suite2p

Release 0.7.2

Carsen Stringer, Marius Pachitariu

May 14, 2023

BASICS:

suite2p is an imaging processing pipeline written in Python 3 which includes the following modules:

- Registration
- Cell detection
- Spike detection
- Visualization GUI

For examples of how the output looks and how the GUI works, check out this twitter thread.

This code was written by Carsen Stringer and Marius Pachitariu. For support, please open an issue.

The reference paper is here. The deconvolution algorithm is based on this paper, with settings based on this paper.

We make pip installable releases of suite2p, here is the pypi. You can install it as `pip install suite2p`

- modindex
- search
- genindex

CHAPTER

ONE

INSTALLATION

Please refer to the suite2p README for the latest up-to-date installation instructions.

Common issues

- If when running `suite2p`, you receive the error: `No module named PyQt5.sip`, then try uninstalling and reinstalling pyqt5

```
pip uninstall pyqt5 pyqt5-tools
pip install suite2p
```

- If when running `suite2p`, you receive an error associated with **matplotlib**, try upgrading it:

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

- If you are on Yosemite Mac OS, PyQt doesn't work, and you won't be able to install suite2p. More recent versions of Mac OS are fine.
- If you are using Ubuntu 22.04 and run into the following issue:

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in even though it was␣
↪found.
This application failed to start because no Qt platform plugin could be initialized.
↪ Reinstalling the application
may fix this problem.
```

  Follow this link to install Qt5 and the issue above should be fixed.

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. The registration step runs faster on Ubuntu than Windows, so if you have a choice we recommend using the Ubuntu OS.

1.1 Dependencies

- rastermap
- pyqtgraph
- PyQt5
- numpy (>=1.13.0)
- scipy
- h5py

- scikit-learn
- scanimage-tiff-reader
- tifffile
- natsort
- matplotlib (not for plotting (only using hsv_to_rgb and colormap function), should not conflict with PyQt5)

CHAPTER

TWO

INPUTS

2.1 Input format

This applies to all file types!

2.1.1 Directory structure

suite2p looks for all tiffs/hdf5 in the folders listed in `ops['data_path']`. If you want suite2p to look in those folders AND all their children folders, set `ops['look_one_level_down']=True`. If you want suite2p to only look at some of the folder's children, then set `ops['subfolders']` to those folder names.

If you want suite2p to only use specific tiffs in ONE folder, then set the data path to only have one folder (`ops['data_path']=['my_folder_path']`), and name the tiffs you want processed in `ops['tiff_list']`.

See examples in this notebook.

2.1.2 Frame ordering

If you have data with multiple planes and/or multiple channels, suite2p expects the frames to be interleaved, e.g.

- frame0 = time0_plane0_channel1
- frame1 = time0_plane0_channel2
- frame2 = time0_plane1_channel1
- frame3 = time0_plane1_channel2
- frame4 = time1_plane0_channel1
- frame5 = time1_plane0_channel2
- …

channels are ones-based (channel 1 and 2 NOT 0 and 1).

2.1.3 Recordings with photostim / other artifacts

Photostim and other artifacts require you to exclude these frames during ROI detection. Otherwise there will be "ROIs" that are related to the stimulation, not actually cells. To exclude them, make an array of integers corresponding to the frame times of the photostimulation. Save this array into a numpy array called `bad_frames.npy`:

```
import numpy as np

bad_frames = np.array([20,30,40])
np.save('bad_frames.npy', bad_frames)
```

Put this file into the first folder in your ops['data_path'] (the first folder you choose in the GUI).

2.2 Different file types

2.2.1 Tiffs

Most tiffs should work out of the box. suite2p relies on two external tiff readers: scanimage-tiff-reader and sklearn.external.tifffile. The default is the scanimage one, but it will use the other one if it errors.

You can use single-page tiffs. These will work out of the box if they end in *.tif or *.tiff. If they have a different ending then use the flag `ops['all_files_are_tiffs'] = True` and the pipeline will assume any files in your folders are tiffs. NOTE that these will be slower to load in and create the binary, so if you're planning on using the pipeline extensively you may want to change your acquisition output.

If you save a stack of tiffs using ImageJ, and it's larger than 4GB, then it won't run through suite2p anymore. A workaround is to save as an OME-TIFF in FIJI: "File->save as->OME-TIFF->compression type uncompressed" in FIJI (thanks @kylemxm! see issue here).

If you have old Scanimage tiffs (version <5) that are larger than 2GB, then most tiff readers will not work. @elhananby has recommended this repository for reading the data into matlab (see issue **`here`_**). After reading it into matlab, you can re-save the tiff in a format that imageJ and suite2p can recognize (see matlab tiff writing here).

2.2.2 Bruker

Using Bruker Prairie View system, .RAW files are batch converted to single .ome.tifs. Now, you can load the resulting multiple tif files (i.e. one per frame per channel) to suite2p to be converted to binary. This looks for files containing 'Ch1', and will assume all additional files are 'Ch2'. Select "input_format" as "bruker" in the drop down menu in the GUI or set `ops['input_format'] = "bruker"`.

2.2.3 Mesoscope tiffs

We have a matlab script here for extracting the parameters from scanimage tiffs collected from the Thorlabs mesoscope. The script creates an `ops.json` file that you can then load into the run GUI using the button "load ops file". This should populate the run GUI with the appropriate parameters. Behind the scenes there are `ops['lines']` loaded and `ops['dy'],ops['dx']` that specify which lines in the tiff correspond to each ROI and where in space each ROI is respectively. `ops['nplanes']` will only be greater than 1 if you collected in multi-plane mode. Once the pipeline starts running, this parameter will change to "nplanes * nrois" and each "plane" is now an ROI from a specific plane. Please open issues if you're using this and having trouble because it's not straightforward.

2.2.4 Thorlabs raw files

Christoph Schmidt-Hieber (@neurodroid) has written haussmeister which can load and convert ThorLabs *.raw files to suite2p binary files! suite2p will automatically use this if you have pip installed it (`pip install haussmeister`).

2.2.5 HDF5 files (and *.sbx)

These should work out of the box, but are less well-tested. Dario Ringach has a utility to convert neurolabware *.sbx files to *.h5 files (see blog post here).

The H5 loading from the GUI now works the same as it always has for tiffs. Select "h5" from the drop-down menu and input the h5 KEY for the data as a string. Now choose the folder with your *.h5 or *.hdf5 files and the pipeline will use all h5 files in that folder. You can use ops['look_one_level_down'] to process all subfolders of the data_path.

2.2.6 sbx binary files

Scanbox binary files (*.sbx) work out of the box if you set `ops['input_format'] = "sbx"`.

When recording in bidirectional mode some columns might have every other line saturated; to trim these during loading set `ops['sbx_ndeadcols']`. Set this option to `-1` to let suite2p compute the number of columns automatically, a positive integer to specify the number of columns to trim. Joao Couto (@jcouto) wrote the binary sbx parser.

2.2.7 Nikon nd2 files

Suite2p reads nd2 files using the nd2 package and returns a numpy array representing the data with a minimum of two dimensions (Height, Width). The data can also have additional dimensions for Time, Depth, and Channel. If any dimensions are missing, Suite2p adds them in the order of Time, Depth, Channel, Height, and Width, resulting in a 5-dimensional array. To use Suite2p with nd2 files, simply set `ops['input_format'] = "nd2".`

2.3 BinaryRWFile

The `BinaryRWFile` is a special class in suite2p that is used to read/write imaging data and acts like a Numpy Array. Inputs of any format listed above will be converted into a `BinaryRWFile` before being passed in through the suite2p pipeline. An input file can easily be changed to a `BinaryRWFile` in the following way:

```
import suite2p

fname = "gt1.tif" # Let's say input is of shape (4200, 325, 556)
Lx, Ly = 556, 326 # Lx and Ly are the x and y dimensions of the imaging input
# Read in our input tif and convert it to a BinaryRWFile
f_input = suite2p.io.BinaryRWFile(Ly=Ly, Lx=Lx, filename=fname)
```

`BinaryRWFile` can work with any of the input formats above. For instance, if you'd like to convert an input binary file, you can do the following:

```
# Read in an input binary file and convert it to a BinaryRWFile
f_input2 = suite2p.io.BinaryRWFile(Ly=Ly, Lx=Lx, filename='gt1.bin')
```

Elements of these `BinaryRWFile` instances can be accessed similar to how one would access a Numpy Array.

```
f_input.shape # returns shape of your input (num_frames, Ly, Lx)
f_input[0] # returns the first frame with shape (Ly, Lx)
```

Also, `BinaryRWFile` instances can be directly passed to the several wrapper functions `suite2p` offers (e.g., `suite2p.detection_wrapper`, `suite2p.extraction_wrapper`, etc.). These wrapper functions can also directly work with Numpy arrays so feel free to pass them as inputs. If you'd like to run only specific modules, you will have to use the `BinaryRWFile` class. For example, this is how you can run the detection module on an input file that has already been registered.

```
f_reg = suite2p.io.BinaryRWFile(Ly=Ly, Lx=Lx, filename='registered_input.tif')
ops, stat = suite2p.detection_wrapper(f_reg=f_reg, ops=ops)
```

CHAPTER

THREE

SETTINGS (OPS.NPY)

Suite2p can be run with different configurations using the `ops` dictionary. The `ops` dictionary will describe the settings used for a particular run of the pipeline. Here is a summary of all the parameters that the pipeline takes and their default values.

3.1 Main settings

These are the essential settings that are dataset-specific.

- **nplanes**: (*int, default: 1*) each tiff has this many planes in sequence
- **nchannels**: (*int, default: 1*) each tiff has this many channels per plane
- **functional_chan**: (*int, default: 1*) this channel is used to extract functional ROIs (1-based, so 1 means first channel, and 2 means second channel)
- **tau**: (*float, default: 1.0*) The timescale of the sensor (in seconds), used for deconvolution kernel. The kernel is fixed to have this decay and is not fit to the data. We recommend:
  - 0.7 for GCaMP6f
  - 1.0 for GCaMP6m
  - 1.25-1.5 for GCaMP6s
- **force_sktiff**: (*boolean, default: False*) specifies whether or not to use scikit-image for reading in tiffs
- **fs**: (*float, default: 10.0*) Sampling rate (per plane). For instance, if you have a 10 plane recording acquired at 30Hz, then the sampling rate per plane is 3Hz, so set ops['fs'] = 3.
- **do_bidiphase**: (*bool, default: False*) whether or not to compute bidirectional phase offset from misaligned line scanning experiment (applies to 2P recordings only). suite2p will estimate the bidirectional phase offset from ops['nimg_init'] frames if this is set to 1 (and ops['bidiphase']=0), and then apply this computed offset to all frames.
- **bidiphase**: (*int, default: 0*) bidirectional phase offset from line scanning (set by user). If set to any value besides 0, then this offset is used and applied to all frames in the recording.
- **bidi_corrected**: (*bool, default: False*) Specifies whether to do bidi correction.
- **frames_include**: (*int, default: -1*) if greater than zero, only *frames_include* frames are processed. useful for testing parameters on a subset of data.
- **multiplane_parallel**: (*boolean, default: False*) specifies whether or not to run pipeline on server
- **ignore_flyback**: (*list[ints], default: empty list*) specifies which planes will be ignored as flyback planes by the pipeline.

3.2 File input/output settings

Suite2p can accomodate many different file formats. Refer to this *page* for a detailed list of formats suite2p can work with.

- **fast_disk**: (*list[str], default: empty list*) specifies location where temporary binary file will be stored. Defaults to `save_path0` if no directory is provided by user.
- **delete_bin** (*bool, default:False*) specifies whether to delete binary file created during registration stage.
- **mesoscan** (*bool, default: False*) specifies whether file being read in is a scanimage mesoscope recording
- **bruker** (*bool, default: False*) specifies whether provided tif files are single page BRUKER tiffs
- **bruker_bidirectional** (*bool, default: False*) specifies whether BRUKER files are bidirectional multiplane recordings. The `True` setting corresponds to the following plane order (first plane is indexed as zero): [0,1,2,2,1,0]. `False` corresponds to [0,1,2,0,1,2].
- **h5py** (*list[str], default: empty list*) specifies path to h5py file that will be used as inputs. Keep in mind the pathname provided here overwrites the pathname specified in `ops[data_path]`.
- **h5py_key** (*str, default: 'data'*) key used to access data array in h5py file. Only use this when the `h5py` setting is set to `True`.
- **nwb_file** (*str, default: ''*) specifies path to NWB file you use to use as input
- **nwb_driver** (*str, default: ''*) location of driver for NWB file. Leave this empty if the pathname refers to a local file.
- **nwb_series** (*str, default: ''*) Name of TwoPhotonSeries values you wish to retrieve from your NWB file.
- **save_path0** (*list[str], default: empty list*) List containing pathname of where you'd like to save your pipeline results. If list is empty, the first element of `ops['data_path']` is used.
- **save_folder** (*list[str], default: empty list*) List containing directory name you'd like results to be saved under. Defaults to `"suite2p"`.
- **look_one_level_down**: (*bool, default: False*) specifies whether to look in all subfolders when searching for tiffs. Make sure to specify subfolders in the `subfolders` parameter below.
- **subfolders** (*list[str], default: empty list*) Specifies subfolders you'd like to look through. Make sure to have the above parameter `ops[look_one_level_down] = True` when using this parameter.
- **move_bin** (*bool, default: False*) If True and `ops['fast_disk']` is different from `ops[save_disk]`, the created binary file is moved to `ops['save_disk']`.

3.3 Output settings

- **preclassify**: (*float, default: 0.0*) (**new**) apply classifier before signal extraction with probability threshold of "preclassify". If this is set to 0.0, then all detected ROIs are kept and signals are computed.
- **save_nwb**: (*bool, default: False*) whether to save output as NWB file
- **save_mat**: (*bool, default: False*) whether to save the results in matlab format in file "Fall.mat". NOTE the cells you click in the GUI will NOT change "Fall.mat". But there is a **new** button in the GUI you can click to resave "Fall.mat" in the "File" window.
- **combined**: (*bool, default: True*) combine results across planes in separate folder "combined" at end of processing. This folder will allow all planes to be loaded into the GUI simultaneously.

- **aspect**: (*float, default: 1.0) (**new**) ratio of um/pixels in X to um/pixels in Y (ONLY for correct aspect ratio in GUI, not used for other processing)
- **report_time**: (*bool, default: True) (**new**) whether or not to return a timing dictionary for each plane. Timing dictionary will contain keys corresponding to stages and values corresponding to the duration of that stage.

3.4 Registration settings

These settings are specific to the registration module of suite2p.

- **do_registration**: (*bool, default: True*) whether or not to run registration
- **align_by_chan**: (*int, default: 1*) which channel to use for alignment (1-based, so 1 means 1st channel and 2 means 2nd channel). If you have a non-functional channel with something like td-Tomato expression, you may want to use this channel for alignment rather than the functional channel.
- **nimg_init**: (*int, default: 300*) how many frames to use to compute reference image for registration
- **batch_size**: (*int, default: 500*) how many frames to register simultaneously in each batch. This depends on memory constraints - it will be faster to run if the batch is larger, but it will require more RAM.
- **maxregshift**: (*float, default: 0.1*) the maximum shift as a fraction of the frame size. If the frame is Ly pixels x Lx pixels, then the maximum pixel shift in pixels will be max(Ly,Lx) * ops['maxregshift'].
- **smooth_sigma**: (*float, default: 1.15*) standard deviation in pixels of the gaussian used to smooth the phase correlation between the reference image and the frame which is being registered. A value of *>4* is recommended for one-photon recordings (with a 512x512 pixel FOV).
- **smooth_sigma_time**: (*float, default: 0*) standard deviation in time frames of the gaussian used to smooth the data before phase correlation is computed. Might need this to be set to 1 or 2 for low SNR data.
- **keep_movie_raw**: (*bool, default: False*) whether or not to keep the binary file of the non-registered frames. You can view the registered and non-registered binaries together in the GUI in the "View registered binaries" view if you set this to *True*.
- **two_step_registration**: (*bool, default: False*) whether or not to run registration twice (for low SNR data). *keep_movie_raw* must be True for this to work.
- **reg_tif**: (*bool, default: False*) whether or not to write the registered binary to tiff files
- **reg_tif_chan2**: (*bool, default: False*) whether or not to write the registered binary of the non-functional channel to tiff files
- **subpixel**: (*int, default:10*) Precision of Subpixel Registration (1/subpixel steps)
- **th_badframes**: (*float, default: 1.0*) Involved with setting threshold for excluding frames for cropping. Set this smaller to exclude more frames.
- **norm_frames**: (*bool, default: True*) Normalize frames when detecting shifts
- **force_refImg**: (*bool, default: False*) Specifies whether to use refImg stored in `ops`. Make sure that `ops['refImg']` has a valid file pathname.
- **pad_fft**: (*bool, default: False*) Specifies whether to pad image or not during FFT portion of registration.

3.4.1 1P registration

- **1Preg**: (*bool, default: False*) whether to perform high-pass spatial filtering and tapering (parameters set below), which help with 1P registration
- **spatial_hp_reg**: (*int, default: 42*) window in pixels for spatial high-pass filtering before registration
- **pre_smooth**: (*float, default: 0*) if > 0, defines stddev of Gaussian smoothing, which is applied before spatial high-pass filtering
- **spatial_taper**: (*float, default: 40*) how many pixels to ignore on edges - they are set to zero (important for vignetted windows, for FFT padding do not set BELOW 3*ops['smooth_sigma'])

3.4.2 Non-rigid registration

- **nonrigid**: (*bool, default: True*) whether or not to perform non-rigid registration, which splits the field of view into blocks and computes registration offsets in each block separately.
- **block_size**: (*two ints, default: [128,128]*) size of blocks for non-rigid registration, in pixels. HIGHLY recommend keeping this a power of 2 and/or 3 (e.g. 128, 256, 384, etc) for efficient fft
- **snr_thresh**: (*float, default: 1.2*) how big the phase correlation peak has to be relative to the noise in the phase correlation map for the block shift to be accepted. In low SNR recordings like one-photon, I'd recommend a larger value like *1.5*, so that block shifts are only accepted if there is significant SNR in the phase correlation.
- **maxregshiftNR**: (*float, default: 5.0*) maximum shift in pixels of a block relative to the rigid shift

3.5 ROI detection settings

- **roidetect**: (*bool, default: True*) whether or not to run ROI detect and extraction
- **sparse_mode**: (*bool, default: True*) whether or not to use sparse_mode cell detection
- **spatial_scale**: (*int, default: 0*), what the optimal scale of the recording is in pixels. if set to 0, then the algorithm determines it automatically (recommend this on the first try). If it seems off, set it yourself to the following values: 1 (=6 pixels), 2 (=12 pixels), 3 (=24 pixels), or 4 (=48 pixels).
- **connected**: (*bool, default: True*) whether or not to require ROIs to be fully connected (set to *0* for dendrites/boutons)
- **threshold_scaling**: (*float, default: 1.0*) this controls the threshold at which to detect ROIs (how much the ROIs have to stand out from the noise to be detected). if you set this higher, then fewer ROIs will be detected, and if you set it lower, more ROIs will be detected.
- **spatial_hp_detect**: (*int, default: 25*) window for spatial high-pass filtering for neuropil subtracation before ROI detection takes place.
- **max_overlap**: (*float, default: 0.75*) we allow overlapping ROIs during cell detection. After detection, ROIs with more than ops['max_overlap'] fraction of their pixels overlapping with other ROIs will be discarded. Therefore, to throw out NO ROIs, set this to 1.0.
- **high_pass**: (*int, default: 100*) running mean subtraction across time with window of size 'high_pass'. Values of less than 10 are recommended for 1P data where there are often large full-field changes in brightness.
- **smooth_masks**: (*bool, default: True*) whether to smooth masks in final pass of cell detection. This is useful especially if you are in a high noise regime.
- **max_iterations**: (*int, default: 20*) how many iterations over which to extract cells - at most ops['max_iterations'], but usually stops before due to ops['threshold_scaling'] criterion.

- **nbinned**: (*int, default: 5000*) maximum number of binned frames to use for ROI detection.
- **denoise**: (*bool, default: False*) Whether or not binned movie should be denoised before cell detection in sparse_mode. If True, make sure to set `ops['sparse_mode']` is also set to True.

3.5.1 Cellpose Detection

These settings are only used if `ops['anatomical_only']` is set to an integer greater than 0.

- **anatomical_only**: (*int, default: 0*) If greater than 0, specifies what to use Cellpose on.
  - 1: Will find masks on max projection image divided by mean image.
  - 2: Will find masks on mean image
  - 3: Will find masks on enhanced mean image
  - 4: Will find masks on maximum projection image
- **diameter**: (*int, default: 0*) Diameter that will be used for cellpose. If set to zero, diameter is estimated.
- **cellprob_threshold**: (*float, default: 0.0*) specifies threshold for cell detection that will be used by cellpose.
- **flow_threshold**: (*float, default: 1.5*) specifies flow threshold that will be used for cellpose.
- **spatial_hp_cp**: (*int, default: 0*) Window for spatial high-pass filtering of image to be used for cellpose.
- **pretrained_model**: (*str, default: 'cyto'*) Path to pretrained model or string for model type (can be user's model).

3.6 Signal extraction settings

- **neuropil_extract**: (*bool, default: True*) Whether or not to extract signal from neuropil. If False, Fneu is set to zero.
- **allow_overlap**: (*bool, default: False*) whether or not to extract signals from pixels which belong to two ROIs. By default, any pixels which belong to two ROIs (overlapping pixels) are excluded from the computation of the ROI trace.
- **min_neuropil_pixels**: (*int, default: 350*) minimum number of pixels used to compute neuropil for each cell
- **inner_neuropil_radius**: (*int, default: 2*) number of pixels to keep between ROI and neuropil donut
- **lam_percentile**: (*int, default: 50*)Percentile of Lambda within area to ignore when excluding cell pixels for neuropil extraction

3.7 Spike deconvolution settings

We neuropil-correct the trace `Fout = F - ops['neucoeff'] * Fneu`, and then baseline-correct these traces with an `ops['baseline']` filter, and then detect spikes.

- **spikedetect**: (*bool, default: True*) Whether or not to run spike_deconvolution
- **neucoeff**: (*float, default: 0.7*) neuropil coefficient for all ROIs.
- **baseline**: (*string, default 'maximin'*) how to compute the baseline of each trace. This baseline is then subtracted from each cell. *'maximin'* computes a moving baseline by filtering the data with a Gaussian of width `ops['sig_baseline'] * ops['fs']`, and then minimum filtering with a window of

`ops['win_baseline'] * ops['fs']`, and then maximum filtering with the same window. *'constant'* computes a constant baseline by filtering with a Gaussian of width `ops['sig_baseline'] * ops['fs']` and then taking the minimum value of this filtered trace. *'constant_percentile'* computes a constant baseline by taking the `ops['prctile_baseline']` percentile of the trace.

- **win_baseline**: (*float, default: 60.0*) window for maximin filter in seconds
- **sig_baseline**: (*float, default: 10.0*) Gaussian filter width in seconds, used before maximin filtering or taking the minimum value of the trace, `ops['baseline'] = 'maximin'` or `'constant'`.
- **prctile_baseline**: (*float, optional, default: 8*) percentile of trace to use as baseline if `ops['baseline'] = 'constant_percentile'`.

3.8 Classification settings

- **soma_crop**: (*bool, default: True*) Specifies whether to crop dendrites for cell classification stats (e.g., compactness)
- **use_builtin_classifier**: (*bool, default: False*) Specifies whether or not to use built-in classifier for cell detection. This will override classifier specified in `ops['classifier_path']` if set to True.
- **classifier_path**: (*str, default: ''*) Path to classifier file you want to use for cell classification

3.9 Channel 2 specific settings

- **chan2_thres**: threshold for calling an ROI "detected" on a second channel

3.10 Miscellaneous settings

- **suite2p_version**: specifies version of suite2p pipeline that was run with these settings. Changing this parameter will NOT change the version of suite2p used.

CHAPTER

FOUR

USING THE GUI

Once you've run the processing, you can open the output `stat.npy` file from the GUI. This allows you to explore the data in depth both spatially and in time. In addition you can classify ROIs as 'cells' or 'NOT cells' (left or right side of screen) and train a classifier to automatically identify the cells as one of these two classes. Note that these categories do not have to be 'cells' and 'NOT cells', they could be 'boutons' and 'NOT boutons', we just chose to say 'cells' because that's the most common ROI studied.

You can now **drag and drop** your `stat.npy` files into the GUI!

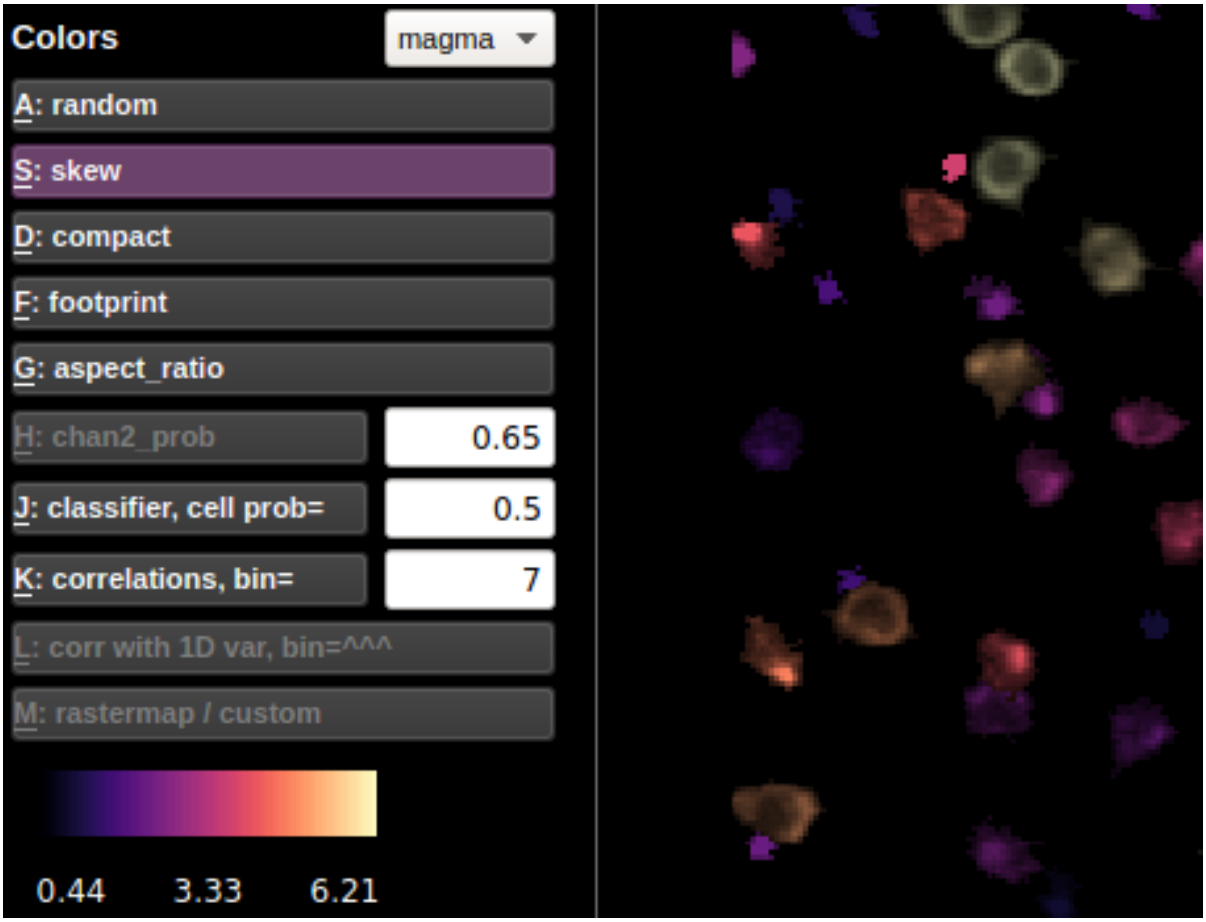
4.1 Different views and colors for ROI panels

4.1.1 Views

To turn off ROIs in views 2-4, uncheck *ROIs on*

1. *ROIs*: ROIs only are drawn
2. *mean img*: mean image is shown in background
3. *mean img (enhanced)*: mean image filtered with a min-max filter shown in background
4. *correlation map*: map of correlated pixels shown in background
5. *mean img (non-functional)*: the non-functional mean image shown in background (if nchannels=2)

4.1.2 Colors



Randomly colored ROI view is the default view. The ROIs in the random view are colored between purple and yellow, with red reserved for ROIs assigned to be RED based on the non-functional channel (you can change the threshold for calling a cell RED with the number next to the chan2 prob button). The other color views color the ROIs based on their statistics. The values of those statistics are shown in the colorbar below the buttons.

Here is more info about the less explanatory views:

4.1.3 Correlations

In correlation color view, the selected cell's activity (or the mean of the selected cells' activities) is correlated with the activity of all the other ROIs. The ROIs are colored according to these correlations. The bin in which to compute the correlations can be chosen (in units of frames). The default bin size is the number of frames per second (`ops['fs']`).

If a 1D external variable is loaded, then the *corr with 1D var* button is activated. The cells are then colored according to their correlation with the external variable. The bin size is determined by the box next to the *correlations* button.

4.1.4 Correlations with 1D var

You can load an external stimulus or behavioral trace (1D) using "File - Load behavior or stim trace (1D only)". The GUI expects a *.npy file that is the same length as the data in time (F.shape[1] from "F.npy"). You can then look at the correlation of each cell with this trace. And it will be plotted along with the cell traces if you select multiple cells or in the "Visualize" menu.

4.1.5 Rastermap / custom

Rastermap: Click 'Visualize selected cells' in the Visualizations menu and run rastermap on the cells. The selected cells (which could be all cells on LEFT or RIGHT) will then be colored based on their position in the rastermap.

Custom map: Use 'Load custom hue' in the Visualizations menu to load a *.npy file with the same number of values as ROIs (length of stat) and these values will become the hues of the cells (scaled to between 0 and 1) for the HSV map. If you do rastermap after this then the colors will change and vice versa this will overwrite the rastermap colors.

4.2 Buttons / shortcuts for cell selection

4.2.1 Mouse control

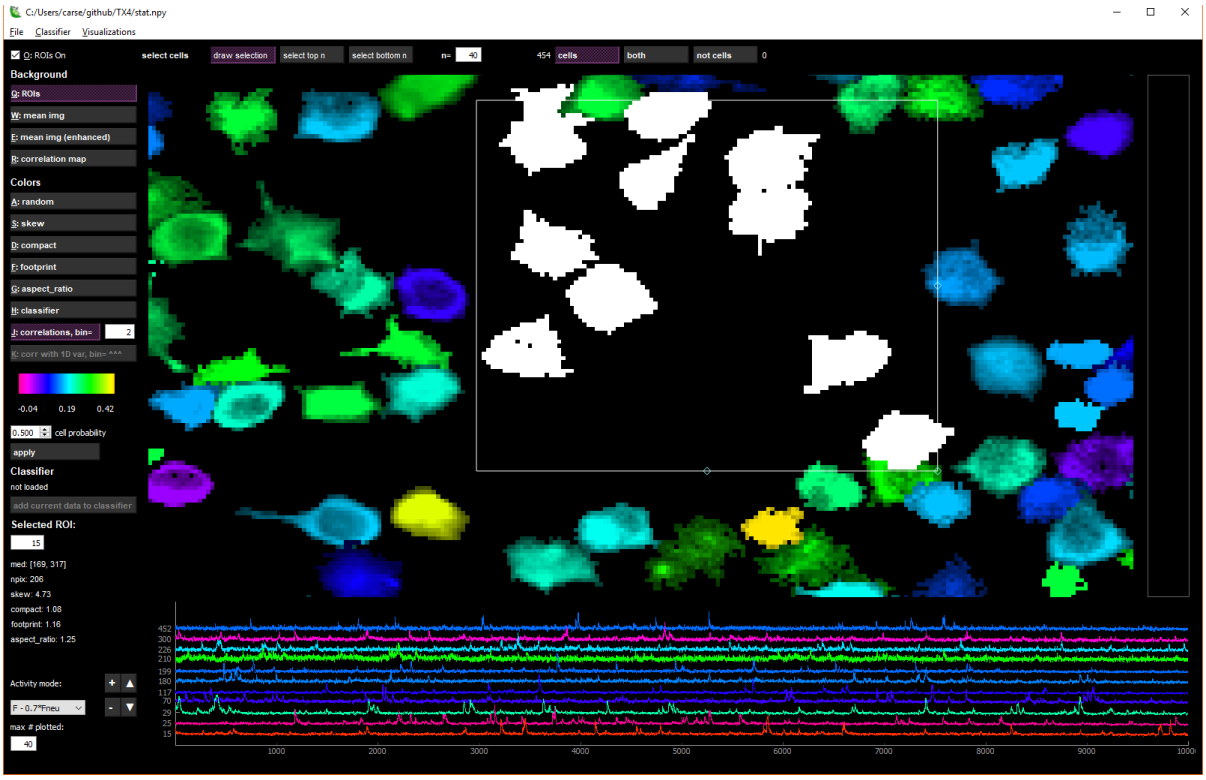
- **double left click = returns to full view in ALL PLOTS**
- left click = select cell
- left click + CTRL = select multiple cells
- left click + drag = moves field of view
- right click = flip selected cell(s) from left<->right, – or if clicked in trace view, will open up "export" option
- scroll wheel = zoom in and out

4.2.2 Keyboard shortcuts

- Esc = returns to full view
- Delete = removes box from *draw selection* from window For the letters, just press the letter (do not capitalize)
- O = turn of ROIs in non-ROI view
- Q-U = different views (can change saturation with slider)
- A-M = different color maps
- Left and right keys = cycle between cells of same panel
- Up Key = flip selected cell to other panel
- Alt+Enter = merge selected ROIs
- note you can also ask the GUI to auto-suggest merges with the Merge>Auto-suggest merges window *

4.2.3 Multi-cell selection

You can select multiple cells by holding down CTRL while left-clicking on cells. If you are in 'cells' or 'NOT cells' view (not 'both' view), then several buttons for multi-cell selection activate.



The *draw selection* button activates a box that you can drag and resize to select multiple cells. To delete the box, click the Delete key. *select top n* selects n=X top neurons from the current colormap. For instance, in 'skew' view, *select top n* will select the most skewed neurons. In 'correlation' view, it will choose the most correlated neurons with the currently selected neuron.

4.3 Trace view (bottom row)

When one cell is selected, the fluorescence, neuropil and deconvolved traces are shown for the chosen cell in the bottom row of the GUI. When multiple cells are selected, you can choose what type of traces to view with the drop-down menu in the lower left:

- F: fluorescence
- Fneu: neuropil fluorescence
- F - 0.7*Fneu: corrected fluorescence
- deconvolved: deconvolution of corrected fluorescence

You can resize the trace view with the triangle buttons (bigger = , smaller =). If multiple cells are selected, you can vary how much the traces overlap with the +/- buttons.

You can select as many cells as you want, but by default only 40 of those will be plotted. You can increase or decrease this number by changing the number in the box below *max # plotted*.

You can hide the fluorescence, neuropil and/or the deconvolved traces by toggling the chechboxes or using the keys as follows:

Deconvolved - N key Neuropil - B Key Fluorescence - V Key

4.4 Classifying cells

suite2p comes with a *built-in* classifier (based on our own manual curation of GCaMP6s imaging of cells in cortex). The *default* classifier is initialized as the *built-in* classifier, but can be modified by the user.

After running suite2p, the cells are automatically classified by the default classifier (at the time of running the pipeline), and these cell probabilities are shown as the colors in the *classifier* view. You can then further manually curate this data (flipping cells left and right depending on your criteria).

4.4.1 Adding data to a classifier

You can add this manually curated data to an already built classifier:

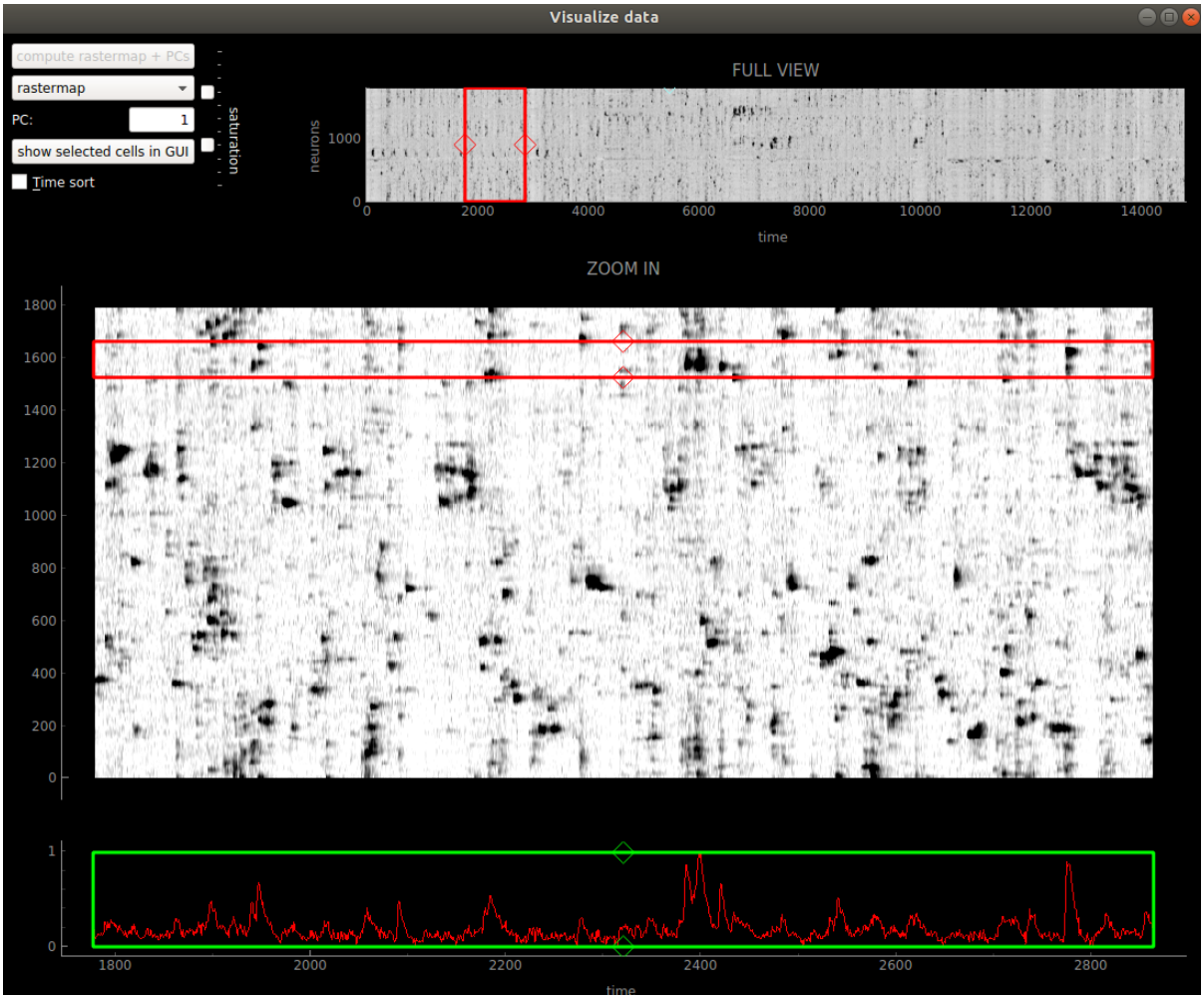
1. Load a classifier by going to the "Classifier" menu and clicking "Load". Choose the *default* classifier, or load another classifier that you've built and saved with the *from file* option.
2. Click the *add current data to classifier* button. This will either overwrite the classifier file that is loaded, or you can specify a file location for the classifier with this newly added data.

4.4.2 Building your own classifier

Go to the "Classifier" menu and click "Build". A window will pop up and in the window you can add datasets as training samples for the classifier. Click the *Load iscell.npy* button and add an `iscell.npy` file. You can add as many as you like, then click *build classifier*, and it will ask you to specify a file location for the new classifier. Then you can load the classifier that you built into the GUI, or you can save it as your default classifier.

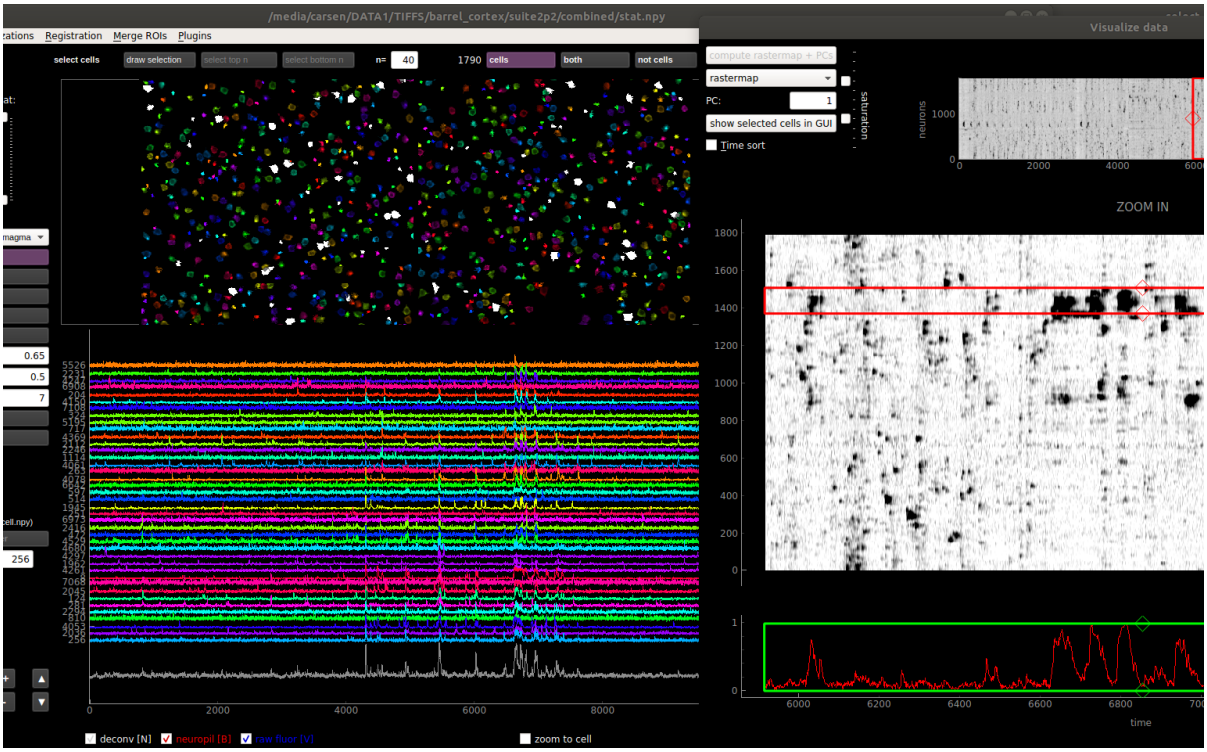
4.5 Visualizing activity

Go to the "Visualizations" menu and click "Visualize selected cells". If only one ROI is selected, then all ROIs in that view (cell or not cell) will be plotted. Otherwise the selected cells are plotted. You can sort the neurons by their principal component weights, or by our algorithm rastermap by clicking the compute buttons. Once you click the *compute* buttons, they will be grayed out, because you can't compute them again (they won't change). The plot below shows a mesoscope recording sorted by rastermap. You can change between sorting by rastermap and by the PCs by using the drop-down menu.



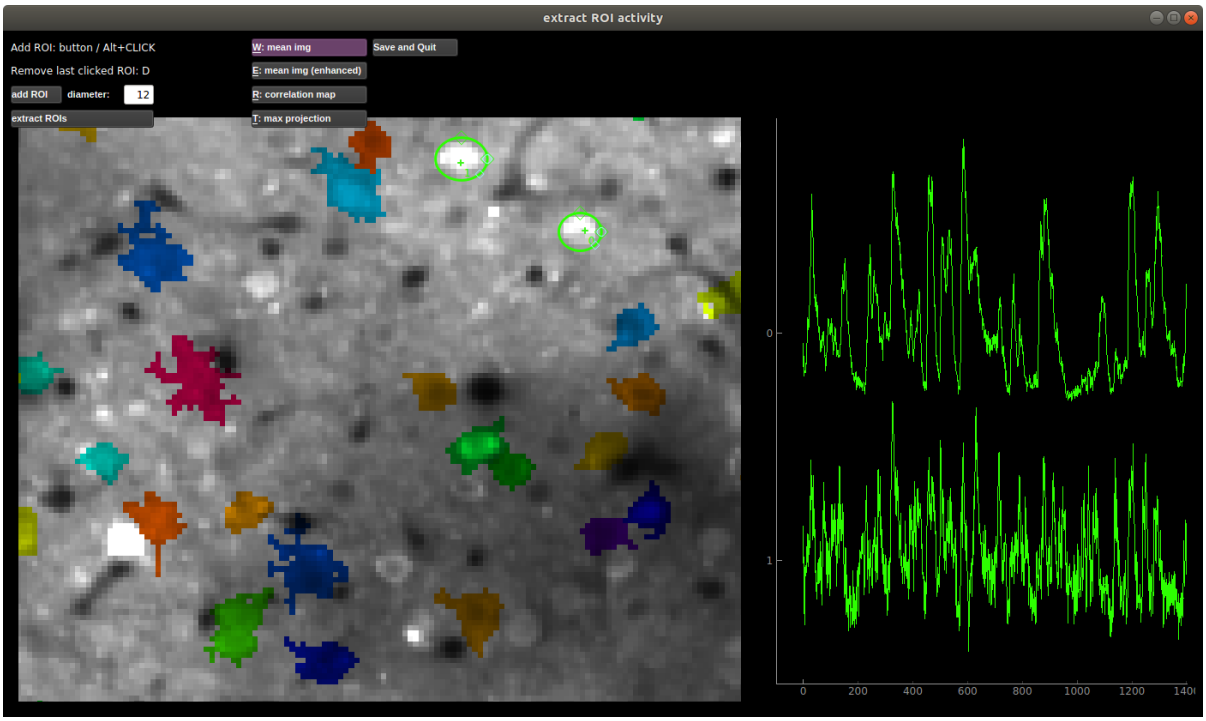
The red box allows you to zoom in on specific segments of the recording. You can move it by dragging the mouse when in the box, or with the arrow keys. You can resize it by using the diamond handles on the sides of the box, or by holding down the shift key and using the arrow keys.

If you click the *show selected cells in GUI* button, then the cells surrounded by the red box will show up as white in the GUI.



4.6 Manual adding of ROIs

You can add ROIs in the File>Manual labelling. You MUST keep the binary file for the computing of the mask's activity across time. When you save and exit the ROIs will be added to the *.npy files as the first N ROIs (where N is the number that you drew).



4.7 Merging ROIs

You can merge selected ROIs (multi-select with CTRL) by pressing ALT+ENTER, or get suggested merges in the "Merge ROI" menu. The merged ROIs then must be saved before you close the GUI to write the new ROIs to the *.npy files. Each merged ROI is appended to the end of the list of ROIs (in stat), and the ROIs that were merged to create it are in the key 'imerge'. Note in the stat file and other files the original ROIs (that create the ROI) are NOT removed so that you retain the original signals and original suite2p output. In the GUI ROI view the merged ROIs are shown.

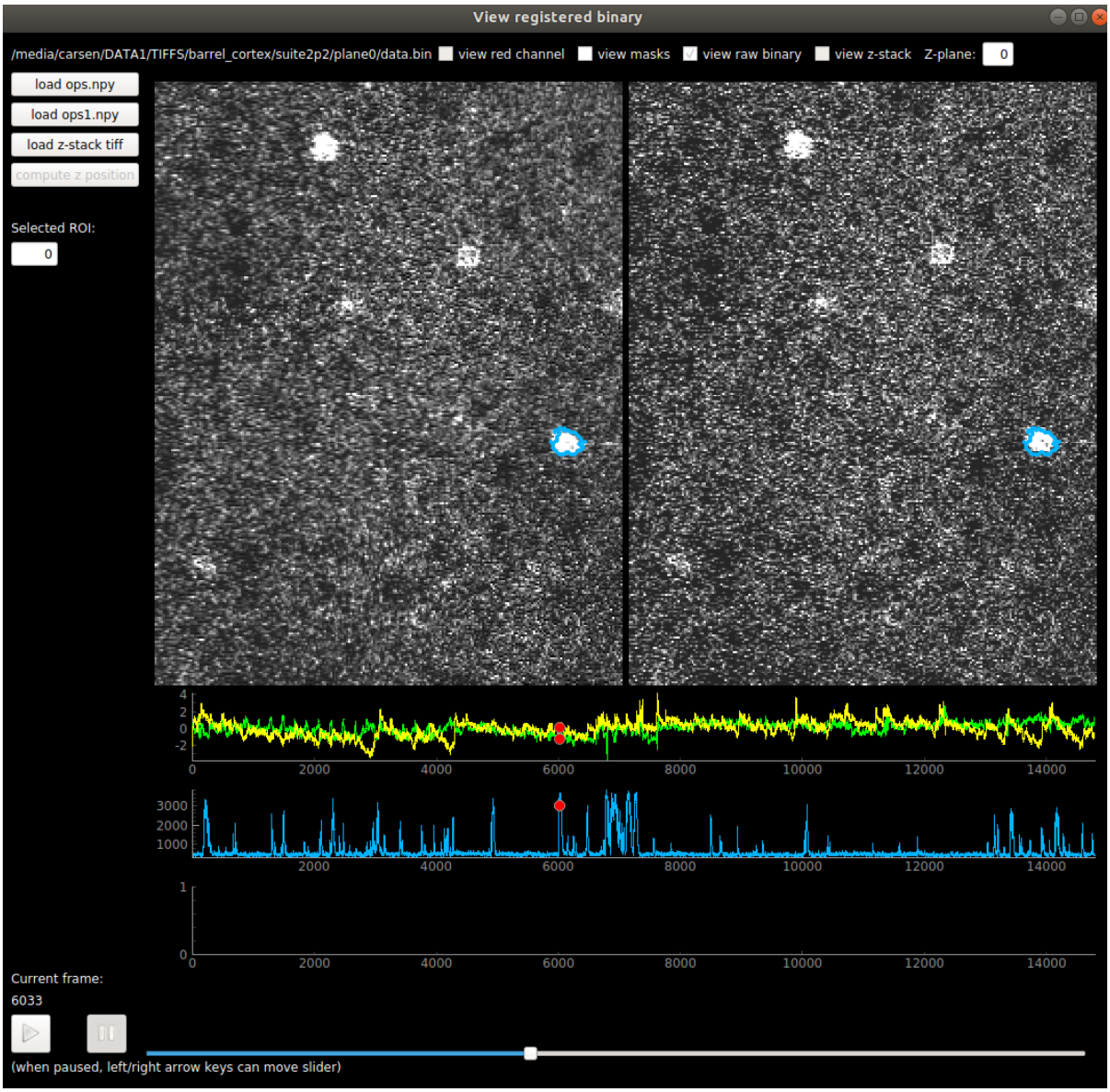
The merging of fluorescence is done by taking the mean of the selected cells' fluorescences. The list of merges are available in the stat for you to choose alternative strategies for combining signals.

4.8 View registered binary

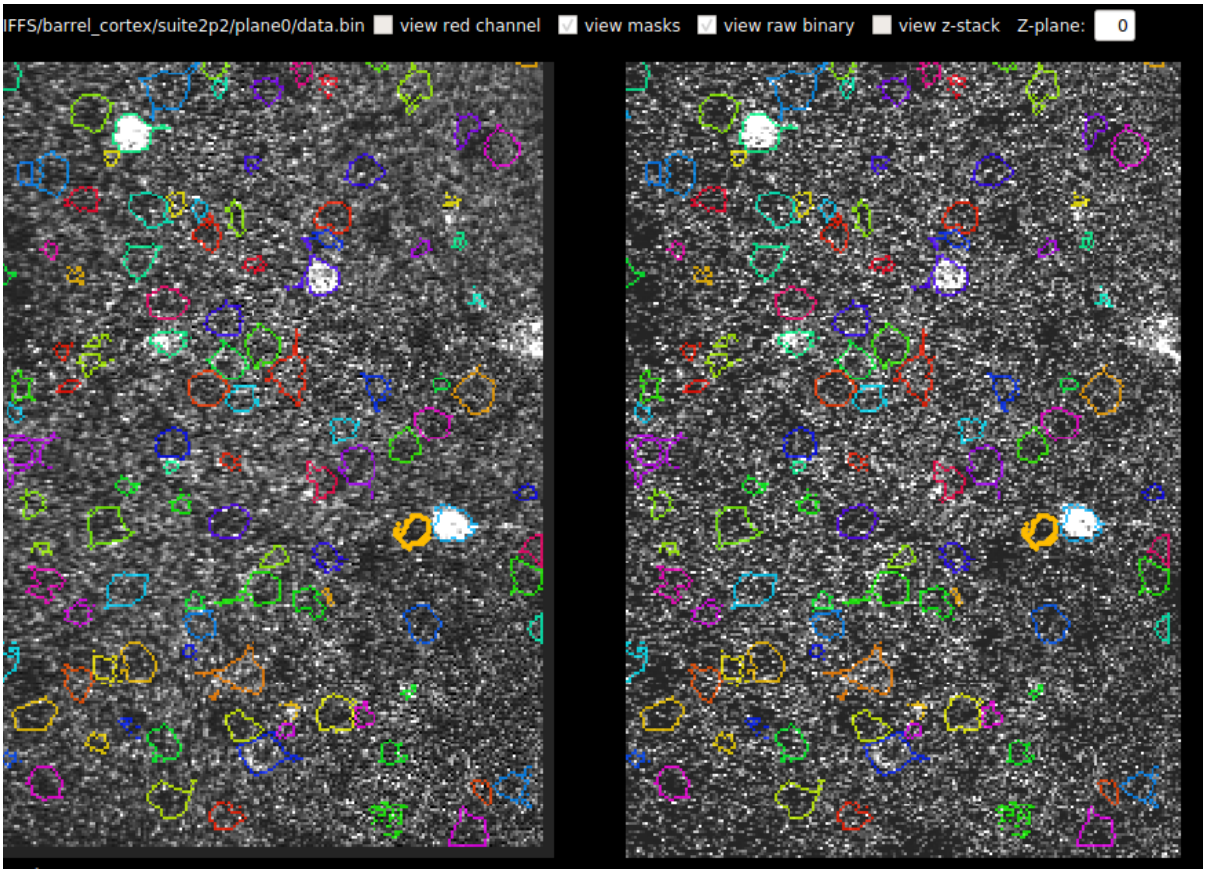
Open the "Registration" menu and click "View registered binary". A window will pop up with the binary file loaded (first row) along with the registration shifts (second row), and the fluorescence of a selected ROI (third row). If ops['keep_movie_raw']=1, then both the unregistered and registered binaries will be shown in the first row. You can select an ROI by typing in the ROI number in the upper right.

You can zoom in and out on any of the plots. The shift plot and the fluorescence plot have linked x-axes. To return to full view, double-click on the plot that you want to recenter.

When not playing the movie, you can click on the shift plot and the fluorescence plot to go to a specific point in time in the movie. You can also seek through the movie by clicking the slide bar. The left and right arrow keys will move the slide bar incrementally. The space bar will pause and play the movie.



You can also view all the masks, and go from cell to cell by clicking on them.



4.8.1 Z-stack Alignment

You can also now load a Z-stack into this view. You can compute the z-position of the recording across time IF you keep the registered binary file. It expects the Z-stack to be a tiff with number of planes by number of pixels in Y by number of pixels in X. The results of the correlation between z-stack and each frame are saved in ops['zcorr'] which is number of planes (in Z) x number of frames. The GUI smooths this matrix across Z and then takes the max and plots the max across time in the third row.

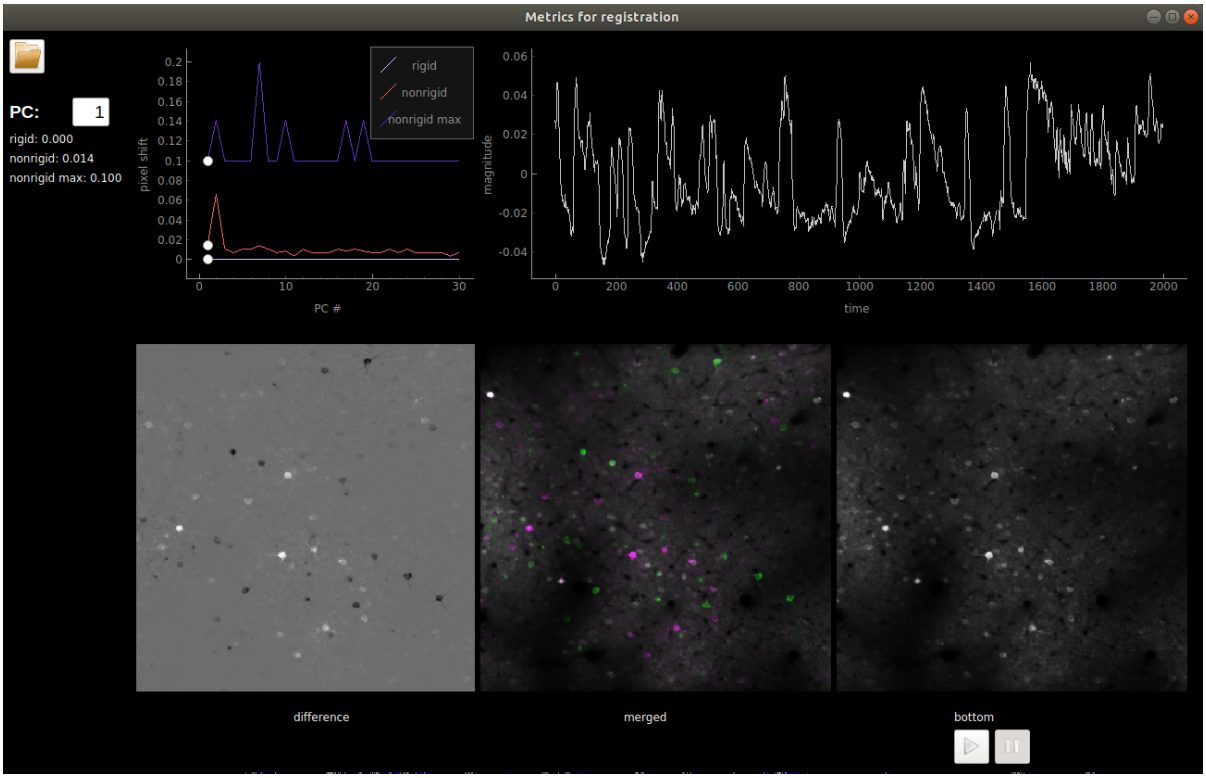
4.9 View registration metrics

Open the "Registration" menu and click "View registration metrics". A window will pop up with ops['regDX'] and ops['regPC'] plotted. The ops['regPC']'s are computed by taking the principal components of the registered movie. `ops['regPC'][0,0,:,:]` is the average of the top 500 frames of the 1st PC, `ops['regPC'][1,0,:,:]` is the average of the bottom 500 frames of the 1st PC. `ops['regDX']` quantifies the movement in each PC (`iPC`) by registering `ops['regPC'][0,iPC,:,:]` and `ops['regPC'][1,iPC,:,:]` to the reference images and computing the registration shifts.

The first plot in the upper left shows the magnitude of the shifts (both rigid and non-rigid) in the PCs (`ops['regDX']`). The second row of plots are meant to help explore the direction of the PC. The first image is the "difference" between the top and the bottom of the PC. The second image is the "merged" image of the top and bottom of the PC. The third image allows you to flip between the top and bottom PCs using the "play" button.

The left and right arrow keys will change the PC number (or you can type in a number). The space bar will pause and play the movie.

The example below shows a movie that has been rigid registered but not non-rigid registered. The metrics suggest that non-rigid registration should also be performed on this recording.



CHAPTER

FIVE

OUTPUTS

`F.npy`: array of fluorescence traces (ROIs by timepoints)

`Fneu.npy`: array of neuropil fluorescence traces (ROIs by timepoints)

`spks.npy`: array of deconvolved traces (ROIs by timepoints)

`stat.npy`: list of statistics computed for each cell (ROIs by 1)

`ops.npy`: options and intermediate outputs (dictionary)

`iscell.npy`: specifies whether an ROI is a cell, first column is 0/1, and second column is probability that the ROI is a cell based on the default classifier

All can be loaded in python with numpy

```
import numpy as np

F = np.load('F.npy', allow_pickle=True)
Fneu = np.load('Fneu.npy', allow_pickle=True)
spks = np.load('spks.npy', allow_pickle=True)
stat = np.load('stat.npy', allow_pickle=True)
ops =  np.load('ops.npy', allow_pickle=True)
ops = ops.item()
iscell = np.load('iscell.npy', allow_pickle=True)
```

5.1 MATLAB output

If `'save_mat'=1`, then a MATLAB file is created `Fall.mat`. This will contain ops, F, Fneu, stat, spks and iscell. The "iscell" assignments are only saved ONCE when the pipeline is finished running. If you make changes in the GUI to the cell assignments, ONLY `iscell.npy` changes. To load a modified `iscell.npy` into MATLAB, I recommend using this package: npy-matlab. Alternatively there is a *new* save button in the GUI (in the file menu) that allows you to save the iscell again to the `Fall.mat` file.

5.2 NWB Output

If `ops['save_NWB']=1`, then an NWB file is created `ophys.nwb`. This will contain the fields from ops and stat required to load back into the GUI, along with F, Fneu, spks and iscell. If the recording has multiple planes, then they are all saved together like in combined view. See fields below:

stat: stat['ypix'], stat['xpix'] (if multiplane *stat['iplane']*) are saved in 'pixel_mask' (called *'voxel_mask'* in multiplane).

ops: 'meanImg', 'max_proj', 'Vcorr' are saved in Images 'Backgrounds_k' where k is the plane number, and have the same names. optionally if two channels, 'meanImg_chan2' is saved.

iscell: saved as an array 'iscell'

F,Fneu,spks are saved as roi_response_series 'Fluorescence', 'Neuropil', and 'Deconvolved'.

5.3 Multichannel recordings

Cells are detected on the ops['functional_chan'] and the fluorescence signals are extracted from both channels. The functional channel signals are saved to `F.npy` and `F_neu.npy`, and non-functional channel signals are saved to `F_chan2.npy` and `Fneu_chan2.npy`.

5.4 stat.npy fields

- ypix: y-pixels of cell
- xpix: x-pixels of cell
- med: (y,x) center of cell
- lam: pixel mask (sum(lam * frames[ypix,xpix,:]) = fluorescence)
- npix: number of pixels in ROI
- npix_norm: number of pixels in ROI normalized by the mean of npix across all ROIs
- radius: estimated radius of cell from 2D Gaussian fit to mask
- aspect_ratio: ratio between major and minor axes of a 2D Gaussian fit to mask
- compact: how compact the ROI is (1 is a disk, >1 means less compact)
- footprint: spatial extent of an ROI's functional signal, including pixels not assigned to the ROI; a threshold of 1/5 of the max is used as a threshold, and the average distance of these pixels from the center is defined as the footprint
- skew: skewness of neuropil-corrected fluorescence trace
- std: standard deviation of neuropil-corrected fluorescence trace
- overlap: which pixels overlap with other ROIs (these are excluded from fluorescence computation)
- ipix_neuropil: pixels of neuropil mask for this cell

Here is example code to make an image where each cell (without its overlapping pixels) is a different "number":

```
stat = np.load('stat.npy')
ops = np.load('ops.npy').item()

im = np.zeros((ops['Ly'], ops['Lx']))

for n in range(0,ncells):
    ypix = stat[n]['ypix'][~stat[n]['overlap']]
    xpix = stat[n]['xpix'][~stat[n]['overlap']]
    im[ypix,xpix] = n+1

plt.imshow(im)
plt.show()
```

(There is no longer ipix like in the matlab version. In python note you can access a 2D array like X[ys, xs] = lam. In Matlab, this would cause a broadcast of all the pairs of ys and xs, which is why ipix = ys + (xs-1) * Ly was a useful temporary variable to have around for linear indexing into arrays. In Python, the equivalent ipix would be ipix = yx + xs * Lxy.)

5.5 ops.npy fields

This will include all of the options you ran the pipeline with, including file paths. During the running of the pipeline, some outputs are added to `ops.npy`:

- **reg_file**: location of registered binary file
- **Ly**: size of Y dimension of tiffs/h5
- **Lx**: size of X dimension of tiffs/h5
- **nframes**: number of frames in recording
- **yrange**: valid y-range used for cell detection (excludes edges that were shifted out of the FOV during registration)
- **xrange**: valid x-range used for cell detection (excludes edges that were shifted out of the FOV during registration)
- **refImg**: reference image used for registration
- **yoff**: y-shifts of recording at each timepoint
- **xoff**: x-shifts of recording at each timepoint
- **corrXY**: peak of phase correlation between frame and reference image at each timepoint
- **meanImg**: mean of registered frames
- **meanImgE**: a median-filtered version of the mean image
- **Vcorr**: correlation map (computed during cell detection)
- **filelist**: List of the image file names (e.g. tiff) that were loaded, in the order that Suite2p processed them.
- **date_proc**: Date and time that the analysis was run.

CHAPTER

SIX

MULTIDAY RECORDINGS

In the matlab version of suite2p, Henry Dalgleish wrote the utility "registers2p" for multiday alignment, but it has not been ported to python.

I recommend trying to run all your recordings together (add all the separate folders to data_path). This has worked well for people who have automated online registration on their microscope to register day by day (scanimage 2018b (free) offers this capability). I highly recommend checking this out - we have contributed to a module in that software for online Z-correction that has greatly improved our recording quality.

However, if there are significant non-rigid shifts between days (angle changes etc) and low SNR then concatenating recordings and running them together will not work so well.

In this case, (if you have a matlab license) here is a package written by Adam Ranson which is based on similar concepts as 'registers2p' by Henry Dalgleish that takes the output of suite2p-python directly: https://github.com/ransona/ROIMatchPub.

CHAPTER

SEVEN

DEVELOPER DOCUMENTATION

7.1 Versioning

There's a rare issue that developers may face when calling *suite2p –version* on their command line. You may get an incorrect version number. To fix this issue, one should use the following command:

```
$ git fetch --prune --unshallow
```

7.2 Testing

Before contributing to Suite2P, please make sure your changes pass all our tests.

7.2.1 Downloading Test Data

To run the tests (located in the `tests` subdirectory of your working `suite2p` directory) , you'll first need to download our test data. Suite2p depends on dvc to download the test data.

Note: Before testing, make sure you have dvc and pydrive2 installed. Navigate to the suite2p directory and use the following command to install both dvc and pydrive2.

```
$ pip install -e .[data]
```

zsh users should use the following:

```
$ pip install -e .docs
```

Use to following command to download the test data into the `data` subdirectory of your working `suite2p` directory.

```
$ dvc pull
```

7.2.2 Running the tests

Tests can then be easily run with the following command:

```
$ python setup.py test
```

If all the tests pass, you're good to go!

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS

8.1 Cropped field-of-view

Some issues on this: #273, #207, #125.

Why does this happen? suite2p crops the field-of-view so that areas that move out of view on the edges are not used for ROI detection and signal extraction. These areas are excluded because they are not always in the FOV - they move in and out and therefore activity in these regions is unreliable to estimate.

suite2p determines the region to crop based on the maximum rigid shifts in XY. You can view these shifts with the movie in the "View registered binary" window. If these shifts are too large and don't seem to be accurate (low SNR regime), you can decrease the maximum shift that suite2p can estimate by setting ops['maxregshift'] lower than its default (which is 0.1 = 10% of the size of the FOV). suite2p does exclude some of the large outlier shifts when computing the crop, and determines the threshold of what is an "outlier" using the parameter ops['th_badframes']. Set this lower to increase the number of "outliers". These "outliers" are labelled as ops['badframes'] and these frames are excluded also from ROI detection.

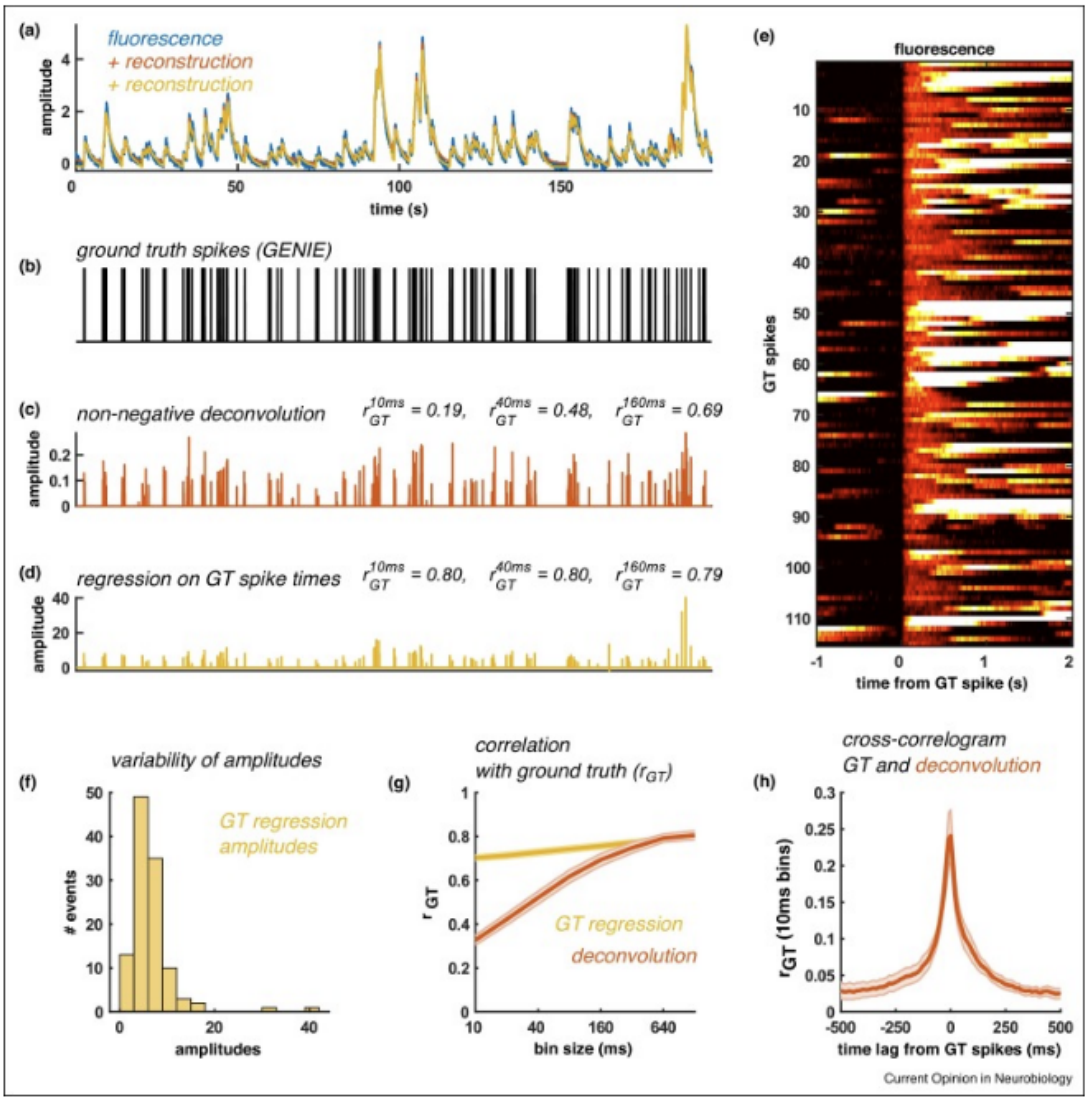
You can add frames to this list of ops['badframes'] by creating a numpy array (0-based, the first frame is zero) and save it as bad_frames.npy in the folder with your tiffs (if you have multiple folders, save it in the FIRST folder with tiffs, or if you have subfolders with 'look_one_level_down' it should be in the parent folder). See this page inputs for more info.

8.2 Deconvolution means what?

There is a lot of misinformation about what deconvolution is and what it isn't. Some issues on this: #267, #202, #169

TLDR: Deconvolution will NOT tell you how many spikes happened in a neuron - there is too much variability in the calcium signal to know that. Our deconvolution has NO sparsity constraints and we recommend against thresholding the output values because they contain information about approximately how many spikes occurred. We found that using the raw deconvolved values gave us the most reliable responses to stimuli (as measured by signal variance).

See this figure from our review paper for reference:



Download : Download high-res image (1MB) Download : Download full-size image

Figure 4. The limits of spike deconvolution. **(a)** Recorded fluorescence trace in blue, and reconstructions from spike deconvolution (red) and from direct regression on spike times (yellow). **(b)** Ground truth spike times recorded by simultaneous electrophysiology [47]. **(c)** Spike deconvolution result, and the correlation with ground truth in bins of 10, 40 and 160 ms (non-negative, using the OASIS implementation [48•]). **(d)** Regression on ground truth spike times to obtain "optimal" amplitudes. **(e)** Fluorescence traces aligned to spike times and baseline subtracted at 0 timelag. **(f)** Variability of single-spike "optimal" amplitudes from GT regression sets the limit of possible spike deconvolution performance. **(g)** Correlation of binned ground truth spike trains with deconvolved and "optimal" amplitudes. At large bin sizes, deconvolution saturates the possible maximal performance. **(h)** Failures of deconvolution at small bin sizes correspond to ambiguities of spike timing on the order of 100 ms, reflected in the shape of the cross-correlogram between deconvolved spike trains and ground truth.

Long answer (mostly from #267):

There is an unknown scaling factor between fluorescence and # spikes, which is very hard to estimate. This is true both for the raw dF, or dF/F, and for the deconvolved amplitudes, which we usually treat as arbitrary units. The same calcium amplitude transient may have been generated by a single spike, or by a burst of many spikes, and for many neurons it is very hard to disentangle these, so we don't try. Few spike deconvolution algorithms try to estimate single spike amplitude (look up "MLspike"), but we are in general suspicious of the results, and usually have no need for absolute numbers of spikes.

As for the question of thresholding, we always recommend not to, because you will lose information. More importantly, you will treat 1-spike events the same as 10-spike events, which isn't right. There are several L0-based methods that return discrete spike times, including one we've developed in the past, which we've since shown to be worse than the vanilla OASIS method (see our Jneurosci paper). We do not use L1 penalties either, departing from the original OASIS paper, because we found that hurts in all cases (see Jneurosci).

How do you compare across cells then if these values are arbitrary to some extent?

If you need to compare between cells, you would usually be comparing effect sizes, such as tuning width, SNR, choice index etc. which are relative quantities, i.e. firing rate 1 / firing rate 2. If you really need to compare absolute firing rates, then you need to normalize the deconvolved events by the F0 of the fluorescence trace, because the dF/F should be more closely related to absolute firing rate. Computing the F0 has problems in itself, as it may sometimes be estimated to be negative or near-zero for high SNR sensors like gcamp6 and 7. You could take the mean F0 before subtracting the neuropil and normalize by that, and then decide on a threshold to use across all cells, but at that point you need to realize these choices will affect your result and interpretation, so you cannot really put much weight on them. For these reasons, I would avoid making statements about absolute firing rates from calcium imaging data, and I don't know of many papers that make such statements.

8.3 Multiple functional channels

If you have two channels and they both have functional activity, then to process both you need to run suite2p in a jupyter notebook. Here is an example notebook for that purpose: multiple_functional_channels.ipynb

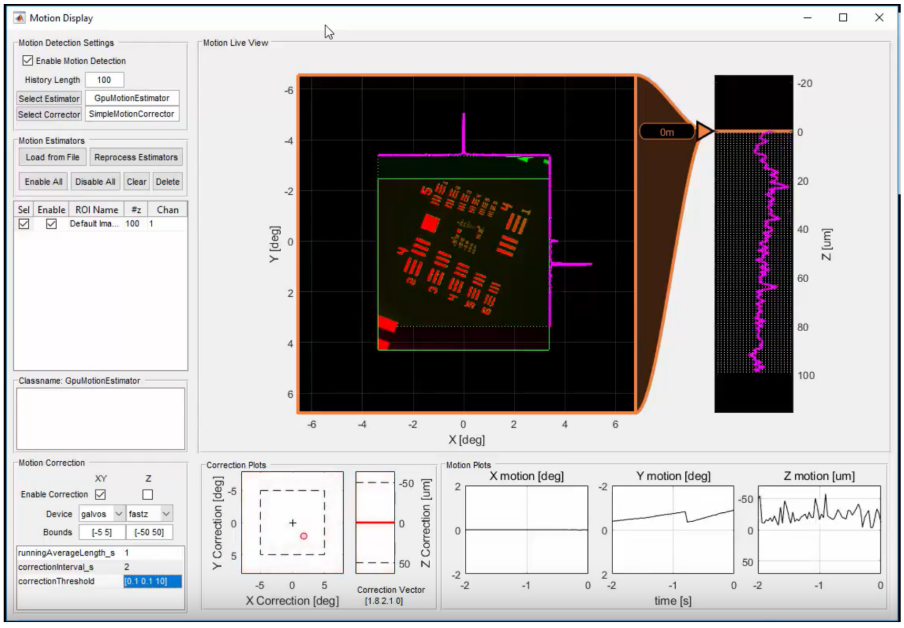
8.4 Z-drift

It's not frequently asked about but it should be :)

In the GUI in the "View registered binary" window you can now load in a z-stack and compute the z-position of the recording across time.

Scanimage now can do z-correction ONLINE for you!

Scanimage 2018 (free version) has automated z-drift correction!



8.5 No signals in manually selected ROIs

If you circle an ROI in the manual selection GUI on top of another ROI and ops['allow_overlap'] is 0 or False, then that ROI will have no activity because it has no non-overlapping pixels. You can change this after processing with

```
import numpy as np
np.load('ops.npy', allow_pickle=True).item()
np.save('ops_original.npy', ops)
ops['allow_overlap'] = True
np.save('ops.npy', ops)
```

Thanks @marysethomas, see full issue here: #651,

CHAPTER

NINE

REGISTRATION

You can register your frames using the first channel of the recording, or using the second channel. Say your first channel shows GCaMP and your second channel shows td-Tomato, you might want to use the second channel for registration if it has higher SNR. If so, set `ops['align_by_chan']=2`. Otherwise, leave `ops['align_by_chan']=1` (default).

Your registered output for the first channel of the recording will be saved as `data.bin` in the suite2p output folder. If you run the pipeline using more than 2 channels(`ops['nchannels'] = 2`), you will also see a registered output for the second channel's data saved as `data_chan2.bin`.

9.1 Finding a target reference image

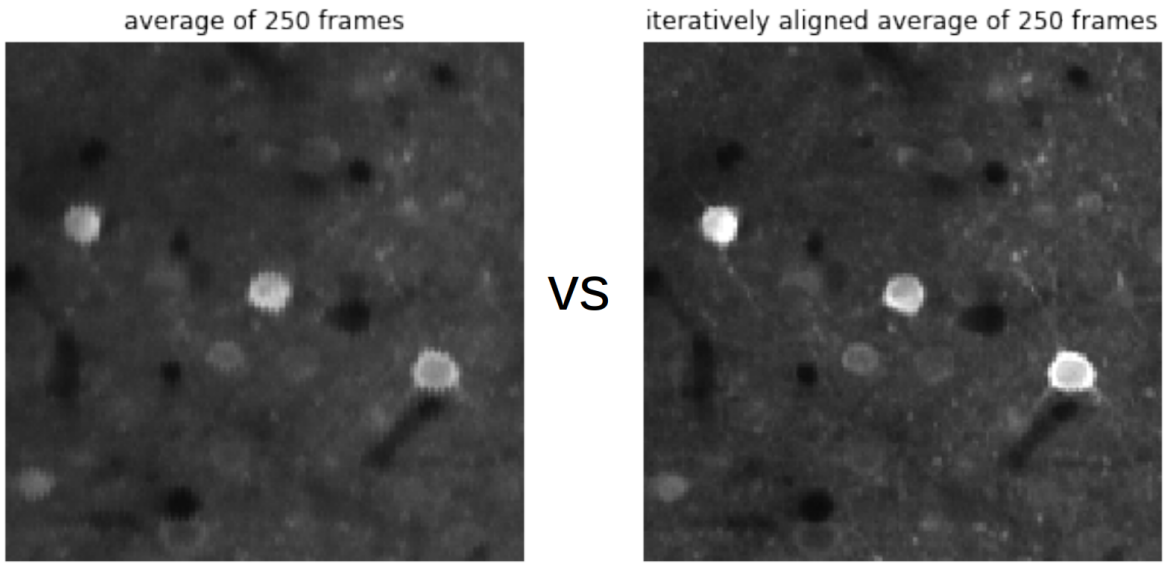
To perform registration, we need a reference image to align all the frames to. This requires an initial alignment step. Consider we just took the average of a subset of frames. Because these frames are not motion-corrected, the average will not be crisp - there will be fuzzy edges because objects in the image have been moving around across the frames. Therefore, we do an initial iterative alignment procedure on a random subset of frames in order to get a crisp reference image for registration. We first take `ops['nimg_init']` random frames of the movie. Then from those frames, we take the top 20 frames that are most correlated to each other and take the mean of those frames as our initial reference image. Then we refine this reference image iteratively by aligning all the random frames to the reference image, and then recomputing the reference image as the mean of the best aligned frames.

The function that performs these steps can be run as follows (where ops needs the reg_file, Ly, Lx, and nimg_init parameters):

```
from suite2p.registration import register

refImg = register.pick_initial_reference(ops)
```

Here is an example reference image on the right, compared to just taking the average of a random subset of frames (on the left):



If the reference image doesn't look good, try increasing `ops['nimg_init']`.

9.2 Registering the frames to the reference image

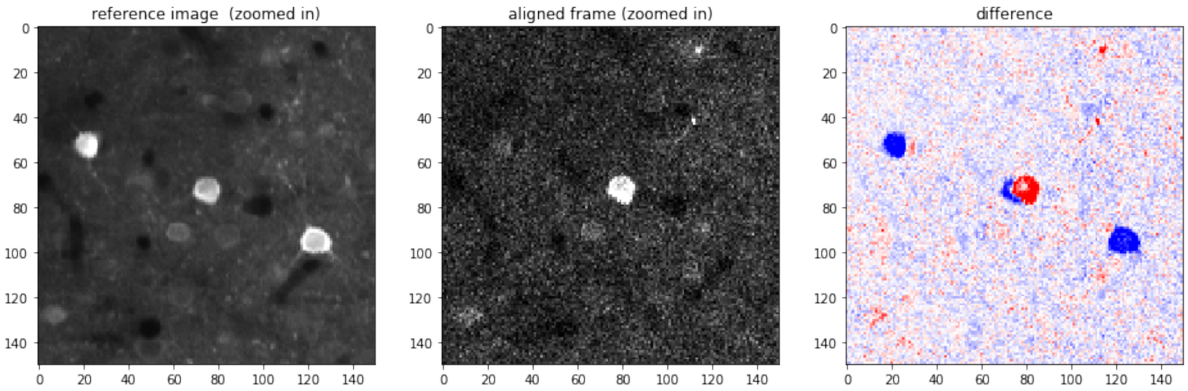
Once the reference image is obtained, we align each frame to the reference image. The frames are registered in batches of size `ops['batch_size']` (default is 500 frames per batch).

We first perform rigid registration (assuming that the whole image shifts by some (dy,dx)), and then optionally after that we perform non-rigid registration (assuming that subsegments of the image shift by separate amounts). To turn on non-rigid registration, set `ops['nonrigid']=True`. We will outline the parameters of each registration step below.

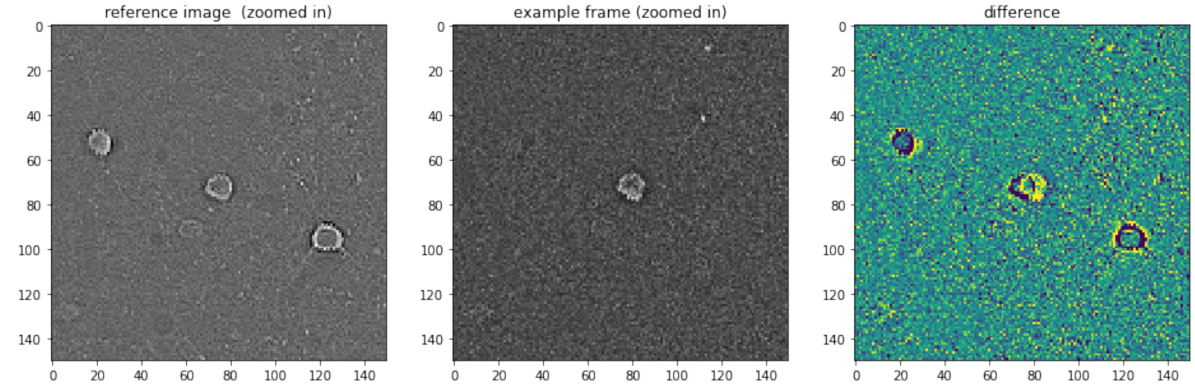
9.3 1. Rigid registration

Rigid registration computes the shifts between the frame and the reference image using phase-correlation. We have found on simulated data that phase-correlation is more accurate than cross-correlation. Phase-correlation is a well-established method to compute the relative movement between two images. Phase-correlation normalizes the Fourier spectra of the images before multiplying them (whereas cross-correlation would just multiply them). This normalization emphasizes the correlation between the higher frequency components of the images, which in most cases makes it more robust to noise.

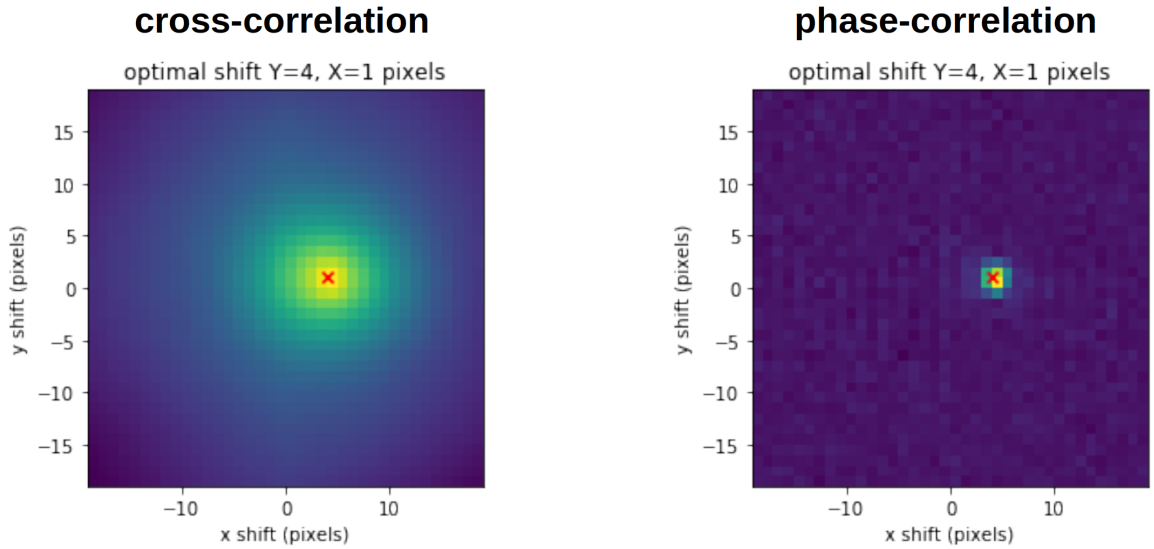
Cross-correlation



Phase-correlation



Comparison



You can set a maximum shift size using the option `ops['maxregshift']`. By default, it is 0.1, which means that the maximum shift of the frame from the reference in the Y direction is `0.1 * ops['Ly']` and in X is `0.1 * ops['Lx']` where Ly and Lx are the Y and X sizes of the frame.

After computing the shifts, the frames are shifted in the Fourier domain (allowing subpixel shifts of the images). The shifts are saved in `ops['yoff']` and `ops['xoff']` for y and x shifts respectively. The peak of the phase-correlation of each frame with the reference image is saved in `ops['corrXY']`.

You can run this independently from the pipeline, if you have a reference image (ops requires the parameters non-rigid=False, num_workers, and maxregshift):

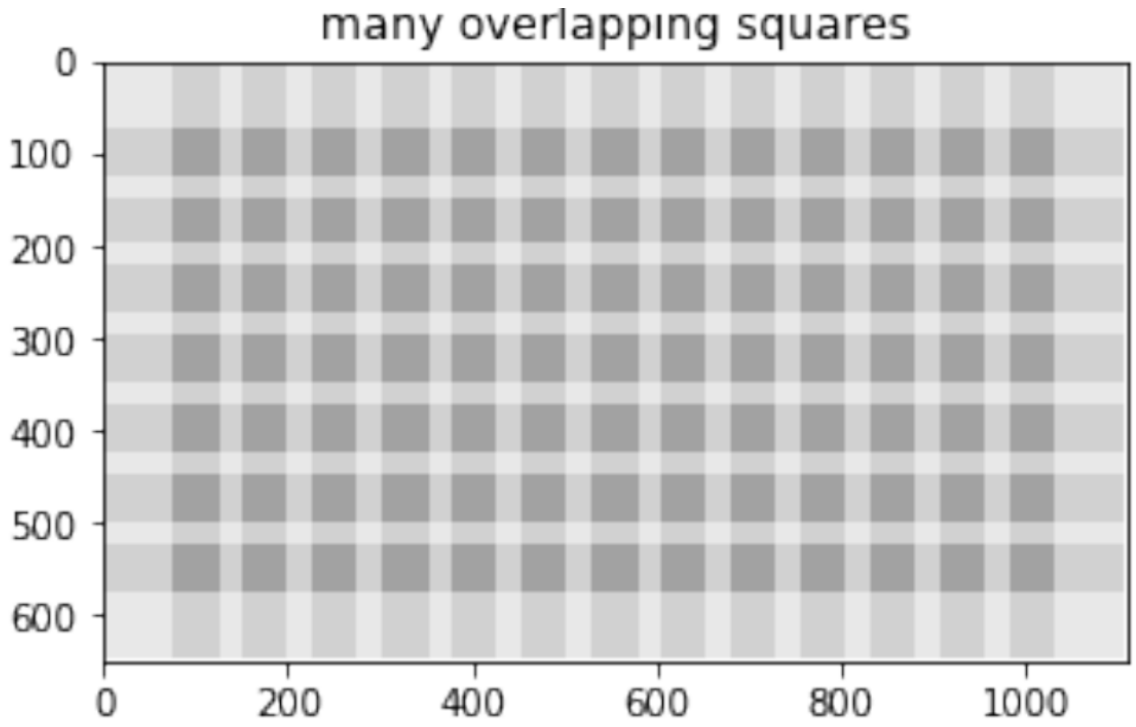
```
maskMul,maskOffset,cfRefImg = register.prepare_masks(refImg)
refAndMasks = [maskMul,maskOffset,cfRefImg]
aligned_data, yshift, xshift, corrXY, yxnr = register.phasecorr(data, refAndMasks, ops)
```

(see bioRxiv preprint comparing cross/phase here)

9.4 2. Non-rigid registration (optional)

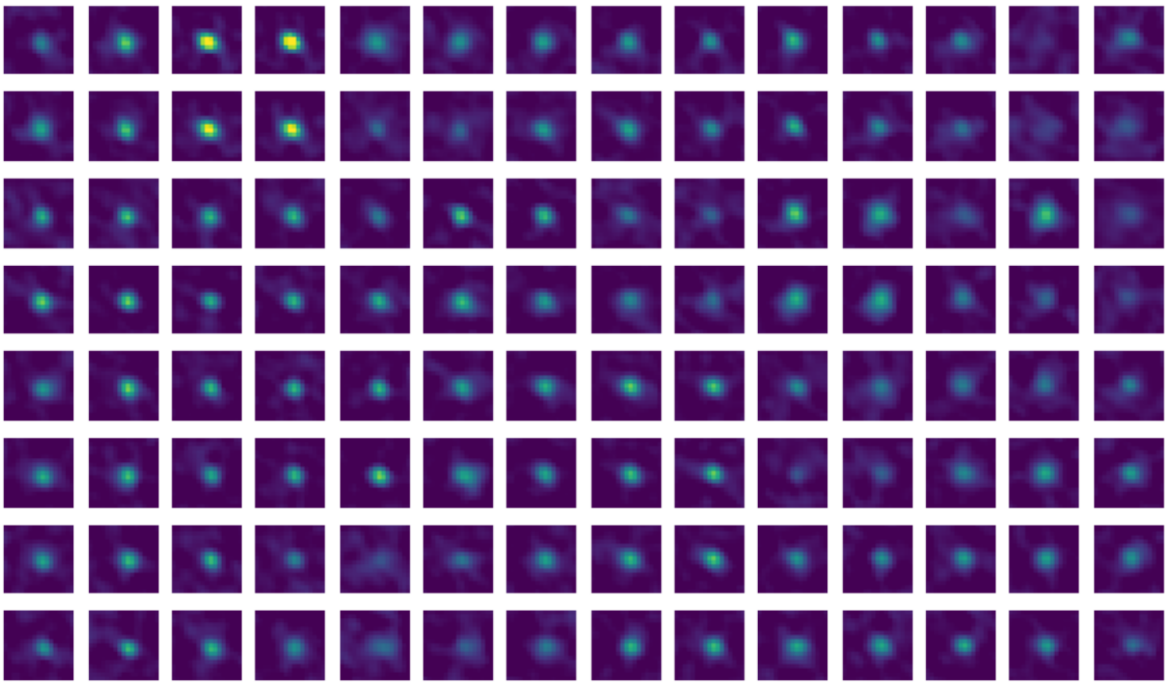
If you run rigid registration and find that there is still motion in your frames, then you should run non-rigid registration. Non-rigid registration divides the image into subsections and computes the shift of each subsection (called a block) separately. Non-rigid registration will approximately double the registration time.

The size of the blocks to divide the image into is defined by `ops['block_size'] = [128,128]` which is the size in Y and X in pixels. If Y is the direction of line-scanning for 2p imaging, you may want to divide it into smaller blocks in that direction.

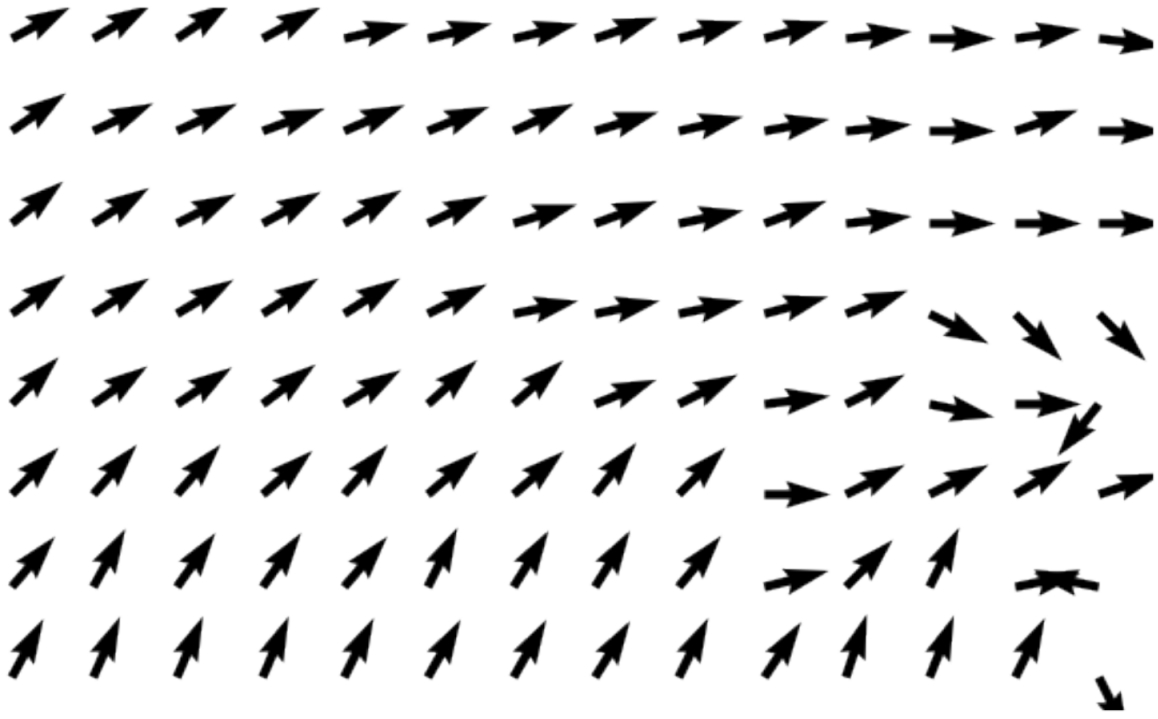


Each block is able to shift up to `ops['maxregshiftNR']` pixels in Y and X. We recommend to keep this small unless you're in a very high signal-to-noise ratio regime and your motion is very large. For subpixel shifts, we use Kriging interpolation and run it on each block.

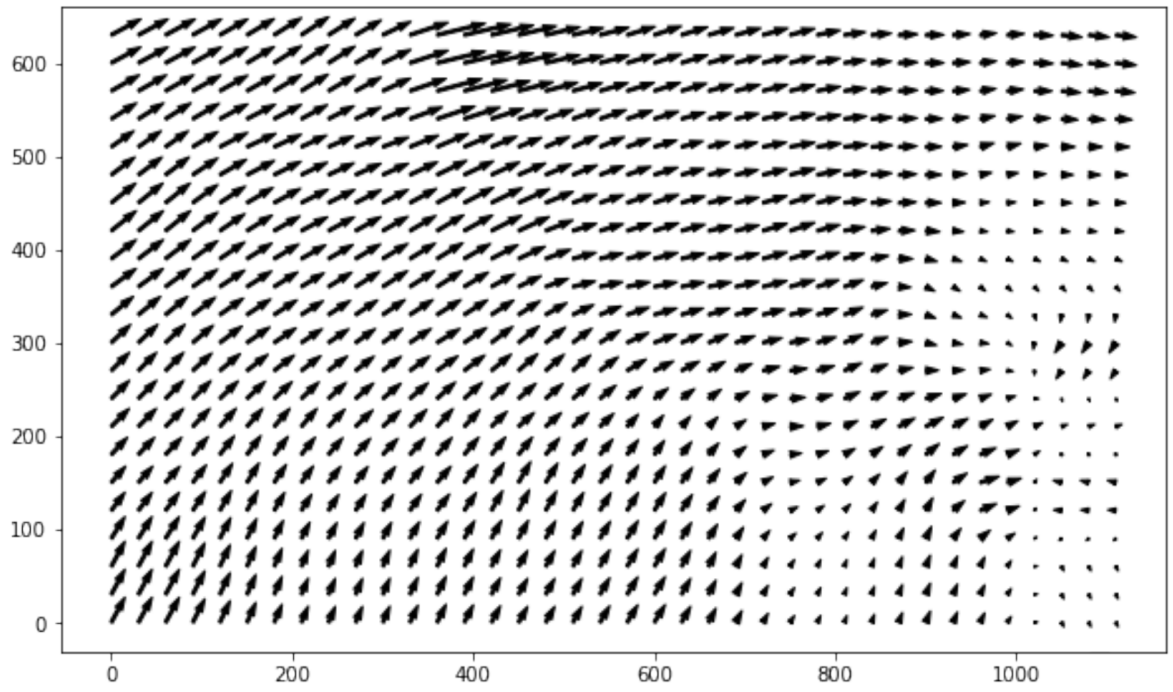
Phase correlation of each block:



Shift of each block from phase corr:



In a low signal-to-noise ratio regime, there may be blocks which on a given frame do not have sufficient information from which to align with the reference image. We compute a given block's maximum phase correlation with the reference block, and determine how much greater this max is than the surrounding phase correlations. The ratio between these two is defined as the `snr` of that block at that given time point. We smooth over high snr blocks and use bilinear interpolation to upsample create the final shifts:



We then use bilinear interpolation to warp the frame using these shifts.

9.5 Metrics for registration quality

The inputs required for PC metrics are the following fields in ops: `nframes`, `Ly`, `Lx`, `reg_file`. Your movie must have at least 1500 frames in each plane for the metrics to be calculated. You can run on the red channel (ops['reg_file_chan2']) if use_red=True. The outputs saved from the PC metrics are `ops['regDX']`, `ops['tPC']` and `ops['regPC']`.

```
from suite2p.registration import metrics

ops = metrics.get_pc_metrics(ops, use_red=False)
```

`ops['tPC']` are the time courses of each of the principal components of the registered movie. Note the time-course is not the entire movie, it's only the subset of frames used to compute the PCs (2000-5000 frames equally sampled throughout the movie).

`ops['regPC']` are computed from the spatial principal components of the registered movie. `ops['regPC'][0,0,:,:]` is the average of the top 500 frames of the 1st PC, `ops['regPC'][1,0,:,:]` is the average of the bottom 500 frames of the 1st PC. `ops['regDX']` quantifies the movement in each PC (`iPC`) by registering `ops['regPC'][0,iPC,:,:]` and `ops['regPC'][1,iPC,:,:]` to the reference image `ops['refImg']` (if available, if not the mean of all the frames is used as the reference image) and computing the registration shifts.

Here's a twitter thread with multiple examples.

9.5.1 CLI Script

Suite2p provides a CLI (Command-Line Interface) script that calculates the registration metrics for a given input tif and outputs some statistics on those metrics. You can use this script to determine the quality of registration and tune your registration parameters (e.g: determine if non-rigid registration is necessary).

To run the script, use the following command:

```
$ reg_metrics <INSERT_OPS_DATA_PATH> # Add --tiff_list <INSERT_INPUT_TIF_FILENAME_HERE>.tif to select a
```

Once you run the `reg_metrics` command, registration will be performed for the input file with default ops parameters and an output similar to the following will be shown:

```
# Average NR refers to the average nonrigid offsets of the blocks for a PC
# Max NR refers to the max nonrigid offsets of the blocks for a PC
Plane 0:
Avg_Rigid: 0.000000     Avg_Average NR: 0.028889        Avg_Max NR: 0.120000
Max_Rigid: 0.000000     Max_Average NR: 0.044444        Max_Max NR: 0.200000
```

For each `nplane`, these statistics (Average and Max) are calculated across PCs on the offsets found in `ops['regDX']`. If the registration works perfectly and most of the motion is removed from the registered dataset, these scores should all be very close to zero.

Important: Make sure to also inspect the registered video to check the quality of registration. You can see an example of how this is done in the GUI here.

You may notice that upon visual inspection, the registered video may look fine/contain little motion even if the statistics are not close to zero. You should always visually check the registration output and prioritize what your eyes say over what the CLI script reports.

Note: All suite2p registration settings can be modified in this CLI script. Just pass the setting with its value as an optional argument. For instance,

```
$ reg_metrics path_to_data_tif --nplanes 2 --smooth_sigma 1.2
```

runs the script with `ops['nplanes'] = 2` and `ops['smooth_sigma'] = 1.2`. You can see all the arguments `reg_metrics` takes with the following command:

```
$ reg_metrics --help
```

CHAPTER

TEN

CELL DETECTION

10.1 Summary

The cell detection algorithm consists of reducing the dimensionality of the data (principal components computation), smoothing spatial principal components, finding peaks in these components, and extending ROIs spatially around these peaks. On each iteration of peak extraction, the neuropil is estimated from large spatial masks and subtracted from the spatial components. This is to improve cell detection and to help avoid extracting neuropil components with large spatial extents.

10.2 SVDs (= PCs) of data

Before computing the principal components of the movie, we bin the data such that we have at least as many frames to take the SVD of as specified in the option `ops['navg_frames_svd']`. The bin size will be the maximum of `nframes/ops['navg_frames_svd']` and `ops['tau'] * ops['fs']` (the number of samples per transient). We then bin the movie into this bin size and subtract the mean of the binned movie across time. Then we smooth the movie in Y and X with a gaussian filter of standard deviation `sig = ops['diameter']/10`. The we normalize the pixels by their noise variance. The noise variance is variance of each pixel in the movie across time (at least 1e-10). Then we compute the covariance of the movie (`mov @ mov.T`). Then we compute the SVD of the covariance and keep the top `ops['nsvd_for_roi']` spatial components (components that are Y x X).

The function that performs this is `celldetect2.getSVDdata` and it requires the ops described above, and Ly, Lx, yrange, xrange, and a reg_file location.

10.3 Sourcery

After the spatial components are found, we perform an iterative algorithm to find the cells in the components. Each iteration consists of the following steps:

1. **Smoothing of spatial components**: The components are smoothed with a Gaussian filter in Y and X with standard deviation `sig = ops['diameter']` (this matrix is called `us`). Note that diameter can be a list (for unequal pixel/um in Y and X). Next the mean of the squared smoothed components is computed. The mean of the squared un-smoothed components is also computed. The *correlation map* is defined as the element-wise division of the smoothed components by the unsmoothed components. The function that computes the correlation map is `celldetect2.getVmap`.
2. **Detection of peaks in correlation map**: On each iteration, up to 200 peaks are extracted from the correlation map. These are the largest remaining peaks such that they are greater than the threshold, which is set to be proportional to the median of the peaks in the whole correlation map: `ops['threshold_scaling'] *`

`np.median(peaks[peaks>1e-4])`. The initial activity `code` for this newly detected peak is the value of `us` (Gaussian smoothed PCs) at this peak. This is a vector of values across the PCs (nPCs in length).

3. **ROI extension**: The ROI is iteratively extended around its currently defined pixels +/- 1 in each direction. First, the new pixel weights (`lam`) of the extended ROI are computed. The weights `lam` are the unsmoothed PCs projected into the `code` dimension. The pixels that are greater than `max(lam)/5` are kept. The `lam`'s are normalized to be unit norm. The new `code` is recomputed from the new weights, and is the unsmoothed PCs projected onto the `lam` weights. Then this extension procedure is repeated until no pixels are greater than `max(lam)/5`.
4. **Neuropil computation**: Now that the new codes are computed, the neuropil is estimated. We set spatial basis functions for the neuropil, which are raised cosines that tile the FOV. The parameter `ops['ratio_neuropil']` determines how big you expect the neuropil basis functions to be relative to the cell diameter (`ops['diameter']`). The default is 6. This results in a tiling of 7x7 raised cosines if your FOV is 512x512 pixels and your diameter is 12 pixels. For one-photon recordings, we recommend setting `ops['ratio_neuropil']` to 2 or 3. Next we perform regression to compute the contribution of the neuropil on the PCs, and we subtract the estimated neuropil contribution from the `U` PCs. And these steps are repeated until the stopping criterion is reached.

Stopping criterion: The number of cells detected in the first iteration is defined as `Nfirst`. The cell detection is stopped if the number of cells detected in the current iteration is less than `Nfirst/10` or if the iteration is the last iteration (defined by `ops['max_iterations']`).

Refinement step: We remove masks which have more than a fraction `ops['max_overlap']` of their pixels overlapping with other masks. Also, if `ops['connected']=1`, then only the connected regions of ROIs are kept. If you are looking for dendritic components, you may want to set `ops['connected']=0`.

CHAPTER

ELEVEN

SIGNAL EXTRACTION

CHAPTER

TWELVE

SPIKE DECONVOLUTION

Our spike deconvolution in the pipeline is based on the OASIS algorithm (see OASIS paper). We run it with only a non-negativity constraint - no L0/L1 constraints (see this paper for more details on why).

We first baseline the traces using the rolling max of the rolling min. Here is an example of how the pipeline processes the traces (and how to run your own data separately if you want):

```
# compute deconvolution
from suite2p.extraction import dcnv
import numpy as np

tau = 1.0 # timescale of indicator
fs = 30.0 # sampling rate in Hz
neucoeff = 0.7 # neuropil coefficient
# for computing and subtracting baseline
baseline = 'maximin' # take the running max of the running min after smoothing with␣
↪gaussian
sig_baseline = 10.0 # in bins, standard deviation of gaussian with which to smooth
win_baseline = 60.0 # in seconds, window in which to compute max/min filters

ops = {'tau': tau, 'fs': fs, 'neucoeff': neucoeff,
       'baseline': baseline, 'sig_baseline': sig_baseline, 'win_baseline': win_baseline}

# load traces and subtract neuropil
F = np.load('F.npy')
Fneu = np.load('Fneu.npy')
Fc = F - ops['neucoeff'] * Fneu

# baseline operation
Fc = dcnv.preprocess(
     F=Fc,
     baseline=ops['baseline'],
     win_baseline=ops['win_baseline'],
     sig_baseline=ops['sig_baseline'],
     fs=ops['fs'],
     prctile_baseline=ops['prctile_baseline']
 )

# get spikes
spks = dcnv.oasis(F=Fc, batch_size=ops['batch_size'], tau=ops['tau'], fs=ops['fs'])
```

CHAPTER

THIRTEEN

SUITE2P.IO PACKAGE

CHAPTER

FOURTEEN

SUITE2P.REGISTRATION PACKAGE

CHAPTER

FIFTEEN

SUITE2P.DETECTION PACKAGE

15.1 Submodules

15.2 suite2p.detection.anatomical module

15.3 suite2p.detection.chan2detect module

suite2p.detection.chan2detect.**cellpose_overlap**(*stats*, *mimg2*)

suite2p.detection.chan2detect.**correct_bleedthrough**(*Ly*, *Lx*, *nblks*, *mimg*, *mimg2*)

suite2p.detection.chan2detect.**detect**(*ops*, *stats*)

suite2p.detection.chan2detect.**intensity_ratio**(*ops*, *stats*)

compute pixels in cell and in area around cell (including overlaps) (exclude pixels from other cells)

suite2p.detection.chan2detect.**quadrant_mask**(*Ly*, *Lx*, *ny*, *nx*, *sT*)

15.4 suite2p.detection.denoise module

15.5 suite2p.detection.detect module

15.6 suite2p.detection.metrics module

15.7 suite2p.detection.sourcery module

suite2p.detection.sourcery.**circleMask**(*d0*)

creates array with indices which are the radius of that x,y point

Parameters

d0 – (patch of (-d0,d0+1) over which radius computed

Returns

- *rs* – array (2*d0+1,2*d0+1) of radii
- *dx* – indices in rs where the radius is less than d0

- *dy* – indices in rs where the radius is less than d0

suite2p.detection.sourcery.**connected_region**(*stat*, *ops*)

suite2p.detection.sourcery.**create_neuropil_basis**(*ops*, *Ly*, *Lx*)

computes neuropil basis functions

Parameters

- **ops** – ratio_neuropil, tile_factor, diameter, neuropil_type
- **Ly** (*int*) –
- **Lx** (*int*) –

Returns

basis functions (pixels x nbasis functions)

Return type

S

suite2p.detection.sourcery.**drawClusters**(*stat*, *ops*)

suite2p.detection.sourcery.**extendROI**(*ypix*, *xpix*, *Ly*, *Lx*, *niter=1*)

suite2p.detection.sourcery.**getSVDdata**(*mov*, *ops*)

suite2p.detection.sourcery.**getSVDproj**(*mov*, *ops*, *u*)

suite2p.detection.sourcery.**getStU**(*ops*, *U*)

suite2p.detection.sourcery.**getVmap**(*Ucell*, *sig*)

suite2p.detection.sourcery.**get_connected**(*Ly*, *Lx*, *stat*)

grow i0 until it cannot grow any more

suite2p.detection.sourcery.**get_stat**(*ops*, *stats*, *Ucell*, *codes*, *frac=0.5*)

computes statistics of cells found using sourcery

Parameters

- **Ly** –
- **Lx** –
- **d0** –
- **mPix** (*(pixels,ncells)*) –
- **mLam** (*(weights,ncells)*) –
- **codes** (*(ncells,nsvd)*) –
- **Ucell** (*(nsvd,Ly,Lx)*) –

Returns

assigned to stat: ipix, ypix, xpix, med, npix, lam, footprint, compact, aspect_ratio, ellipse

Return type

stat

suite2p.detection.sourcery.**iter_extend**(*ypix*, *xpix*, *Ucell*, *code*, *refine=-1*, *change_codes=False*)

suite2p.detection.sourcery.**localMax**(*V*, *footprint*, *thres*)

find local maxima of V (correlation map) using a filter with (usually circular) footprint

Parameters

- **V** –
- **footprint** –
- **thres** –

Returns

i,j

Return type

indices of local max greater than thres

suite2p.detection.sourcery.**localRegion**(*i*, *j*, *dy*, *dx*, *Ly*, *Lx*)

returns valid indices of local region surrounding (i,j) of size (dy.size, dx.size)

suite2p.detection.sourcery.**minDistance**(*inputs*)

suite2p.detection.sourcery.**morphOpen**(*V*, *footprint*)

computes the morphological opening of V (correlation map) with circular footprint

suite2p.detection.sourcery.**pairwiseDistance**(*y*, *x*)

suite2p.detection.sourcery.**postprocess**(*ops*, *stat*, *Ucell*, *codes*)

suite2p.detection.sourcery.**r_squared**(*yp*, *xp*, *ypix*, *xpix*, *diam_y*, *diam_x*, *estimator=<function median>*)

suite2p.detection.sourcery.**sourcery**(*mov*, *ops*)

suite2p.detection.sourcery.**sub2ind**(*array_shape*, *rows*, *cols*)

15.8 suite2p.detection.sparsedetect module

class suite2p.detection.sparsedetect.**EstimateMode**(*value*)

Bases: Enum

An enumeration.

Estimated = 'estimated'

Forced = 'FORCED'

suite2p.detection.sparsedetect.**add_square**(*yi*, *xi*, *lx*, *Ly*, *Lx*)

return square of pixels around peak with norm 1

Parameters

- **yi** (*int*) – y-center
- **xi** (*int*) – x-center
- **lx** (*int*) – x-width
- **Ly** (*int*) – full y frame
- **Lx** (*int*) – full x frame

Returns

-
- **y0** (*array*) – pixels in y
- **x0** (*array*) – pixels in x
- **mask** (*array*) – pixel weightings

suite2p.detection.sparsedetect.**estimate_spatial_scale**(*I*)

Return type
int

suite2p.detection.sparsedetect.**extendROI**(*ypix*, *xpix*, *Ly*, *Lx*, *niter=1*)

extend ypix and xpix by niter pixel(s) on each side

suite2p.detection.sparsedetect.**extend_mask**(*ypix*, *xpix*, *lam*, *Ly*, *Lx*)

extend mask into 8 surrrounding pixels

suite2p.detection.sparsedetect.**find_best_scale**(*I*, *spatial_scale*)

Returns best scale and estimate method (if the spatial scale was forced (if positive) or estimated (the top peaks).

Return type
Tuple[int, *EstimateMode*]

suite2p.detection.sparsedetect.**iter_extend**(*ypix*, *xpix*, *mov*, *Lyc*, *Lxc*, *active_frames*)

extend mask based on activity of pixels on active frames ACTIVE frames determined by threshold

Parameters

- **ypix** (*array*) – pixels in y
- **xpix** (*array*) – pixels in x
- **mov** (*2D array*) – binned residual movie [nbinned x Lyc*Lxc]
- **active_frames** (*1D array*) – list of active frames

Returns

- **ypix** (*array*) – extended pixels in y
- **xpix** (*array*) – extended pixels in x
- **lam** (*array*) – pixel weighting

suite2p.detection.sparsedetect.**multiscale_mask**(*ypix0*, *xpix0*, *lam0*, *Lyp*, *Lxp*)

suite2p.detection.sparsedetect.**neuropil_subtraction**(*mov*, *filter_size*)

Returns movie subtracted by a low-pass filtered version of itself to help ignore neuropil.

Return type
None

suite2p.detection.sparsedetect.**sparsery**(*mov*, *high_pass*, *neuropil_high_pass*, *batch_size*, *spatial_scale*, *threshold_scaling*, *max_iterations*, *percentile=0*)

Returns stats and ops from "mov" using correlations in time.

Return type
Tuple[Dict[str, Any], List[Dict[str, Any]]]

suite2p.detection.sparsedetect.**square_convolution_2d**(*mov*, *filter_size*)

Returns movie convolved by uniform kernel with width "filter_size".

Return type
ndarray

suite2p.detection.sparsedetect.**two_comps**(*mpix0*, *lam*, *Th2*)

check if splitting ROI increases variance explained

Parameters

- **mpix0** (*2D array*) – binned movie for pixels in ROI [nbinned x npix]
- **lam** (*array*) – pixel weighting
- **Th2** (*float*) – intensity threshold

Returns

-
- **vrat** (*array*) – extended pixels in y
- **ipick** (*tuple*) – new ROI

15.9 suite2p.detection.stats module

class suite2p.detection.stats.**EllipseData**(*mu*, *cov*, *radii*, *ellipse*, *dy*, *dx*)

Bases: tuple

property area

property aspect_ratio: float

cov: float

Alias for field number 1

dx: int

Alias for field number 5

dy: int

Alias for field number 4

ellipse: ndarray

Alias for field number 3

mu: float

Alias for field number 0

radii: Tuple[float, float]

Alias for field number 2

property radius: float

class suite2p.detection.stats.**ROI**(*ypix*, *xpix*, *lam*, *med*, *do_crop*, *rsort=array([0., 1., 1., ..., 42.42640687, 42.42640687, 42.42640687])*)

Bases: object

ROI(ypix: 'np.ndarray', xpix: 'np.ndarray', lam: 'np.ndarray', med: 'np.ndarray', do_crop: 'bool', rsort: 'np.ndarray' = array([0. , 1. , 1. , ..., 42.42640687, 42.42640687, 42.42640687]))

do_crop: bool

classmethod filter_overlappers(*rois*, *overlap_image*, *max_overlap*)

returns logical array of rois that remain after removing those that overlap more than fraction max_overlap from overlap_img.

Return type
List[bool]

fit_ellipse(*dy*, *dx*)

Return type
EllipseData

classmethod from_stat_dict(*stat*, *do_crop=True*)

Return type
ROI

classmethod get_mean_r_squared_normed_all(*rois*, *first_n=100*)

Return type
ndarray

classmethod get_n_pixels_normed_all(*rois*, *first_n=100*)

Return type
ndarray

classmethod get_overlap_count_image(*rois*, *Ly*, *Lx*)

Return type
ndarray

get_overlap_image(*overlap_count_image*)

Return type
ndarray

lam: ndarray

property mean_r_squared: float

property mean_r_squared0: float

property mean_r_squared_compact: float

med: ndarray

property n_pixels: int

property npix_soma: int

ravel_indices(*Ly*, *Lx*)

Returns a 1-dimensional array of indices from the ypix and xpix coordinates, assuming an image shape Ly x Lx.

Return type
ndarray

rsort: ndarray = array([0. , 1. , 1. , ..., 42.42640687, 42.42640687, 42.42640687])

property solidity: float

property soma_crop: ndarray

classmethod stats_dicts_to_3d_array(*stats*, *Ly*, *Lx*, *label_id=False*)

Outputs a (roi x Ly x Lx) float array from a sequence of stat dicts. Convenience function that repeatedly calls ROI.from_stat_dict() and ROI.to_array() for all rois.

Parameters

- **stats** (*List of dictionary "ypix", "xpix", "lam"*) –
- **Ly** (*y size of frame*) –
- **Lx** (*x size of frame*) –
- **label_id** (*whether array should be an integer value indicating ROI id or just 1 (indicating precence of* ROI*).*) –

to_array(*Ly*, *Lx*)

Returns a 2D boolean array of shape (Ly x Lx) indicating where the roi is located.

Return type
ndarray

xpix: ndarray

ypix: ndarray

suite2p.detection.stats.**aspect_ratio**(*width*, *height*, *offset=0.01*)

Return type
float

suite2p.detection.stats.**count_overlaps**(*Ly*, *Lx*, *ypixs*, *xpixs*)

Return type
ndarray

suite2p.detection.stats.**distance_kernel**(*radius*)

Returns 2D array containing geometric distance from center, with radius "radius"

Return type
ndarray

suite2p.detection.stats.**filter_overlappers**(*ypixs*, *xpixs*, *overlap_image*, *max_overlap*)

returns ROI indices that remain after removing those that overlap more than fraction max_overlap from overlap_img.

Return type
List[bool]

suite2p.detection.stats.**fitMVGaus**(*y*, *x*, *lam0*, *dy*, *dx*, *thres=2.5*, *npts=100*)

computes 2D gaussian fit to data and returns ellipse of radius thres standard deviations. :type y: :param y: pixel locations in y :type y: float, array :type x: :param x: pixel locations in x :type x: float, array :type lam0: :param lam0: weights of each pixel :type lam0: float, array

Return type
EllipseData

suite2p.detection.stats.**mean_r_squared**(*y*, *x*, *estimator=<function median>*)

Return type
float

suite2p.detection.stats.**median_pix**(*ypix*, *xpix*)

suite2p.detection.stats.**norm_by_average**(*values*, *estimator=<function mean>*, *first_n=100*, *offset=0.0*)

Returns array divided by the (average of the "first_n" values + offset), calculating the average with "estimator".

Return type
ndarray

suite2p.detection.stats.**roi_stats**(*stat*, *Ly*, *Lx*, *aspect=None*, *diameter=None*, *max_overlap=None*, *do_crop=True*)

computes statistics of ROIs :type stat: :param stat: "ypix", "xpix", "lam" :type stat: dictionary :param FOV size: :type FOV size: (Ly, Lx) :type aspect: :param aspect: :type aspect: aspect ratio of recording :type diameter: :param diameter: :type diameter: (dy, dx)

Returns
stat – adds "npix", "npix_norm", "med", "footprint", "compact", "radius", "aspect_ratio"

Return type
dictionary

15.10 suite2p.detection.utils module

suite2p.detection.utils.**downsample**(*mov*, *taper_edge=True*)

Returns a pixel-downsampled movie from "mov", tapering the edges of "taper_edge" is True.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to downsample
- **taper_edge** (*bool*) – Whether to taper the edges

Returns
The downsampled frames

Return type
filtered_mov

suite2p.detection.utils.**hp_gaussian_filter**(*mov*, *width*)

Returns a high-pass-filtered copy of the 3D array "mov" using a gaussian kernel.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to filter
- **width** (*int*) – The kernel width

Returns
filtered_mov – The filtered video

Return type
nImg x Ly x Lx

suite2p.detection.utils.**hp_rolling_mean_filter**(*mov*, *width*)

Returns a high-pass-filtered copy of the 3D array "mov" using a non-overlapping rolling mean kernel over time.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to filter
- **width** (*int*) – The filter width

Returns
filtered_mov – The filtered frames

Return type
nImg x Ly x Lx

suite2p.detection.utils.**mask_ious**(*masks_true*, *masks_pred*)

return best-matched masks

Parameters

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

Returns

-
- **iou** (*float, ND-array*) – array of IOU pairs
- **preds** (*int, ND-array*) – array of matched indices
- **iou_all** (*float, ND-array*) – full IOU matrix across all pairs

suite2p.detection.utils.**mask_stats**(*mask*)

median and diameter of mask

suite2p.detection.utils.**match_masks**(*iou*)

suite2p.detection.utils.**square_mask**(*mask*, *ly*, *yi*, *xi*)

crop from mask a square of size ly at position yi,xi

suite2p.detection.utils.**standard_deviation_over_time**(*mov*, *batch_size*)

Returns standard deviation of difference between pixels across time, computed in batches of batch_size.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to filter
- **batch_size** (*int*) – The batch size

Returns
filtered_mov – The statistics for each pixel

Return type
Ly x Lx

suite2p.detection.utils.**temporal_high_pass_filter**(*mov*, *width*)

Returns hp-filtered mov over time, selecting an algorithm for computational performance based on the kernel width.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to filter

- **width** (*int*) – The filter width

Returns
filtered_mov – The filtered frames

Return type
nImg x Ly x Lx

suite2p.detection.utils.**threshold_reduce**(*mov*, *intensity_threshold*)

Returns standard deviation of pixels, thresholded by "intensity_threshold". Run in a loop to reduce memory footprint.

Parameters

- **mov** (*nImg x Ly x Lx*) – The frames to downsample
- **intensity_threshold** (*float*) – The threshold to use

Returns
Vt – The standard deviation of the non-thresholded pixels

Return type
Ly x Lx

15.11 Module contents

CHAPTER

SIXTEEN

SUITE2P.EXTRACTION PACKAGE

16.1 Submodules

16.2 suite2p.extraction.dcnv module

suite2p.extraction.dcnv.**oasis**(*F*, *batch_size*, *tau*, *fs*)

computes non-negative deconvolution

no sparsity constraints

Parameters

- **F** (*float, 2D array*) – size [neurons x time], in pipeline uses neuropil-subtracted fluorescence
- **batch_size** (*int*) – number of frames processed per batch
- **tau** (*float*) – timescale of the sensor, used for the deconvolution kernel
- **fs** (*float*) – sampling rate per plane

Return type
ndarray

Returns

-
- **S** (*float, 2D array*) – size [neurons x time], deconvolved fluorescence

suite2p.extraction.dcnv.**oasis_matrix**(*F*, *v*, *w*, *t*, *l*, *s*, *tau*, *fs*)

spike deconvolution on many neurons parallelized with prange

suite2p.extraction.dcnv.**oasis_trace**(*F*, *v*, *w*, *t*, *l*, *s*, *tau*, *fs*)

spike deconvolution on a single neuron

suite2p.extraction.dcnv.**preprocess**(*F*, *baseline*, *win_baseline*, *sig_baseline*, *fs*, *prctile_baseline=8*)

preprocesses fluorescence traces for spike deconvolution

baseline-subtraction with window "win_baseline"

Parameters

- **F** (*float, 2D array*) – size [neurons x time], in pipeline uses neuropil-subtracted fluorescence
- **baseline** (*str*) – setting that describes how to compute the baseline of each trace

- **win_baseline** (*float*) – window (in seconds) for max filter
- **sig_baseline** (*float*) – width of Gaussian filter in frames
- **fs** (*float*) – sampling rate per plane
- **prctile_baseline** (*float*) – percentile of trace to use as baseline if using *constant_prctile* for baseline

Return type
ndarray

Returns

-
- **F** (*float, 2D array*) – size [neurons x time], baseline-corrected fluorescence

16.3 suite2p.extraction.extract module

16.4 suite2p.extraction.masks module

suite2p.extraction.masks.**create_cell_mask**(*stat*, *Ly*, *Lx*, *allow_overlap=False*)

creates cell masks for ROIs in stat and computes radii

Parameters

- **stat** (*dictionary "ypix", "xpix", "lam"*) –
- **Ly** (*y size of frame*) –
- **Lx** (*x size of frame*) –
- **allow_overlap** (*whether or not to include overlapping pixels in cell masks*) –

Return type
Tuple[ndarray, ndarray]

Returns

-
- **cell_masks** (*len ncells, each has tuple of pixels belonging to each cell and weights*)
- *lam_normed*

suite2p.extraction.masks.**create_cell_pix**(*stats*, *Ly*, *Lx*, *lam_percentile=50.0*)

Returns Ly x Lx array of whether pixel contains a cell (1) or not (0).

lam_percentile allows some pixels with low cell weights to be used, disable with lam_percentile=0.0

Return type
ndarray

suite2p.extraction.masks.**create_masks**(*stats*, *Ly*, *Lx*, *ops={'1Preg': False, 'align_by_chan': 1, 'allow_overlap': False, 'anatomical_only': 0, 'aspect': 1.0, 'baseline': 'maximin', 'batch_size': 500, 'bidi_corrected': False, 'bidiphase': 0, 'block_size': [128, 128], 'bruker': False, 'bruker_bidirectional': False, 'cellprob_threshold': 0.0, 'chan2_thres': 0.65, 'classifier_path': '', 'combined': True, 'connected': True, 'delete_bin': False, 'denoise': False, 'diameter': 0, 'do_bidiphase': False, 'do_registration': True, 'fast_disk': [], 'flow_threshold': 1.5, 'force_refImg': False, 'force_sktiff': False, 'frames_include': -1, 'fs': 10.0, 'functional_chan': 1, 'h5py': [], 'h5py_key': 'data', 'high_pass': 100, 'ignore_flyback': [], 'inner_neuropil_radius': 2, 'keep_movie_raw': False, 'lam_percentile': 50.0, 'look_one_level_down': False, 'max_iterations': 20, 'max_overlap': 0.75, 'maxregshift': 0.1, 'maxregshiftNR': 5, 'mesoscan': False, 'min_neuropil_pixels': 350, 'move_bin': False, 'multiplane_parallel': False, 'nbinned': 5000, 'nchannels': 1, 'neucoeff': 0.7, 'neuropil_extract': True, 'nimg_init': 300, 'nonrigid': True, 'norm_frames': True, 'nplanes': 1, 'nwb_driver': '', 'nwb_file': '', 'nwb_series': '', 'pad_fft': False, 'prctile_baseline': 8.0, 'pre_smooth': 0, 'preclassify': 0.0, 'pretrained_model': 'cyto', 'reg_tif': False, 'reg_tif_chan2': False, 'roidetect': True, 'save_NWB': False, 'save_folder': [], 'save_mat': False, 'save_path0': [], 'sig_baseline': 10.0, 'smooth_sigma': 1.15, 'smooth_sigma_time': 0, 'snr_thresh': 1.2, 'soma_crop': True, 'sparse_mode': True, 'spatial_hp_cp': 0, 'spatial_hp_detect': 25, 'spatial_hp_reg': 42, 'spatial_scale': 0, 'spatial_taper': 40, 'spikedetect': True, 'subfolders': [], 'subpixel': 10, 'suite2p_version': '0.12.2.dev26+ge1f3790', 'tau': 1.0, 'th_badframes': 1.0, 'threshold_scaling': 1.0, 'two_step_registration': False, 'use_builtin_classifier': False, 'win_baseline': 60.0}*)

create cell and neuropil masks

suite2p.extraction.masks.**create_neuropil_masks**(*ypixs*, *xpixs*, *cell_pix*, *inner_neuropil_radius*, *min_neuropil_pixels*, *circular=False*)

creates surround neuropil masks for ROIs in stat by EXTENDING ROI (slower if circular)

Parameters

cellpix (*2D array*) – 1 if ROI exists in pixel, 0 if not; pixels ignored for neuropil computation

Returns

-
- **neuropil_masks** (*list*) – each element is array of pixels in mask in (Ly*Lx) coordinates

16.5 Module contents

CHAPTER

SEVENTEEN

SUITE2P.CLASSIFICATION PACKAGE

17.1 Submodules

17.2 suite2p.classification.classifier module

17.3 suite2p.classification.classify module

17.4 Module contents

CHAPTER

EIGHTEEN

SUITE2P.GUI PACKAGE

PYTHON MODULE INDEX

S

INDEX

T

X

Y