

Interfacing CellProfiler with other software tools via files and plugins

Beth Cimini

Broad Institute of MIT and Harvard, Cambridge, MA.

Background information

General background information

This exercise is meant to extend and build upon the Beginner Segmentation exercise available from tutorials.cellprofiler.org. Please consult that tutorial for general information on how to configure CellProfiler as well as information about the images. It will be generally assumed you understand the modules covered in that tutorial, including input, object creation, overlays, and saving.

What is this exercise?

There are a number of great software tools available to a biologist wishing to analyze images these days - forum.image.sc has more than 60 open-source tools alone. Sometimes, though, it helps to have a multi-tool workflow - do one step in ToolA, and then another in ToolB (and then possibly C, D, etc, but hopefully not!). In this tutorial, you'll try 3 different ways of accessing work you did in other tools.

1. Loading masks created by other segmentation tools (in this case, [Cellpose](#), but many tools use this format)
2. Accessing [ilastik](#) to use a trained pixel classification model via CellProfiler's plugins system
3. Running Cellpose in CellProfiler via CellProfiler's plugins system and a Docker container

Plugins

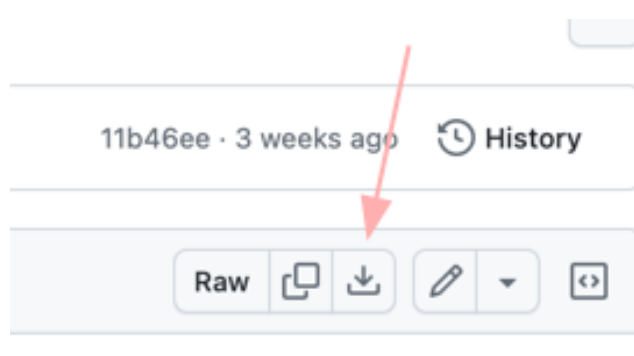
While CellProfiler doesn't have as many plugins as, say, Fiji, it does have many for you to try! You can visit plugins.cellprofiler.org to learn more. To quote from that site:

Plugins advance the capabilities of CellProfiler but are not officially supported in the same way as modules. A module may be in CellProfiler-plugins instead of CellProfiler itself because:

- it is under active development
- it has a niche audience
- it is not documented to CellProfiler's standards
- it only works with certain version of CellProfiler
- it requires extra libraries or other dependencies we are unable or unwilling to require for CellProfiler
- it has been contributed by a community member

Tip:

While that documentation has instructions on [installing plugins](#), in step 2 it suggests downloading all of the CellProfiler plugins; this isn't a bad thing to do, but you can download individual plugins from GitHub with the website button below as well or instead. We strongly recommend making a dedicated folder for CellProfiler plugins, as loading can be slow if there are a lot of other miscellaneous files around.



GitHub's "Download Raw Files" button

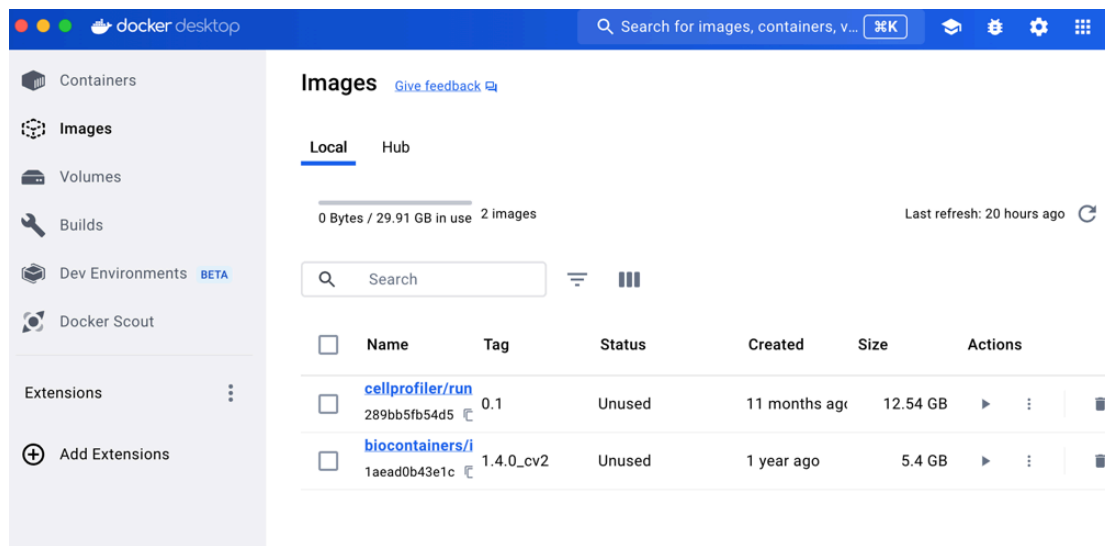
Docker Desktop

In Exercise 3 (and for Windows users, optionally Exercise 2 as well), we want to access software that we either don't want to install locally because it's painful if you're not pretty comfortable in Python (Cellpose) or that we CAN install locally but may not play nicely with other tools' multiprocessing setup (ilastik).

Computer scientists will often use software containers to ship tools or data that are hard to install - [here](#) is a good introduction and overview for non-computer scientists. You can think of them as "a pre-configured operating system in a box". Because they come to you pre-configured, installation of any software happens once-and-only-once (by the creator of the container, not by you), and should stay working for long after ie the latest Mac upgrade breaks a certain older software installation. Groups like [biocontainers](#) have already containerized many of the tools you know and love. There are a number of types of software containers, but one of the most common is called a Docker container.

Most biologists aren't aware of or don't use containers, especially because typical usage involves accessing them via your terminal. But Docker doesn't have to mean terminal! There are [containers that spit out interactive websites for you to use](#), and CellProfiler has plugins that are specifically set up to call other tools that live inside Docker containers.

To do this, CellProfiler needs to have the infrastructure for Docker containers up and running, which you can give it by installing a free program called [Docker Desktop](#). You need not make an account (but you probably will need to reboot your computer). You then simply must make sure the program is open and running when you try to use a Docker-calling plugin in CellProfiler - CellProfiler will take care of everything else!



The Docker Desktop interface for Mac. Containers can be searched for in the top search bar!

Warning:

Anytime you're installing a program that can run other programs, you must of course be careful. Dockers give you access to a huge variety of tools that might otherwise be hard to install (or install together), but exercise caution in only running containers you recognize or trust!

Exercise 1: Importing masks from another segmentation tool

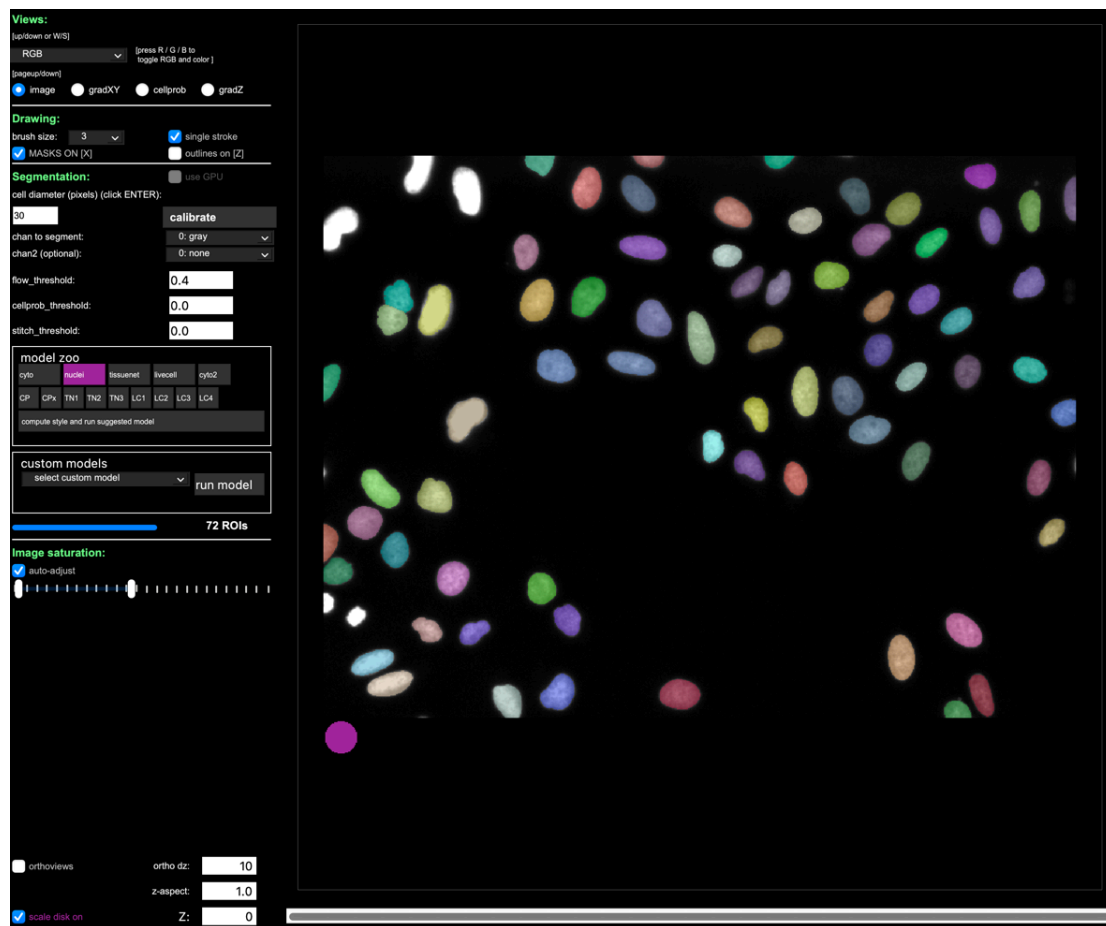
In this exercise, we will be loading in *label masks* (sometimes also called *label matrices*) made in Cellpose. Simply, a label mask is an image in which all the background pixels have a value of 0, all the pixels of object 1 have a value of 1, all the pixels of object 2 have a value of 2, etc. It is a commonly used format by many tools that do object segmentation (though unsuitable for applications where you have overlapping objects). Since many tools (including CellProfiler) can make these masks, CellProfiler has the ability to read them in and automatically detect them as objects.

We've used Cellpose 2.2.2 to generate nuclear masks from the DNA images and cell masks from the ActinGolgi images.

FYI only - how did we make these?

Cellpose was installed in a conda environment according to the [official instructions](#) for installation with the GUI.

Nuclear masks were made by starting cellpose in that conda environment with the command `cellpose`, and then dragging each nuclear image into the GUI, selecting the nuclei model, and then saving to PNG with `Cmd+N` (Mac) `Ctl+N` (Windows). Since there were only 10 and the hotkeys were available, this was not too painful.



Nuclei segmented in the Cellpose GUI

Cell masks were made by navigating into the image folder and then running the command below; this requires some knowledge of the [Cellpose command line syntax](#) but is much easier for large numbers of images.

```
python -m cellpose --dir . --img_filter Ch4 --pretrained_model cyto2 --diameter 40
--save_png --verbose
```


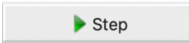
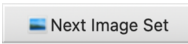
```

((cellpose) bcimini@wm4f8-761 cellpose_masks_cells % python -m cellpose --dir . --pretrained_model cyto2 --diameter 40 --save_png --verbose
2024-04-30 10:59:04,196 [INFO] WRITING LOG OUTPUT TO /Users/bcimini/.cellpose/run.log
2024-04-30 10:59:04,196 [INFO]
cellpose version: 2.2.2
platform: darwin
python version: 3.9.16
torch version: 2.0.1
2024-04-30 10:59:04,196 [INFO] >>> using CPU
2024-04-30 10:59:04,198 [INFO] >>> running cellpose on 10 images using chan_to_seg GRAY and chan (opt) NONE
2024-04-30 10:59:04,198 [INFO] >>> using CPU
2024-04-30 10:59:04,198 [INFO] >> cyto2 << model set to be used
2024-04-30 10:59:04,198 [INFO] Downloading: "https://www.cellpose.org/models/cyto2torch_1" to /Users/bcimini/.cellpose/models/cyto2torch_1
100%|
2024-04-30 10:59:37,787 [INFO] Downloading: "https://www.cellpose.org/models/cyto2torch_2" to /Users/bcimini/.cellpose/models/cyto2torch_2
100%|
2024-04-30 10:59:53,084 [INFO] Downloading: "https://www.cellpose.org/models/cyto2torch_3" to /Users/bcimini/.cellpose/models/cyto2torch_3
100%|
2024-04-30 11:00:04,675 [INFO] WARNING: MKL version on torch not working/installed - CPU version will be slightly slower.
2024-04-30 11:00:04,675 [INFO] see https://pytorch.org/docs/stable/backends.html?highlight=mkl
2024-04-30 11:00:04,877 [INFO] >>> model diam_mean = 30.000 (ROIs rescaled to this size during training)
2024-04-30 11:00:04,881 [INFO] Downloading: "https://www.cellpose.org/models/size_cyto2torch_0.npy" to /Users/bcimini/.cellpose/models/size_0
100%|
2024-04-30 11:00:05,073 [INFO] >>> using diameter 40.000 for all images
2024-04-30 11:00:05,074 [INFO] 0%| 0/10 [00:00<?, ?it/s]
2024-04-30 11:00:05,078 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:13,672 [INFO] >>> TOTAL TIME 8.59 sec
2024-04-30 11:00:13,719 [INFO] 10%|# 1/10 [00:08<01:17, 8.64s/it]
2024-04-30 11:00:13,731 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:23,115 [INFO] >>> TOTAL TIME 9.38 sec
2024-04-30 11:00:23,152 [INFO] 20%|## 2/10 [00:18<01:12, 9.11s/it]
2024-04-30 11:00:23,156 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:31,916 [INFO] >>> TOTAL TIME 8.76 sec
2024-04-30 11:00:31,940 [INFO] 30%|### 3/10 [00:26<01:02, 8.96s/it]
2024-04-30 11:00:31,945 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:39,665 [INFO] >>> TOTAL TIME 7.72 sec
2024-04-30 11:00:39,727 [INFO] 40%|### 4/10 [00:34<00:50, 8.50s/it]
2024-04-30 11:00:39,728 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:48,511 [INFO] >>> TOTAL TIME 8.78 sec
2024-04-30 11:00:48,541 [INFO] 50%|#### 5/10 [00:43<00:43, 8.61s/it]
2024-04-30 11:00:48,544 [INFO] --- FINDING MASKS ---
2024-04-30 11:00:57,089 [INFO] >>> TOTAL TIME 8.54 sec
2024-04-30 11:00:57,122 [INFO] 60%|##### 6/10 [00:52<00:34, 8.60s/it]
2024-04-30 11:00:57,125 [INFO] --- FINDING MASKS ---
2024-04-30 11:01:05,502 [INFO] >>> TOTAL TIME 8.38 sec
2024-04-30 11:01:05,540 [INFO] 70%|##### 7/10 [01:00<00:25, 8.54s/it]
2024-04-30 11:01:05,542 [INFO] --- FINDING MASKS ---
2024-04-30 11:01:13,788 [INFO] >>> TOTAL TIME 8.25 sec
2024-04-30 11:01:13,826 [INFO] 80%|##### 8/10 [01:08<00:16, 8.46s/it]
2024-04-30 11:01:13,829 [INFO] --- FINDING MASKS ---
2024-04-30 11:01:22,739 [INFO] >>> TOTAL TIME 8.91 sec
2024-04-30 11:01:22,802 [INFO] 90%|##### 9/10 [01:17<00:08, 8.62s/it]
2024-04-30 11:01:22,813 [INFO] --- FINDING MASKS ---
2024-04-30 11:01:35,845 [INFO] >>> TOTAL TIME 13.03 sec
2024-04-30 11:01:35,897 [INFO] 100%|##### 10/10 [01:30<00:00, 10.00s/it]
2024-04-30 11:01:35,897 [INFO] 100%|##### 10/10 [01:30<00:00, 9.08s/it]
2024-04-30 11:01:35,897 [INFO] >>> completed in 151.701 sec
(cellpose) bcimini@wm4f8-761 cellpose_masks_cells %

```

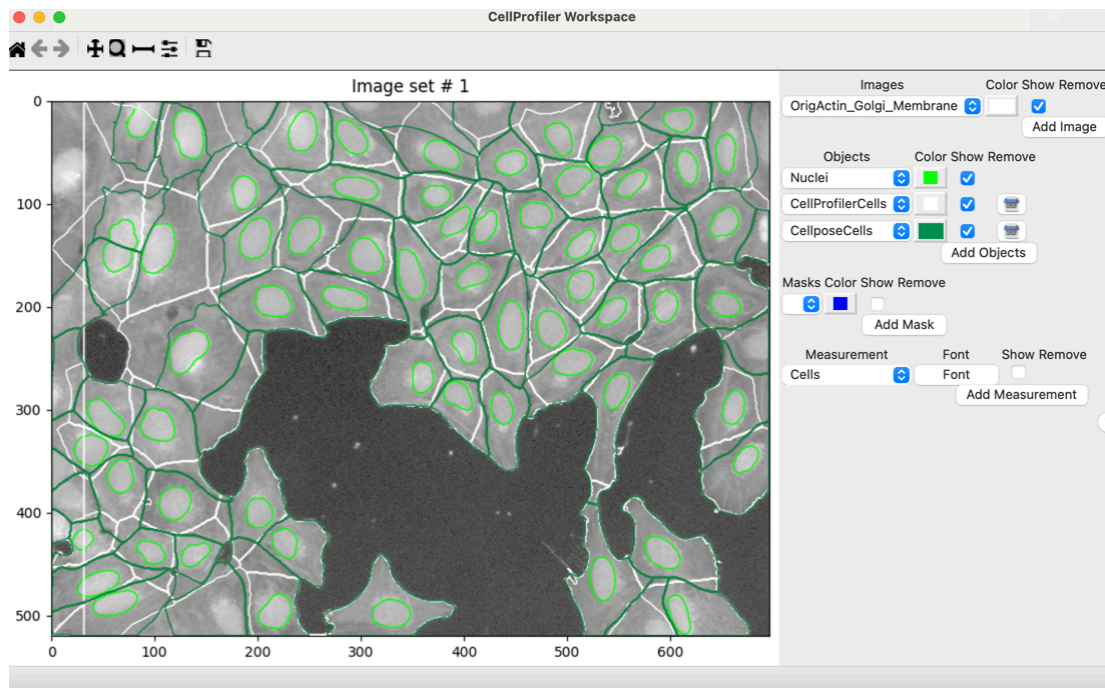
Command line execution of Cellpose

Loading these masks into CellProfiler

1. Open up a clean copy of CellProfiler (or run File -> New Project) and drag `bonus_1_import_masks.cppipe` into the pipeline panel.
2. Drag the `images_Illum-corrected` subfolder from the main exercise and the `cellpose_masks_cells` subfolder from this exercise. *Do not drag in the `cellpose_masks_nuclei` folder.*
3. Put CellProfiler into TestMode , open the eye next to OverlayOutlines, and then hit  3 times to create a classical segmentation and compare it with the Cellpose-generated version.
4. Optionally, open the Workspace Viewer to create easily on-the-fly customizable overlays
5. Hit  and repeat a couple of times to examine a couple more images

Question for you:

Are there cases where you think classical typically performs better? Where Cellpose performs better?



Visualization of images and objects in the Workspace Viewer

Reverse engineer how this worked

1. Open the NamesAndTypes module
2. Find the entry in NamesAndTypes that adds the masks - is there any setting you notice about it that is different than the other channels?
3. Find the entry in NamesAndTypes that has 2 rules the image has to pass, not just one - do you understand why that is?

Bonus-to-the-Bonus - add the provided nuclear segmentations as well

1. Add the `cellpose_masks_nuclei` folder in the Images module
 2. Go to the NamesAndTypes module and configure it so both kinds of masks can be loaded (hint: there's a duplicate button!)
- You'll probably have to modify the existing settings for loading the cell masks by changing its rule or adding a second rule. Do you understand how/why?

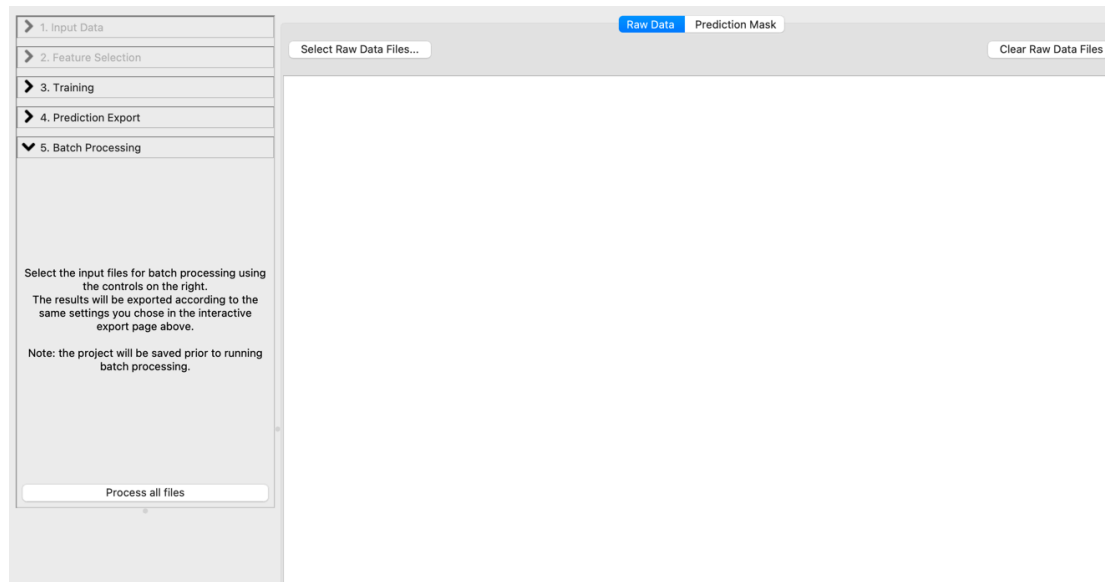
Exercise 2: Running ilastik locally or from a Docker container

Classical segmentation requires the thing you care about in an image be bright and every single other pixel dark(er). If you have a good clean fluorescent signal for the thing you care about, great! If not, you may need to resort to some tricks.

One trick is by training a classifier on a pixel-by-pixel basis to say “here is what I think is the likelihood that this is the pixel that you care about”. If your classifier is good, that will give you an image where the pixels you care about are high-probability (bright) and everywhere else is low-probability (dark). That's ex-

actly what we want! Biologists tend to call this “Pixel Classification”; computer scientists will sometimes refer to it as “Semantic Segmentation”.

There are a few popular Fiji plugins for doing this, including Weka Trainable Segmentation and Labkit. We tend to use ilastik, because it makes it easy to automate creating a classifier from a very small number of images and then bulk-applying it to many others in “Batch Processing” mode. You can check out a tutorial we have written for running ilastik, and *then* CellProfiler at tutorials.cellprofiler.org (look for Pixel Classification).



ilastik’s Batch Processing mode

Why do two steps though, when you can do one instead? In this tutorial we’ll take a pre-trained ilastik classifier and run it inside our CellProfiler pipeline, so we can find and measure objects all in one step.

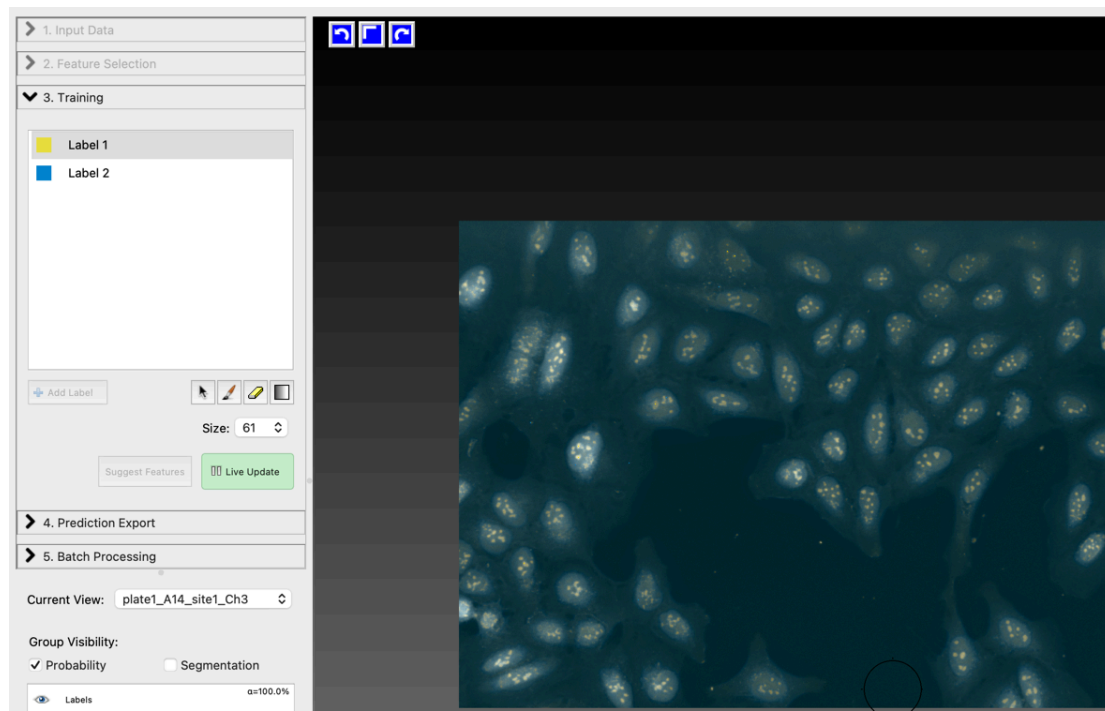
You will need either ilastik or Docker Desktop installed on your computer for this exercise. If ilastik, make sure it is CLOSED, if Docker Desktop, make sure it is OPEN. We recommend installing ilastik because it is a good and helpful tool, and will allow you to do this exercise’s bonus exercise.

Warning:

If you are on Windows, RunIlastik in Local mode (working on an installed copy of ilastik, rather than a copy inside a Docker file), this exercise will work in TestMode, but not will not run in analysis mode - we are working with the ilastik developers to determine why that is. This is fine for the purpose of this exercise; if you have a lot of your own data you want to run later, you can still use ilastik in a two step process, and/or use Dockerized ilastik.

FYI only - how did we make this?

This classifier was made in ilastik1.4.1b5 by training on 4 images (A14_site1, E18_site1, D16_site1, and C12_site1) identifying two classes - one class for nucleoli (yellow below), and one class for every other part of the image (blue below). One COULD have made more classes, but this in practice worked.



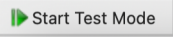

Screenshot of this ilastik classifier

When using ilastik for fluorescence microscopy, you will likely get the best performance if you keep your annotations extremely minimal - the classifier you're going to use was trained by using 31 total pixels of annotation across the 4 images (13 pixels of annotation inside nucleoli, and 18 pixels outside of them); no image had more than 14 pixels annotated in total. We strongly suggest you make your classifiers one pixel at a time! This feels counterintuitive but we promise it's true.

Grab the Runilastik plugin

1. Download [the plugin](#) into a folder on your local computer. As stated above, we strongly suggest a folder that contains ONLY plugins
2. In CellProfiler's File -> Preferences menu, set the CellProfiler plugins directory to the folder containing the plugin
3. Close and reopen CellProfiler to load the plugin

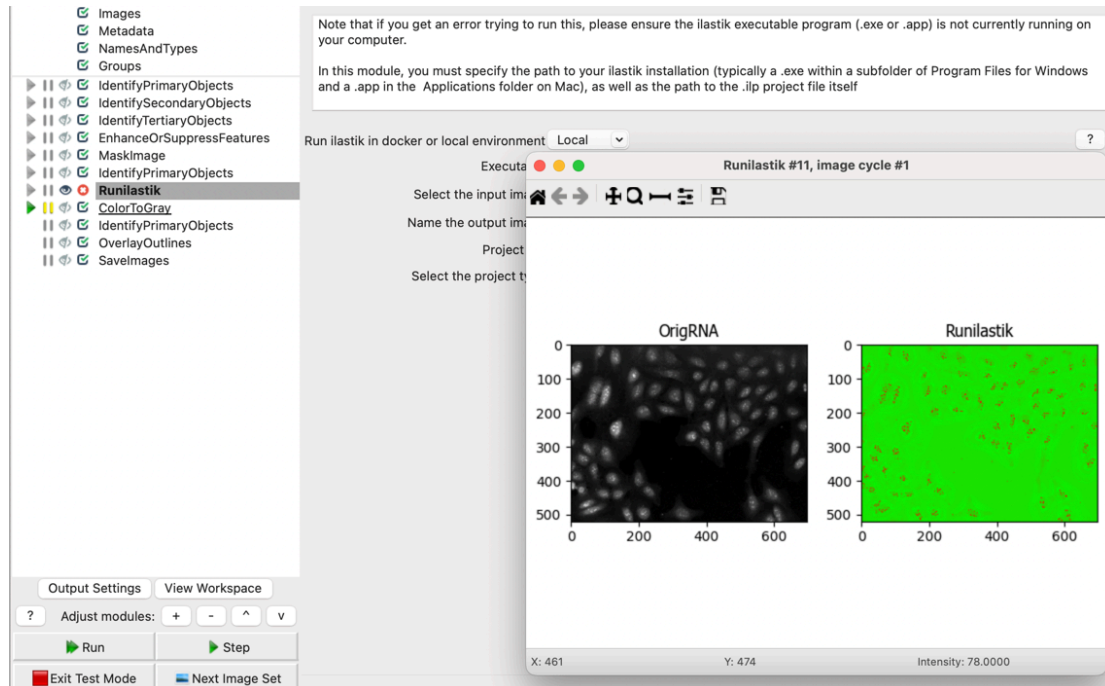
Load the classifier and evaluate it

1. Drag `bonus_2_ilastik.cppipe` into the pipeline panel.
2. Drag the `images_Illum-corrected` subfolder from the main exercise into the Images module
3. Open the Runilastik module, set the path to your project file (`NucleoliDetection.ilp`) and then also
 - ilastik-installed (recommended): set the path to your local installation of ilastik
 - Docker-installed: change the top setting from Local to Docker
3. Put CellProfiler into TestMode , open the eyes next to Runilastik and OverlayOutlines, and then hit 
 - You may wish to put a pause next to SaveImages, or uncheck it, to keep it from saving images, but that's up to you

Note:

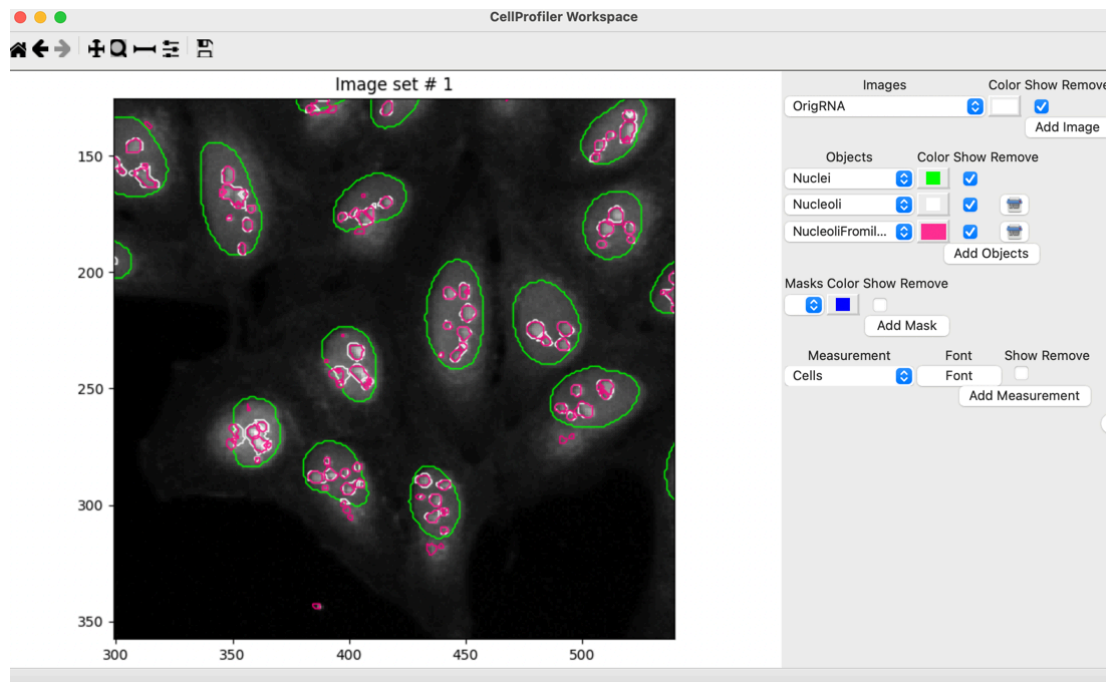
If using Docker, the very first time you hit the Runilastik module, it will need to download an ~5GB file, which may be slow depending on your connection. You only need to do this step once however!

4. Evaluate your prediction in Runilastik across a few image sets - how well does it perform? Does it perform worse on images it wasn't trained on?



Screenshot of the Runilastik module

5. Evaluate your segmentations across a few image sets using OverlayOutlines and/or the WorkspaceViewer - are there cases where you think the ilastik model-based segmentation performs better? The filtering-and-masking segmentation?



Evaluating segmentations in the Workspace Viewer

Bonus-to-the-Bonus - train your own ilastik model

Based on your evaluations above, can you identify some places where additional training might help fix some issues in the ilastik model?

1. Open `NucleoliDetection.ilp` in ilastik
2. Navigate to the Training tab
3. Turn on LiveUpdate
4. Pick an image you think needed some help and add (a very small number of very small) annotations. Did things get better or worse?
5. Add some very large annotations to one image, then switch images in order to prove to yourself large annotations can actually harm in some cases rather than help (if you want to save your classifier, go back and use the eraser tool to delete these!)
6. If you think you've materially improved the prediction, save this model and return to CellProfiler. Did it help in the cases where you thought it would?

Exercise 3: Running Cellpose from a Docker container

RunCellpose is by far our most popular plugin, simply because a) Cellpose is awesome and b) conda installing software when you aren't very computationally comfortable isn't. You can use the plugin in either of two modes - using a local conda or python installation that contains both CellProfiler or Cellpose, OR using Docker. The run time with Docker is substantially slower (about a minute more per image, in our testing), but if installation would take you a long time and be frustrating, in this sense you can "trade" your personal hands-on frustration time for time where CellProfiler is running on your computer (but you aren't there). For many biologists, this is a good trade!



Start Docker Desktop

1. If you have not already installed Docker Desktop from the link above, please do so! This may involve rebooting your computer.
2. Start Docker Desktop
3. Optional but strongly recommended - once Docker Desktop opens, use the search functionality to search Cellpose and look for the `runcellpose_with_pretrained` model and pull it. If for whatever reason this isn't working, move on, but it will save you some time later.

Grab the RunCellpose plugin

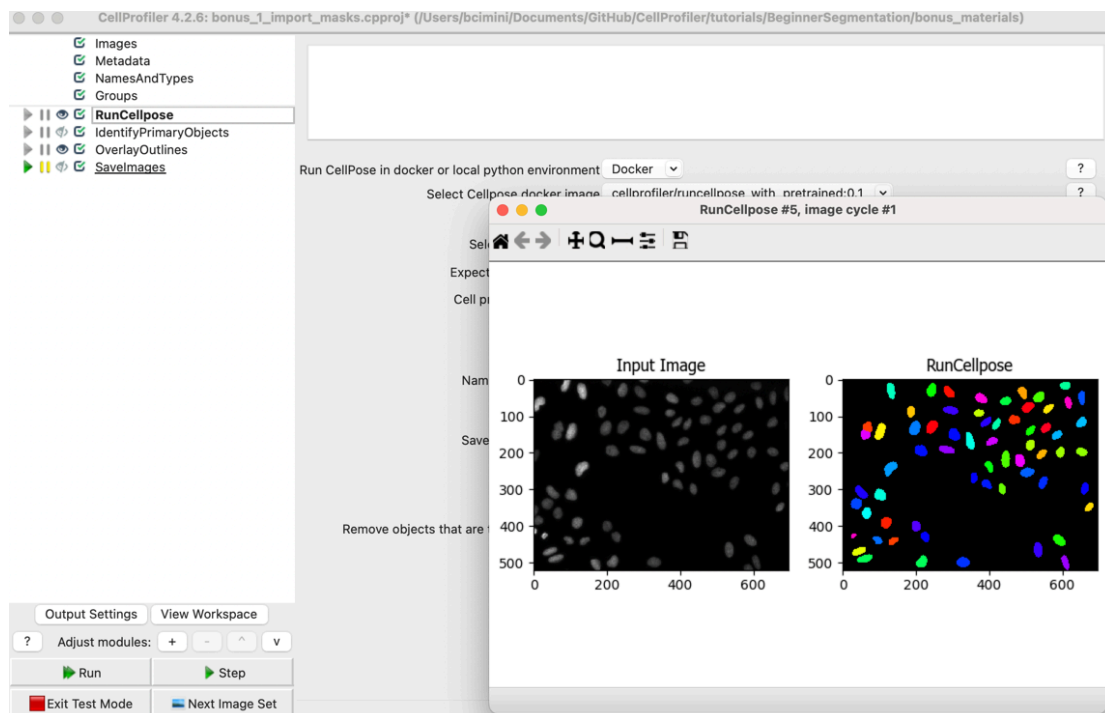
1. Download [the plugin](#) into a folder on your local computer. As stated above, we strongly suggest a folder that contains ONLY plugins
2. In CellProfiler's File -> Preferences menu, set the CellProfiler plugins directory to the folder containing the plugin
3. Close and reopen CellProfiler to load the plugin

Load the pipeline and evaluate segmentation


1. Drag `bonus_3_cellpose.cppipe` into the pipeline panel.
 2. Drag the `images_Illum-corrected` subfolder from the main exercise into the Images module
 3. Put CellProfiler into TestMode , open the eyes next to RunCellpose and OverlayOutlines, and then hit 
- You may wish to put a pause next to SaveImages, or uncheck it, to keep it from saving images, but that's up to you

Note:

If you didn't already pull the container in the Docker Desktop section above, the very first time you hit the RunCellpose module, it will need to download an ~13GB file, which may be slow depending on your connection. You only need to do this step once however!



The output of the RunCellpose module

4. As before, using OverlayOutlines and/or the WorkspaceViewer, evaluate segmentation on a few images. Where is CellProfiler doing better, and where is Cellpose doing better?
5. Use the  button to learn more about the different parameters you can pass to Cellpose (we don't offer all of them, but many) - how does tweaking these affect your output? How does changing the model you're using, and/or the image you're segmenting?