USER-GUIDE

Cell-matching Dashboard

Setting up the Dashboard

Installation

STEPS:

1. Clone the Git repository: https://github.com/AnnaS-R/Dashboard_CellMatching (info: File descriptions)
2. Install requirements (using PyCharm (Settings/Interpreter Settings)) or run 'pip install -r requirements.txt'.

Data preparation

The necessary data from the *cnm_results.hdf5* files must be extracted and saved as a numpy array (.npy). You will need these npy files whenever you want to use the dashboard for the relevant sessions. This step is necessary to speed up the uploading of the data (images) when using the dashboard.

STEPS:

1. Update the user-specific settings in the prepare_data_dashboard.py file (info: User-specific settings)
    1. Update the list of paths to the folder names (input_folderpaths) that include the files *cnm_results.hdf5* and *mean_intensity_image.tif* for each session.
    2. Update the list of paths storing the file paths (output_filepaths), where you want to save the resulting npy files.
2. Run the script prepare_data_dashboard.py

Setting parameters

STEPS:

1. Update the user-specific settings in the app.py file (info: User-specific settings).
    1. Update the path of the folder path (UPLOAD_DIRECTORY) where the npy files that you want to load to the dashboard are stored.
    2. Choose a filename (filename_result_csv) for the result csv which will list the neuron matched you label (info: Error: Reference source not found).
    3. Other settings can be set such as the zoom ratio, colors etc.
    4. Leave the classifier settings as their default values, unless you read (info: Classifier Background Info) and retrained the classifier. Then set the path to the classifier file (classifier_path) to the new classifier file.

Run Dashboard

STEPS:

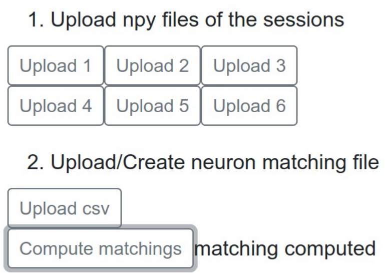
1. Run the app.py script. In the console you should see "Dash is running on http://127.0.0.1:8050/". Click on this and the dashboard will appear in your browser.

Note: If you want to match the neurons of more than 6 sessions at the same time run app_12sessions.py or app_16sessions.py, which can be used to match up to 12 and 16 sessions respectively. (You can always upload less files, but this will still be slightly slower, so only use app_12sessions.py or app_16sessions.py if you need to.)
Make sure you set the user-specific settings (info: Setting parameters) in the file, you are using to run the dashboard.

Using the Dashboard

1. Upload the session data

(These are the npy files from step: Data preparation)
Make sure these files are in the folder you set as UPLOAD_DIRECTORY in app.py.



- You can upload 2-6 sessions.
- The reference session is the top left image and is controlled by the “Upload 1” button.
(Meaning of reference session: Error: Reference source not found)

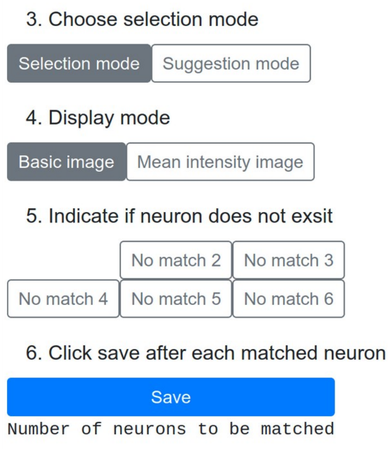
2. Upload/Create neuron matching file (.csv)

- If this is the first time labeling these sessions, click “compute matchings” and the features will be computed and the result csv will be constructed. (This may take longer if there you upload many sessions with many neurons.) Once it is done you will see the writing “matching computed”.
- If you are continuing the labeling of sessions you can upload the results csv file that was saved the during your previous labeling session. (Make sure the results file is in the folder you set as UPLOAD_DIRECTORY in app.py.)

3. Selection mode

There are two selection modes to choose from:

- Selection mode:
Here you can click on a neuron in the reference image (top left), which will turn white. The suggested neuron matches in the other session images will also turn white. You can correct these by clicking on the correct neurons in these image.
- Suggestion mode:
Here a neuron in the reference image will automatically be chosen and colored white you need to check that the



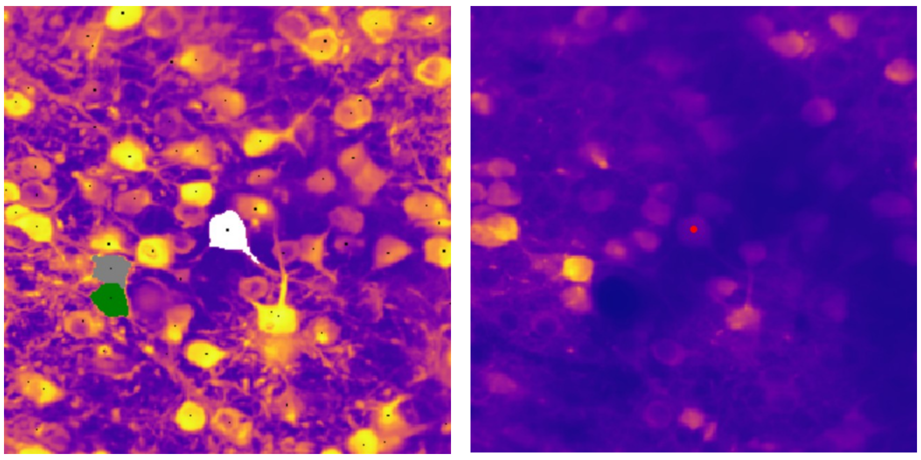
neuron matched in the other sessions are correct (you can correct these by clicking on the correct neurons in these image).
(This mode is useful if you have labeled most neurons and its difficult to find the neurons in the reference image that still need to be matched.)

4. **Display mode**

1. You can switch the basic image (local correlation image) or mean intensity image by clicking the appropriate buttons. The default is the basic image mode.

- Local correlation image:
  - The selected neuron is colored white.
  - The already matched neurons are colored in varying colors. (*This is useful as reference points and to ensure that you don't match the same neuron twice.*)
  - The center of mass of every neuron is represented by an black dot (*This is useful because sometimes one neuron is recognized as two neurons by caiman, or neurons are not recognized at all.*)

- Mean intensity image:
  - The selected neuron is indicated by a red dot.
  - The already matched neurons are not colored (*Sometimes this is makes it easier to have a more clear view of the neurons*).



The same crop of the FoV in the two display modes: basic image (left) and mean intensity image (right) mode

2. A modebar appears in the top right-hand side of a **plotly** graph on mouse hover, which allows you to manually zoom, pan, auto-scale etc. each image individually.

Floating modebar

5. No match (neuron not present in a session image)

If there is no neuron in an image that matches the neuron in the reference image, click the appropriate “No match button” (they are ordered in the same way as you see the images. There is no “no match 1” button because every match needs to refer to a neuron in the reference image.)

The white highlighting of the suggested neuron in the session you clicked the ‘no match’ button for, will disappear. So the white colored neurons (or lack of white colored neurons) are exactly the neurons that will be saved as a match, once you click save.

Once you click save all ‘no match’ buttons will be reset to their default state: not selected.

6. Save

Click save after you checked that the highlighted neuron matched in each image. Only once you have saved you should select a new neuron in the reference image (or if you are n suggestion mode the next neuron to match will be highlighted automatically).
When you click save the updates are always also saved in the results csv file. So if you accidentally close the browser you don’t lose any of the matched you labeled.

After you click save the ‘number of neurons to be matched’ field is updated.

Result format

The output csv file of the dashboard looks like this (ref, sess, s2, s3 are placeholders for the names of the files of the sessions):

ref_sess	s2_ranking	s2	s2feature_vals	s3_ranking	s3	s3_feature_vals	colour	confirmed
1	[4,3,..]	5	[[2.1,..],[..],]	[4,3,..]	7	[[23.2,..],[..],]	0	1
2	[44,5,3,..]	44	[[22,..],[..],]	[2,37,..]	2	[[1,..],[..],]	1	0

The first column lists the neuron indexes of the reference session. In the columns s2, s3.. you can find the indexes of the neurons that have been matched to the neuron index of the reference session of that row. (i.e. here neuron 1 of the reference session, neuron 5 of session 2 and neuron 7 of session 3 are considered to be the same neuron.)

The rows where the column ‘confirmed’ is set to 1, have already been labeled using the dashboard app.

The ‘ranking’ columns list the neuron indexes in the order of CoM distance to the ref neuron of that row. (If the row has not been confirmed, the value in the s2, s3, columns is the first value of the ranking column i.e. the ‘closest’ neuron). The ‘feature_vals’ columns contain the features used for the input of the classifier.

Retraining the classifier

Once you have accumulated more labeled data by using the dashboard, you should retrain the classifier. You can train multiple classifiers using session matching data that have specific characteristics (i.e. for transgenic mice, different brain areas). Make sure to set the classifier file path in app.py!

How to train

Use the training.py script to train the classifier and update the resulting classifier pkl file used in the cellmatching dashboard. An image of the resulting decision tree is also saved.

STEPS:

1. Set the list labelled_results_csv_paths to the paths to the final csv files (.csv) that were saved when using the cellmatching dashboard. These are already in the correct format. If you plan to generate csv files in another manner make sure they are in the assumed format [*].

2. Set train_clf_full_data to True.

3. You can set the classifier parameters max_depth_decision_tree and min_samples_leaf_decision_tree.

4. Run training.py script.

[*] Assumed format of csv file:
1. A column (header: 'confirmed', values: 0/1) to indicate which rows of the spreadsheet will be considered for training.
2. The csv file includes two columns for each session:
   - a col ('nameofsession') of integers, referring to the id of the chosen matched cell
   - a col (nameofsessionfeature_vals') of lists, where each entry of the list is a list of features

How to measure performance of the classifier

Use the training.py script to split the data into train and test set (70:30) and output accuracy, precision to the console and plot the confusion matrix.

STEPS:

1. Set the list labelled_results_csv_paths to the paths to the final csv files (.csv) that were saved when using the cellmatching dashboard. These are already in the correct format. If you plan to generate csv files in another manner make sure they are in the assumed format [*].

2. Set train_clf_full_data to False.

Current classifier

The classifier was trained using the following files (attached to mail) with data I labeled (I just choose the cells and sessions from the data I was given ("W:\\Neurophysiology-Storage1\\Wahl\\Anna\\") that were similar any easiest for me to label):

- result_hendrik_M38.csv (187 cells for 6 Sessions),
- result_hendrik_M41.csv (282 cells for 4 Sessions),
- result_jithin_pre.csv (249 cells über 3 Sessions).

This results in 2279 data points used for training. More data is needed to improve the accuracy of the classifier, so I recommend retraining once you have labeled more sessions (see Retraining the classifier).

Implementation Details

File descriptions

Dashboard files:

prepare_data_dashboard.py	Script that extracts of the data needed for the dashboard from the h5py files from the caiman pipeline. *Needs to be run once on data of each session.*
training.py	*Script that allows the user to train a decision tree classifier* that predicts which neurons are the same over sessions. Y*ou only need to run this script if you want to retrain because you have collected new labeled data.*
app.py	This script runs the dashboard! *You need to run this whenever you want to start using the dashboard (up to 6 sessions).*
app_12sessions.py	This script runs the dashboard with additional sessions! *You need to run this whenever you want to start using the dashboard with 7-12 sessions.*
app_16sessions.py	This script runs the dashboard with additional sessions ! *You need to run this whenever you want to start using the dashboard with 13-16 sessions.*
content_layout.py	File includes functions needed for the dashboard. Y*ou don't need to touch this.*
internal_functions.py	File includes functions needed for the dashboard. *You don't need to touch this.*
requirements.txt	*You can use to install the needed python packages. You can also ignore this and install the packages directly through PyCharm (Settings/Interpreter Settings).*

Classifier files:

training_data.csv	The correctly formatted training data used to train the current classifier
pickle_model.pkl	Saved Decision Tree classifier object
decision_tree.png	Image of current trained decision tree

Documentation files: documentation.pdf, documentation.odt

Classifier Background Info

General

The classifier is a a decision tree classifier, which is supervised learning method. It is trained using labeled data (a list of neuron features and the correct target neuron) and creates a model that predicts the value of a target variable by learning simple decision rules.

The main advantage of this classifier is that the decision trees are simple to understand and to interpret. Trees can be visualized (See the file decision_tree.png for the current decision tree and How to train to create a decision tree image).

Another advantage is that it can handle multi-output problems and requires little data preparation (I.e no data normalization etc needed), so you can add any numerical feature you want.

Classifier input and output

The input for the classifier is a list of features of the neurons which are closest (CoM are shifted using Hendrik's function) to the reference neuron. The target is which of the neurons (index) of the list should be chosen.

The format is as follows:

```
[feat1_neur1, feat2_neur1, feat3_neur1, ...,
 feat1_neur2, feat2_neur2, feat2_neur2, ...,
 feat1_neur3, feat2_neur3, feat2_neur3, ...,
 .... ]
```

The classifier is then trained to output 1 (2, 3, ..) to indicate that neuron 1 (2, 3, ..) is the correct match.

If you plan to add/delete features you can do this, but then you must update num_features in app.py when using a classifier trained with more/less features.

Good to know

User-specific settings

In all files you run the only settings you should adapt are in the 'USER-SPECIFIC SETTINGS' which look like this:

```
# *******************************************************************************************
#               USER-SPECIFIC SETTINGS
# *******************************************************************************************

# what setting 1 is
variable1 = “….”

# what setting 2 is
variable2 = “….”

# *********************************************************************************************
```