Cellfinder

Release 0.3.4

Adam Tyson, Charly Rousseau & Christian Niedworok

Jan 27, 2020

Contents

1 Introduction:

Cellfinder is a collection of tools from the Margrie Lab and others at the Sainsbury Wellcome Centre for the analysis of whole-brain imaging data such as serial-section imaging and lightsheet imaging in cleared tissue.

The aim is to provide a single solution for:

- Cell detection (initial cell candidate detection and refinement using deep learning implemented in Keras)
- Atlas registration (using aMAP)
- Analysis of cell positions in a common space

2 User Guide

2.1 Installation

For installation instructions in a cluster computing environment, see here

Only tested on Ubuntu 16.04 and 18.04

If required, install CUDA for GPU-support

Necessary for training cell classification, highly recommended for inference. For compatibility, we require CUDA 10.1 and cuDNN >=7.6, instructions can be found here. If you need other versions of CUDA (for other work), please see here for how to link CUDA versions to conda environments.

Alternatively, CUDA can be installed into your conda environment (see below).

Set up conda environment and install dependencies

Recommended, allthough not necessary Conda environments are used to isolate python versions and packages.

Once conda environment is set up, everything must be carried out in this environment

- Download miniconda installation file here
- Run miniconda installation file, e.g.:

```
bash ~/Downloads/Miniconda3-latest-Linux-x86_64.sh
```

- Create and activate new minimal conda environment

```
conda create --name cellfinder python
conda activate cellfinder
```

Optional: Install CUDA and cuDNN into the conda environment:

This avoids the need to install CUDA and cuDNN system-wide, but the GPU drivers are still required

```
conda install cudatoolkit=10.1 cudnn
```

Install cellfinder

```
pip install cellfinder
```

If you use a conda environment, remember to activate it (`conda activate cellfinder`) before using cellfinder.

Download atlas and trained classification models (optional)

When cellfinder runs, the appropriate machine learning models and reference atlas will be downloaded (if not previously done so). If you would like to download in advance to save time (or if you will not have an internet connection) please use `cellfinder_download`.

Currently, the only supported atlas is the Allen reference mouse brain, originally from here. The atlas will download into `./cellfinder/atlas`.

In the future, other atlases will be available, and you will be able to choose which is downloaded.

If you want to modify the cellfinder download, use:

- `--atlas-install-path` Supply a path to download the atlas elsewhere. This should also update the default `registration.conf` file so that the correct atlas is sourced. Alternatively, use this command to tell cellfinder where an existing atlas is, to save it being downloaded twice. (Requires 20GB disk space)
- `--atlas download path` The path to download the atlas into. (Requires 1.2GB disk space). Defaults to `/tmp`.

For developers

If you want to edit cellfinder, run tests and build the documentation etc, then use:

```
git clone --branch dev https://github.com/SainsburyWellcomeCentre/cellfinder
cd cellfinder
pip install -e .[dev]
```

2.2 Usage

```
    cellfinder -s /path/to/first/signal/channel.txt  /path/to/any/other/signal/
↪channels.txt  -b /path/to/background/channel.txt -o /path/to/output_directory -x␣
↪0.001 -y 0.001 -z 0.005
```

Arguments

Mandatory

- `-s` or `--signal-planes-paths` Path to the directory of the signal files. Can also be a text file pointing to the files. **There can be as many signal channels as you like, and each will be treated independently**.
- `b` or `--signal-planes-path` Path to the directory of the background files. Can also be a text file pointing to the files. **This background channel will be used for all signal channels**.
- `-o` or `--output-dir` Output directory for all intermediate and final results

Either

- `-x` or `--x-pixel-mm` Pixel spacing of the data in the first dimension, specified in mm.
- `-y` or `--y-pixel-mm` Pixel spacing of the data in the second dimension, specified in mm.
- `-z` or `--z-pixel-mm` Pixel spacing of the data in the third dimension, specified in mm.

Or

- `--metadata` Metadata file containing pixel sizes (any format supported by micrometa can be used). If both pixel sizes and metadata are provided, the command line arguments will take priority.

The following options can also be used:

Additional options

- `--register` Register the background channel to the Allen brain atlas
- `--summarise` Generate summary csv files showing how many cells are in each brain area (will also run registration if not specified.
- `--signal-channel-ids` Channel ID numbers, in the same order as
- `--figures` Generate figures

Only run parts of cellfinder

If for some reason you don't want some parts of cellfinder to run, you can use the following options. If a part of the pipeline is required by another part it will be run (i.e. `--no-detection` won't do anything unless `--no-classification` is also used). Cellfinder will attempt to work out what parts of the pipeline have allready been run (in a given output directory) and not run them again if appropriate.

- `--no-detection` Don't run cell candidate detection
- `--no-classification` Don't run cell classification

- `--no-standard_space` Dont convert cell positions to standard space. Otherwise will run automatically if registration and classification has run.

If you have channel numbers that you'd like to carry over into cellfinder

- `--signal-channel-ids`. Channel ID numbers, in the same order as 'signal-planes-paths. Will default to '0, 1, 2' etc, but maybe useful to specify.
- `--background-channel-id` Channel ID number, corresponding to 'background-planes-path'

Figures options

Figures cannot yet be customised much, but the current options are here:

- `--no-heatmap` Don't generate a heatmap of cell locations
- `--heatmap-bin` Heatmap bin size (mm of each edge of histogram cube)
- `--heatmap-smoothing` Gaussian smoothing sigma, in mm.
- `--no-outlines` Don't generate outlines of segmentation regions (for overlay)
- `--no-mask-figs` Don't mask the figures (removing any areas outside the brain, from e.g. smoothing)

Performance/debugging

- `--debug` Increase verbosity of statements printed to console and save all intermediate files.
- `--n-free-cpus` The number of CPU cores on the machine to leave unused by the program to spare resources.
- `--max-ram` Maximum amount of RAM to use (in GB) - **not currently fully implemented for all parts of cellfinder**

Input/output options

Useful for testing or if you know your cells are only in a specific region

- `--start-plane` The first plane to process in the Z dimension
- `--end-plane` The last plane to process in the Z dimension

Cell candidate detection

- `--save-planes` Whether to save the individual planes after processing and thresholding. Useful for debugging.
- `--outlier-keep` Dont remove putative cells that fall outside initial clusters
- `--artifact-keep` Save artifacts into the initial xml file
- `--max-cluster-size` Largest putative cell cluster where splitting should be attempted
- `--soma-diameter` The expected soma size in pixels in the x/y dimensions
- `--ball-xy-size` The size in pixels of the ball used for the morphological filter in the x/y dimensions
- `--ball-z-size` The size in pixels of the ball used for the morphological filter in the z dimension
- `--ball-overlap-fraction` The fraction of the ball that has to cover thresholded pixels for the centre pixel to be considered a nucleus pixel

Cell candidate classification

- `--trained-model` To use your own network (not the one supplied with cellfinder) specify the model file.
- `--model-weights` To use pretrained model weights. Ensure that this model matches the `--network-depth` parameter.
- `--batch-size` Batch size for classification. Can be adjusted depending on GPU memory

If the pixel sizes of the cubes used to train your network are different to that of the network supplied with cellfinder (0.001 x 0.001 x 0.005 mm), this can be set:

- `--x-pixel-network`
- `--y-pixel-network`
- `--z-pixel-network`

Registration to atlas

- `--registration-config` To supply your own, custom registration configuration file.

If the supplied data does not match the NifTI standard orientation (origin is the most ventral, posterior, left voxel), then the atlas can be flipped to match the input data:

- `--flip-x` Flip the sample brain along the first dimension for atlas registration
- `--flip-y` Flip the sample brain along the second dimension for atlas registration
- `--flip-z` Flip the sample brain along the third dimension for atlas registration

Input data definitions

- `--orientation` The orientation of the sample brain `coronal`, `saggital` or `horizontal`
- `--x-pixel-mm-network` The pixel size (in the first dimension) that the machine learning network was trained on. Set this to adjust the pixel sizes of the extracted cubes.
- `--y-pixel-mm-network` The pixel size (in the second dimension) that the machine learning network was trained on. Set this to adjust the pixel sizes of the extracted cubes.

Standard space options

- `--transform-all` Transform all cell positions (including artifacts).

Atlas specification

When cellfinder runs, the appropriate machine learning models and reference atlas will be downloaded (if not previously done so). If you would like to download in advance to save time (or if you will not have an internet connection) please use `cellfinder_download`.

Currently, the only supported atlas is the Allen reference mouse brain, originally from here. The atlas will download into `~/.cellfinder/atlas`.

In the future, other atlases will be available, and you will be able to choose which is downloaded.

If you want to modify the cellfinder download, use:

- `--atlas-install-path` Supply a path to download the atlas elsewhere. This should also update the default `registration.conf.custom` file so that the correct atlas is sourced. Alternatively, use this command to tell cellfinder where an existing atlas is, to save it being downloaded twice. (Requires 20GB disk space)
- `--atlas download path` The path to download the atlas into. (Requires 1.2GB disk space). Defaults to `/tmp`.

Further help

All cellfinder options can be found by using the `-h` flag

```
cellfinder -h
```

If you have any issues, common issues can be found here. If you're still having issues, get in touch, or even better raise an issue on Github.

2.3 Training

Cellfinder includes a pretrained network for cell candidate classification. This will likely need to be retrained for different applications. There are two choices:

- Continue training the existing network
- Start from scratch

Prerequsites

To train the network, you will need:

- Tiff cubes generated by running cellfinder with the `--no-classification` option or by using `cellfinder_gen_cubes`.
- (Optionally) an .xml file (as per cellfinder output) or a directory as per ROI sorter which defines which tiff cubes are `cell` or `no_cell`

Generating the training YAML file

To specify the datasets used for training, a YAML file must be generated. This file will have a single list named `data`, and each entry will have the following parameters:

- cube_dir: The path to the directory containing the tiff cubes
- cell_def: A file that defines which of the cubes in cube_dir are relevant. Can either be an cell counter xml file, or a directory with rois - as output when manually sorting/correcting a previous run (using ROI sorter) If all cubes should be used, set this parameter to empty string ("") or null (without quote marks).
- type: If this entry describes cells, set it to "cell", otherwise use "no_cell". Each entry must contain either ONLY cells or ONLY non_cells.
- signal_channel: The index or number of the signal channel. Note This number should match the one in the cube tif files!
- bg_channel: The index or number of the background channel. Note This number should match the one in the cube tif files! Set to -1 if no background channel is present.

e.g.:

```
data:
- {cube_dir: /path/to/cubes1,
   cell_def: /path/to/cubes1_cells.xml,
   type: "cell",
   signal_channel: 1,
   bg_channel: 2}
- {cube_dir: /path/to/cubes1,
   cell_def: /path/to/cubes1_no_cells.xml,
   type: "no_cell",
   signal_channel: 1,
   bg_channel: 2}
- {cube_dir: /path/to/cubes2,
   cell_def: /path/to/cubes2_cells_dir,
   type: "cell",
   signal_channel: 2,
   bg_channel: 0}
- {cube_dir: /path/to/cubes2,
   cell_def: /path/to/cubes2_no_cells_dir,
   type: "no_cell",
   signal_channel: 2,
   bg_channel: 0}
```

Start training

```
cellfinder_train -/path/to/training_yaml.yml  /path/to/output/directory/
```

Arguments

Positional

- The path to the yaml run file
- Output directory for the trained model (or model weights) results

The following options can also be used:

- `--continue-training` Continue training from an existing trained model. If no model or model weights are specified, this will continue from the included model.
- `--trained-model` Path to a trained model to continue training
- `--model-weights` Path to existing model weights to continue training
- `--network-depth` Resnet depth (based on He et al. (2015)). Choose from (18, 34, 50, 101 or 152). In theory, a deeper network should classify better, at the expense of a larger model, and longer training time. Default: 50
- `--batch-size` Batch size for training (how many cell candidates to process at once). Default: 16
- `--epochs` How many times to use each sample for training. Default: 1000
- `--test-fraction` What fraction of data to keep for validation. Default: 0.1
- `--no-augment` Do not use data augmentation
- `--save-weights` Only store the model weights, and not the full model. Useful to save storage space.
- `--tensorboard` Log to `output_directory/tensorboard`. Use `tensorboard --logdir outputdirectory/tensorboard` to view.

Further help

All cellfinder_train options can be found by using the `-h` flag

```
cellfinder_train -h
```

2.4 Troubleshooting

As cellfinder relies on a number of third party libraries (notably TensorFlow, CUDA and cuDNN) there may be issues while running the software.

Error messages

OSError: [Errno 24] Too many open files

```
OSError: [Errno 24] Too many open files
```

This is likely because your default limits are set too low (allthough this should probably be prevented from happening at all see here). To fix this, follow the instructions here. If for any reason you don't want to or can't change the system-wide limits, running `ulimit -n 60000` before running cellfinder should work. This setting will persist for the present shell session, but will have to repeated if you open a new terminal.

libcublas.so.9.0: cannot open shared object file: No such file or directory

```
ImportError: libcublas.so.9.0: cannot open shared object file: No such file or␣
↪directory
Failed to load the native TensorFlow runtime.
See https://www.tensorflow.org/install/errors
for some common reasons and solutions.  Include the entire stack trace
above this error message when asking for help.
```

If you see an error like this, it is likely that you don't have CUDA installed, or it isn't added to your path (so tensorflow can't find it). If you have a CUDA 10 version of tensorflow-gpu installed, you may instead see:

```
ImportError: libcublas.so.10.0: cannot open shared object file: No such file or␣
↪directory
```

If you're not sure which version of CUDA you need, please see here. If you don't have one of these newer GPUs, it is easier to use CUDA 9.

If CUDA and cuDNN is definately installed (following instructions here). Then please make sure that they are added to the path. If for example your version of CUDA is located at `/usr/local/cuda-10.0`, then run:

```
export PATH=/usr/local/cuda-10.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
```

To save doing this each time you run cellfinder, you can add these lines to e.g. your `~/.bashrc` file. If you have multiple versions of CUDA, or don't want to edit your `.bashrc` file, see here for how to only add to the path for specific conda environments.

INFO:tensorflow:Error reported to Coordinator: Failed to get convolution algorithm

```
INFO:tensorflow:global_step/sec: 0
2019-05-17 13:04:53.550306: E tensorflow/stream_executor/cuda/cuda_dnn.cc:334]␣
↪Could not create cudnn handle: CUDNN_STATUS_INTERNAL_ERROR
2019-05-17 13:04:53.565467: E tensorflow/stream_executor/cuda/cuda_dnn.cc:334]␣
↪Could not create cudnn handle: CUDNN_STATUS_INTERNAL_ERROR
INFO:tensorflow:Error reported to Coordinator: Failed to get convolution algorithm.
↪ This is probably because cuDNN failed to initialize, so try looking to see if a␣
↪warning log message was printed above.
```

If you see an error like this, it's likely to be one of two things, either your GPU memory is full, or an issue with your CUDA and cuDNN version.

Your GPU memory may be full if it is still being used by another process. To test this, run `nvidia-smi`, and you will see something like this:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.56       Driver Version: 418.56       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  TITAN RTX           On   | 00000000:2D:00.0  On |                  N/A |
| 41%   38C    P2    55W / 280W |  23408MiB / 24187MiB |      4%      Default |
+-------------------------------+----------------------+----------------------+
```

The bit to look for is the memory use (`23408MiB / 24187MiB`), is this is nearly full (like the example) then find the culprit:

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     37793      C   ...miniconda3/envs/cellfinder/bin/python 23408MiB   |
+-----------------------------------------------------------------------------+
```

In this case, a previous run of cellfinder hasn't completed. Either wait for it to run, or cancel it with CTRL+C (in the cellfinder terminal).

Alternatively, your version of CUDA and cuDNN may be not compatible with tensorflow 2.0. You can update them by following the instructions hereor by installing them into your conda environment:

```
conda install cudatoolkit cudnn
```

SyntaxError: invalid syntax (logging)

```
  File "/home/adam/projects/cellfinder/cellfinder/tools/tools.py", line 444
    logging.debug(f"Free memory is: {free_mem} bytes.")
                                                     ^
SyntaxError: invalid syntax
```

If you see an error like this, with the second line starting with something like `logging.debug(f"`, `logging.info(f"` or `print(f"`, then you likely have an unsupported (<3.6) version of python. Use conda or pip to install python 3.6.

Things that look like errors, but aren't:

Most things that are actually errors will interrupt cellfinder, and the program won't run. Other things will get logged with an `ERROR` or a `WARNING` flag and will get printed to the console in addition to the log file.

A number of third party modules may raise their own errors. As long as you understand what they mean, they can usually be safely ignored.

Can't find openCV

```
CRITICAL:tensorflow:Optional Python module cv2 not found, please install cv2 and␣
↪retry if the application fails.
```

Tensorflow thinks this is critical, it's not.

CPU instruction sets

```
tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions␣
↪that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
```

Unless you built tensorflow from source, something like this will come up. It'll still work fine

3 Additional tools

3.1 Two dimensional cell viewer

To visualise cells (or cell candidates) from an XML file overlaid onto a full-resolution image. Planes are loaded only when needed, so uses very little ram.

```
    cellfinder_view_cells /path/to/images /path/to/xmlfile.xml
```

3.2 Three dimensional viewer

3D image viewing using napari

Work in progress, for the time being, I recommend ClearVolume in ImageJ

Unlike the two dimensional cell viewer, the data must fit into RAM.

Usage

```
    cellfinder_view_3D -/path/to/image
```

Arguments

Mandatory

- `-i` or `--img-paths` Can be the path of a nifti file, tiff file, tiff files folder or text file containing a list of paths

3.3 Transforming cells into standard space

To transform cell positions from the coordinate space of the raw data to that of another coordinate framework (usually the atlas).

```
cellfinder_cell_standard --cells /path/to/classified_cells.xml --transformation /
↪path/to/control_point_file.nii --ref /path/to/reference_image.nii -o /path/to/
↪output/directory -x 0.002 -y 0.002 -z 0.005
```

Arguments

Mandatory

- `--cells` Path of the xml file containing cells to be transformed.
- `--transformation` Path to the control point file (from niftyreg or cellfinder registration)
- `--reference` Reference nii image, in the same space as the downsampled raw data
- `-o` or `--output-dir` Output directory

Either

- `-x` or `--x-pixel-mm` Pixel spacing of the data in the first dimension, specified in mm.
- `-y` or `--y-pixel-mm` Pixel spacing of the data in the second dimension, specified in mm.
- `-z` or `--z-pixel-mm` Pixel spacing of the data in the third dimension, specified in mm.

Or

- `--metadata` Metadata file containing pixel sizes (supported formats include BakingTray recipe files mesoSPIM metadata files or cellfinder custom metadata files. If both pixel sizes and metadata are provided, the command line arguments will take priority.

The following options can also be used:

- `--transform-all` Transform all cell positions (including artifacts).
- `----registration-config` To supply your own, custom registration configuration file.

Performance/debugging

- `-V` or `--verbose` Increase verbosity of statements printed to console (all debug information is saved to file regardless of this flag)
- `--n-free-cpus` The number of CPU cores on the machine to leave unused by the program to spare resources.

All other options (and their defaults) can be round by running:

`cellfinder_cell_standard -h`

3.4 Converting cells to brainrender format

Temporary tool until a Cellfinder -> BrainRender bridge is finalised

To convert cell positions (e.g. in `cells_in_standard_space.xml`) to a format that can be used in BrainRender.

```
cellfinder_cells_to_brainrender cells_in_standard_space.xml exported_cells.h5
```

Arguments

Run `cellfinder_cells_to_brainrender -h` to see all options.

Positional

- Cellfinder cells file to be converted
- Output filename. Should end with '.h5'

The following options may also need to be used:

- `-x` or `--x-pixel-size` Pixel spacing of the data that the cells are defined in, in the first dimension, specified in um. (Default: 10)
- `-y` or `--y-pixel-size` Pixel spacing of the data that the cells are defined in, in the second dimension, specified in um. (Default: 10)
- `-z` or `--z-pixel-size` Pixel spacing of the data that the cells are defined in, in the third dimension, specified in um. (Default: 10)
- `--max-z` Maximum z extent of the atlas, specified in um. (Default: 13200)
- `--hdf-key` HDF identifying key. If this has changed, it must be specified in the call to `BrainRender.scene.Scene.add_cells_from_file()`

To visualise this file in BrainRender

```
from BrainRender.scene import Scene
scene = Scene(jupyter=True)
scene.add_cells_from_file("exported_cells.h5")
scene.render()
```

3.5 Organise cells into atlas regions

To understand results of the cell classification, the cell coordinates from the `cell_classification.xml` file must be "assigned" to brain regions from the registration step.

```
cellfinder_count_summary --registered-atlas /path/to/registered_atlas.nii --
↪hemispheres /path/to/hemispheres.nii --xml /path/to/classified cells.xml --
↪structures-file /path/to/structures.csv --csv-out /path/to/csv_out.csv
```

Arguments

Mandatory

- `--registered-atlas` The path to the atlas registered to the sample brain
- `--hemispheres` The atlas with just the hemispheres encoded
- `--xml` The xml file containing the cell locations
- `----output-dir` Directory to write the results

The registered atlases and the xml will be read, and a csv showing how many cells are in each brain area (per hemisphere) is generated

3.6 Cube extraction

To extract tiff cubes for cell classification or training. Useful so you don't need to save thousands of cubes for future use.

```
    cellfinder_gen_cubes --cells /path/to/cells.xml -i /path/to/image/channel.txt /
↪path/to/any/other/image/channels.txt -o /path/to/output_directory -x x_pixel_in_
↪mm -y y_pixel_in_mm -z z_pixel_in_mm
```

Arguments

Mandatory

- `--cells` Path of the xml file (or Roi sorter output directory)containing the ROIs to be extracted.
- `-i` or `--img-paths` Path to the directory of the image files. Can also be a text file pointing to the files. **There can be as many channels as you like**.
- `-o` or `--output-dir` Directory to save the cubes into

Either

- `-x` or `--x-pixel-mm` Pixel spacing of the data in the first dimension, specified in mm.
- `-y` or `--y-pixel-mm` Pixel spacing of the data in the second dimension, specified in mm.
- `-z` or `--z-pixel-mm` Pixel spacing of the data in the third dimension, specified in mm.

Or

- `--metadata` Metadata file containing pixel sizes (supported formats include BakingTray recipe files mesoSPIM metadata files or cellfinder custom metadata files. If both pixel sizes and metadata are provided, the command line arguments will take priority.

Cube extraction generates a large number (100,000+) of small files. Reading and writing these on shared (e.g. network) storage can cause issues for other users. It is strongly recommended to set `--output-dir` as local (preferably fast) storage.

The following options can also be used:

Cube definitions

- `--cube-width` The width of the cubes to extract (must be even) [default 50]
- `--cube-height` The height of the cubes to extract (must be even) [default 50]
- `--cube-depth` The depth of the cubes to extract [default 20]

Performance/debugging

- `--debug` Debug mode. Will increase verbosity of logging and save all intermediate files for diagnosis of software issues.
- `--n-free-cpus` The number of CPU cores on the machine to leave unused by the program to spare resources.

3.7 Generate figures

N.B. A command line tool has not yet been developed. For now, figures can only be generated by using the `--figures` flag in `cellfinder_run`.

By default, if you use the `--figures` flag, a number of files will be generated. Until custom figures can be generated in cellfinder, these files are generated to allow you to more easily make figures in e.g. FIJI. More details can be found here.

The idea is that these images can be combined to generate more custom figures. These images are all saved in the same coordinate space as the downsampled raw data saved into `output_directory/registration`. Currently three files will be generated by default.

- **heatmap.nii** - This is a 3D histogram, showing cell densities throughout the brain. Likely to be more useful than visualising cell positions for dense areas. By default, this is smoothed, and masked (so that the smoothing doesn't "bleed" outside of the brain)
- **glass_brain.nii** This is a binary image, just showing the brain surface. Plotting this in 3D along with **heatmap.nii** gives a nice sense of cell density throughout the brain. ClearVolume works well for this.
- **boundaries.nii** Another binary image, but showing the divisions between each region in the atlas. More useful for smaller, 3D figures to delineate region boundaries. Also useful to assess registration by overlaying on the raw, downsampled data.

There are not yet many options, but the current options are here:

- `--no-heatmap` Don't generate a heatmap of cell locations
- `--heatmap-bin` Heatmap bin size (mm of each edge of histogram cube)
- `--heatmap-smoothing` Gaussian smoothing sigma, in mm.
- `--no-outlines` Don't generate outlines of segmentation regions (for overlay)
- `--no-mask-figs` Don't mask the figures (removing any areas outside the brain, from e.g. smoothing)

3.8 Region image generation

Generates an image of specified Allen institute brain regions (either solid, or hollow).

```
cellfinder_gen_region_vol -s "Name of Structure 1" "Name of Structure 2" -a /path/
↪to/atlas/annotations.nii -o output.nii
```

Arguments

Run `cellfinder_gen_region_vol -h` to see all options.

- `-s` or `--structure-names` Structure names as a series of string (as per the reference atlas). If these are higher level structures (e.g. "Retrosplenial area"), it will include all subregions. If multiple structures are provided the final image will be the combination of all of them.
- `-a` or `--atlas` Path to the atlas annotations. The resulting image will be in the same coordinate space
- `-o` or `--output` Output name for the resulting `.nii` file

The following options may also need to be used:

- `--glass` Generate a hollow volume
- `--atlas-config` To supply your own, custom atlas configuration file

3.9 Summarise cell counts from multiple brains

To summarise and organise the results from `cellfinder_cells_into_region` (which is a list of every brain region and the number of cells) you can use `cellfinder_region_summary`.

This takes a directory of csv's output from `cellfinder_cells_into_region` and generates summary versions in which:

- Brain regions are organised in the same way for each csv file
- (Optionally, only the brain regions specified by `--regions-list` and/or `--regions`)

```
cellfinder_region_summary /path/to/directory/with/csvs
```

Arguments

Mandatory

- Directory containing csv files to be summarised

The following options can also be used:

- `--regions-list` A text file listing a subset of brain regions to be included in the summary csv file (as per the allen brain atlas csv)
- `--sum-regions` Rather than finding all child regions, sum them.
- `--regions` A list of additional regions to include, e.g:

```
--regions "Anterior cingulate area" "Retrosplenial area"
```

- `--structures-file` The csv file containing the structures definition (if not using the default Allen brain atlas) **[NOT YET IMPLEMENTED]**

N.B. The regions are hierarchical, so if you specify `Retrosplenial area, dorsal part`, you will get all subregions:

- Retrosplenial area, dorsal part
- Retrosplenial area, dorsal part, layer 1
- Retrosplenial area, dorsal part, layer 2/3
- Retrosplenial area, dorsal part, layer 4
- Retrosplenial area, dorsal part, layer 5
- Retrosplenial area, dorsal part, layer 6a
- Retrosplenial area, dorsal part, layer 6b

3.10 ROI transformation

Transforms ROI positions from ImageJ/FIJI into standard space.

When given an ImageJ ROI collection `.zip` files an a cellfinder registration output directory, an image of the ROIs in standard space will be generated.

```
cellfinder_roi_transform /path/to/rois.zip /path/to/registration/directory
```

Arguments

Run `cellfinder_roi_transform -h` to see all options.

Positional

- ImageJ/FIJI ROI .zip collection.
- Cellfinder registration directory containing the downsampled image that the ROIs were drawn on

The following options may also need to be used:

- `-r` or `--reference` If the ROIs are not defined on the downsampled image in the `registration` directory (i.e. on the raw, non-downsampled data), provide the path to this image here (e.g. the directory with a series of tiff files.)
- `-o` or `--output` Output filename. If not provided, will default to saving in the same directory as the ROIs
- `--registration-config` To supply your own, custom registration configuration file.
- `--selem` Size of the structuring element used to clean up the ROI image. Default - 10. Increase if the ROI looks "fragmented", decrease if it looks too "rounded".
- `--zfill` Increase this number to fill in gaps in your ROIs in Z (in case you drew your ROIs on downsampled data). Default: 2.
- `--debug` Debug mode. Save all intermediate files for diagnosis of software issues.

3.11 XML file cropping

To "crop" an xml file, i.e. only include cell coordinates corresponding to given anatomical areas.

```
    cellfinder_xml_crop --xml-dir xml_directory --structures-file chosen_
↪structures.csv --registered-atlas registered_atlas.nii --hemispheres hemispheres.
↪nii
```

Arguments

Run `cellfinder_xml_crop -h` to see all options.

Mandatory

- `--xml-dir` Directory containing xml files to be cropped
- `--structures-file` Curated csv structure list (as per the allen brain atlas csv
- `--registered-atlas` The path to the atlas registered to the sample brain
- `--hemispheres` The atlas with just the hemispheres encoded

The following options may also need to be used:

- `--hemisphere-query` Which hemisphere to keep (1 or 2). Default: 0 (all)

Either

- `-x` or `--x-pixel-mm` Pixel spacing of the data in the first dimension, specified in mm.
- `-y` or `--y-pixel-mm` Pixel spacing of the data in the second dimension, specified in mm.
- `-z` or `--z-pixel-mm` Pixel spacing of the data in the third dimension, specified in mm.

Or

- `--metadata` Metadata file containing pixel sizes (supported formats include BakingTray recipe files mesoSPIM metadata files or cellfinder custom metadata files. If both pixel sizes and metadata are provided, the command line arguments will take priority.

3.12 XML file rescaling

To rescale the cell positions within an XML file. For compatibility with other software, or if your data has been scaled after cell detection.

```
cellfinder_xml_scale /path/to.xml -x x_scale -y y_scale -z z_scale
```

Arguments

Run `cellfinder_xml_scale -h` to see all options.

Optional arguments

Scales

If not set, will default to 1 (i.e. no scaling). Can be any float (e.g. 0.5 or 2)

- `-x` or `--x-scale` Rescaling factor in the first dimension
- `-y` or `--y-scale` Rescaling factor in the second dimension
- `-z` or `--z-scale` Rescaling factor in the third dimension

Others

- `-o` or `--output` Output directory for the scaled xml file. Defaults to the same directory as the input file.

4 About

4.1 Releases

Version 0.3.3 (TBC)

Version 0.3.2a (2020-01-10)

Main updates

- Very basic Windows support (allthough some issues remain.)
- Pixel sizes are now defined in um (rather than mm). Users specifying a metadata file will not be affected. Those specifying the pixel sizes manually will need to change their usage. Flags such as `--x-pixel-mm` are now `--x-pixel-um`.
- All cell detection parameters (e.g. `--soma-diameter`) are now defined in um or um3 (rather than mm)
- Registration parameters are now defined via command-line arguments (e.g. `--freeform-use-n-steps`) rather than a config file.
- Installation is simplified (no need to install tensorflow separately)

Changed

- Updated default parameters to better work on noisy data.

Fixed

- Memory consumption during inference reduced

Developers

- Registration code is now in amap
- Compatibility with tensorflow 2.1.0

Version 0.3.1a (2019-10-31)

Main updates

- Cube extraction is now carried out on the fly during classification (and not saved to disk). This prevents file system issues due to the generation of (potentially) hundreds of thousands of cells.

Changed

- Likelihood during training of each individual augmentation transformation reduced to 10% (from 50%)

Fixed

- Bug fix in `cellfinder_view_cells` to deal with Napari api change

Developers

- New viewer to inspect batches of generated cubes (`cellfinder.classify.tools.batch_viewer`)
- Removed some old, unecessary tests
- Fixed a bug in `tests/tests/test_integration/test_detection.py`

Version 0.3.0a (2019-10-29)

Main updates

- Removed NiftyNet dependency. Classification is now implemented in `tf.keras`. **THIS MEANS YOU NEED TO TRAIN A NEW MODEL. OLD MODELS ARE INCOMPATIBLE!**

Added

- New `cellfinder_view_cells` command which uses napari view cells overlaid on raw data.
- New `cellfinder_gen_region_vol` command line tool to generate images of specific brain regions (for figures etc)
- `roi_transform` now supports ROIs generated in the original, full-size images
- New `cellfinder_cells_to_brainrender` to allow classified cells to be viewed in BrainRender

Removed

`cellfinder_view_cells2D` function is removed, and replaced with `cellfinder_view_cells`.

Changed

- Cube scaling is now 3D (theoretically leading to better classification performance)
- `cellfinder_view_3D` now uses napari.

Developers

- Brain image IO moved to brainio
- Reading a slurm environment (if present) moved to slurmio.
- Logging moved to fancylog

Version 0.2.12a (2019-09-24)

Added `roi_transform` command line tool to transform ImageJ ROI sets into images in standard space.

Version 0.2.11a (2019-09-23)

- Fixes a bug causing heatmaps to be scaled incorrectly

Version 0.2.10a (2019-09-20)

Only minor changes to documentation and testing.

Version 0.2.9a (2019-09-20)

Added

- New `--max-ram` to specify a maximum amount of RAM to use (in GB). Currently only affects processes that will scale to use as much RAM as is available (such as cube extraction).
- New `cellfinder_xml_scale` command line tool to rescale the cell positions within an XML file. For compatibility with other software, or if your data has been scaled after cell detection.

Version 0.2.8a (2019-09-17)

THIS VERSION HAS NOT BEEN FULLY TESTED

Added

- Instructions for installing CUDA 9 and cuDNN via conda added.
- `cellfinder_region_summary` now has a `--sum-regions` flag combine child regions.

Fixed

- Bug which caused heatmaps to be calculated incorrectly is fixed.
- Bug which causes `--regions-list` input to `cellfinder_region_summary` to be parsed incorrectly is fixed.
- Bug which crashed summary csv generation if a cell is transformed incorrectly due to rounding errors is fixed.
- Bug which crashed volume calculation in registration if a region is missing is fixed.
- Bug causing registration to crash if a custom registration file without a `base_folder` entry is fixed.
- `urllib3` version is specified to prevent atlas downloading from failing.

Version 0.2.7a (2019-09-05)

Fixed

- A bug which caused the atlas not be downloaded when using `cellfinder_register` is fixed.
- Registration now respects the `--n-free-cpus` flag

Version 0.2.6a (2019-09-05)

Main updates

- Python 3.5 is no longer supported.

- Detected cell positions can now be transformed into standard space. By default, if cell classification and registration has been performed, an additional `cells_in_standard_space.xml` file will be generated. Use `--no-standard_space` to prevent this behaviour. This functionality can also be run using the standalone script `cellfinder_cell_standard`.

Added

- Pixel sizes must still be specified, but instead of using the `-x`, `-y`, and `-z` flags, a `--metadata` flag can be used instead. Supported formats include BakingTray recipe files mesoSPIM metadata files or cellfinder custom metadata files. If both pixel sizes and metadata are provided, the command line arguments will take priority.

Fixed

- An installation bug (due to new requirements of gputools is fixed.)

Deprecations

- The main command-line interface to cellfinder is now `cellfinder`, `cellfinder_run` has been removed (and may call an old version of the software). Please use `cellfinder` from now on.

Version 0.2.5a (2019-07-31)

Fixed

- Bug in `cellfinder_gen_cubes` has been fixed

Deprecations

- The main command-line interface to cellfinder will be `cellfinder`, `cellfinder_run` will stop working soon (and may call an old version of the software). Please use `cellfinder` from now on.

Version 0.2.4a (2019-07-30)

THIS VERSION HAS NOT BEEN FULLY TESTED Use Version 0.2.3a if you don't need the new features

Main Updates

- Pixel sizes must now be specified with `-x`, `-y`, and `-z`.

Added

- `cellfinder.IO.cells.get_cells()` can now read the `.yml` files output from the cell counting plugin of MaSIV.
- Registration will now calculate a `volumes.csv` file which contains the volumes of each brain area in the atlas, in the sample brain.
- The `--summarise` flag will now save more information, including:
  - Cell counts combined across hemispheres

    - Cell densities in cells per mm3.

- Registration will also calculate the reverse transforms (i.e. sample -> atlas)
- A new `--figures` flag will create a subdirectory of 3D images in the coordinate space of the downsampled raw data (and registered atlas) that can be used to generate figures e.g. in FIJI. Currently three files will be generated by default.
    - **heatmap.nii** - This is a 3D histogram, showing cell densities throughout the brain. Likely to be more useful than visualising cell positions for dense areas. By default, this is smoothed, and masked (so that the smoothing doesn't "bleed" outside of the brain).
    - **glass_brain.nii** This is a binary image, just showing the brain surface. Plotting this in 3D along with **heatmap.nii** gives a nice sense of cell density throughout the brain. ClearVolume works well for this.
    - **boundaries.nii** Another binary image, but showing the divisions between each region in the atlas. More useful for smaller, 3D figures to delineate region boundaries. Also useful to assess registration by overlaying on the raw, downsampled data.

Fixed

- Bug on certain systems where cube extraction could use too much RAM, and crash the system has been fixed.

Changed

- Output directory structure has been reorganised, and simplified slightly.

Developers

- Code coverage increased to 58%.
- A lot of the code has been refactored, and (hopefully simplified).
- Log files will now print more information (e.g. information about the git repository, and internal parameters).

Version 0.2.3a (2019-07-01)

Main Updates

- Registration can now handle data in any orientation. The default is coronal, and in the NifTI standard orientation (origin is the most ventral, posterior, left voxel). If your data does not match this, use the `--orientation` , `--flip-x`, `--flip-y` and `--flip-z` tags.

Added

- There is now a standalone registration command, `cellfinder_register`. This will be the maintained python version of aMAP going forward.
- There is now an option `--remove-intermediate` to remove all intermediate files generated by cellfinder. Use with caution.

Fixed

- All downsampled data, registered atlases and cell locations are in the same coordinate space as the raw data (although possibly scaled). This fixes issues #22 and #23, and allows overlay of the cell positions on the downsampled data from the registration step.

Changed

- Cube extraction will now use as many CPU cores as available, slightly speeding up this step. However, if you don't have much RAM (or a lot of CPU cores) see #36.
- Input data paths no longer need to end in the channel number. For channel referencing purposes, this will be set automatically, or can be overridden with `--signal-channel-ids` and `--background-channel-id` (or `--channel-ids` for `cellfinder_gen_cubes`).
- By default, registration will now save downsampled versions of all channels. if you don't want these files, use `--no-save-downsampled`.
- Cube extraction will now default to extracting cubes of 50um x 50um x 100um. This can be overridden with `--x_pixel_mm_network` and `--y_pixel_mm_network`. Currently there is no rescaling in z.
- Cube extraction now supports the output of ROI sorter, rather than just xml.

Developers

- Code coverage increased to 32%.
- Logging now includes diagnostic information.

Version 0.2.2a (2019-06-17)

Main Updates

- Cellfinder can now be installed with a single command:

```
pip install git+https://github.com/adamltyson/cellfinder
```

- The atlas will now no longer be downloaded automatically upon installation of cellfinder, but on the first use of the atlas. To download in advance, `cellfinder_download` can be used.

Added

- New function to load `.nrrd` files (in `cellfinder.tools.brain.io.load_any`).
- Data can now be passed as a directory of 2D .tif files (rather than just a text file of file paths).
- If a network was trained with a cubes of a different pixel size to that of the input data, then the pixel spacing of the output cubes can be changed (e.g. using: `--x-pixel-mm-network`).
- Cell (and candidate) positions can be saved as `.csv` (as well as `.xml`), by using `--save-csv`.
- `cellfinder_xml_crop` has been added. This curates an input `.xml` file and outputs a file with only those positions within given anatomical areas. Currently, the cells, and the atlas need to be in the same anatomical space.

Fixed

- Bug which caused cellfinder to hang when using part of a node managed by SLURM is fixed.
- Bug fixed which prevented cellfinder_run from progressing if a cube directory was present, but empty. Thanks to ablot.
- Bug fixed which prevented the `checkpoint` file from a previously trained network being copied to the output directory. Thanks to ablot.

Changed

- If a cube cannot be extracted for any reason (such as proximity to the edge of an image), the default is now to not save it (rather than save an empty cube in it's place). This can be overridden with `--save-empty-cubes`.
- Error messages when a cube cannot be extracted (due to proximity to the edge of the image) have been clarified.

Developers

- Code is now automatically formatted with Black, which requires Python >= 3.6 to run.
- Testing has been added (although currently there is only one test).
- All commits & pull requests will be build by Travis.
- For development, cellfinder should be installed using `pip install -e .[dev]` in the repository.

Version 0.2.1a (2019-05-09)

Main Updates

- Simple summary information can be generated with the `--summarise` flag. This will run registration (if it wasn't run allready) and associate each cell output from the cell detection step with a brain region. This info is saved as a csv file.
- The mouse brain atlas can be downloaded to any directory when installing cellfinder using the `--atlas-install-path` flag. This flag can also be used to point to an existing atlas installation (e.g. from aMAP). This will update the default config file so the correct atlas is sourced (and a second isn't downloaded.)
- Cellfinder *should* know what parts have allready been run, so if you re-run a `cellfinder_run` command, it won't repeat any parts of the pipeline.

Added

- PDF documentation
- `cellfinder_gen_cubes` function to generate tiff cubes from an xml file independent of the rest of cellfinder. Useful for generating training data.
- `cellfinder_count_summary` function to combine a brain registered to the allen atlas with cell counts. Generates a csv file of cells per region.
- `cellfinder_region_summary` function to organise (align by brain area) csv files of summary cell counts from multiple brains.
- `cellfinder_view_cells2D` function to view cells overlaid on raw data. Uses hardly any RAM, but I recommend ROI sorter instead.

- `cellfinder_view_3D` function. A very rudimentary 3D viewer, but can load any file type in use by cellfinder.
- Some API docs

Fixed

- Logging that interfered with multiprocessing when the `--verbose` tag was used has been removed. This was an issue in case cellfinder was stopped in the middle of cell classification as it could cause GPU memory to not be released (and not show up in `nvidia-smi`).

Changed

- Documentation moved from pydocmd to sphinx.
- Installation requires configobj (as well as Cython).
- Install with `--dev` if you want to build the documentation yourself.

Version 0.2.0a (2019-04-30)

Main Updates

- Can run cell detection on N-channels
- Streamlined training of cell classification network via `cellfinder_train`
- Includes (rudimentary) integration of aMAP for registration to the Allen brain atlas

Added

- Further options to only run parts of the analysis
- Option `--tensorboard` to launch tensorboard automatically during training
- Entry point `cellfinder_view` to launch the (very simple) image/cell viewer
- Docs generated via pydocmd

Fixed

- Output of cell candidate detection step `cells.xml` can now be opened in Roi Sorter

Changed

- Docs generated by pydocmd means that installation info is currently here

Version 0.1.0a (2019-04-12)

Main Updates

First version compatible with NiftyNet 0.5.0, TensorFlow 1.12.0 and therefore CUDA 9/10.

Added

- Option to specify external configuration file for cell candidate classification
- All results saved to a single directory
- Option to keep or discard bright spots classified as artifact during cell candidate detection
- Log file

Fixed

- Z positions of cell candidates when analysing a z-subset of the data fixed in the resulting .xml file
- Cell candidate detection .xml file only provides positive candidate types `<Type>1</Type>` for compatibility with Roi Sorter

Changed

- No longer relies on NiftyNet model zoo (although this may return)
- Much simplified installation (no manual cp/mv)
- Moving towards a single API, calls to n_count_python etc. broken
- Multiprocessing compatible with shared resources (e.g. HPC job schedulers)

Notes

- Developed with Python 3.5, but should work with 3.6/3.7. no Python 2 compatibility
- HPC compatibility only tested with SLURM, but should work with LSF & SGE

Version 0.0.1a (2019-01-23)

Final version compatible with niftynet 0.2.2 and tensorflow 1.4.1 (and therefore CUDA 8).

4.2 Future Plans

General

- Graphical user interface
- Visualisation options
- Interface for training data generation/curation
- Pass config options on command line

Development

- Test coverage

5 Developers

5.1 Developer notes

Code organisation

File paths

All file paths should be defined in `cellfinder.tools.prep.Paths`. Any intermediate file paths, (i.e. those which are not of interest to the typical end user) should be prefixed with `tmp__`. These should then be cleaned up as soon as possible after generation.

Conventions

Cellfinder has recently (2019-08-02) dropped support for Python 3.5. Following this, a number of new python features will be adopted throughout.

- pathlib conventions (rather then `os.path`).
- f-strings (rather than `.format()` or using the old `%` operator).

In all cases, please aim to replace old calls to `os.path` or `.format()` with pathlib object methods and f-strings.

Travis

All commits & pull requests will be build by Travis. To ensure there are no issues, please check that it builds: `pip install .` and run all of the tests: `pytest` before commiting changes. Some tests may take a long time (e.g. those requiring tensorflow if you don't have a GPU). These tests should be marked with `@pytest.mark.slow`, e.g.:

```python
import pytest
@pytest.mark.slow
def test_something_slow():
    slow_result = run_slow_processes()
    assert slow_result == expected_slow_thing
```

During development, these "slow" tests can be skipped by running `pytest -m "not slow"`.

Formatting

To ensure a consistent code style, please use Black before commiting changes. Please use the syntax: `black ./ -l 79 --target-version py36`

Documentation

Documentation is built using Sphinx with Markdown (rather than reStructuredText). Please edit (or create new) `.md` files in the appropriate directory in `cellfinder/doc_build`. The organisation of these files is then defined in `index.rst`.

Please ensure that:

- Any new changes are added to the release notes (`doc_build/main/about/release_notes.md`)
- All command-line functions are fully documented, including an explanation of all arguments, and example syntax.

To build the documentation (assuming you installed cellfinder with `pip install -e .[dev]` to install the dependencies):

```
cd doc_build
make html
```

Prior to commmiting to master, ensure that contents of `doc_build/_build/html/` is copied to `cellfinder/docs` for hosting with github pages.

This can be done automatically with a pre-commit hook. An example is here.

Dependencies

Any dependencies in the `master` branch will be checked by dependabot, and a pull request will be generated to update which are outdated. Please only merge these if all tests are passing, and you are confident that there will be no issues.

Misc

Configuration files

Any configuration files that cellfinder edits with local information (e.g. atlas installation location) should be in the `.gitignore` file. Please ensure that any custom files are not commited to the repository.

6 Misc

6.1 Cellfinder installation in a cluster computing environment.

Currently cellfinder has only been written with SLURM in mind. In theory, it should be easy enough to allow the use of any other job scheduler.

SLURM

Based on the SWC SLURM cluster, and so most of the command syntax will likely vary. Specifically, you are unlikely to have modules configured in exactly the same way as us.

Prepare the environment

- On our cluster, modules are only available on a compute node, so start an interactive job on a GPU node, and request a GPU for testing.

```
srun -p gpu --gres=gpu:1 --pty bash
```

- Load miniconda

```
module load miniconda
```

Set up conda environment and install cellfinder

- Now you can proceed as with a local installation
  - Create and activate new minimal conda environment

    ```
    conda create --name cellfinder python=3.7
    conda activate cellfinder
    ```

  - Install CUDA and cuDNN

```
conda install cudatoolkit=10.1 cudnn
```

- Install cellfinder

```
pip install cellfinder
```

- Check that tensorflow and CUDA are configured properly:

```
python
```

```
import tensorflow as tf
tf.test.is_gpu_available()
```

If you see something like this, then all is well.

```
   2019-06-26 10:51:34.697900: I tensorflow/core/platform/cpu_feature_guard.
↪cc:141] Your CPU supports instructions that this TensorFlow binary was not␣
↪compiled to use: AVX512F
   2019-06-26 10:51:34.881183: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1432] Found device 0 with properties:
   name: TITAN RTX major: 7 minor: 5 memoryClockRate(GHz): 1.77
   pciBusID: 0000:2d:00.0
   totalMemory: 23.62GiB freeMemory: 504.25MiB
   2019-06-26 10:51:34.881217: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1511] Adding visible gpu devices: 0
   2019-06-26 10:51:35.251465: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:982] Device interconnect StreamExecutor with strength 1 edge␣
↪matrix:
   2019-06-26 10:51:35.251505: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:988]      0
   2019-06-26 10:51:35.251511: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1001] 0:   N
   2019-06-26 10:51:35.251729: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1115] Created TensorFlow device (/device:GPU:0 with 195 MB memory)␣
↪-> physical GPU (device: 0, name: TITAN RTX, pci bus id: 0000:2d:00.0,␣
↪compute capability: 7.5)
   True
```

- **End your interactive job**

```
exit
```

Run cellfinder

Allthough you can run cellfinder interactively, it is better to submit a batch job.

- Write the job submission script. An example can be found here. If possible, set the output directory to local, fast scratch storage.
- Submit the job to the job scheduler

```
sbatch cellfinder_sbatch.sh
```

- If you use the example script, you will recieve an email when the job is done. To watch the progress, log onto a node with the same storage drive mounted and run:

```
watch tail -n 100 /path/to/cellfinder_log.log
```

- Copy the results from the storage platform.