Cellpose segmentation

Rémy Torro

August 9, 2023

1 Introduction

In this report, we will outline the process of training a Cellpose model from scratch to perform a segmentation task on our own multichannel data. Our example will be paired brightfield and RICM images, for which we want to segment accurately the adhesion patch of adhered cells and pinpoint the location of non-adhered cells.

1.1 Data preparation

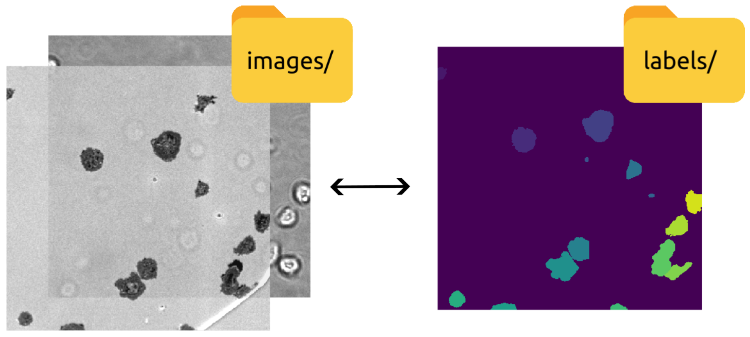

Figure 1: Example of annotation for one training image. The adhered cells are outlined as precisely as possible whereas non adhered cells only have a very small patch.

Cellpose expects a training directory and an optional test directory. These directories should contain both the images and the paired annotations. One can specify a suffix for the annotations (`--mask_filter` in the CLI command). Following good practice in Deep Learning, we will perform some data augmentation on our dataset and clearly separate training and testing data. We will export the images with the proper Cellpose-friendly structure of test and train sets.

1.1.1 Dataset splitting

We take at random 25 % of our dataset as test data. These images and paired annotations are copied to a `dataset/test/` folder. No augmentation is performed on the test set.

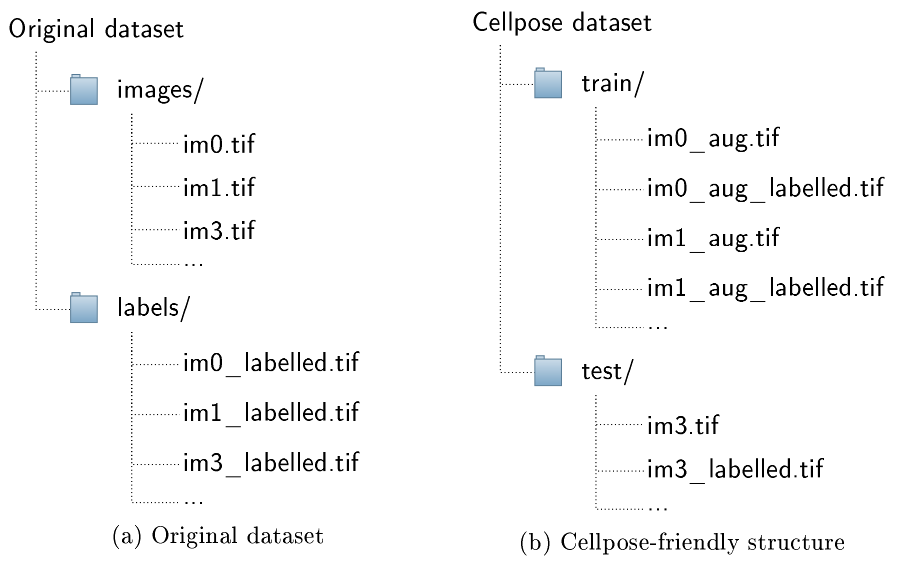

Figure 2: from the original dataset structure, convenient to annotate many images over many days to a structure directly compatible with the Cellpose CLI command.

For the remaining images, we will sample, transform and export to the `dataset/train/` folder. The transformations include random flipping, random shift, slight intensity fluctuations and a white noise. We re-sample as many time as needed, to give enough inputs to the Cellpose model (here $M = N_{\text{train}} \times 2$).

As a result, the test set contains only original images and the paired annotations, whereas the train sets contains transformed images and the associated annotations. The quantity of train images can exceed the number of images in our dataset.

1.1.2 Train Cellpose model

Once the dataset structure is established, we can use the command line to call for a Cellpose training. A minimum Cellpose CLI command would look like this:

```
python -m cellpose --use_gpu --train --dir dataset/train/
```

We run Cellpose in training mode with the flag `--train`, using the GPU with `--use_gpu`, and we point toward the training set with `--dir dataset/train/`.

There are more underlying flags that are set to their default value. Let's go step by step towards refining the command to the problem at hand. We introduced a test set, which can be presented to the model with the flag `--test_dir dataset/test/`. We put the `_labelled` suffix to the name of the labels. We can pass the flag `--mask_filter _labelled`. The images are already normalized, so we pass `--no_norm`, to prevent Cellpose from applying a percentile normalization with clipping. Finally, and most importantly, we have to pass the two channels as inputs (the default will only read the channel interpreted as gray). Standard Cellpose is structured for either:

- a unique cytoplasm channel (`--pretrained_model cyto`)
- a cytoplasm channel plus a nucleus channel (`--pretrained_model cyto2`)

Since the core of Cellpose is a UNET, we can technically pass any number of channel as the input, without having to make changes to the output. To enter this mode, we have to pass `--pretrained_model None`, and `--all_channels` instead of `--chan 0`.

Eventually, we end up with a CLI command as follows:

```
python -m cellpose --use_gpu --train --dir dataset/train/
↪  --test_dir dataset/test/ --learning_rate 0.001 --weight_decay
↪  0.01 --n_epochs 5000 --mask_filter _labelled --verbose
↪  --no_norm --batch_size 8 --all_channels --pretrained_model
↪  None --save_every 100 --save_each
```

The model will train for 5000 epochs and save its weights each 100 epochs. We wil keep each of those intermediate models to compare performance in post.

1.2 Quality control: selecting the best model

We can reload any of the intermediate models as follows:

```
from cellpose.models import CellposeModel

model = CellposeModel(gpu=True,
                      model_type=None,
                      pretrained_model="path/to/model/",
                      diam_mean=30.0,
                      nchan=2,
                      )
```