int (optional, default None)*) – if None, channels dimension is attempted to be automatically determined
- **z_axis** (*int (optional, default None)*) – if None, z dimension is attempted to be automatically determined
- **invert** (*bool (optional, default False)*) – invert image pixel intensity before running network (if True, image is also normalized)
- **normalize** (*bool (optional, default True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel
- **diameter** (*float (optional, default 30.)*) – if set to None, then diameter is automatically estimated if size model is loaded
- **do_3D** (*bool (optional, default False)*) – set to True to run 3D segmentation on 4D image input
- **anisotropy** (*float (optional, default None)*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y)
- **net_avg** (*bool (optional, default True)*) – runs the 4 built-in networks and averages them if True, runs one network if False
- **augment** (*bool (optional, default False)*) – tiles image with overlapping tiles and flips overlapped regions to augment
- **tile** (*bool (optional, default True)*) – tiles image to ensure GPU/CPU memory usage limited (recommended)
- **tile_overlap** (*float (optional, default 0.1)*) – fraction of overlap of tiles when computing flows
- **resample** (*bool (optional, default True)*) – run dynamics at original image size (will be slower but create more accurate boundaries)
- **interp** (*bool (optional, default True)*) – interpolate during 2D dynamics (not available in 3D) (in previous versions it was False)
- **flow_threshold** (*float (optional, default 0.4)*) – flow error threshold (all cells with errors below threshold are kept) (not used for 3D)
- **mask_threshold** (*float (optional, default 0.0)*) – all pixels with value above threshold kept for masks, decrease to find more and larger masks
- **dist_threshold** (*float (optional, default None) DEPRECATED*) – use mask_threshold instead
- **cellprob_threshold** (*float (optional, default None) DEPRECATED*) – use mask_threshold instead
- **min_size** (*int (optional, default 15)*) – minimum number of pixels per mask, can turn off with -1
- **stitch_threshold** (*float (optional, default 0.0)*) – if stitch_threshold>0.0 and not do_3D and equal image sizes, masks are stitched in 3D to return volume segmentation
- **rescale** (*float (optional, default None)*) – if diameter is set to None, and rescale is not None, then rescale is used instead of diameter for resizing image

- **progress** (*pyqt progress bar (optional, default None)*) – to return progress bar status to GUI
- **omni** (*bool (optional, default False)*) – use omnipose mask recontruction features
- **calc_trace** (*bool (optional, default False)*) – calculate pixel traces and return as part of the flow
- **verbose** (*bool (optional, default False)*) – turn on additional output to logs for debugging
- **verbose** – turn on additional output to logs for debugging
- **transparency** (*bool (optional, default False)*) – modulate flow opacity by magnitude instead of brightness (can use flows on any color background)
- **model_loaded** (*bool (optional, default False)*) – internal variable for determining if model has been loaded, used in __main__.py

Returns

- **masks** (*list of 2D arrays, or single 3D array (if do_3D=True)*) – labelled image, where 0=no masks; 1,2,…=mask labels
- **flows** (*list of lists 2D arrays, or list of 3D arrays (if do_3D=True)*) – flows[k][0] = XY flow in HSV 0-255 flows[k][1] = flows at each pixel flows[k][2] = scalar cell probability (Cellpose) or distance transform (Omnipose) flows[k][3] = final pixel locations after Euler integration flows[k][4] = boundary output (nonempty for Omnipose) flows[k][5] = pixel traces (nonempty for calc_trace=True)
- **styles** (*list of 1D arrays of length 256, or single 1D array (if do_3D=True)*) – style vector summarizing each image, also used to estimate size of objects in image
- **diams** (*list of diameters, or float (if do_3D=True)*)

9.2 CellposeModel

class cellpose.models.**CellposeModel**(*gpu=False*, *pretrained_model=False*, *model_type=None*, *net_avg=True*, *torch=True*, *diam_mean=30.0*, *device=None*, *residual_on=True*, *style_on=True*, *concatenation=False*, *nchan=2*, *nclasses=3*, *omni=False*)

Parameters

- **gpu** (*bool (optional, default False)*) – whether or not to save model to GPU, will check if GPU available
- **pretrained_model** (*str or list of strings (optional, default False)*) – path to pretrained cellpose model(s), if None or False, no model loaded
- **model_type** (*str (optional, default None)*) – 'cyto'=cytoplasm model; 'nuclei'=nucleus model; if None, pretrained_model used
- **net_avg** (*bool (optional, default True)*) – loads the 4 built-in networks and averages them if True, loads one network if False
- **torch** (*bool (optional, default True)*) – use torch nn rather than mxnet
- **diam_mean** (*float (optional, default 27.)*) – mean 'diameter', 27. is built in value for 'cyto' model

- **device** (*mxnet device (optional, default None)*) – where model is saved (mx.gpu() or mx.cpu()), overrides gpu input, recommended if you want to use a specific GPU (e.g. mx.gpu(4))
- **model_dir** (*str (optional, default None)*) – overwrite the built in model directory where cellpose looks for models
- **omni** (*use omnipose model (optional, default False)*) –

eval(*x*, *batch_size=8*, *channels=None*, *channel_axis=None*, *z_axis=None*, *normalize=True*, *invert=False*, *rescale=None*, *diameter=None*, *do_3D=False*, *anisotropy=None*, *net_avg=True*, *augment=False*, *tile=True*, *tile_overlap=0.1*, *resample=True*, *interp=True*, *cluster=False*, *flow_threshold=0.4*, *mask_threshold=0.0*, *diam_threshold=12.0*, *cellprob_threshold=None*, *dist_threshold=None*, *compute_masks=True*, *min_size=15*, *stitch_threshold=0.0*, *progress=None*, *omni=False*, *calc_trace=False*, *verbose=False*, *transparency=False*, *loop_run=False*, *model_loaded=False*)

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

Parameters

- **x** (*list or array of images*) – can be list of 2D/3D/4D images, or array of 2D/3D/4D images
- **batch_size** (*int (optional, default 8)*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage)
- **channels** (*list (optional, default None)*) – 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]].
- **channel_axis** (*int (optional, default None)*) – if None, channels dimension is attempted to be automatically determined
- **z_axis** (*int (optional, default None)*) – if None, z dimension is attempted to be automatically determined
- **normalize** (*bool (default, True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel
- **invert** (*bool (optional, default False)*) – invert image pixel intensity before running network
- **rescale** (*float (optional, default None)*) – resize factor for each image, if None, set to 1.0
- **diameter** (*float (optional, default None)*) – diameter for each image (only used if rescale is None), if diameter is None, set to diam_mean
- **do_3D** (*bool (optional, default False)*) – set to True to run 3D segmentation on 4D image input
- **anisotropy** (*float (optional, default None)*) – for 3D segmentation, optional rescaling factor (e.g. set to 2.0 if Z is sampled half as dense as X or Y)
- **net_avg** (*bool (optional, default True)*) – runs the 4 built-in networks and averages them if True, runs one network if False
- **augment** (*bool (optional, default False)*) – tiles image with overlapping tiles and flips overlapped regions to augment

- **tile** (*bool (optional, default True)*) – tiles image to ensure GPU/CPU memory usage limited (recommended)
- **tile_overlap** (*float (optional, default 0.1)*) – fraction of overlap of tiles when computing flows
- **resample** (*bool (optional, default True)*) – run dynamics at original image size (will be slower but create more accurate boundaries)
- **interp** (*bool (optional, default True)*) – interpolate during 2D dynamics (not available in 3D) (in previous versions it was False)
- **flow_threshold** (*float (optional, default 0.4)*) – flow error threshold (all cells with errors below threshold are kept) (not used for 3D)
- **mask_threshold** (*float (optional, default 0.0)*) – all pixels with value above threshold kept for masks, decrease to find more and larger masks
- **dist_threshold** (*float (optional, default None) DEPRECATED*) – use mask_threshold instead
- **cellprob_threshold** (*float (optional, default None) DEPRECATED*) – use mask_threshold instead
- **compute_masks** (*bool (optional, default True)*) – Whether or not to compute dynamics and return masks. This is set to False when retrieving the styles for the size model.
- **min_size** (*int (optional, default 15)*) – minimum number of pixels per mask, can turn off with -1
- **stitch_threshold** (*float (optional, default 0.0)*) – if stitch_threshold>0.0 and not do_3D, masks are stitched in 3D to return volume segmentation
- **progress** (*pyqt progress bar (optional, default None)*) – to return progress bar status to GUI
- **omni** (*bool (optional, default False)*) – use omnipose mask recontruction features
- **calc_trace** (*bool (optional, default False)*) – calculate pixel traces and return as part of the flow
- **verbose** (*bool (optional, default False)*) – turn on additional output to logs for debugging
- **transparency** (*bool (optional, default False)*) – modulate flow opacity by magnitude instead of brightness (can use flows on any color background)
- **loop_run** (*bool (optional, default False)*) – internal variable for determining if model has been loaded, stops model loading in loop over images
- **model_loaded** (*bool (optional, default False)*) – internal variable for determining if model has been loaded, used in __main__.py

Returns

- **masks** (*list of 2D arrays, or single 3D array (if do_3D=True)*) – labelled image, where 0=no masks; 1,2,…=mask labels
- **flows** (*list of lists 2D arrays, or list of 3D arrays (if do_3D=True)*) – flows[k][0] = XY flow in HSV 0-255 flows[k][1] = flows at each pixel flows[k][2] = scalar cell probability (Cellpose) or distance transform (Omnipose) flows[k][3] = boundary output (nonempty for Omnipose) flows[k][4] = final pixel locations after Euler integration flows[k][5] = pixel traces (nonempty for calc_trace=True)

- **styles** (*list of 1D arrays of length 64, or single 1D array (if do_3D=True)*) – style vector summarizing each image, also used to estimate size of objects in image

loss_fn(*lbl*, *y*)

loss function between true labels lbl and prediction y

train(*train_data*, *train_labels*, *train_files=None*, *test_data=None*, *test_labels=None*, *test_files=None*, *channels=None*, *normalize=True*, *save_path=None*, *save_every=100*, *save_each=False*, *learning_rate=0.2*, *n_epochs=500*, *momentum=0.9*, *SGD=True*, *weight_decay=1e-05*, *batch_size=8*, *nimg_per_epoch=None*, *rescale=True*, *min_train_masks=5*, *omni=False*, *netstr=None*)

train network with images train_data

Parameters

- **train_data** (*list of arrays (2D or 3D)*) – images for training
- **train_labels** (*list of arrays (2D or 3D)*) – labels for train_data, where 0=no masks; 1,2,…=mask labels can include flows as additional images
- **train_files** (*list of strings*) – file names for images in train_data (to save flows for future runs)
- **test_data** (*list of arrays (2D or 3D)*) – images for testing
- **test_labels** (*list of arrays (2D or 3D)*) – labels for test_data, where 0=no masks; 1,2,…=mask labels; can include flows as additional images
- **test_files** (*list of strings*) – file names for images in test_data (to save flows for future runs)
- **channels** (*list of ints (default, None)*) – channels to use for training
- **normalize** (*bool (default, True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel
- **save_path** (*string (default, None)*) – where to save trained model, if None it is not saved
- **save_every** (*int (default, 100)*) – save network every [save_every] epochs
- **learning_rate** (*float or list/np.ndarray (default, 0.2)*) – learning rate for training, if list, must be same length as n_epochs
- **n_epochs** (*int (default, 500)*) – how many times to go through whole training set during training
- **weight_decay** (*float (default, 0.00001)*) –
- **SGD** (*bool (default, True)*) – use SGD as optimization instead of RAdam
- **batch_size** (*int (optional, default 8)*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage)
- **nimg_per_epoch** (*int (optional, default None)*) – minimum number of images to train on per epoch, with a small training set (< 8 images) it may help to set to 8
- **rescale** (*bool (default, True)*) – whether or not to rescale images to diam_mean during training, if True it assumes you will fit a size model after training or resize your images accordingly, if False it will try to train the model to be scale-invariant (works worse)
- **min_train_masks** (*int (default, 5)*) – minimum number of masks an image must have to use in training set

- **netstr** (*str (default, None)*) – name of network, otherwise saved with name as params + training start time

9.3 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

Parameters

- **cp_model** (*UnetModel or CellposeModel*) – model from which to get styles
- **device** (*mxnet device (optional, default mx.cpu())*) – where cellpose model is saved (mx.gpu() or mx.cpu())
- **pretrained_size** (*str*) – path to pretrained size model
- **omni** (*bool*) – whether or not to use distance-based size metrics corresponding to 'omni' model

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

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 or array of images*) – can be list of 2D/3D images, or array of 2D/3D images
- **channels** (*list (optional, default None)*) – 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]].
- **channel_axis** (*int (optional, default None)*) – if None, channels dimension is attempted to be automatically determined
- **normalize** (*bool (default, True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel
- **invert** (*bool (optional, default False)*) – invert image pixel intensity before running network
- **augment** (*bool (optional, default False)*) – tiles image with overlapping tiles and flips overlapped regions to augment
- **tile** (*bool (optional, default True)*) – tiles image to ensure GPU/CPU memory usage limited (recommended)
- **progress** (*pyqt progress bar (optional, default None)*) – to return progress bar status to GUI

Returns

- **diam** (*array, float*) – final estimated diameters from images x or styles style after running both steps
- **diam_style** (*array, float*) – estimated diameters from style alone

train(*train_data*, *train_labels*, *test_data=None*, *test_labels=None*, *channels=None*, *normalize=True*, *learning_rate=0.2*, *n_epochs=10*, *l2_regularization=1.0*, *batch_size=8*, *omni=False*)

train size model with images train_data to estimate linear model from styles to diameters

Parameters

- **train_data** (*list of arrays (2D or 3D)*) – images for training
- **train_labels** (*list of arrays (2D or 3D)*) – labels for train_data, where 0=no masks; 1,2,…=mask labels can include flows as additional images
- **channels** (*list of ints (default, None)*) – channels to use for training
- **normalize** (*bool (default, True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel
- **n_epochs** (*int (default, 10)*) – how many times to go through whole training set (taking random patches) for styles for diameter estimation
- **l2_regularization** (*float (default, 1.0)*) – regularize linear model from styles to diameters
- **batch_size** (*int (optional, default 8)*) – number of 224x224 patches to run simultaneously on the GPU (can make smaller or bigger depending on GPU memory usage)

9.4 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 ND-arrays (int) or ND-array (int)*) – where 0=NO masks; 1,2… are mask labels
- **masks_pred** (*list of ND-arrays (int) or ND-array (int)*) – ND-array (int) where 0=NO masks; 1,2… are mask labels

Returns **aji**

Return type aggregated jaccard index for each set of masks

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 ND-arrays (int) or ND-array (int)*) – where 0=NO masks; 1,2… are mask labels

- **masks_pred** (*list of ND-arrays (int) or ND-array (int)*) – ND-array (int) where 0=NO masks; 1,2… are mask labels

Returns

- **ap** (*array [len(masks_true) x len(threshold)]*) – 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

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

boundary precision / recall / Fscore

cellpose.metrics.**flow_error**(*maski*, *dP_net*, *use_gpu=False*, *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. Setting can be changed in Cellpose.eval or CellposeModel.eval.

Parameters

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

Returns

- **flow_errors** (*float array with length maski.max()*) – mean squared error between predicted flows and flows from masks
- **dP_masks** (*ND-array (float)*) – ND flows produced from the predicted masks

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

return best-matched masks

9.5 Flows to masks

cellpose.dynamics.**compute_masks**(*dP*, *cellprob*, *bd=None*, *p=None*, *inds=None*, *niter=200*, *mask_threshold=0.0*, *diam_threshold=12.0*, *flow_threshold=0.4*, *interp=True*, *do_3D=False*, *min_size=15*, *resize=None*, *verbose=False*, *use_gpu=False*, *device=None*, *nclasses=3*)

compute masks using dynamics from dP, cellprob, and boundary

cellpose.dynamics.**follow_flows**(*dP*, *mask=None*, *inds=None*, *niter=200*, *interp=True*, *use_gpu=True*, *device=None*, *omni=False*, *calc_trace=False*)

define pixels and run dynamics to recover masks in 2D

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

Parameters

- **dP** (*float32, 3D or 4D array*) – flows [axis x Ly x Lx] or [axis x Lz x Ly x Lx]

- **mask** (*(optional, default None)*) – pixel mask to seed masks. Useful when flows have low magnitudes.
- **niter** (*int (optional, default 200)*) – number of iterations of dynamics to run
- **interp** (*bool (optional, default True)*) – interpolate during 2D dynamics (not available in 3D) (in previous versions + paper it was False)
- **use_gpu** (*bool (optional, default False)*) – use GPU to run interpolated dynamics (faster than CPU)

Returns **p** – final locations of each pixel after dynamics

Return type float32, 3D array

cellpose.dynamics.**get_masks**(*p*, *iscell=None*, *rpad=20*, *flows=None*, *threshold=0.4*, *use_gpu=False*, *device=None*)

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. :param p: final locations of each pixel after dynamics,

size [axis x Ly x Lx] or [axis x Lz x Ly x Lx].

Parameters

- **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, default 20)*) – histogram edge padding
- **threshold** (*float (optional, default 0.4)*) – masks with flow error greater than threshold are discarded (if flows is not None)
- **flows** (*float, 3D or 4D array (optional, default None)*) – flows [axis x Ly x Lx] or [axis x Lz x Ly x Lx]. If flows is not None, then masks with inconsistent flows are removed using *remove_bad_flow_masks*.

Returns **M0** – 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 int, 2D or 3D array

cellpose.dynamics.**labels_to_flows**(*labels*, *files=None*, *use_gpu=False*, *device=None*, *redo_flows=False*)

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

if files is not None, flows are saved to files to be reused

Parameters **labels** (*list of ND-arrays*) – labels[k] can be 2D or 3D, if [3 x Ly x Lx] then it is assumed that flows were precomputed. Otherwise labels[k][0] or labels[k] (if 2D) is used to create flows and cell probabilities.

Returns **flows** – 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 ycoordinates yc and xcoordinates xc to Y

Parameters

- **I** (*C x Ly x Lx*) –

- **yc** (*ni*) – new y coordinates
- **xc** (*ni*) – new x coordinates
- **Y** (*C x ni*) – I sampled at (yc,xc)

cellpose.dynamics.**masks_to_flows**(*masks*, *use_gpu=False*, *device=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 median 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

- **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*) – for each pixel, the distance to the center of the mask in which it resides

cellpose.dynamics.**masks_to_flows_cpu**(*masks*, *device=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 median of all pixels that is inside the mask. Result of diffusion is converted into flows by computing the gradients of the diffusion density map. :param masks: labelled masks 0=NO masks; 1,2,...=mask labels :type masks: int, 2D array

Returns

- **mu** (*float, 3D array*) – flows in Y = mu[-2], flows in X = mu[-1]. if masks are 3D, flows in Z = mu[0].
- **mu_c** (*float, 2D array*) – for each pixel, the distance to the center of the mask in which it resides

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

convert masks to flows using diffusion from center pixel Center of masks where diffusion starts is defined using COM :param masks: labelled masks 0=NO masks; 1,2,...=mask labels :type masks: int, 2D or 3D array

Returns

- **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*) – for each pixel, the distance to the center of the mask in which it resides

cellpose.dynamics.**remove_bad_flow_masks**(*masks*, *flows*, *threshold=0.4*, *use_gpu=False*, *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 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, default 0.4)*) – masks with flow error greater than threshold are discarded.

Returns **masks** – 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 int, 2D or 3D array

`cellpose.dynamics.step_factor(t)`

Euler integration suppression factor.

`cellpose.dynamics.steps2D(p, dP, inds, niter, omni=False, calc_trace=False)`

run dynamics of pixels to recover masks in 2D

Euler integration of dynamics dP for niter steps

Parameters

- **p** (*float32, 3D array*) – pixel locations [axis x Ly x Lx] (start at initial meshgrid)
- **dP** (*float32, 3D array*) – flows [axis x Ly x Lx]
- **inds** (*int32, 2D array*) – non-zero pixels to run dynamics on [npixels x 2]
- **niter** (*int32*) – number of iterations of dynamics to run

Returns **p** – final locations of each pixel after dynamics

Return type float32, 3D array

`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** (*float32, 4D array*) – pixel locations [axis x Lz x Ly x Lx] (start at initial meshgrid)
- **dP** (*float32, 4D array*) – flows [axis x Lz x Ly x Lx]
- **inds** (*int32, 2D array*) – non-zero pixels to run dynamics on [npixels x 3]
- **niter** (*int32*) – number of iterations of dynamics to run

Returns **p** – final locations of each pixel after dynamics

Return type float32, 4D array

9.6 Image transforms

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

average results of network over tiles

Parameters

- **y** (*float, [ntiles x nclasses x bsize x bsize]*) – output of cellpose network for each tile
- **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 **yf** – network output averaged over tiles

Return type float32, [nclasses x Ly x Lx]

cellpose.transforms.**convert_image**(*x*, *channels*, *channel_axis=None*, *z_axis=None*, *do_3D=False*, *normalize=True*, *invert=False*, *nchan=2*, *omni=False*)

return image with z first, channels last and normalized intensities

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

make tiles of image to run at test-time

if augmented, tiles are flipped and tile_overlap=2.

- original
- flipped vertically
- flipped horizontally
- flipped vertically and horizontally

Parameters

- **imgi** (*float32*) – array that's nchan x Ly x Lx
- **bsize** (*float (optional, default 224)*) – size of tiles
- **augment** (*bool (optional, default False)*) – flip tiles and set tile_overlap=2.
- **tile_overlap** (*float (optional, default 0.1)*) – fraction of overlap of tiles

Returns

- **IMG** (*float32*) – array that's ntiles x nchan 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

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 minimum dimension last as channels if < 10, or force==True

cellpose.transforms.**normalize99**(*Y*, *lower=0.01*, *upper=99.99*, *omni=False*)

normalize image so 0.0 is 0.01st percentile and 1.0 is 99.99th percentile

cellpose.transforms.**normalize_img**(*img*, *axis=- 1*, *invert=False*, *omni=False*)

normalize each channel of the image so that so that 0.0=1st percentile and 1.0=99th percentile of image intensities

optional inversion

Parameters

- **img** (*ND-array (at least 3 dimensions)*) –
- **axis** (*channel axis to loop over for normalization*) –

Returns **img** – normalized image of same size

Return type ND-array, float32

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

augmentation by random rotation and resizing X and Y are lists or arrays of length nimg, with dims channels x Ly x Lx (channels optional) :param X: list of image arrays of size [nchan x Ly x Lx] or [Ly x Lx] :type X: LIST of ND-arrays, float :param Y: 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.

Parameters

- **scale_range** (*float (optional, default 1.0)*) – Range of resizing of images for augmentation. Images are resized by (1-scale_range/2) + scale_range * np.random.rand()
- **xy** (*tuple, int (optional, default (224,224))*) – size of transformed images to return
- **do_flip** (*bool (optional, default True)*) – whether or not to flip images horizontally
- **rescale** (*array, float (optional, default None)*) – how much to resize images by before performing augmentations
- **unet** (*bool (optional, default False)*) –

Returns

- **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.**pad_image_ND**(*img0*, *div=16*, *extra=1*)

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

Parameters

- **img0** (*ND-array*) – image of size [nchan (x Lz) x Ly x Lx]
- **div** (*int (optional, default 16)*) –

Returns

- **I** (*ND-array*) – padded image
- **ysub** (*array, int*) – yrange of pixels in I corresponding to img0
- **xsub** (*array, int*) – xrange of pixels in I corresponding to img0

cellpose.transforms.**random_rotate_and_resize**(*X*, *Y=None*, *scale_range=1.0*, *gamma_range=0.5*, *xy=(224, 224)*, *do_flip=True*, *rescale=None*, *unet=False*, *inds=None*, *depth=0*, *omni=False*)

augmentation by random rotation and resizing

X and Y are lists or arrays of length nimg, with dims channels x Ly x Lx (channels optional)

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, default None)*) – 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.
- **scale_range** (*float (optional, default 1.0)*) – Range of resizing of images for augmentation. Images are resized by (1-scale_range/2) + scale_range * np.random.rand()
- **gamma_range** (*float (optional, default 0.5)*) – Images are gamma-adjusted im**gamma for gamma in (1-gamma_range,1+gamma_range)
- **xy** (*tuple, int (optional, default (224,224))*) – size of transformed images to return
- **do_flip** (*bool (optional, default True)*) – whether or not to flip images horizontally
- **rescale** (*array, float (optional, default None)*) – how much to resize images by before performing augmentations
- **unet** (*bool (optional, default False)*) –

Returns

- **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 array that's (Z x) Ly x Lx x nchan*) – if data.ndim==8 and data.shape[0]<8, assumed to be nchan x Ly x Lx
- **channels** (*list of int of length 2 (optional, default [0,0])*) – 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 train on grayscale images, input [0,0]. To train on images with cells in green and nuclei in blue, input [2,3].
- **invert** (*bool*) – invert intensities

Returns **data**

Return type numpy array that's (Z x) Ly x Lx x nchan (if chan_first==False)

cellpose.transforms.**reshape_and_normalize_data**(*train_data*, *test_data=None*, *channels=None*, *normalize=True*, *omni=False*)

inputs converted to correct shapes for *training* and rescaled so that 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel

Parameters

- **train_data** (*list of ND-arrays, float*) – list of training images of size [Ly x Lx], [nchan x Ly x Lx], or [Ly x Lx x nchan]
- **test_data** (*list of ND-arrays, float (optional, default None)*) – list of testing images of size [Ly x Lx], [nchan x Ly x Lx], or [Ly x Lx x nchan]

- **channels** (*list of int of length 2 (optional, default None)*) – 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 train on grayscale images, input [0,0]. To train on images with cells in green and nuclei in blue, input [2,3].
- **normalize** (*bool (optional, True)*) – normalize data so 0.0=1st percentile and 1.0=99th percentile of image intensities in each channel

Returns

- **train_data** (*list of ND-arrays, float*) – list of training images of size [2 x Ly x Lx]
- **test_data** (*list of ND-arrays, float (optional, default None)*) – list of testing images of size [2 x Ly x Lx]
- **run_test** (*bool*) – whether or not test_data was correct size and is useable during training

cellpose.transforms.**reshape_train_test**(*train_data*, *train_labels*, *test_data*, *test_labels*, *channels*, *normalize=True*, *omni=False*)

check sizes and reshape train and test data for training

cellpose.transforms.**resize_image**(*img0*, *Ly=None*, *Lx=None*, *rsz=None*, *interpolation=1*, *no_channels=False*)

resize image for computing flows / unresize for computing dynamics

Parameters

- **img0** (*ND-array*) – 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*) –
- **Lx** (*int, optional*) –
- **rsz** (*float, optional*) – resize coefficient(s) for image; if Ly is None then rsz is used
- **interpolation** (*cv2 interp method (optional, default cv2.INTER_LINEAR)*) –

Returns **imgs** – image of size [Ly x Lx x nchan] or [Lz x Ly x Lx x nchan]

Return type ND-array

cellpose.transforms.**unaugment_tiles**(*y*, *unet=False*)

reverse test-time augmentations for averaging

Parameters

- **y** (*float32*) – array that's ntiles_y x ntiles_x x chan x Ly x Lx where chan = (dY, dX, cell prob)
- **unet** (*bool (optional, False)*) – whether or not unet output or cellpose output

Returns **y**

Return type float32

9.7 Plot functions

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

returns pixels of disk with radius r and center med

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

dP is 2 x Y x X => 'optic' flow representation

Parameters

- **dP** (*2xLyxLx array*) – Flow field components [dy,dx]
- **transparency** (*bool, default False*) – magnitude of flow controls opacity, not lightness (clear background)
- **mask** (*2D array*) – Multiplies each RGB component to suppress noise

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

image is 2 x Ly x Lx or Ly x Lx x 2 - change to RGB Ly x Lx x 3

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

get patch of size bsize x bsize with most masks

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

overlay masks on image (set image to grayscale)

Parameters

- **img** (*int or float, 2D or 3D array*) – img is 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, default None)*) – size [nmasks x 3], each entry is a color in 0-255 range

Returns **RGB** – array of masks overlaid on grayscale image

Return type 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, default None)*) – size [nmasks x 3], each entry is a color in 0-255 range

Returns **RGB** – array of masks overlaid on grayscale image

Return type uint8, 3D array

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

Generates a red outline overlay onto image.

cellpose.plot.**show_segmentation**(*fig*, *img*, *maski*, *flowi*, *channels=[0, 0]*, *file_name=None*, *omni=False*, *seg_norm=False*, *bg_color=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** (*2D or 3D array*) – image input into cellpose
- **maski** (*int, 2D array*) – for image k, masks[k] output from Cellpose.eval, where 0=NO masks; 1,2,...=mask labels
- **flowi** (*int, 2D array*) – for image k, flows[k][0] output from Cellpose.eval (RGB of flows)
- **channels** (*list of int (optional, default [0,0])*) – channels used to run Cellpose, no need to use if image is RGB
- **file_name** (*str (optional, default None)*) – file name of image, if file_name is not None, figure panels are saved
- **omni** (*bool (optional, default False)*) – use omni version of normalize99, image_to_rgb
- **seg_norm** (*bool (optional, default False)*) – improve cell visibility under labels
- **bg_color** (*float (Optional, default none)*) – background color to draw behind flow (visible if flow transparency is on)

PYTHON MODULE INDEX

C

INDEX

S

T

U