deepCINAC

Release alpha 0.0.11

Cossart lab

Mar 24, 2023

CONTENTS

CHAPTER

ONE

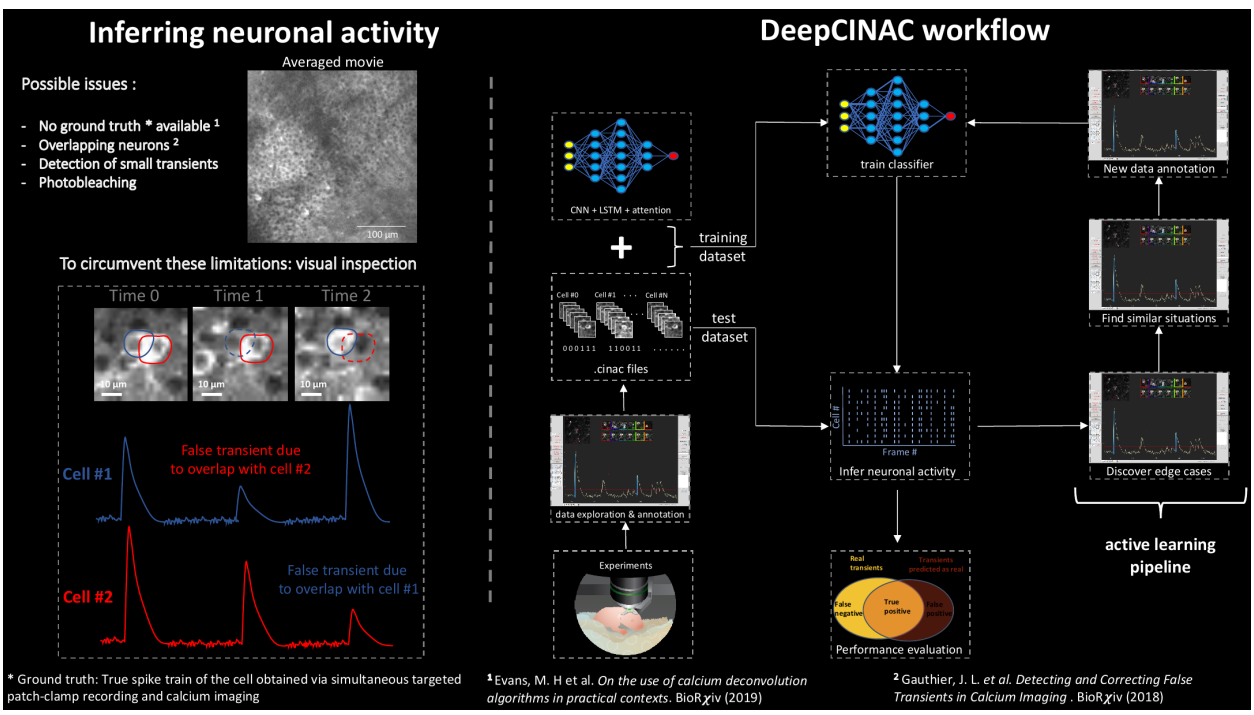
CALCIUM IMAGING TOOLBOX

We have developed a Graphical User Interface (GUI) that offers various tools to visually evaluate inferred neuronal activity from inference methods and to build a eye inspection-based ground truth on calcium imaging data.

Then, we have designed a deep-learning based method, named DeepCINAC (Calcium Imaging Neuronal Activity Classifier). Instead of basing activity inference on the extracted fluorescence signal, DeepCINAC builds up on the visual inspection of each cell from the raw movie using the GUI.

This toolbox being very flexible, it can be adapted to any kind of calcium imaging dataset, in-vivo or in-vitro.

The toolbox also allows to predict cell type.



To train a classifier, the first step is to annotate data using the GUI. See the GUI tutorial for more information.

Then using the .cinac files produced, follow the instructions to train your classifier.

Note that .cinac files don't contain the full original calcium imaging movie, only patches surrounding the cells you have annotated, so you can share the files without fearing that your data will be used by someone else other than to train a classifier.

To predict data, you can either use one of our pre-trained classifier or one you have trained, follow those instructions here .

1.1 Dependencies

deepCINAC has the following minimum requirements, which must be installed before you can get started using PyNWB.

1. Python 3.6, or 3.7
2. pip

deepCINAC has been tested on Ubuntu 18.04.1 LTS, Windows 10 and macOS Mojave, using Python 3.6

1.2 Installation

1.2.1 Install release from PyPI

The Python Package Index (PyPI) is a repository of software for the Python programming language.

To install or update deepCINAC distribution from PyPI simply run:

```
$ pip install deepcinac
```

This will not automatically install the required dependencies. You can download our requirements.txt file and run :

```
$ pip -r requirements.txt
```

The following packages will be installed :

- numpy
- scanimage-tiff-reader
- tifffile
- keras
- matplotlib
- Pillow
- scipy
- networkx
- seaborn
- alt_model_checkpoint
- hdf5storage
- PyYAML
- h5py
- read-roi

Make sure your setuptools package is up to date, you might otherwise get this error message during installation: "ImportError: cannot import name 'find_namespace_packages' from 'setuptools"

Two other packages will still be missing:

- **shapely**

On Linux or MacOs simply run:

```
$ pip install shapely
```

For windows users, follow the instruction there. If you went on the wheels option, here are the instruction to install the wheel file: first open a console then cd to where you've downloaded your file and use:

```
$ pip install Shapely-1.6.4.post2-cp37-cp37m-win_amd64.whl
```

- **tensorflow**

If you wish to use the GPU to increase the significantly the speed of prediction, then install tensorflow-gpu instead. For using the GPU, you will also need to install the NVIDIA CUDA Toolkit and Driver.

Here is a link to guide you to install it on windows (for french speakers).

For linux users, those links might be useful: nvidia website and a blog post.

1.2.2 Use the notebook with google colab to infer neuronal activity

If you just want to infer neuronal activity of your calcium imaging data and you don't possess a GPU or don't want to go through the process of configuring your environment to make use of it, you can run this notebook using google colab.

Google provides free virtual machines for you to use: with about 12GB RAM and 50GB hard drive space, and TensorFlow is pre-installed.

You will need a google account. Upload the notebook on google colab, then just follow the instructions in the notebook to go through.

1.3 GUI tutorial

To start the GUI, either execute this command in a terminal:

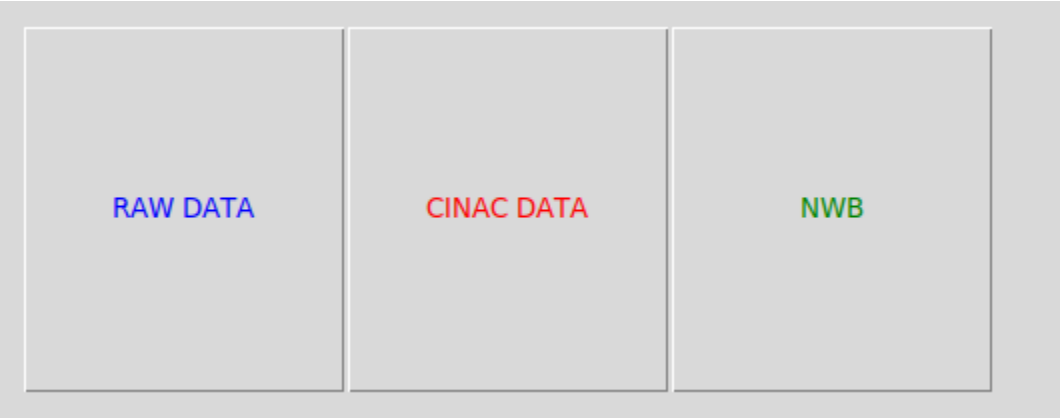
```
python -m deepcinac
```

Or execute the python code:

```
deepcinac.gui.cinac_gui.launch_gui()
```

If you don't have data, we provide a demo dataset with a calcium imaging movie, ROIs (cell coordinates) extracted by Suite2P (this file and this one) and the predictions from our general classifier.

1.3.1 First window: data to display

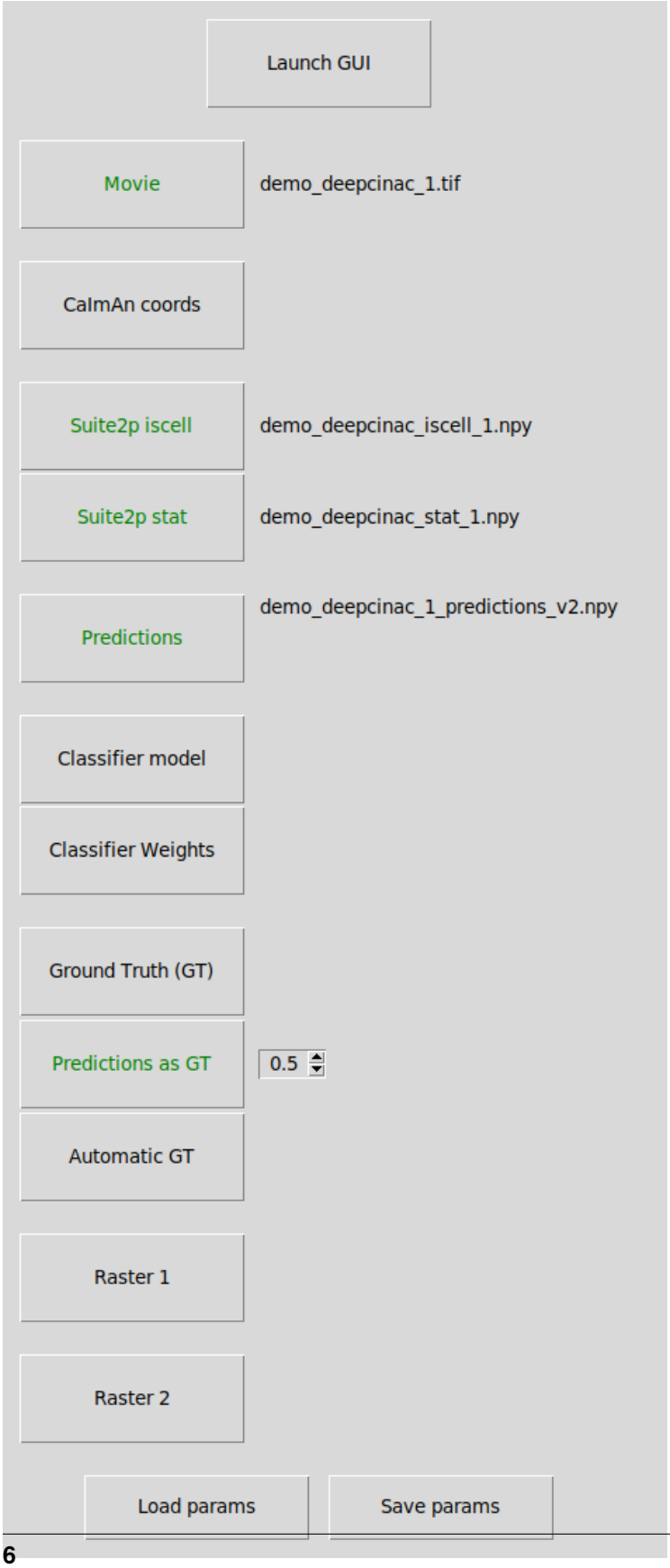


The button RAW data allows you to load data that haven't been processed using the GUI previously (see below).

The button CINAC data allows you to load data that have been processed using the GUI.

The button NWB data allows to load calcium imaging contained in a Neurodata Without Borders (NWB) file. For the option to be available, you need to install the package pynwb. You can also use nwb to launch predictions using our toolbox CICADA.

1.3.2 Raw data parameters



This window will allow you to choose which data you want to display in the GUI.

When you select a file in the .npz or .mat format, a menu will appear letting you choose which attributes of the file should be loaded.

We will describe the function of each button below:

Launch GUI: launch the exploratory GUI, the button will be enabled when you have selected a movie file and some ROIs.

Load params: Select a .yaml file that contains the parameters to be load.

Save params: Allow to save in a .yaml file the parameters that have been selected in this window, to be loaded later.

Movie: Select the calcium imaging movie file, so far only tiff format is supported.

CaImAn coords: Select the file containing the CaImAn cell coordinates (ROIs). It can be a file in numpy (.npy or .npz) or matlab format (.mat). If the coordinates have been computed with matlab, meaning the indexing starts at 1, check the box "matlab indexing". CaImAn coordinates as we used them are represented by a list (if 2d-array) of 2 lines representing x and y coordinates of the contours's points.

Suite2p iscell: Select the file iscell produced by suite2p indicating which cell is valid.

Suite2p stat: Select the file stat produced by suite2p containing the cell's pixels

You can either load coordinates from CaImAn or from Suite2p but not both.

Predictions: Select a file containing the predictions from the classifier. It contains a 2d array of n_cells * n_frames with float values between 0 and 1. By loading this file, you will be able to display the predictions in the GUI. You would also be allowed to click on the button "Predictions as GT" to used those predictions with a specific threshold to fill the GUI with this ground truth as a base.

Classifier model: Instead of loading predictions, you can load the model (.json file)

Classifier Weights: and the classifier weights (.h5 file). This will allow you to run the classifier cell by cell and display the predictions on the GUI. If you haven't configured your GPU and use the CPU, it might take a few minutes to predict the activity for each cell depending on how long is your recording.

By clicking one of the 3 following buttons, the GUI will display peaks and onsets determined by one of this button. Otherwise, by default, the fluorescence signal will be displayed without any peaks and onsets. You'll be able to add some. If some are loaded, you will still be able to modify them.

Ground Truth (GT): Select a file that contains a 2d array of dimension n_cells * n_frames, it should be a binary matrix, containing only 0 and 1, 1 indicating that a cell is active at a given frame.

Predictions as GT: If a predictions file is loaded, you can set the threshold that will be used to decide the periods of activity of the cells and onsets/peaks according to those transients.

Automatic GT: Based on the smooth version of the fluorescence signal, all potential onsets and peaks will be added.

Raster 1: Select a file that contains a 2d array of dimension n_cells * n_frames, for each value > 0, a line will be displayed at the bottom of the fluorescence signal section (the matrix is binarized), thus representing the activity of the cell (either as a spike inference or representing transients like CINAC). It allows the comparison of the results of the predictions or your own ground truth with another method. It is displayed in green.

Raster 2: Same as raster 1, but allows you to add one more raster.It is displayed in blue.

Cell type config: Mandatory to display the predictions. It should be the same yaml file than the one used to train the cell type classifier. The file indicates which cell type are the output of the classifier. You can find examples here.

Cell type predictions: Select a file containing the predictions from the classifier. It contains a 1d array representing the indices of the cells predicted and a 2d array of n cell types x n_cells with float values between 0 and 1. By loading this file, you will be able to display the predictions in the GUI.

Cell type model: If the predictions don't cover all the cells, you can load the model (.json file)

Cell type weights: and the classifier weights (.h5 file). This will allow you to run the classifier for any given cell and display the predictions on the GUI. If you haven't configured your GPU and use the CPU, it might take a few minutes to predict the cell type.

1.3.3 CINAC data parameters

This window will allow you to choose which data you want to display in the GUI.

The cinac file (extension .cinac) is the format used to save ground truth data produced using the GUI.

We will describe the function of each button below:

Launch GUI: launch the exploratory GUI, the button will be enabled when you have selected a cinac file.

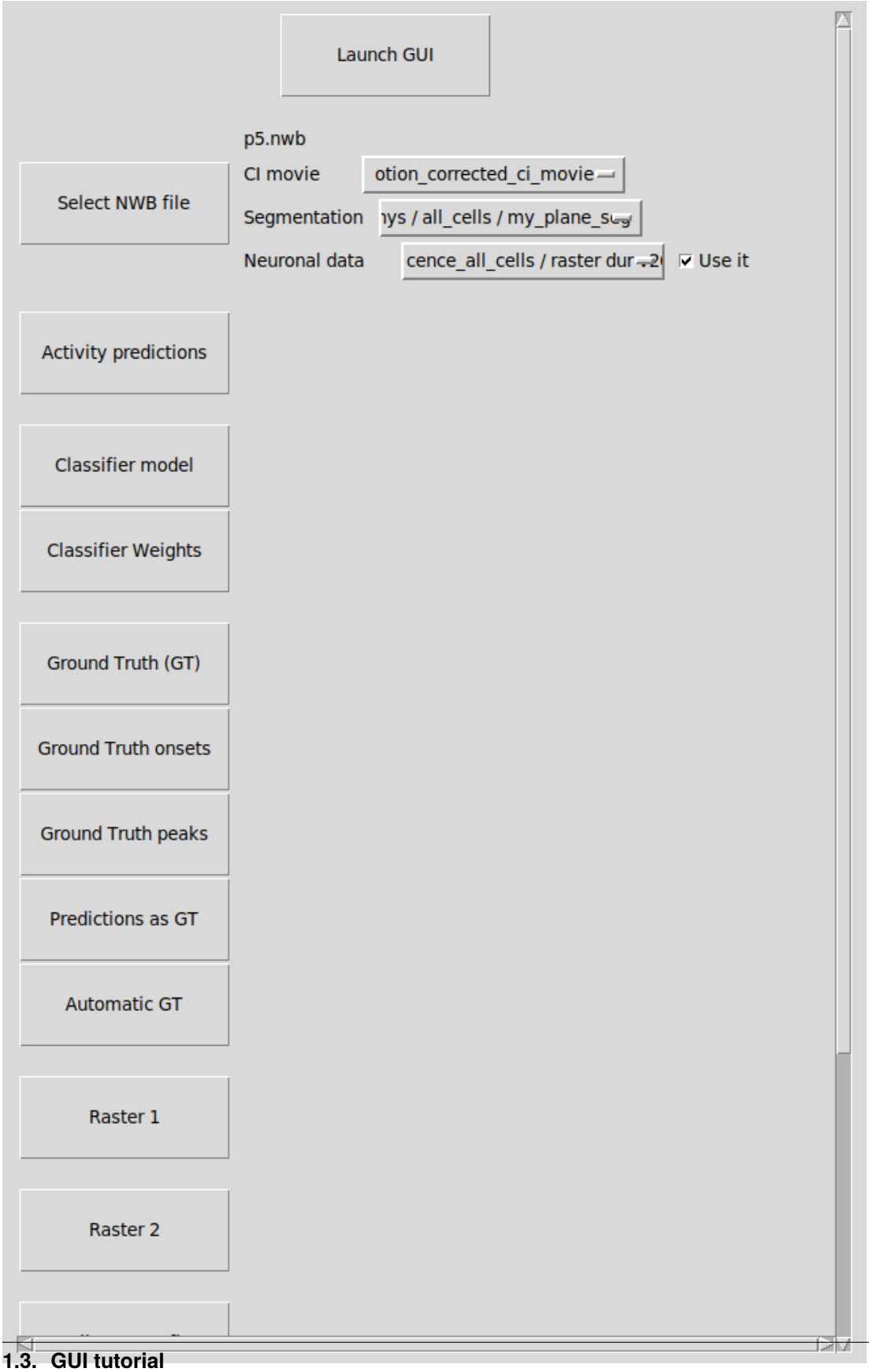
Select CINAC file: Select the cinac file to open.

CI movie: this button will be enabled if no movie file reference is contained in the CINAC file or if this reference doesn't match an existing file. Thus allowing you to select the calcium imaging movie that matches the cinac file. The calcium imaging movie should be the original one used to produce the CINAC file that matches the cell contours contained in the CINAC file. Moreover, selecting a CI movie is not mandatory to open a CINAC file, if no movie is indicated, then segments will be opened separately (see below).

Listbox: displays the segments for which a ground truth has been made. Each line of the listbox represents a segment, with the given format "{cell} / {first_frame}-{last_frame}" the cell the segment is from and the frames it contains. If no CI movie is available, then you need to select the segment you want to display. A distinct window will be opened for each segment, up to 10 can be opened simultaneously. In that case, only the pixels surrounding the cell will be displayed in the GUI, the number of cells available will correspond to the number of cells that overlaps with the cell of the segment. In that mode, no modification of the segment can be saved, it is necessary to have the original movie to make modifications. On top of the listbox, it is displayed the number of frames that are comprised in the segment as well the number of active frames, meaning frames that are contained between an onset and a peak (corresponding to the transient rise time)

others buttons: same usage as the ones in the window "Raw data parameters" (see above). However, those buttons will be enabled only if a CI movie is available.

1.3.4 NWB data parameters



This window will allow you to select the content from a Neurodata Without Borders (NWB) file you want to display.

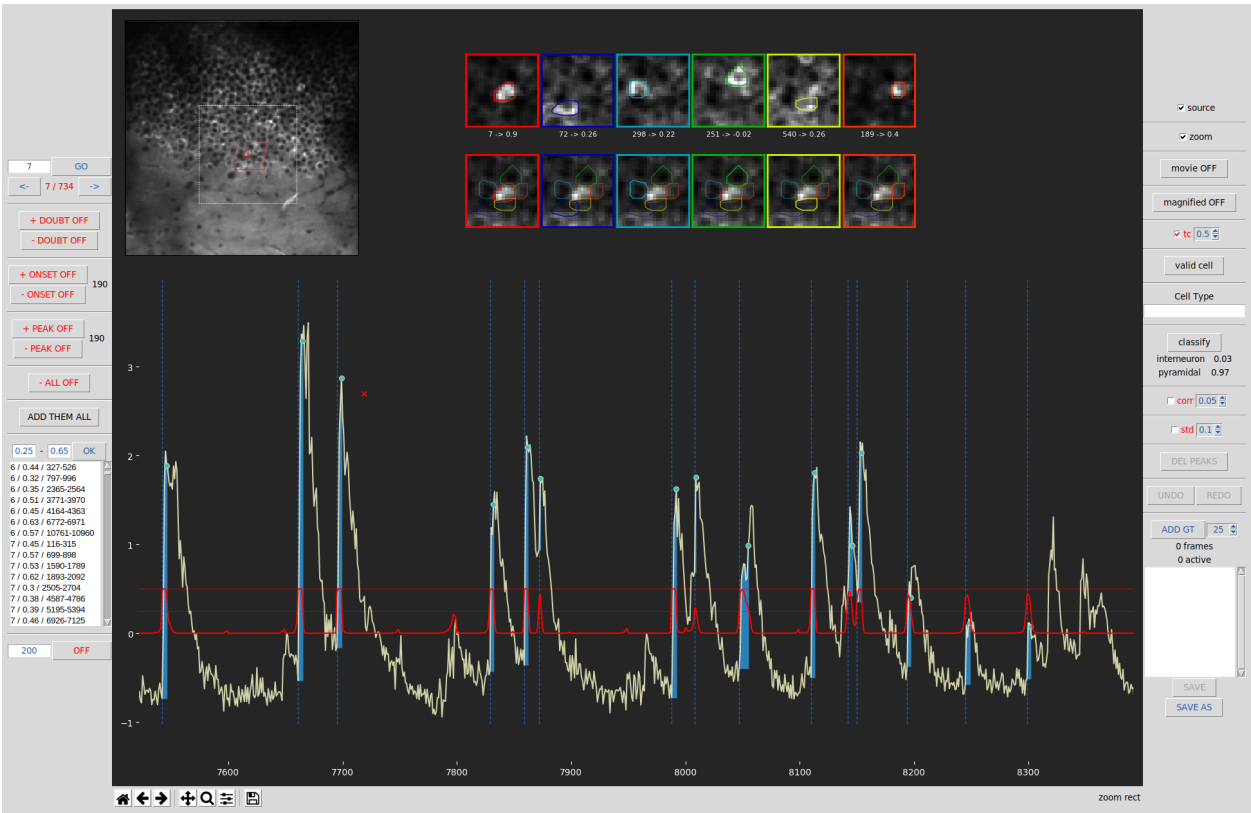
We will describe the function of each button below:

Launch GUI: launch the exploratory GUI, the button will be enabled when you have selected a NWB file.

Select CINAC file: Select the NWB file to open. The NWB file should contain at least one (two photon) calcium imaging movie and some data regarding segmentation (as 'pixel_mask'). If neuronal data ara available, they will be displayed.

others buttons: same usage as the ones in the window "Raw data parameters" (see above).

1.3.5 Exploratory GUI



The exploratory GUI has two main goals:

- to visually evaluate inferred neuronal activity from inference methods
- to build a eye inspection-based ground truth

In order to build a ground truth, you need to annotate the data either adding onsets/peaks for neuronal activity or indicating the cell type. For each cell, you can save any segment (of continuous frames) of data. The annotated data will be saved in the format of .cinac files.

Central Panel:

- Average movie view: When the movie is not being played, on the top left is displayed the average of all movie's frames. The cell currently viewed is represented on the map by a red contour. You can click on any cell to set the selection to this cell. The dashed white edge square represents the view displayed when playing the movie in the zoom mode. The red edge square represents the pixels that will be saved and used for the current cell by the classifier.

- Movie player: On top left, when the movie is being played (by clicking on the first and last frame to be played after activating the movie mode). On top-left corner the current frame number is displayed. The current cell has a red edge contour. Finally, the fluorescence signal of the current cell is displayed (the red part represents the frames already played and the white part, the ones coming).
- Source and transient profiles: displayed on the top right if source mode has been activated and a transient has been selected. The top row represents the source profile of the cells (the current one, always on the left side, and the ones overlapping it). Source profile represents the weighted average pixels intensities of the cell during the rise time of its highest transients. On the bottow row is displayed the transient profiles that represents the weighted average pixels intensities of the cell during a given transient (the one before or at the red cross). Finally, between those 2 rows are displayed for each cell, its number and the Pearson correlation between the source profile and transient profile (for the pixels contains in the ROIs of the cell).
- Magnifier: displayed on the top right when the magnifier mode is on and the onsets/peaks mode are one. It allows to displayed a zoom version of the fluorescence signal displayed belown thus allowing a more precise annotation of the signal.
- Fluorescence signal: on the bottom, the average fluorescence signal (z-score normalization) of the current cell is displayed. The blue dashed vertical line represents the onsets. The blue dots represent the peaks. If the predictions mode is on, then a red line will represent the predictions of the classifier for this cell (see the right panel section bellow for more details). A black vertical span might be displayed to represent the a doubtful segment.
- Navigation toolbar: displayed on the bottom, under the fluorescence signal window. The home button allows to come back to full signal view. The left/right arrows allow to come back to the last part of the signal visualized. The 4 way arrows allows to move the signal displayed. The magnifier button allows to zoom on one portion of the signal. The floppy button allows to save the signal plot as an image. Note that this is the toolbar provide by matplotlib and tkinter, however we notice that sometimes the buttons are not reactive, in that case the trick is to click on the plot then back to the button a few time until it does work again. We will rewrite the toolbar in a further version.

Left Panel:

- Cells selection: the text field allows to write the cell number you which to display. Then press the enter button or click on the OK button. You can also use the arrows button to display the previous/next cell. You can also use shortcuts ctrl-left/right arrow to display the previous/next cell. Finally, you can also click directly on the cell you want to display in the cells map
- DOUBT buttons: add or remove segment of doubt. Doubt segments are sections that should not be used for training the classifier as ground truth is difficult to establish. Just click on the start and end of the segment on the fluorescence signal section.
- + onset button: activate/deactivate the add onset mode. Allows you to add onset, just click where you want to add an onset. A dashed blue vertical line will appear at that position. Next to the button is displayed the number of onsets for this cell.
- - onset button: activate/deactivate the remove onset mode. Allows you to remove onset(s), click 2 times with the mouse on the "fluorescence signal" part to indicate the start and the end of the movie segment from which you want to remove onsets.
- + peak button: activate/deactivate the add peak mode. Allows you to add peak, just click where you want to add an peak. A blue dot will appear at that position. Next to the button is displayed the number of peaks for this cell.
- - peak button: activate/deactivate the remove peak mode. Allows you to remove peak(s), click 2 times with the mouse on the "fluorescence signal" part to indicate the start and the end of the movie segment from which you want to remove peaks.
- - ALL button: activate/deactivate the remove all mode. Allows you to remove onset(s) and peak(s), click 2 times with the mouse on the "fluorescence signal" part to indicate the start and the end of the movie segment from which you want to remove onsets and peaks.

- Add them all button: when clicked, it will remove all onsets/peaks present in the current window displayed and add all potential onsets/peaks that can be found using the change of derivative sign from a smooth version of the fluorescence signal.
- Predictions listbox: this listbox is displayed if either predictions results are available or if a model and weights files have been selected. The listbox display the transients of the cells according to their predictions score. The text fields above it, allow to change the value boundaries of the predictions displayed. When clicking on a line of the listbox, it displays the given transient, that will be centered on the window, the number of frames on the window will correspond to the number displayed on the text field below the listbox. Finally, you can either double-click on a given transient in the listbox / push the ADD GT button / or press the G keyboard touch to add the given window the ground truth segment list.
- Center Segment mode: you can change the size of the segment used when a frame is centered over the window displayed. By clicking on the OFF button, you active the center segment mode, then you can click on the "fluorescence signal" section, and the frame you click on will be then at the center of the window, the window number of frames will then be the one indicated on the text field.

Right Panel:

- Cell / Neuropil / Cell - Neuropil checkboxes: allows to choose which trace(s) you want to display. 'cell' displays the raw signal (z-score normalization), Neuropil displays the neuropil surrounding the cell, and 'Cell - Neuropil' displays the raw signal substracted by the Neuropil one.
- Source checkbox: activate the source mode (shortcut keyboard: s touch). Then to display the cells sources and transient, you need to click in the transient you want to display or just after. However, the transient need to be identified by an onset and a peak, if none exists you must add it before you can display the transient profile.
- zoom checkbox: activate/deactivate the zoom mode. When on, a white dashed square will be displayed on the average calcium imaging movie view. The square represents the view displayed when playing the movie in the zoom mode.
- movie button: allows you to play or stop the movie. (shortcut keyboard: space bar). To play it, click 2 times with the mouse on the "fluorescence signal" part to indicate the start and the end of the movie segment you want to play. To stop the movie, either press the space bar or the button "movie ON". The movie will repeat until you do one of these actions. You might encounter some memory issues if you let it loop too many times.
- Magnified button: Activate/deactivate magnifier mode. Will only appear if the onset or peak mode is active. Allows the display a zoom mode of the fluorescence trace on the upper right section of the GUI, thus allowing to be more precise without having to zoom on the main section.
- tc checkbox: To display predictions. Click on it and either the predictions will be immediately displayed if you have pre-loaded them or it will be predicted using the model weights you have provided. A bold red line will then be displayed representing the prediction value, between 0 and 1. The 0 is labeled in the y-axis. The 0.5 is represented by the dashed red-line, the 1 by the top thin solid line. The interior of the fluorescence signal will be fill with blue when the prediction passes the threshold. You can set the threshold using the spin box next to the checkbox (Default value is 0.5). The checkbox is not displayed if no predictions or model/weights files have been selected.
- valid cell button: identify the cell as valid (by default) or invalid.
- cell type field: allows to indicate the cell type of the cell. It will be saved in the cinac file if you save any segment of the cell.
- classify button: will be display only if a model and weights files have been provided. Run the classifier on the current cell, and display below the predictions score for each cell type as determined in the cell type yaml file given. If the predictions are loaded, they will be directly displayed.
- corr: similar to std checkbox below, but set a threshold of correlation between the transients profile and the source profile of the cell. It can take a bit of time to compute depending how many peaks and onsets are present. The

transient under the threshold will be displayed in red (if other correlations with overlapping cell are also low), and can be deleted using the "DEL PEAK" button.

- std checkbox: set a threshold. All peaks (from smooth trace) that will be under the threshold will be colored in red. Then using the "DEL PEAKS" button, they will be deleted with their associated onsets. It can be useful when you use automatic onsets & peaks initiation and just want to eliminate the obvious noise.
- DEL PEAKS button: allows to delete the peaks selected using the corr or std checkbox
- UNDO & REDO buttons: to UNDO & REDO actions.
- Ground truth segment: section that allows to select ground truth segment that will be used to train a classifier. The spinbox next to the "ADD GT" button allows to change the size (in pixels) on the square surrounding the cell and that represent the content that will given to the classifier. The "ADD GT" button allows to add the current cell and window displayed (frames displayed in the fluorescence signal section) as a ground truth segment, a new line will be added in the listbox. You can click on the line in the listbox to display the segment. To remove a segment, double click on it.

shortcuts:

- Space bar: allows you to play the movie. Same effect as clicking on the button "movie OFF". Then click 2 times with the mouse on the "fluorescence signal" part to indicate the start and the end of the movie segment you want to play. To stop the movie, either press the space bar or the button "movie ON". The movie will repeat until you do one of these actions. You might encounter some memory issues if you let it loop too many times.
- s: activate the source mode, same effect as clicking on the source checkbox. Then to display the cells sources and transient, you need to click in the transient you want to display or just after. However, the transient need to be identified by an onset and a peak, if none exists you must add it before you can display the transient profile.
- o, O: activate/deactivate the onset mode. Allows you to add onset, just click where you want to add an onset. A dashed blue vertical line will appear at that position.
- p, P: activate/deactivate the peak mode. Allows you to add onset, just click where you want to add a peak. A blue dot will appear at that position.
- r, R: remove mode, will remove all onsets or peaks that are present between your first click and second click on the fluorescence signal section.
- ctrl + left or right arrow: Switch to previous or next cell display. You can also indicate the cell to display in the text field. Another option, is to directly click on the average calcium imaging picture to select the cell you want to display.
- left or right arrow: if you are zoomed on the fluorescence signal, then it will shift to the previous or following frames.
- a, A: same effect as left arrow
- q, Q: same effect as right arrow
- m: magnifier mode. Will only appear if the onset or peak mode is active. Allows the display a zoom mode of the fluorescence trace on the upper right section of the GUI, thus allowing to be more precise without having to zoom on the main section.
- c, C: activate/deactivate the center segment mode. If activated, center the window to the frame that will be clicked after pressing this touch. The size of the new window will correspond to the number of frames indicated in the text field under the listbox on the left panel. It allows to make sure the event you labeled will at the centered of the window, thus allowing a bigger representation when temporal overlaps before using it to train the classifier.
- z, Z: activate/deactivate the zoom mode. When on, a white dashed square will be displayed on the average calcium imaging movie view. The square represents the view displayed when playing the movie in the zoom mode.
- g, G: allows to add the current cell and window displayed as a ground truth segment.

- T: if cell type predictions are loaded, then it allows to display the next cell (in indices order) with the same cell type as the cell currently display.

1.4 Training tutorial

Using the annotated .cinac files created with the GUI, you can now train your classifier.

Below are the few lines of codes needed to train the classifier:

```
cinac_model = CinacModel(results_path="/media/deepcinac/results",
                         using_splitted_tiff_cinac_movie=False,
                         n_epochs=20, batch_size=8)
cinac_model.add_input_data_from_dir(dir_name="/media/deepcinac/data/cinac_ground_truth/
↪for_training")
cinac_model.prepare_model()
cinac_model.fit()
```

First, you need to build an instance of CinacModel (see below for the parameters), the add data, prepare the model and finally fit it to your data to train the classifier.

Input data are the cinac files, you can either load all files in a directory or load files one by one.

To train your classifier to infer neuronal activity, you would have had to add onsets and peaks through the segments recorded in the .cinac files using the GUI.

To train your classifier to predict cell type, you would have had to annotate the cell type of the segments's cell recorded in the .cinac files using the GUI.

1.4.1 CinacModel arguments

By changing the arguments passed to CinacModel init() method, you can change most of the parameters used to train the classifier. Including precising if you want to train a classifier to infer neuronal activity or cell type.

Let's see first the mandatory one, with no default value:

- **results_path**: path of the directory where to save the output files.

The other arguments are:

- **cell_type_classifier_mode**: (bool, default False) if False, means you want to train a classifier to infer neuronal activity, if True a cell type classifier.
- **using_splitted_tiff_cinac_movie**: (bool, default is True) Should be set to False on first use. In some cases, if multiple GPUs are used or depending on operating systems, it seems to be a thread lock issue when reading .cinac file (hdf5 format). A solution to fix it, if to keep this variable to True, it will then split the calcium imaging on individual tiff file (one by frame), in the directory given by the 'tiffs_dirname' argument. Used only for neuronal activity classifier. The individual tiffs frame will be created when adding data to the model, an exception will be raised when the tiffs will have been created, the code needs to be re-run then.
- **tiffs_dirname**: (str, default is None) path where to save the the calcium imaging movie as individual tiff for each frame.
- **n_epochs**: (int, default is 30), number of epochs for training the classifier
- **batch_size**: (int, default is 8) size of the batch used to train the classifier
- **split_values**: (tuple of 3 floats, default is (0.8, 0.2, 0)), tuple of 3 floats, the sum should be equal to 1. Give the distribution of the dataset into training, validation and test

- **window_len**: (int, default 100) number of frames of the temporal segments given to the classifier
- **max_width**: (int, default 25) number of pixels for the width of the frame surrounding the cell. Should be big enough so most of the overlaping cell fit in for activity classifier.
- **max_height**: (int, default 25) number of pixels for the height of the frame surrounding the cell. Should be big enough so most of the overlaping cell fit in for activity classifier.
- **model_descr**: (str, default is ''), string describing the model, the string will be added to the name of the files used to save the model
- **with_augmentation_for_training_data**: (bool, default is True): is True, then geometric transformations are applied to the dataset
- **overlap_value**: (float, default 0.9), overlap between 2 segments using sliding window of size window_len 0.9 means contiguous segments will share 90% of their frames. (temporal data augmentation)
- **max_n_transformations**: (int, default is 6) max number of geometric transformations to apply to each segment. (data augmentation)
- **n_gpus**: (int, default is 1) Maximum number of processes to spin up when using process-based threading.
- **workers**: (int, default is 10), number of workers used to run the classifier
- **cell_type_categories_yaml_file**: (str, default is None) path and filename of the yaml file used to configure the classifier for cell type (types of cell, number of classes). If None, then cell_type_categories_default.yaml will be used with pyramidal cells and interneuron and 2 classes.
- **n_windows_len_to_keep_by_cell**: (int, default 2), used only for cell type classifier, indicate how many segments of length window_len to keep for training. If too many segment are given then the classifier might not be able to generalize well
- **frames_to_avoid_for_cell_type**: (sequence of int, default []), list of frame indices. If given, then segment than contains one of those indices won't be add to the training. Useful for example if movies are concatenated.
- **using_multi_class**: (int, default is 1) number of classes used to classify the activity. So far only 1 or 3 are valid options. 3 means we distinguish active cell, from overlap activity, from neuropil and other. 1 class gives better results so far. Don't use it for cell type, the number of classes will be set through the yaml configuration file.
- **pixels_around**: (int, default is 0), number of pixels to add around the mask of the cell (for activity classifier)
- **buffer**: (int, default is 1): indicated of how many pixels to inflate the mask of the cell.
- **loss_fct**: (str, default is 'binary_crossentropy' if using_multi_class is 1, 'categorical_crossentropy' else), loss function used to train the classifier.
- **with_learning_rate_reduction**: (bool, default is True), if True, means the learning rate will be reduced according to the following arguments
- **learning_rate_reduction_patience**: (int, default is 2) number of epochs before reducing the learning rate if the validation accuracy has not improved, with_learning_rate_reduction needs to be True
- **learning_rate_start**: (float, default is 0.001) default learning rate to start with
- **with_early_stopping**: (bool, default is True) if True, then early stopping is activated, if the classifier doesn't progress for a given number of epochs indicated through the arg early_stop_patience
- **early_stop_patience**: (int, default is 15): number of epochs before the training stops if no progress is made (based on validation accuracy values)
- **with_shuffling**: (bool, default True), if True, the segments will be shuffle before added to training or validation dataset
- **seed_value**: (int, default is 42), if not None, used as seed for the shuffling of segments, giving a seed means the shuffle will always be the same for a given dataset.

- **main_ratio_balance**: (tuple of 3 floats, default is (0.6, 0.2, 0.2)), sequence of 3 float, the sum should be equal to 1. Used for activity classifier, precise the proportion in the final training dataset between sequence according to real transient, fake transient and "neuropil"
- **crop_non_crop_ratio_balance**: (tuple of 2 floats, default is (-1, -1)), use for stratification in activity classifier, if values are -1, we don't use it. Otherwise allows to balance segments between segment with transient cropped versus not cropped.
- non_crop_ratio_balance: (tuple of 2 floats, default is (-1, -1)), used for stratification in activity classifier, if values are -1, we don't use it. Otherwise allows to balance segments between segment with one transient versus more than one transient.
- **with_model_check_point**: (bool, default is True) allows to save weights of the classifier at each epoch
- **verbose**: (int, default is 2), 0 no verbose, 1 main outputs printed, 2 all outputs printed
- **dropout_value**: (float, default 0.5), dropout between CNN layers
- **dropout_value_rnn**: (float, default 0.5), dropout between RNN layers
- **dropout_at_the_end**: (float, default 0), dropout value at the end of the model
- **with_batch_normalization**: (bool, default False), if True use batch normalization
- **optimizer_choice**: (str, default is "RMSprop"), optimizer to use, choices "SGD", "RMSprop","Adam"
- **activation_fct**: (str, default is "swish") , activation function to use, you can choose any activation function available in TensorFlow, swish is also available.
- **without_bidirectional**: (bool, default is False), if True, LSTM is not bidirectional
- **conv_filters**: (tuple of 4 int, default is (64, 64, 128, 128)), the dimensionality of the output space (i.e. the number of filters in the convolution) sequence of 4 integers, representing the filters of the 4 convolution layers
- **lstm_layers_size** (sequence of int, default is (128, 256)), number of LSTM layer and size of each layer.
- **bin_lstm_size** (int, default is 256), size of the LSTM layer used in bin et al.
- **use_bin_at_al_version** (bool, default is True), using model inspired from the paper from Bin et al.
- **apply_attention**: (bool, default is True), if True, attention mechanism is used
- **apply_attention_before_lstm**: (bool, default is True), if True it means attention mechanism will be apply before LSTM
- **use_single_attention_vector** (bool, default is False), if True us single attention vector

1.4.2 Training outputs

The output files will be saved in the directory indicated by the argument "results_path" of CinacModel.

- **.json file**: representing the model used (its architecture). It is needed to run the classifier to infer neuronal data.
- **.h5 files**: representing the weights of each epoch. You can use the weights of one epoch to run the classifier on your data in association with the model file.
- **metrics_history.npz**: If the code run up till the last epoch parametrized (of if early stop is activated), a npz file will be saved with the metrics history at each epoch. Each key represents a metric and the value is a 1d array.

1.4.3 Demo code

We provide demo code that shows you how to use DeepCINAC either as a notebook (that you can run on google colab) or in a Python file.

On google colab

If you don't possess a GPU or don't want to go through the process of configuring your environment to make use of it, you can run this notebook using google colab.

Google provides free virtual machines for you to use: with about 12GB RAM and 50GB hard drive space, and TensorFlow is pre-installed.

You will need a google account. Upload the notebook on google colab, then just follow the instructions in the notebook to go through.

Note that with google colab you won't probably be able to train an efficient classifier has the run time is limited to 12h. However, it will let you test the code. You'll then need a local GPU or HPC access to train it with enough data to get good results.

On your local device

You can follow the steps described in this demo file.

To know the list of all the arguments, check the API reference.

1.5 Predictions tutorial

1.5.1 Inferring neuronal activity

The classifier takes as inputs the motion corrected calcium imaging movie and spatial footprints of the sources (cells).

The outputs are float values between 0 and 1 for each frame and each source, representing the probability for a cell to be active at that given frame.

The classifier we provide was trained to consider a cell as active during the rise time of its transients.

For more information, check-out our demo code.

On google colab

you can run this notebook.

On your local device

You can follow the steps described in this demo file.

1.5.2 Predicting cell type

The classifier takes as inputs the motion corrected calcium imaging movie and spatial footprints of the sources (cells).

The outputs are float values between 0 and 1 for each cell type, representing the cell type probability of a given cell.

We have trained a classifier on two cell type interneurons and pyramidal cells. For training, interneurons were identified using GadCre mouse while pyramidal cell were putative.

A .yaml file allows to configure the cell types you want to use.

We are currently improving the classifier.

For more information, check-out our demo code.

On google colab

you can run this notebook.

On your local device

You can follow the steps described in this demo file.

1.6 API Reference

This page contains auto-generated API reference documentation[1].

1.6.1 deepcinac

Subpackages

deepcinac.gui

Submodules

deepcinac.gui.cinac_gui

Module Contents

[1] Created with sphinx-autoapi

Classes

DataAndParam	
MySessionButton	Button widget.
RawFormatOptionModule	Frame widget which may contain other widgets and can have a 3D border.
ChoiceRawFormatFrame	Frame widget which may contain other widgets and can have a 3D border.
ChoiceFormatFrame	Frame widget which may contain other widgets and can have a 3D border.
ChoiceNwbFormatFrame	Frame widget which may contain other widgets and can have a 3D border.
ChoiceCinacFormatFrame	Frame widget which may contain other widgets and can have a 3D border.
ManualAction	
RemoveOnsetAction	
RemovePeakAction	
AgreePeakAction	
DontAgreePeakAction	
DontAgreeOnsetAction	
AgreeOnsetAction	
AddOnsetAction	
AddThemAllAction	
AddSegmentToSaveAction	
RemoveSegmentToSaveAction	
AddPeakAction	
AddDoubtfulFramesAction	
RemoveDoubtfulFramesAction	
AddMvtFramesAction	
RemoveMvtFramesAction	
MyCanvas	The canvas the figure renders into.
ManualOnsetFrame	Frame widget which may contain other widgets and can have a 3D border.

Functions

event_lambda(f, *args, **kwds)	
raise_above_all(window)	Take a window as argument (like tk.root()) and put it on front of all other
set_menu_from_list(keys, menu_tk, menu_variable, ...)	
load_menu_from_file(file_name, menu_tk, menu_variable)	Take a tkinter OptionMenu instance and load it using the fields of a matlab or npy file.
load_data_from_npy_or_mat_file(file_name, data_descr)	Load data from a numpy or matlab file (.npz, .npz or .mat)
display_loading_window(message, fct_to_run, args_for_fct)	
get_file_name_and_path(path_file)	
do_traces_smoothing(traces)	
print_save(text, file, to_write[, no_print])	
merge_close_values(raster, raster_to_fill, cell, ...)	**param raster** Raster is a 2d binary array, lines represents cells, columns binary values
fusion_gui_selection(path_data)	
launch_cinac_gui()	

Attributes

NWB_PACKAGE_AVAILABLE
NWB_PACKAGE_AVAILABLE

deepcinac.gui.cinac_gui.**NWB_PACKAGE_AVAILABLE = True**

deepcinac.gui.cinac_gui.**NWB_PACKAGE_AVAILABLE = False**

deepcinac.gui.cinac_gui.**event_lambda**(*f*, **args*, ***kwds*)

class deepcinac.gui.cinac_gui.**DataAndParam**(*path_data=None*)

class deepcinac.gui.cinac_gui.**MySessionButton**(*master*)

Bases: `tkinter.Button`

Button widget.

`deepcinac.gui.cinac_gui.raise_above_all(window)`

Take a window as argument (like tk.root()) and put it on front of all other :param window:

Returns:

`deepcinac.gui.cinac_gui.set_menu_from_list(keys, menu_tk, menu_variable, favorite_key)`

`deepcinac.gui.cinac_gui.load_menu_from_file(file_name, menu_tk, menu_variable, keyword_for_key=None)`

Take a tkinter OptionMenu instance and load it using the fields of a matlab or npy file. :param file_name: string :param menu_tk: OptionMenu instance :param menu_variable: Variable from the menu :param keyword_for_key: string, keyword, if found in one of the field, this field will be selected by default :param in the menu:

Returns: a list of string representing the options of the menu

`class deepcinac.gui.cinac_gui.RawFormatOptionModule(button_text, label_text, with_file_dialog, file_dialog_filetypes, file_dialog_title, root, mandatory, with_only_label=False, with_check_box=False, label_check_box=None, check_box_active_by_default=True, active_label_text=None, with_option_menu=False, with_spin_box=False, menu_keyword_for_key=None, fct_to_call_at_update=None, last_path_open=None, height_button=3, command_fct=None, default_path=None, master=None)`

Bases: `Tkinter.Frame`

Frame widget which may contain other widgets and can have a 3D border.

`check_box_action()`

`button_action()`

`activate_only_label(enabling_button=False)`

`activate_spin_box()`

`set_file_value(with_file_dialog=True, file_name=None)`

`deactivate(disabling_button=False)`

Deactivate the module Returns:

`add_exclusive_modules(exclusive_modules)`

Means that only one among self and exclusive_modules can be activated at once. When one is activated, the other are deactivated :param exclusive_modules: list of instance of RawFormatOptionModule

Returns:

`get_config_as_dict()`

From the data gui in widgets, build a dictionnary used to save the parameters Returns:

`set_config_from_dict(config_dict)`

From the data gui in widgets, build a dictionnary used to save the parameters Returns:

class deepcinac.gui.cinac_gui.**ChoiceRawFormatFrame**(*default_path=None*, *master=None*)

Bases: Tkinter.Frame

Frame widget which may contain other widgets and can have a 3D border.

__configure(*event*)

create_buttons()

Create buttons Returns:

display_only_raw_traces_check_box_action()

launch_exploratory_gui()

Returns:

load_last_used_config()

load_config()

Returns: None

save_config(*file_name=None*)

get_config_as_dict()

From the data gui in widgets, build a dictionnary used to save the paramters Returns:

set_config_from_dict(*config_dict*)

From the data gui in widgets, build a dictionnary used to save the parameters Returns:

update_launch_gui_button()

Update the launch gui button depending on the file being selected Returns:

deepcinac.gui.cinac_gui.**load_data_from_npy_or_mat_file**(*file_name*, *data_descr*, *attr_name=None*)

Load data from a numpy or matlab file (.npz, .npz or .mat) :param file_name: :param data_descr: string used to display error message if the file is not in the good format :param attr_name:

Returns:

class deepcinac.gui.cinac_gui.**ChoiceFormatFrame**(*default_path=None*, *master=None*)

Bases: Tkinter.Frame

Frame widget which may contain other widgets and can have a 3D border.

create_buttons()

open_option_format_frame(*format_str*)

deepcinac.gui.cinac_gui.**display_loading_window**(*message*, *fct_to_run*, *args_for_fct*)

class deepcinac.gui.cinac_gui.**ChoiceNwbFormatFrame**(*default_path=None*, *master=None*)

Bases: Tkinter.Frame

Frame widget which may contain other widgets and can have a 3D border.

neuronal_data_check_box_action()

activate_all_buttons()

deactivate_all_buttons()

__configure(*event*)

select_nwb_file()

Open a file dialog to select to cinac file to open and then change the GUI accordingly Returns:

_get_segmentations()

Returns: a dict that for each step till plane_segmentation represents the different option. First dict will have as keys the name of the modules, then for each modules the value will be a new dict with keys the ImageSegmentation names and then the value will be a list representing the segmentation plane

get_pixel_mask(*segmentation_info*)

Return pixel_mask which is a list of list of pair of integers representing the pixels coordinate (x, y) for each cell. the list length is the same as the number of cells. :param segmentation_info: a list of 3 elements: first one being the name of the module, then the name :param of image_segmentation and then the name of the segmentation plane.:

Returns:

_get_roi_response_serie_data(*keys*)

Parameters
keys – lsit of string allowing to get the roi repsonse series wanted

Returns:

_get_roi_response_series(*keywords_to_exclude=None*)

param: keywords_to_exclude: if not None, list of str, if one of neuronal data has this keyword, then we don't add it to the choices

Returns: a list or dict of objects representing all roi response series (rrs) names rrs could represents raw traces, or binary raster, and its link to a given segmentation. The results returned should allow to identify the segmentation associated. Object could be strings, or a list of strings, that identify a rrs and give information how to get there.

launch_exploratory_gui()

Returns:

class deepcinac.gui.cinac_gui.**ChoiceCinacFormatFrame**(*default_path=None*, *master=None*)

Bases: Tkinter.Frame

Frame widget which may contain other widgets and can have a 3D border.

launch_exploratory_gui_for_a_segment(*segment_selected*)

Launch the exploratory GUI for a given segment :param segment_selected: tuple of 3 int representing the cell, first_frame and last_frame

Returns:

launch_exploratory_gui()

Returns:

select_ci_movie()

Open a file dialog to select to calcium imagine movie Returns:

activate_all_buttons()

deactivate_all_buttons()

select_cinac_file()

Open a file dialog to select to cinac file to open and then change the GUI accordingly Returns:

__configure(*event*)

class deepcinac.gui.cinac_gui.**ManualAction**(*session_frame*, *neuron*, *is_saved*, *x_limits=None*, *y_limits=None*)

undo()

redo()

class deepcinac.gui.cinac_gui.**RemoveOnsetAction**(*removed_times*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**RemovePeakAction**(*removed_times*, *amplitudes*, *removed_onset_action=None*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AgreePeakAction**(*agreed_peaks_index*, *agree_onset_action*, *agreed_peaks_values*, *peaks_added*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**DontAgreePeakAction**(*not_agreed_peaks_index*, *dont_agree_onset_action*, *not_agreed_peaks_values*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**DontAgreeOnsetAction**(*not_agreed_onsets_index*, *not_agreed_onsets_values*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AgreeOnsetAction**(*agreed_onsets_index*, *agreed_onsets_values*, *onsets_added*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddOnsetAction**(*added_time*, *add_peak_action=None*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddThemAllAction**(*first_frame*, *last_frame*, *old_peak_values*, *new_peak_values*, *old_onset_values*, *new_onset_values*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddSegmentToSaveAction**(*segment_list*, *segment_added*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**RemoveSegmentToSaveAction**(*segment_list*, *segment_added*, *index_to_remove*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddPeakAction**(*added_time*, *amplitude*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddDoubtfulFramesAction**(*x_from*, *x_to*, *backup_values*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**RemoveDoubtfulFramesAction**(*removed_times*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**AddMvtFramesAction**(*x_from*, *x_to*, *backup_values*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

class deepcinac.gui.cinac_gui.**RemoveMvtFramesAction**(*removed_times*, ***kwargs*)

Bases: *ManualAction*

undo()

redo()

deepcinac.gui.cinac_gui.**get_file_name_and_path**(*path_file*)

deepcinac.gui.cinac_gui.**do_traces_smoothing**(*traces*)

class deepcinac.gui.cinac_gui.**MyCanvas**(*figure*, *parent_frame*, *manual_onset_frame*)

Bases: matplotlib.backends.backend_tkagg.FigureCanvasTkAgg

The canvas the figure renders into.

figure

A high-level figure instance.

Type
matplotlib.figure.Figure

button_press_event(*event*, ***args*)

Callback processing for mouse button press events.

Backend derived classes should call this function on any mouse button press. (x, y) are the canvas coords ((0, 0) is lower left). button and key are as defined in *MouseEvent*.

This method will call all functions connected to the 'button_press_event' with a *MouseEvent* instance.

class deepcinac.gui.cinac_gui.**ManualOnsetFrame**(*data_and_param*, *default_path=None*, *segment_mode=False*, *parent=None*, *title=None*)

Bases: Tkinter.Frame

Frame widget which may contain other widgets and can have a 3D border.

RAW_TRACE = 'Cell'

RAW_TRACE_WITHOUT_OVERLAP = 'Cell no over'

NEUROPIL_TRACE = 'Neuropil (Np)'

RAW_M_NEUROPIL_TRACE = 'Cell no over - Np'

neuron_entry_change(**args*)

switch_michou()

_build_traces()

Build different fluorescence signal for each cell. Fill the self.traces_dict, each key is a trace description (str), and each value is a 2d array (n cells * n frames) Returns:

switch_movie_mode(*from_movie_button=True*)

switch_mvt_display()

switch_magnifier()

set_inter_neuron()

remove_cell()

clear_and_update_center_segment_entry_widget()

clear_and_update_entry_neuron_widget()

clear_and_update_entry_cell_type_widget()

update_uncertain_prediction_values(*event=None*)

Update the widget that contain the limit of the prediction we want to look at :param event: :return:

center_segment_button_action(*event=None*)

cell_type_action(*event=None*)

Action called when the entry widget for cell type is called :param event:

Returns:

go_to_neuron_button_action(*event=None*)

key_press_action(*event*)

key_release_action(*event*)

detect_onset_associated_to_peak(*peak_times*)

Return an array with the onset times (from trace time) associated to the peak_times. We look before each peak_time (1sec before), for current_neurons :param peak_times: :return:

update_onset_times()

numbers_of_onset()

numbers_of_peak()

numbers_of_onset_to_agree()

numbers_of_peak_to_agree()

swith_all_click_actions(*initiator*)

center_segment_swith_mode(*from_center_segment_button=True*)

switch_prediction_improvement_mode()

Allows to display a signal that might change a prediction according for ex to neuropil variation or transient correlation. Mode only active if predictions are available Returns:

compute_prediction_improvement_for_cell(*cell*)

COmpute prediction improvement only if predictions available for this cell :param cell:

Returns:

add_onset_switch_mode(*with_peak=False*, *from_add_onset_button=True*)

Parameters

- **with_peak** – if True, then a peak after onset will be added automatically
- **from_add_onset_button** –

Returns:

remove_onset_switch_mode(*from_remove_onset_button=True*)

add_doubtful_frames_switch_mode(*from_add_doubtful_frames_button=True*)

remove_doubtful_frames_switch_mode(*from_remove_doubtful_frames_button=True*)

add_mvt_frames_switch_mode(*from_add_mvt_frames_button=True*)

remove_mvt_frames_switch_mode(*from_remove_mvt_frames_button=True*)

update_transient_prediction_periods_to_check()

update_predictions_list_box(*keep_same_selected_index=False*)

update_segments_to_save_list_box(*initial_loading=False*)

segments_to_save_list_box_double_click(*evt*)

Called when a double click is done on segments to save list_box. It removes from the list the element that is double clicked :param evt:

Returns:

remove_segment_to_save(*index_to_remove*)

change_prediction_transient_to_look_at(*period*)

CHange the plot such as the period indicated is displayed :param period: a tuple of len 4: cell, x_left, x_right, pred :return:

predictions_list_box_click(*evt*)

segments_to_save_list_box_click(*evt*)

predictions_list_box_double_click(*evt*)

add_current_segment_to_save_list()

Add the current view as a segment. The view should have a minimum of 10 frames. The view should not be already added. Returns:

update_contour_for_cell(*cell*)

agree_switch_mode(*from_agree_button=True*)

dont_agree_switch_mode(*from_dont_agree_button=True*)

remove_peak_switch_mode(*from_remove_peak_button=True*)

remove_all_switch_mode(*from_remove_all_button=True*)

onrelease_map(*event*)

Action when a mouse button is released on cell map :param event: :return:

onrelease(*event*)

Action when a mouse button is released :param event: :return:

motion(*event*)

Action when the mouse is moved :param event: :return:

onclick(*event*)

Action when a mouse button is pressed :param event: :return:

remove_onset(*x_from*, *x_to*)

dont_agree_on_fusion(*x_from*, *x_to*)

agree_on_fusion(*x_from*, *x_to*)

remove_all(*x_from*, *x_to*)

remove_peaks_under_threshold()

remove_peak(*x_from*, *x_to*)

switch_trace_to_be_displayed(*trace_str*)

Called when one of the trace checkbox change selection. Allows to change the trace displayed

Returns:

switch_source_profile_mode(*from_check_box=True*, *from_key_shortcut=False*, *from_magnifier=False*)

activate_movie_zoom(*from_check_box=True*)

set_transient_classifier_prediction_for_cell(*cell*)

transient_classifier_check_box_action()

set_cell_type_classifier_prediction_for_cell(*cell*)

cell_type_classifier_button_action()

Method called when pushing cell type classification button Returns:

display_cell_type_predictions()

Display in the cell type labels the predictions for the current_neuron Returns:

correlation_check_box_action(*from_std_treshold=False*)

threshold_check_box_action(*from_correlation=False*)

spin_box_pixels_around_cell_update()

spin_box_transient_classifier_update()

spin_box_threshold_update()

spin_box_correlation_update()

unsaved()

means a changed has been done, and the actual plot is not saved

update_doubtful_frames_periods(*cell*)

update_mvt_frames_periods(*cell*)

normalize_traces()

Normalize the fluorescence signal using z-score and change the value of smooth smooth_traces so there are displayed under the raw smooth_traces Returns:

add_onset(*at_time*)

add_peak(*at_time*, *amplitude=0*)

update_last_action(*new_action*, *from_redo_action=False*)

Keep the size of the last_actions up to five actions :param new_action: :return:

add_peak_switch_mode(*from_add_peak_button=True*)

Parameters
from_add_peak_button – indicate the user click on the add_peak button, otherwise it means the

function has been called after another button has been clicked :return:

remove_doubtful_frames(*x_from*, *x_to*)

add_doubtful_frames(*x_from*, *x_to*)

remove_mvt_frames(*x_from*, *x_to*)

add_mvt_frames(*x_from*, *x_to*)

validation_before_closing()

redo_action()

undo_action()

Revoke the last action :return:

save_sources_profile_map(*key_cmap=None*)

get_square_coord_around_cell(*cell*, *x_len_max*, *y_len_max*, *square_size*)

Parameters

- **cell** –
- **x_len_max** – max x value (border of the movie)
- **y_len_max** – max y value (border of the movie)
- **square_size** – number of pixels that compose the border of the square

Returns: Two int representing the minx and miny of the square that will be used to train the classifier

save_segments(*and_close=False*)

save_segments_as(*and_close=False*)

Open a filedialog to save the data in a .cinac extension (hdf5 format). The data saved correspond to the onset and peak for the cells and frames displayed over the smooth_traces. Returns: None

get_threshold()

plot_magnifier(*first_time=False*, *mouse_x_position=None*, *mouse_y_position=None*)

Plot the magnifier :param first_time: if True, means the function is called for the first time, allow sto initialize some variables. :param mouse_x_position: indicate the x position of the mouse cursor :param mouse_y_position: indicate the y position of the mouse cursor :return: None

draw_magnifier_marker(*mouse_x_position=None*, *mouse_y_position=None*)

corr_between_source_and_transient(*cell*, *transient*, *pixels_around=1*, *redo_computation=False*)

Parameters

- **cell** – int
- **transient** – (int, int) first_frame and last_frame
- **pixels_around** –

- **redo_computation** – if True, means that even if the correlation has been done before for the peak,

it will be redo (useful if the onset has changed for exemple :return:

compute_source_and_transients_correlation(*main_cell*, *redo_computation=False*, *with_overlapping_cells=True*)

Compute the source and transient profiles of a given cell. Should be call for each new neuron displayed :param cell: :param redo_computation: if True, means that even if the correlation has been done before for this cell, it will be redo (useful if the onsets or peaks has changed for exemple) :return:

plot_source_transient(*transient*)

get_source_profile(*cell*, *pixels_around=0*, *bounds=None*, *buffer=None*, *with_full_frame=False*)

Parameters

- **cell** –
- **pixels_around** –
- **bounds** –
- **buffer** –
- **with_full_frame** – Average the full frame

Returns

get_transient_profile(*cell*, *transient*, *pixels_around=0*, *bounds=None*)

get_cell_new_coord_in_source(*cell*, *minx*, *miny*)

update_plot_magnifier(*mouse_x_position*, *mouse_y_position*, *change_frame_ref*)

start_playing_movie(*x_from*, *x_to*)

square_coord_around_cell(*cell*, *size_square*, *x_len_max*, *y_len_max*)

For a given cell, give the coordinates of the square surrounding the cell.

Parameters

- **cell** –
- **size_square** –
- **x_len_max** –
- **y_len_max** –

Returns

(x_beg, x_end, y_beg, y_end)

animate_movie(*i*)

plot_map_img(*first_time=True*, *after_movie=False*)

draw_cell_contour()

update_plot_map_img(*after_michou=False*, *after_movie=False*)

plot_graph(*first_time=False*)

Parameters
first_time –

Returns

current_max_amplitude()

Ceiling value :return:

move_zoom(*to_the_left*)

add_them_all()

Add all possible onset and peaks based on change of derivative in the smooth trace Returns:

update_plot(*new_neuron=False*, *amplitude_zoom_fit=True*, *new_x_limit=None*, *new_y_limit=None*, *changing_face_color=False*, *new_trace=False*, *raw_trace_display_action=False*)

Parameters

- **new_neuron** –
- **amplitude_zoom_fit** –
- **new_x_limit** –
- **new_y_limit** –
- **changing_face_color** –
- **new_trace** – if True, means the trace has changed, and so we want to adapt the y-axis limit
- **raw_trace_display_action** –

Returns:

update_to_agree_label()

update_after_onset_change(*new_neuron=-1*, *new_x_limit=None*, *new_y_limit=None*)

Update the frame if an onset change has been made :param new_neuron: if -1, then the neuron hasn't changed, neuron might change if undo or redo are done. :return:

select_previous_neuron()

select_next_neuron()

go_to_next_cell_with_same_type()

If cell typeis know, allows to display the next cell from the cell type as the cell actualy displayed Returns:

update_neuron(*new_neuron*, *new_x_limit=None*, *new_y_limit=None*, *amplitude_zoom_fit=True*)

Call when the neuron number has changed :return:

deepcinac.gui.cinac_gui.**print_save**(*text*, *file*, *to_write*, *no_print=False*)

deepcinac.gui.cinac_gui.**merge_close_values**(*raster*, *raster_to_fill*, *cell*, *merging_threshold*)

Parameters

- **raster** – Raster is a 2d binary array, lines represents cells, columns binary values
- **cell** – which cell to merge
- **merging_threshold** – times separation between two values under which to merge them

Returns

`deepcinac.gui.cinac_gui.`**`fusion_gui_selection`**(*path_data*)

`deepcinac.gui.cinac_gui.`**`launch_cinac_gui`**()

deepcinac.utils

Submodules

deepcinac.utils.cells_map_utils

Module Contents

Classes

CellsCoord	

Functions

create_cells_coord_from_suite_2p(is_cell_file_name, ...)	**param is_cell_file_name** path and file_name of the file iscell.npy produce by suite2p segmentation process
get_coords_extracted_from_fiji(file_name)	Extract coords from file_name, should be z .zip or .roi file
_angle_to_point(point, centre)	calculate angle in 2-D between points and x axis
area_of_triangle(p1, p2, p3)	calculate area of any triangle given co-ordinates of the corners
convex_hull(points[, smidgen])	from: https://stackoverflow.com/questions/17553035/draw-a-smooth-polygon-around-data-points-in-a-scatter-plot-in-matplotlib

`deepcinac.utils.cells_map_utils.`**`create_cells_coord_from_suite_2p`**(*is_cell_file_name*, *stat_file_name*, *movie_dimensions*)

Parameters

- **`is_cell_file_name`** – path and file_name of the file iscell.npy produce by suite2p segmentation process
- **`None`** (*if*) –
- **`None`** –
- **`keep`** (*means all cells in stat are cells to*) –
- **`stat_file_name`** – path and file_name of the file stat.npy produce by suite2p segmentation process
- **`movie_dimensions`** – tuple of 2 int, dimensions of the movie n col x n line

Returns: a CellsCoord instance

deepcinac.utils.cells_map_utils.**get_coords_extracted_from_fiji**(*file_name*)

Extract coords from file_name, should be z .zip or .roi file :param file_name: .zip or .roi file

Returns: Array of len the number of contours made of array of 2 lines (x & y) and n columns, n being the number of coordinate to set the contour of each cell.

class deepcinac.utils.cells_map_utils.**CellsCoord**(*coords=None*, *pixel_masks=None*, *nb_lines=None*, *nb_col=None*, *from_matlab=False*, *invert_xy_coord=False*)

build_raw_traces_from_movie(*movie*, *without_overlap=False*, *buffer_overlap=1*)

Return a 2d array representing the fluoresence signal raw trace for each cell :param movie: 3d array n_frames x len_y x len_x :param without_overlap: (bool) if True, means the trace will be build only from the pixels from this cell :param buffer_overlap: indicate from how much pixels increasing the size of overlaping cell

Returns: A 2d array (n_cells * n_frames) of float

build_cell_polygon_from_contour(*cell*)

Build the (shapely) polygon representing a given cell using its contour's coordinates. :param cell:

Returns:

get_cell_mask(*cell*, *dimensions*, *without_overlap=False*, *buffer_overlap=1*)

Return the mask of the pixels of the cell :param cell: :param dimensions: height x width :param without_overlap: if True, means with return only the pixel belonging to this cell

buffer_overlap: indicate from how much pixels increasing the size of overlaping cell

Returns

binary 2d array (movie dimension), with 1 for the pixels belonging to the cell

match_cells_indices(*coord_obj*, *path_results*, *plot_title_opt=''*)

Parameters

coord_obj – another instanc of coord_obj

Returns

a 1d array, each index corresponds to the index of a cell of coord_obj, and map it to an index to self

or -1 if no cell match

plot_cells_map(*path_results*, *data_id*, *use_pixel_masks=False*, *title_option=''*, *connections_dict=None*, *background_color=(0, 0, 0, 1)*, *default_cells_color=(1, 1, 1, 1.0)*, *default_edge_color='white'*, *dont_fill_cells_not_in_groups=False*, *link_connect_color='white'*, *link_line_width=1*, *cell_numbers_color='dimgray'*, *show_polygons=False*, *cells_to_link=None*, *edge_line_width=2*, *cells_alpha=1.0*, *fill_polygons=True*, *cells_groups=None*, *cells_groups_colors=None*, *cells_groups_alpha=None*, *cells_to_hide=None*, *img_on_background=None*, *real_size_image_on_bg=True*, *cells_groups_edge_colors=None*, *with_edge=False*, *with_cell_numbers=False*, *text_size=6*, *save_formats='png'*, *save_plot=True*, *return_fig=False*, *ax_to_use=None*, *verbose=False*, *use_welsh_powell_coloring=False*, *dpi=300*)

Parameters

- **path_results** –

- **data_id** –
- **use_pixel_masks** –
- **title_option** –
- **connections_dict** – key is an int representing a cell number, and value is a dict representing the cells it
- **too** (*connects to. The key is a cell is connected*) –
- **connection** (*and the value represent the strength of the*) –
- **it)** (*(like how many times it connects to*) –
- **background_color** –
- **default_cells_color** –
- **default_edge_color** –
- **dont_fill_cells_not_in_groups** –
- **link_connect_color** –
- **link_line_width** –
- **cell_numbers_color** –
- **show_polygons** –
- **cells_to_link** –
- **edge_line_width** –
- **cells_alpha** –
- **fill_polygons** –
- **cells_groups** –
- **cells_groups_colors** –
- **cells_groups_alpha** –
- **cells_to_hide** –
- **img_on_background** –
- **real_size_image_on_bg** – if True, the size of the figure will respect the original size of the background
- **image** –
- **cells_groups_edge_colors** –
- **with_edge** –
- **with_cell_numbers** –
- **text_size** –
- **save_formats** –
- **save_plot** –
- **return_fig** –
- **ax_to_use** –
- **verbose** – if True, some informations will be printed along the way

- **use_welsh_powell_coloring** – if True, use welsh powell algorithm to color all cells that intersect with

different color. In that case, cancel cell_groups arguments.

dpi:

Returns:

add_cells_using_pixel_masks_on_ax(*ax*, *cells_groups*, *cells_not_in_groups*, *cells_to_hide*, *default_cells_color*, *cells_groups_colors*, *with_cell_numbers*, *cell_numbers_color*, *text_size*, *background_color*, *cells_imshow_alpha*, *bg_imshow_alpha*)

Using pixel mask if it exists :param ax: :param cells_groups: :param cells_not_in_groups: :param cells_to_hide: :param default_cells_color: :param cells_groups_colors: :param with_cell_numbers: :param cell_numbers_color: :param text_size: :param background_color:

Returns:

add_cells_using_polygons_on_ax(*ax*, *cells_groups*, *cells_not_in_groups*, *cells_to_hide*, *with_edge*, *edge_line_width*, *default_cells_color*, *default_edge_color*, *cells_groups_edge_colors*, *cells_groups_colors*, *cells_groups_alpha*, *cells_alpha*, *with_cell_numbers*, *cell_numbers_color*, *text_size*, *dont_fill_cells_not_in_groups*)

Add cells to a matplolib ax using the polygons representation. Arguments give parameters to apply :param ax: :param cells_groups: :param cells_not_in_groups: :param cells_to_hide: :param with_edge: :param edge_line_width: :param default_cells_color: :param default_edge_color: :param cells_groups_edge_colors: :param cells_groups_colors: :param cells_groups_alpha: :param cells_alpha: :param with_cell_numbers: :param cell_numbers_color: :param text_size: :param dont_fill_cells_not_in_groups:

Returns:

plot_text_cell(*cell*, *ax*, *cell_numbers_color*, *text_size*)

Plot the cell number on the cell :param cell: integer :param ax: matplolib axis :param cell_numbers_color: color of the text :param text_size: text size (float)

Returns:

get_cell_new_coord_in_source(*cell*, *minx*, *miny*)

scale_polygon_to_source(*poly_gon*, *minx*, *miny*)

get_source_profile(*cell*, *tiff_movie*, *traces*, *peak_nums*, *spike_nums*, *pixels_around=0*, *bounds=None*, *buffer=None*, *with_full_frame=False*)

Return the source profile of a cell :param cell: :param pixels_around: :param bounds: how much padding around the cell pretty much, coordinate of the frame covering the source profile 4 int list :param buffer: :param with_full_frame: Average the full frame :return:

get_transient_profile(*cell*, *transient*, *tiff_movie*, *traces*, *pixels_around=0*, *bounds=None*)

corr_between_source_and_transient(*cell*, *transient*, *source_profile_dict*, *tiff_movie*, *traces*, *source_profile_corr_dict=None*, *pixels_around=1*)

Measure the correlation (pearson) between a source and transient profile for a giveb cell :param cell: :param transient: :param source_profile_dict should contains cell as key, and results of get_source_profile avec values :param pixels_around: :param source_profile_corr_dict: if not None, used to save the correlation of the source profile, f for memory and computing proficiency :return:

`deepcinac.utils.cells_map_utils.`**`_angle_to_point`**(*point*, *centre*)

calculate angle in 2-D between points and x axis

`deepcinac.utils.cells_map_utils.`**`area_of_triangle`**(*p1*, *p2*, *p3*)

calculate area of any triangle given co-ordinates of the corners

`deepcinac.utils.cells_map_utils.`**`convex_hull`**(*points*, *smidgen=0.0075*)

from: https://stackoverflow.com/questions/17553035/draw-a-smooth-polygon-around-data-points-in-a-scatter-plot-in-matplotlib Calculate subset of points that make a convex hull around points Recursively eliminates points that lie inside two neighbouring points until only convex hull is remaining.

Parameters

points : ndarray (2 x m) array of points for which to find hull use pylab to show progress? smidgen : float offset for graphic number labels - useful values depend on your data range

Returns

hull_points : ndarray (2 x n) convex hull surrounding points

deepcinac.utils.cinac_file_utils

Module Contents

Classes

CinacFileReader_open_close
CinacFileWriter
CinacFileReader

Functions

read_cell_type_categories_yaml_file(yaml_file[, ...])	Read cell type categories from a yaml file. If more than 2 type cells are given, then a multi-class
create_tiffs_from_movie(path_for_tiffs, movie_identifier)	Take a Tiff movie or 3d array representing it, and save an unique tiff file for each of its frame.

class `deepcinac.utils.cinac_file_utils.`**`CinacFileReader_open_close`**(*file_name*, *frames_to_keep=None*)

`_open_file`()

`close_file`()

Close the file Returns:

`create_new_cinac_file_for_segment_chunk`(*dir_path*, *segment*, *first_frame*, *last_frame*)

create_cinac_file_for_each_segment(*dir_path*, *return_file_readers*)

For each segment in the instance, it created a .cinac file that will contain just that sequence. :param dir_path: Directory in which save the new .cinac files :param return_file_readers: (bool) if True return a list of instances of :param CinacFileReader from the individual .cinac files created:

Returns:

__building_segments_list()

_get_segment_group(*segment*)

Return the group from cinac_file, it should be open :param segment:

Returns:

get_coords_full_movie()

Returns:

get_invalid_cells()

Return the invalid cells

Returns: 1d array of n cells, as many cells. Binary array, 0 is valid, 1 if invalid Return None if no

with_full_data()

Return True if full data is available, meaning coords of cells in the original movie, invalid cells Returns:

get_ci_movie_file_name()

Returns the name of full calcium imaging movie file_name from which the data are extracted. None if the file_name is unknown. Returns:

get_all_segments()

Return a list of tuple of 3 int (cell, first_frame, last_frame) representing the segments of ground truth available in this file Returns: list

get_n_frames_gt()

Return the number of frames with ground truth Returns:

get_n_active_frames()

Return the number of frames with cells being active Returns:

fill_doubtful_frames_from_segments(*doubtful_frames_nums*)

Fill the doubtful_frames_nums using the ground truth from the segments. :param doubtful_frames_nums: 2d arrays (n_cells x n_frames)

Returns:

fill_raster_dur_from_segments(*raster_dur*)

Fill the raster_dur using the ground truth from the segments. :param raster_dur: 2d arrays (n_cells x n_frames)

Returns:

get_segment_ci_movie(*segment*)

Return the calcium imaging from the ground truth segment. :param segment: segment to use to get ci_movie, tuple of 3 to 4 int

Returns: 3d array

get_segment_ci_movie_frames(*segment*, *frames*)

Return frames from the calcium imaging from the ground truth segment. :param segment: segment to use to get ci_movie, tuple of 3 to 4 int

Returns: 3d array

get_segment_cell_type(*segment*)

Return the name of the cell type from the segment, or None if this information is not known. :param segment: segment

Returns:

get_segment_pixels_around(*segment*)

Return the pixels_around used to produce the cell profile on the frame (not really used anymore). :param segment: segment

Returns:

get_segment_buffer(*segment*)

Return the buffer used to produce the cell profile on the frame (not really used anymore). :param segment: segment

Returns:

get_all_cell_types()

Return a dict with as a key the cell index and value a string representing the cell type. Covers all the cells represented by the segments.

Returns:

get_segment_smooth_traces(*segment*)

Return the smooth fluorescence signal from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: 1d array

get_segment_raw_traces(*segment*)

Return the smooth fluorescence signal from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: 1d array

get_segment_cells_contour(*segment*)

Return the cells contour from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: a list of 2d array that encodes x, y coord (len of the 2d array corresponds to the number of point in the contour.

get_segment_raster_dur(*segment*)

Return the raster_dur from the ground truth segment. :param segment: segment to use to get raster_dur

Returns: 1d array of n frames as specified in segment

get_segment_invalid_cells(*segment*)

Return the invalid cells from the ground truth segment. :param segment: segment (tuple of 3 int)

Returns: 1d array of n cells, as many cells as in the segment (cell of interest + interesections). Binary, 0 is valid, 1 if invalid

get_segment_doubtful_frames(*segment*)

Return the doubtful_frames from the ground truth segment. :param segment: segment (tuple of 3 int)

Returns: 1d array of n frames, as many frames as in the segment. Binary, 0 is not doubtful, 1 if doubtful

class deepcinac.utils.cinac_file_utils.**CinacFileWriter**(*file_name*)

close_file()

Close the file Returns:

delete_groups(*group_names*)

get_group_names()

add_segment_group(*cell*, *first_frame*, *last_frame*, *raster_dur*, *ci_movie*, *cells_contour*, *pixels_around*, *buffer*, *invalid_cells*, *smooth_traces*, *raw_traces*, *cell_type=None*, *doubtful_frames=None*)

Add just a segment (some frames and a given cell) :param cell: :param first_frame: :param last_frame: :param raster_dur: :param ci_movie: :param cells_contour: :param pixels_around: :param buffer: :param invalid_cells: :param smooth_traces: normalized (z-scored) smmoth fluorescence signal of the cell during the give frames :param doubtful_frames: :param raw_traces: normalized (z-scored) raw fluorescence signal of the cell during the give frames :param cell_type: any string describing the cell type, in our case it would be "interneuron" or "pyr" :param (for pyramidal cell). If None: :param means we don't encode this information: :param the cell type is: :param not known ?:

Returns:

create_full_data_group(*save_only_movie_ref*, *save_ci_movie_info*, *cells_contour*, *n_frames*, *n_cells*, *smooth_traces=None*, *raw_traces=None*, *ci_movie_file_name=None*, *ci_movie=None*, *invalid_cells=None*)

Create a group that represents all data (full movie, all cells etc…) :param save_only_movie_ref: boolean, if True means we just save the path & file_name of the calcium imaging movie. :param Otherwise the full movie is saved if ci_movie argument is passed.: :param save_ci_movie_info: boolean, if True then either the ci movie ref or the full data is saved in the file :param n_frames: number of frames in the full movie :param n_cells: number of cells segmented (should be the length of cells_contour) :param cells_contour: a list of 2d np.array representing the coordinates of the contour points :param smooth_traces: Smooth fluorescence signals of the cells (z-score) :param raw_traces: Raw fluorescence signals of the cells (z-score) :param ci_movie_file_name: :param ci_movie: np.array should be 3d: n_frames*len_x*len_y, calcium imaging data :param invalid_cells: a binary np.array of the length the number of cells, set to True or 1 is the cell is invalid.

Returns:

get_n_cells()

Returns: the number of cells in the movie that have been segmented. Return None if this information if not available (need the full_data group to exists)

deepcinac.utils.cinac_file_utils.**read_cell_type_categories_yaml_file**(*yaml_file*, *using_multi_class=2*)

Read cell type categories from a yaml file. If more than 2 type cells are given, then a multi-class classifier will be used. If 2 type cells are given, then either it could be multi-class or binary classifier, then this choice should be given in the parameters of CinacModel. If 2 cell-type are given, for binary classifier, it should be precised which cell type should be predicted if we get more than 0.5 probability. :param yaml_file: :param using_multi_class: int, give the default number of classes used, not necessary if already :param put in the yaml file:

Returns: cell_type_from_code_dict, cell_type_to_code_dict, multi_class_arg cell_type_from_code_dict: dict, key is an int, value is the cell_type cell_type_to_code_dict: dict, key is a string, value is the code of the

cell_type. A code can have more than one string associated, but all of them represent the same cell_type defined in cell_type_from_code_dict multi_class_arg: is None if no multi_class_arg was given in the yaml_file, True or False, if False means we want to use a binary classifier

class deepcinac.utils.cinac_file_utils.**CinacFileReader**(*file_name*, *frames_to_keep=None*)

close_file()

Close the file Returns:

create_new_cinac_file_for_segment_chunk(*dir_path*, *segment*, *first_frame*, *last_frame*)

create_cinac_file_for_each_segment(*dir_path*, *return_file_readers*)

For each segment in the instance, it created a .cinac file that will contain just that sequence. :param dir_path: Directory in which save the new .cinac files :param return_file_readers: (bool) if True return a list of instances of :param CinacFileReader from the individual .cinac files created:

Returns:

__building_segments_list()

get_coords_full_movie()

Returns:

get_n_cells()

Return the number of cells with contours in this movie (if the information is available, None otherwise) Returns:

get_n_frames()

Return the number of frames in the full movie Returns:

get_invalid_cells()

Return the invalid cells

Returns: 1d array of n cells, as many cells. Binary array, 0 is valid, 1 if invalid Return None if no

with_full_data()

Return True if full data is available, meaning coords of cells in the original movie, invalid cells Returns:

get_ci_movie_file_name()

Returns the name of full calcium imaging movie file_name from which the data are extracted. None if the file_name is unknown. Returns:

get_all_segments()

Return a list of tuple of 3 int (cell, first_frame, last_frame) representing the segments of ground truth available in this file Returns: list

get_n_frames_gt()

Return the number of frames with ground truth Returns:

get_n_active_frames()

Return the number of frames with cells being active Returns:

fill_doubtful_frames_from_segments(*doubtful_frames_nums*)

Fill the doubtful_frames_nums using the ground truth from the segments. :param doubtful_frames_nums: 2d arrays (n_cells x n_frames)

Returns:

fill_raster_dur_from_segments(*raster_dur*)

Fill the raster_dur using the ground truth from the segments. :param raster_dur: 2d arrays (n_cells x n_frames)

Returns:

get_segment_ci_movie(*segment*)

Return the calcium imaging from the ground truth segment. :param segment: segment to use to get ci_movie, tuple of 3 to 4 int

Returns: 3d array

get_segment_ci_movie_frames(*segment*, *frames*)

Return frames from the calcium imaging from the ground truth segment. :param segment: segment to use to get ci_movie, tuple of 3 to 4 int

Returns: 3d array

get_segment_cell_type(*segment*)

Return the name of the cell type from the segment, or None if this information is not known. :param segment: segment

Returns:

get_segment_pixels_around(*segment*)

Return the pixels_around used to produce the cell profile on the frame (not really used anymore). :param segment: segment

Returns:

get_segment_buffer(*segment*)

Return the buffer used to produce the cell profile on the frame (not really used anymore). :param segment: segment

Returns:

get_all_cell_types()

Return a dict with as a key the cell index and value a string representing the cell type. Covers all the cells represented by the segments.

Returns:

get_segment_smooth_traces(*segment*)

Return the smooth fluorescence signal from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: 1d array

get_segment_raw_traces(*segment*)

Return the smooth fluorescence signal from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: 1d array

get_segment_cells_contour(*segment*)

Return the cells contour from the ground truth segment. :param segment: segment to use to fill raster_dur, tuple of 3 to 4 int

Returns: a list of 2d array that encodes x, y coord (len of the 2d array corresponds to the number of point in the contour.

get_segment_raster_dur(*segment*)

Return the raster_dur from the ground truth segment. :param segment: segment to use to get raster_dur

Returns: 1d array of n frames as specified in segment

get_segment_invalid_cells(*segment*)

Return the invalid cells from the ground truth segment. :param segment: segment (tuple of 3 int)

Returns: 1d array of n cells, as many cells as in the segment (cell of interest + interesections). Binary, 0 is valid, 1 if invalid

get_segment_doubtful_frames(*segment*)

Return the doubtful_frames from the ground truth segment. :param segment: segment (tuple of 3 int)

Returns: 1d array of n frames, as many frames as in the segment. Binary, 0 is not doubtful, 1 if doubtful

deepcinac.utils.cinac_file_utils.**create_tiffs_from_movie**(*path_for_tiffs*, *movie_identifier*, *movie_file_name=None*, *movie_data=None*)

Take a Tiff movie or 3d array representing it, and save an unique tiff file for each of its frame. Save as well the mean and std as npy file, all in the directory path_for_tiffs, in a directory with the identifier :param path_for_tiffs: str :param movie_identifier: str :param movie_file_name: str :param movie_data: 3d array

Returns: boolean, return False if the directory with this identifier already exists, True if Tiffs have been created

deepcinac.utils.display

Module Contents

Functions

plot_hist_distribution(distribution_data, description)	Plot a distribution in the form of an histogram, with option for adding some scatter values
plot_spikes_raster([spike_nums, title, file_name, ...])	Plot or save a raster given a 2d array either binary representing onsets, peaks or rising time, or made of float

Attributes

BREWER_COLORS

deepcinac.utils.display.**BREWER_COLORS** = ['#a6cee3', '#1f78b4', '#b2df8a', '#33a02c', '#fb9a99', '#e31a1c', '#fdbf6f', '#ff7f00',...

deepcinac.utils.display.**plot_hist_distribution**(*distribution_data*, *description*, *param=None*, *values_to_scatter=None*, *xticks_labelsize=10*, *yticks_labelsize=10*, *x_label_font_size=15*, *y_label_font_size=15*, *labels=None*, *scatter_shapes=None*, *colors=None*, *tight_x_range=False*, *twice_more_bins=False*, *background_color='black'*, *labels_color='white'*, *xlabel=''*, *ylabel=None*, *path_results=None*, *save_formats='pdf'*, *v_line=None*, *x_range=None*, *ax_to_use=None*, *color_to_use=None*)

Plot a distribution in the form of an histogram, with option for adding some scatter values :param distribution_data: :param description: :param param: :param values_to_scatter: :param labels: :param scatter_shapes: :param colors: :param tight_x_range: :param twice_more_bins: :param xlabel: :param ylabel: :param save_formats: :return:

deepcinac.utils.display.**plot_spikes_raster**(*spike_nums=None*, *title=None*, *file_name=None*, *time_str=None*, *spike_train_format=False*, *y_ticks_labels=None*, *y_ticks_labels_size=None*, *y_ticks_labels_color='white'*, *x_ticks_labels_color='white'*, *x_ticks_labels=None*, *x_ticks_labels_size=None*, *x_ticks=None*, *hide_x_labels=False*, *figure_background_color='black'*, *without_ticks=True*, *save_raster=False*, *show_raster=False*, *plot_with_amplitude=False*, *activity_threshold=None*, *save_formats='png'*, *span_area_coords=None*, *span_area_colors=None*, *span_area_only_on_raster=True*, *alpha_span_area=0.5*, *cells_to_highlight=None*, *cells_to_highlight_colors=None*, *color_peaks_activity=False*, *horizontal_lines=None*, *horizontal_lines_colors=None*, *horizontal_lines_sytle=None*, *horizontal_lines_linewidth=None*, *vertical_lines=None*, *vertical_lines_colors=None*, *vertical_lines_sytle=None*, *vertical_lines_linewidth=None*, *scatters_on_traces=None*, *scatters_on_traces_marker='*'*, *scatters_on_traces_size=5*, *sliding_window_duration=1*, *show_sum_spikes_as_percentage=False*, *span_cells_to_highlight=None*, *span_cells_to_highlight_colors=None*, *spike_shape='|'*, *spike_shape_size=10*, *raster_face_color='black'*, *cell_spikes_color='white'*, *activity_sum_plot_color='white'*, *activity_sum_face_color='black'*, *y_lim_sum_activity=None*, *seq_times_to_color_dict=None*, *link_seq_categories=None*, *link_seq_color=None*, *min_len_links_seq=3*, *link_seq_line_width=1*, *link_seq_alpha=1*, *jitter_links_range=1*, *display_link_features=True*, *seq_colors=None*, *debug_mode=False*, *axes_list=None*, *SCE_times=None*, *ylabel=None*, *without_activity_sum=False*, *spike_nums_for_activity_sum=None*, *spikes_sum_to_use=None*, *size_fig=None*, *cmap_name='jet'*, *traces=None*, *display_traces=False*, *display_spike_nums=True*, *traces_lw=0.3*, *path_results=None*, *without_time_str_in_file_name=False*, *desaturate_color_according_to_normalized_amplitude=False*, *lines_to_display=None*, *lines_color='white'*, *lines_width=1*, *lines_band=0*, *lines_band_color='white'*, *use_brewer_colors_for_traces=False*, *dpi=100*)

Plot or save a raster given a 2d array either binary representing onsets, peaks or rising time, or made of float to represents traces or encoding in onset/peaks/rising time a value. :param spike_nums: np.array of 2D, axis=1 (lines) represents the cells, the columns representing the spikes It could be binary, or containing the amplitude, if amplitudes values should be display put plot_with_amplitude to True :param spike_train_format: if True, means the data is a list of np.array, and then spike_nums[i][j] is a timestamps value as float :param title: title to be plot :param file_name: name of the file if save_raster is True :param save_raster: if True, the plot will be save. To do so param should not be None and contain a variable path_results that will indicated where to save file_name :param show_raster: if True, the plot will be shown :param plot_with_amplitude: to display a color bar representing the content values. :param activity_threshold: Int representing a threshold that will be display as a red line on the sum of activity subplot. :param save_formats: string or list of string representing the formats in which saving the

raster. Exemple: "pdf" or ["pdf", "png"] :param span_area_coords: List of list of tuples of two float representing coords (x, x) of span band with a color corresponding to the one in span_area_colors :param span_area_colors: list of colors, same len as span_area_coords :param span_area_only_on_raster: if True, means the span won't be on the sum of activity on the sub-plot as well :param cells_to_highlight: cells index to span (y-axis) with special spikes color, list of int :param cells_to_highlight_colors: cells colors to span, same len as cells_to_span, list of string :param color_peaks_activity: if True, will span to the color of cells_to_highlight_colors each time at which a cell among cells_to_highlight will spike on the activity peak diagram :param horizontal_lines: list of float, representing the y coord at which trace horizontal lines :param horizontal_lines_colors: if horizontal_lines is not None, will set the colors of each line, list of string or color code :param horizontal_lines_style: give the style of the lines, string :param vertical_lines: list of float, representing the x coord at which trace vertical lines :param vertical__lines_colors: if horizontal_lines is not None, will set the colors of each line, list of string or color code :param vertical__lines_style: give the style of the lines, string :param vertical_lines_linewidth: linewidth of vertical_lines :param raster_face_color: the background color of the raster :param cell_spikes_color: the color of the spikes of the raster :param spike_shape: shape of the spike, "|", "*", "o" :param spike_shape_size: use for shape != of "|" :param seq_times_to_color_dict: None or a dict with as the key a tuple of int representing the cell index, and as a value a list of set, each set composed of int representing the times value at which the cell spike should be colored. It will be colored if there is indeed a spike at that time otherwise, the default color will be used. :param seq_colors: A dict, with key a tuple represening the indices of the seq and as value of colors, a color, should have the same keys as seq_times_to_color_dict :param link_seq_color: if not None, give the color with which link the spikes from a sequence. If not None, seq_colors will be ignored. could be a dict with key same tuple as seq_times_to_color_dict or a string and then we use the same color for all seq :param min_len_links_seq: minimum len of a seq for the links to be drawn :param axes_list if not None, give a list of axes that will be used, and be filled, but no figure will be created or saved then. Doesn't work yet is show_amplitude is True :param SCE_times: a list of tuple corresponding to the first and last index of each SCE, (last index being included in the SCE). Will display the position of the SCE and their number above the activity diagram. If None, the overall time will be displayed. Need to be adapted to the format spike_numw or spike_train. Equivalent to span_are_coords :param without_activity_sum: if True, don't plot the sum of activity diagram, valid only if axes_list is not None :param spike_nums_for_activity_sum: if different that the one given for the raster, should be the same second dimension :param spikes_sum_to_use: an array of 1D, that will be use to display the sum of activity, :param size_fig: tuple of int :param cmap_name: "jet" by default, used if with_amplitude for the colormap :param traces if not None and display_traces is True, will display traces instead of a raster :param display_traces, if True display traces :param display_spike_nums, if False, won't display a raster using spike_nums :param traces_lw, default 0.3, linewidth of the traces :param path_results: indicate where to save the plot, replace the param.path_results if it exists :param desaturate_color_according_to_normalized_amplitude: if True, spike_nums should be filled with float between 0 and 1, representing the amplitude of the spike. And if a color is given for a cell, then it will be desaturate according to this value :param lines_to_display, dict that takes for a key a tuple of int representing 2 cells, and as value a list of tuple of 2 float representing the 2 extremities of a line between those 2 cells. By defualt, no lines :param lines_color="white": colors of lines_to_display :param lines_width=1: width of lines_to_display :param lines_band=0: if > 0, display a band around the line with transparency :param lines_band_color="white" :return:

deepcinac.utils.signal

Module Contents

Functions

smooth_convolve(x[, window_len, window])	smooth the data using a window with requested size.

deepcinac.utils.signal.**smooth_convolve**(*x*, *window_len=11*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

TODO: the window parameter could be the window itself if an array instead of a string NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

`deepcinac.utils.utils`

Module Contents

Functions

smooth_convolve(x[, window_len, window])	smooth the data using a window with requested size.
get_continous_time_periods(binary_array)	take a binary array and return a list of tuples representing the first and last position(included) of continuous
find_all_onsets_and_peaks_on_fluorescence_signal(...)	Get all potential onsets and peaks from a fluorescence signal
check_one_dir_by_id_exists(identifiers, results_path)	Check if for each identifier in idenfifiers, a dir with this name exists
norm01(data)	Normalize an array so that values are ranging between 0 and 1
get_tree_dict_as_a_list(tree_dict)	Consider a dict representing a tree (the tree leaves are string), it returns list of all paths from
create_one_npy_file_by_frame(movies_to_split, results_path)	Split a calcium imaging movie so that each frame will be saved as a npy file.
create_one_tiff_file_by_frame(movies_to_split, ...[, ...])	Split a calcium imaging movie so that each frame will be saved as a tiff image.
scale_polygon_to_source(polygon, minx, miny)	Take an instance of shapely.geometry.Polygon or shapely.geometry.LineString and scale it, so that each of
load_movie(file_name, with_normalization[, ...])	Load a movie in memory. So far only tiff format is valid
build_raster_dur_from_onsets_peaks(onsets, peaks)	Build a raster_dur, a 2d binary array indicating when the cell is active (rise time).
get_source_profile_param(cell, movie_dimensions, ...)	For given cell, get the binary mask representing this cell with the possibility to get the binary masks
welsh_powell(graph)	implementation of welsh_powell algorithm
horizontal_flip(movie)	movie is a 3D numpy array
vertical_flip(movie)	movie is a 3D numpy array
v_h_flip(movie)	movie is a 3D numpy array
rotate_movie(movie, angle)	movie is a 3D numpy array
shift_movie(movie, x_shift, y_shift)	movie is a 3D numpy array

`deepcinac.utils.utils.`**`smooth_convolve`**(*x*, *window_len=11*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

Source: https://scipy-cookbook.readthedocs.io/items/SignalSmooth.html

NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

`deepcinac.utils.utils.get_continous_time_periods(binary_array)`

take a binary array and return a list of tuples representing the first and last position(included) of continuous positive period This code was copied from another project or from a forum, but i've lost the reference. :param binary_array: :return:

`deepcinac.utils.utils.find_all_onsets_and_peaks_on_fluorescence_signal(smooth_trace, threshold_factor=0.5, identifier=None)`

Get all potential onsets and peaks from a fluorescence signal :param smooth_trace: fluorescence signal of cell, should be smooth :param threshold_factor: use to define a threshold over which to keep peaks. :param The threshold used is: :type The threshold used is: threshold_factor * std(smooth_trace) + min(smooth_trace :param identifier: :type identifier: str

Returns: a 1d array of integers (binary) representing the time when the cell is active

`deepcinac.utils.utils.check_one_dir_by_id_exists(identifiers, results_path, dir_in_id_name=False)`

Check if for each identifier in idenfifiers, a dir with this name exists in results_path. If they all exists, then True is return, otherwise False Useful to check if a CI_movie has been separated in multiple tiff files. However, doesn't check in all the files are present, we assume that they are :param identifiers: list of string :param results_path: a path where to check for existing dir :param dir_in_id_name: if True, means the dir name should be in the identifier, no need for the directory to be :param exactly named as identifier:

Returns: boolean

`deepcinac.utils.utils.norm01(data)`

Normalize an array so that values are ranging between 0 and 1 :param data: numpy array

Returns:

`deepcinac.utils.utils.get_tree_dict_as_a_list(tree_dict)`

Consider a dict representing a tree (the tree leaves are string), it returns list of all paths from root node to all leaves :param tree_dict:

Returns:

`deepcinac.utils.utils.create_one_npy_file_by_frame(movies_to_split, results_path, without_mean_std_files=False, verbose=0)`

Split a calcium imaging movie so that each frame will be saved as a npy file. If the directory for results already contains a directory of the name of the movie identifier, the data won't be erased. :param movies_to_split: a dictionary with as a key an identifier for the movie that will be used to name :param the directory in which the tiff will be put: :type the directory in which the tiff will be put: using lower case :param of the tiff or an numpy float: :param ndarray representing the calcium imaging data: :type ndarray representing the calcium imaging data: should n_frames * n_pixels_x * n_pixels_y :param results_path: String. The directory in which will be created the directories containing the tiff files. :param verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = times for main operation, 2 = various prints. :param without_mean_std_files: if True, mean and std are not recorded, useful if the movie is already normalized

Returns: None

`deepcinac.utils.utils.create_one_tiff_file_by_frame(movies_to_split, results_path, without_mean_std_files=False, verbose=0)`

Split a calcium imaging movie so that each frame will be saved as a tiff image. If the directory for results already contains a directory of the name of the movie identifier, the data won't be erased. :param movies_to_split: a dictionary with as a key an identifier for the movie that will be used to name :param the directory in which the tiff will be put: :type the directory in which the tiff will be put: using lower case :param of the tiff or an numpy float: :param ndarray representing the calcium imaging data: :type ndarray representing the calcium imaging data: should n_frames * n_pixels_x * n_pixels_y :param results_path: String. The directory in which will be created the directories containing the tiff files. :param verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent,

1 = times for main operation, 2 = various prints. :param without_mean_std_files: if True, mean and std are not recorded, useful if the movie is already normalized

Returns: None

`deepcinac.utils.utils.scale_polygon_to_source(polygon, minx, miny)`

Take an instance of shapely.geometry.Polygon or shapely.geometry.LineString and scale it, so that each of it coordinates are substracted by minx and miny on the x and y axis respectively

coordinates to match minx and miny :param polygon: Polygon instance from shapely package. Could also be a LineString :param minx: integer :param miny: integer

Returns: a new shapely.geometry.Polygon or shapely.geometry.LineString

`deepcinac.utils.utils.load_movie(file_name, with_normalization, both_instances=False, verbose=True)`

Load a movie in memory. So far only tiff format is valid :param file_name: str for a file_name, or array representing the movie :param with_normalization: if True, normalize the movie using z-score formula :param both_instances: if with_normalization is True and both_instances is True, then :param the function return both movie: :param the normal and the normalized one in that order:

Returns:

`deepcinac.utils.utils.build_raster_dur_from_onsets_peaks(onsets, peaks)`

Build a raster_dur, a 2d binary array indicating when the cell is active (rise time). n_cells * n_frames :param onsets: 2d binary array, n_cells * n_frames, 1 if onset at this frame :param peaks: 2d binary array, n_cells * n_frames

Returns:

`deepcinac.utils.utils.get_source_profile_param(cell, movie_dimensions, coord_obj, max_width, max_height, pixels_around=0, buffer=None, with_all_masks=False, get_only_polygon_contour=False)`

For given cell, get the binary mask representing this cell with the possibility to get the binary masks of the cells it intersects with.

Parameters

- **cell** –
- **movie_dimensions** – tuple of integers, width and height of the movie
- **coord_obj** – instance of
- **max_width** – Max width of the frame returned. Might cropped some overlaping cell if necessary
- **max_height** – Max height of the frame returned. Might cropped some overlaping cell if necessary
- **pixels_around** – how many pixels to add around the frame containing the cell and the overlapping one,
- **mask** (*doesn't change the*) –
- **buffer** – How much pixels to scale the cell contour in the mask. If buffer is 0 or None, then size of the cell
- **change.** (*won't*) –

- **with_all_masks** – Return a dict with all overlaps cells masks + the main cell mask. The key is an int.

  The mask consist on a binary array of with 0 for all pixels in the cell, 1 otherwise
  get_only_polygon_contour: the mask represents then only the pixels that makes the contour of the cells

Returns: A mask (numpy 2d binary array), with 0 for all pixels in the cell, 1 otherwise.
A tuple with four integers representing the corner coordinates (minx, maxx, miny, maxy)

deepcinac.utils.utils.**welsh_powell**(*graph*)

implementation of welsh_powell algorithm https://github.com/MUSoC/Visualization-of-popular-algorithms-in-Python/blob/master/Graph%20Coloring/graph_coloring.py :param graph: instance of networkx graph

Returns:

deepcinac.utils.utils.**horizontal_flip**(*movie*)

movie is a 3D numpy array :param movie: :return:

deepcinac.utils.utils.**vertical_flip**(*movie*)

movie is a 3D numpy array :param movie: :return:

deepcinac.utils.utils.**v_h_flip**(*movie*)

movie is a 3D numpy array :param movie: :return:

deepcinac.utils.utils.**rotate_movie**(*movie*, *angle*)

movie is a 3D numpy array :param movie: :return:

deepcinac.utils.utils.**shift_movie**(*movie*, *x_shift*, *y_shift*)

movie is a 3D numpy array :param movie: :param x_shift: :param y_shift: :return:

Submodules

deepcinac.__main__

Module Contents

deepcinac.__main__.**root**

deepcinac.__main__.**app**

deepcinac._version

Git implementation of _version.py.

Module Contents

Classes

VersioneerConfig	Container for Versioneer configuration parameters.

Functions

get_keywords()	Get the keywords needed to look up the version information.
get_config()	Create, populate and return the VersioneerConfig() object.
register_vcs_handler(vcs, method)	Create decorator to mark a method as the handler of a VCS.
run_command(commands, args[, cwd, verbose, ...])	Call the given command(s).
versions_from_parentdir(parentdir_prefix, root, verbose)	Try to determine the version from the parent directory name.
git_get_keywords(versionfile_abs)	Extract version information from the given file.
git_versions_from_keywords(keywords, tag_prefix, verbose)	Get version information from git keywords.
git_pieces_from_vcs(tag_prefix, root, verbose[, runner])	Get version from 'git describe' in the root of the source tree.
plus_or_dot(pieces)	Return a + if we don't already have one, else return a .
render_pep440(pieces)	Build up version string, with post-release "local version identifier".
render_pep440_branch(pieces)	TAG[[.dev0]+DISTANCE.gHEX[.dirty]] .
pep440_split_post(ver)	Split pep440 version string at the post-release segment.
render_pep440_pre(pieces)	TAG[.postN.devDISTANCE] -- No -dirty.
render_pep440_post(pieces)	TAG[.postDISTANCE[.dev0]+gHEX] .
render_pep440_post_branch(pieces)	TAG[.postDISTANCE[.dev0]+gHEX[.dirty]] .
render_pep440_old(pieces)	TAG[.postDISTANCE[.dev0]] .
render_git_describe(pieces)	TAG[-DISTANCE-gHEX][-dirty].
render_git_describe_long(pieces)	TAG-DISTANCE-gHEX[-dirty].
render(pieces, style)	Render the given version pieces into the requested style.
get_versions()	Get version information or return default if unable to do so.

Attributes

LONG_VERSION_PY
HANDLERS

deepcinac._version.**get_keywords**()

Get the keywords needed to look up the version information.

class deepcinac._version.**VersioneerConfig**

Container for Versioneer configuration parameters.

deepcinac._version.**get_config**()

Create, populate and return the VersioneerConfig() object.

exception deepcinac._version.**NotThisMethod**

Bases: Exception

Exception raised if a method is not valid for the current scenario.

deepcinac._version.**LONG_VERSION_PY: Dict[str, str]**

deepcinac._version.**HANDLERS: Dict[str, Dict[str, Callable]]**

deepcinac._version.**register_vcs_handler**(*vcs*, *method*)

Create decorator to mark a method as the handler of a VCS.

deepcinac._version.**run_command**(*commands*, *args*, *cwd=None*, *verbose=False*, *hide_stderr=False*, *env=None*)

Call the given command(s).

deepcinac._version.**versions_from_parentdir**(*parentdir_prefix*, *root*, *verbose*)

Try to determine the version from the parent directory name.

Source tarballs conventionally unpack into a directory that includes both the project name and a version string. We will also support searching up two directory levels for an appropriately named parent directory

deepcinac._version.**git_get_keywords**(*versionfile_abs*)

Extract version information from the given file.

deepcinac._version.**git_versions_from_keywords**(*keywords*, *tag_prefix*, *verbose*)

Get version information from git keywords.

deepcinac._version.**git_pieces_from_vcs**(*tag_prefix*, *root*, *verbose*, *runner=run_command*)

Get version from 'git describe' in the root of the source tree.

This only gets called if the git-archive 'subst' keywords were *not* expanded, and _version.py hasn't already been rewritten with a short version string, meaning we're inside a checked out source tree.

deepcinac._version.**plus_or_dot**(*pieces*)

Return a + if we don't already have one, else return a .

deepcinac._version.**render_pep440**(*pieces*)

Build up version string, with post-release "local version identifier".

Our goal: TAG[+DISTANCE.gHEX[.dirty]] . Note that if you get a tagged build and then dirty it, you'll get TAG+0.gHEX.dirty

Exceptions: 1: no tags. git_describe was just HEX. 0+untagged.DISTANCE.gHEX[.dirty]

deepcinac._version.**render_pep440_branch**(*pieces*)

TAG[[.dev0]+DISTANCE.gHEX[.dirty]] .

The ".dev0" means not master branch. Note that .dev0 sorts backwards (a feature branch will appear "older" than the master branch).

Exceptions: 1: no tags. 0[.dev0]+untagged.DISTANCE.gHEX[.dirty]

`deepcinac._version.pep440_split_post(ver)`

Split pep440 version string at the post-release segment.

Returns the release segments before the post-release and the post-release version number (or -1 if no post-release segment is present).

`deepcinac._version.render_pep440_pre(pieces)`

TAG[.postN.devDISTANCE] – No -dirty.

Exceptions: 1: no tags. 0.post0.devDISTANCE

`deepcinac._version.render_pep440_post(pieces)`

TAG[.postDISTANCE[.dev0]+gHEX] .

The ".dev0" means dirty. Note that .dev0 sorts backwards (a dirty tree will appear "older" than the corresponding clean one), but you shouldn't be releasing software with -dirty anyways.

Exceptions: 1: no tags. 0.postDISTANCE[.dev0]

`deepcinac._version.render_pep440_post_branch(pieces)`

TAG[.postDISTANCE[.dev0]+gHEX[.dirty]] .

The ".dev0" means not master branch.

Exceptions: 1: no tags. 0.postDISTANCE[.dev0]+gHEX[.dirty]

`deepcinac._version.render_pep440_old(pieces)`

TAG[.postDISTANCE[.dev0]] .

The ".dev0" means dirty.

Exceptions: 1: no tags. 0.postDISTANCE[.dev0]

`deepcinac._version.render_git_describe(pieces)`

TAG[-DISTANCE-gHEX][-dirty].

Like 'git describe –tags –dirty –always'.

Exceptions: 1: no tags. HEX[-dirty] (note: no 'g' prefix)

`deepcinac._version.render_git_describe_long(pieces)`

TAG-DISTANCE-gHEX[-dirty].

Like 'git describe –tags –dirty –always -long'. The distance/hash is unconditional.

Exceptions: 1: no tags. HEX[-dirty] (note: no 'g' prefix)

`deepcinac._version.render(pieces, style)`

Render the given version pieces into the requested style.

`deepcinac._version.get_versions()`

Get version information or return default if unable to do so.

deepcinac.cinac_benchmarks

Module Contents

Classes

CinacBenchmarks	Used to plot benchmarks.
SessionForBenchmark	From a directory content including a yaml config file, will read ground truth + infered data

Functions

build_spike_nums_dur(spike_nums, peak_nums[, traces, ...])	Build a "raster dur", meaning a 2d binary array (n_cells * n_frames) representing when a cell is active
get_raster_dur_from_traces(traces[, ...])	Allow to build a raster dur based on all putative transient from the fluorescence traces.
compute_stats_over_gt(raster_gt, raster_to_evaluate, ...)	Compute the stats based on raster dur
load_data_from_np_or_mat_file(file_name[, attr_name])	Load data from a numpy or matlab file (.npz, .npz or .mat)
bin_raster(spike_nums)	Take a binary 2d array (n_cell*n_frames) and return a binned version,
get_raster_dur_spikes_and_traces(spike_nums, traces)	Take a binary raster (n_cells*n_frames) representing spikes or onsets of transient,
do_traces_smoothing(traces)	
benchmark_neuronal_activity_inferences(inferences_dir, ...)	Evaluate the performances of different inferences methods. Every directory in the main
extract_age(label)	

deepcinac.cinac_benchmarks.**build_spike_nums_dur**(*spike_nums*, *peak_nums*, *traces=None*, *fluorescence_threshold=None*)

Build a "raster dur", meaning a 2d binary array (n_cells * n_frames) representing when a cell is active based on the onset and peak on transients. :param spike_nums: (2d array binary n_cells * n_frames), represent the onsets of cell activations (transients) :param peak_nums: (2d array binary n_cells * n_frames), represent the peaks of cell activations (transients) :param traces: (2d array float n_cells * n_frames) fluorescence traces of the cells :param fluorescence_threshold: if a peak amplitude is under the threshold, we don't consider i

Returns:

deepcinac.cinac_benchmarks.**get_raster_dur_from_traces**(*traces*, *fluorescence_threshold=None*)

Allow to build a raster dur based on all putative transient from the fluorescence traces. :param traces:(2d array float, n_cells *n_frames) fluorescence traces :param fluorescence_threshold: None or otherwise 1xlen(traces) array with for each cell the threshold under which we should not take into account a peak and the transient associated. The value is without normalization. :return:

deepcinac.cinac_benchmarks.**compute_stats_over_gt**(*raster_gt*, *raster_to_evaluate*, *traces*, *raster_predictions=None*, *fluorescence_threshold=None*)

Compute the stats based on raster dur :param raster_gt: 1d binary array (n_frames or 2d binary array (n_cells, n_frames), 1 when the cell is active :param raster_to_evaluate: same sahpe as raster_t, binary, 1 when a cell is active :param traces: (1d or 2d array, same shape as raster_gt), should be smoothed, allows to detect :raster_predictions: same shape as raster_gt, but float array, probability that the cell is active at each frame. Can be None, if ground_truth is not based putative transients fluorescence_threshold: if not None, float value representing a low threshold for

for transients used for benchmarks (possible transients), transients below the threshold are not considered

Returns

two dicts: first one with stats on frames, the other one with stats on transients

Frames dict has the following keys (as String): TP: True Positive FP: False Positive FN: False Negative TN: True Negative sensitivity or TPR: True Positive Rate or Recall specificity or TNR: True Negative Rate or Selectivity FPR: False Positive Rate or Fall-out FNR: False Negative Rate or Miss Rate ACC: accuracy Prevalence: sum positive conditions / total population (for frames only) PPV: Positive Predictive Value or Precision FDR: False Discovery Rate FOR: False Omission Rate NPV: Negative Predictive Value LR+: Positive Likelihood ratio LR-: Negative likelihood ratio

transients dict has just the following keys: TP FN sensitivity or TPR: True Positive Rate or Recall FNR: False Negative Rate or Miss Rate

class deepcinac.cinac_benchmarks.**CinacBenchmarks**(*results_path*, *colors_boxplots=None*, *verbose=0*)

Used to plot benchmarks. To do so 4 steps: Create an instance of CinacBenchmarks Add data using add_inference_to_benchmark() Add color for session if you want to using color_by_session() then call one the method to plot stats

color_by_session(*session_id*, *color*)

Attribute a cell to a color, allowing for example to give to cell of the same :param session_id: (str) :param color:

Returns:

add_ground_truth(*session_id*, *ground_truth*, *smooth_traces*)

Parameters

- **session_id** –
- **ground_truth** –
- **smooth_traces** –

Returns:

add_inference_to_benchmark(*session_id*, *inference_to_benchmark_id*, *raster_to_evaluate*, *cells_to_benchmark*, *raster_predictions=None*)

Parameters

- **session_id** –
- **inference_to_benchmark_id** –
- **raster_to_evaluate** –
- **raster_predictions** –
- **cells_to_benchmark** –

Returns:

evaluate_metrics()

Should be called after all inference to benchmark have been added and before generating the plots Returns:

_evaluate_metrics_on_a_cell(*session_id*, *inference_to_benchmark_id*, *cell*)

Parameters

- **session_id** – session from which the activity is recorded
- **inference_to_benchmark_id** – (str) id of the inference to benchmark
- **cell** – int

Returns: a dict with same keys as inferences_to_benchmarks and values…

plot_boxplot_predictions_stat_by_metrics(*description*, *time_str=None*, *colorfull=True*, *white_background=True*, *for_frames=False*, *save_formats='pdf'*, *dpi=500*)

PLot the boxplot regarding the predictions done by cinac classifiers. :param description: (str) will be added to file_name :param time_str: (str) timestamps for file_name, optional :param save_formats: str or list of str (.pdf, .png etc…) :param dpi:

Returns:

plot_boxplots_full_stat(*description*, *time_str=None*, *for_frames=True*, *with_cells=False*, *color_cell_as_boxplot=False*, *box_plots_labels=None*, *colorfull=True*, *white_background=True*, *stats_to_show=('sensitivity', 'specificity', 'PPV', 'NPV')*, *title_correspondance=None*, *alpha_scatter=1.0*, *with_cell_number=True*, *put_metric_as_y_axis_label=False*, *using_patch_for_legend=False*, *with_legend=False*, *save_formats='pdf'*, *dpi=500*)

Parameters

- **description** –
- **time_str** –
- **for_frames** –

;param box_plots_labels: if not NOne, list of str, allowing to choose the order of the boxplots. The labels should be existing one, otherwise it will crash :param with_cells: if True, display a scatter for each cell :param save_formats: :param title_correspondance: dict or None, keys are the stats to show, and the value is another string that will be displayed in the title :return:

plot_boxplots_for_transients_stat(*description*, *colorfull=True*, *time_str=None*, *save_formats='pdf'*)

stats_on_performance(*stats_to_evaluate*, *data_labels=None*, *title_correspondance=None*, *for_frames=False*)

Compare with a wilcoxon test the performance over all pair of labels (methods) :param stats_to_evaluate: list of str :param title_correspondance: dict

Returns:

plot_boxplots_f1_score(*description*, *colorfull=False*, *time_str=None*, *for_frames=True*, *with_cells=False*, *color_cell_as_boxplot=False*, *box_plots_labels=None*, *white_background=False*, *with_legend=False*, *using_patch_for_legend=True*, *alpha_scatter=1*, *with_cell_number=True*, *put_metric_as_y_axis_label=False*, *save_formats='pdf'*, *dpi=500*)

Parameters

- **description** –
- **time_str** –
- **(bool)** (*colorfull*) – if True, the boxplots are filled with color, one different for each

(based on brewer colors) :param white_background: (bool) if True white background, else black background :param for_frames: :param with_cells: if True, display a scatter for each cell :param save_formats: :return:

plot_boxplots_proportion_frames_in_transients(*description*, *time_str=None*, *colorfull=True*, *white_background=False*, *only_this_key=None*, *with_scatter=True*, *with_cell_text=True*, *alpha_scatter=0.6*, *using_patch_for_legend=False*, *with_legend=False*, *put_metric_as_y_axis_label=True*, *save_formats='pdf'*, *dpi=500*)

Parameters

- **description** –
- **time_str** –

param only_this_key: (str): key of the label, if not None, then only the boxplot corresponding will be displayed :param save_formats: :return:

deepcinac.cinac_benchmarks.**load_data_from_np_or_mat_file**(*file_name*, *attr_name=None*)

Load data from a numpy or matlab file (.npz, .npz or .mat) :param file_name: :param data_descr: string used to display error message if the file is not in the good format :param attr_name:

Returns:

deepcinac.cinac_benchmarks.**bin_raster**(*spike_nums*)

Take a binary 2d array (n_cell*n_frames) and return a binned version, with output shape 2d array (n_cell*n_frames//2) :param spike_nums:

Returns:

deepcinac.cinac_benchmarks.**get_raster_dur_spikes_and_traces**(*spike_nums*, *traces*)

Take a binary raster (n_cells*n_frames) representing spikes or onsets of transient, and based on the smoothed traces, detect putative transients and extend the spikes/onsets so that the duration of the rise time of the transient in which they are fully 'active' :param spike_nums: :param traces:

Returns:

class deepcinac.cinac_benchmarks.**SessionForBenchmark**(*dir_to_explore*, *inferences_to_benchmark*, *predictions_keywords_dict*, *default_predictions_threshold*)

From a directory content including a yaml config file, will read ground truth + infered data

get_inference_ids()

extract_data_from_cinac_file(*cinac_file*, *cells_to_add*, *for_ground_truth*, *label=None*)

From a cinac_file, :param session_id: str :param cinac_file: :param cells_to_add: :param for_ground_truth: (bool) to set ground truth or add some inference :param label: (str), necessary if for_ground_truth is False, allow to identify the inference

Returns:

add_it_to_cinac_benchmark(*cinac_benchmarks*)

deepcinac.cinac_benchmarks.**do_traces_smoothing**(*traces*)

deepcinac.cinac_benchmarks.**benchmark_neuronal_activity_inferences**(*inferences_dir*, *results_path*, *colorfull_boxplots=True*, *white_background=False*, *color_cell_as_boxplot=False*, *with_legend=False*, *put_metric_as_y_axis_label=False*, *using_patch_for_legend=True*, *alpha_scatter=0.6*, *plot_proportion_frames_in_transients=False*, *with_cells=True*, *with_cell_number=True*, *predictions_stat_by_metrics=False*, *save_formats=['png', 'eps']*)

Evaluate the performances of different inferences methods. Every directory in the main directory is considered a session with a ground truth and inferences to benchmark :param inferences_dir: directory containing other directories, one by session to benchmark :param results_path: :param colorfull_boxplots: if True boxplots are filled with color :param white_background: if True background is white, else it's black :param with_cells: if True display scatter to represent metrics for a cell :param with_cell_number: Displau the cell number in the scatter if with_cells is True :param with_legend: if True, add legens to the F1 score plot

Returns:

deepcinac.cinac_benchmarks.**extract_age**(*label*)

deepcinac.cinac_model

File that contains methods use to fit the data to the model and train it.

Module Contents

Classes

Swish	
CinacModel	Model used to train a classifier.

Functions

attention_3d_block(inputs, time_steps[, ...])	from: https://github.com/philipperemy/keras-attention-mechanism
sensitivity(y_true, y_pred)	
specificity(y_true, y_pred)	
precision(y_true, y_pred)	
swish(x)	Implementing a the swish activation function.

Attributes

TF_VERSION
config

deepcinac.cinac_model.**TF_VERSION**

deepcinac.cinac_model.**config**

deepcinac.cinac_model.**attention_3d_block**(*inputs*, *time_steps*, *use_single_attention_vector=False*)

from: https://github.com/philipperemy/keras-attention-mechanism :param inputs: :param use_single_attention_vector: if True, the attention vector is shared across the input_dimensions where the attention is applied. :return:

deepcinac.cinac_model.**sensitivity**(*y_true*, *y_pred*)

deepcinac.cinac_model.**specificity**(*y_true*, *y_pred*)

deepcinac.cinac_model.**precision**(*y_true*, *y_pred*)

class deepcinac.cinac_model.**Swish**(*activation*, ***kwargs*)

Bases: `tensorflow.keras.layers.Activation`

deepcinac.cinac_model.**swish**(*x*)

Implementing a the swish activation function. From: https://www.kaggle.com/shahariar/keras-swish-activation-acc-0-996-top-7 Paper describing swish: https://arxiv.org/abs/1710.05941

Parameters

x –

Returns

class deepcinac.cinac_model.**CinacModel**(***kwargs*)

Model used to train a classifier. First add data using add_input_data() method Second prepare_model() And at last fit()

Parameters

- ****kwargs** –

- **results_path** (-) – (str, mandatory), path where to save the results (such as classifier weigths)
- **n_gpus** (-) – (int, default is 1) Maximum number of processes to spin up when using process-based threading
- **workers** (-) – (int, default is 10), number of workers used to run the classifier
- **using_multi_class** (-) – (int, default is 1) number of classes used to classify the activity. So far only 1 or 3 are valid
- **cell** (*options. 3 means we distinguish active*) –
- **activity** (*from overlap*) –
- **gives** (*from neuropil and other. 1 class*) –
- **type** (*better results so far. Don't use it for cell*) –
- **yaml** (*the number of classes will be set throught the*) –
- **file.** (*configuration*) –
- **n_epochs** (-) – (int, default is 30), number of epochs for training the classifier
- **batch_size** (-) – (int, default is 8) size of the batch used to train the classifier
- **window_len** (-) – (int, default 100) number of frames of the segments given to the classifier
- **max_width** (-) – (int, default 25) number of pixels for the width of the frame surrounding the cell. Should be
- **classifier.** (*loss function used to train the*) –
- **max_height** (-) – (int, default 25) number of pixels for the height of the frame surrounding the cell. Should be
- **classifier.** –
- **overlap_value** (-) – (float, default 0.9), overlap between 2 segments using sliding window of size window_len
- **frames.** (*0.9 means contiguous segments will share 90% of their*) –
- **max_n_transformations** (-) – (int, default is 6) max number of geometric transformations to apply to each
- **segment.** –
- **pixels_around** (-) – (int, default is 0), number of pixels to add around the mask of the cell (for activity
- **classifier)** –
- **with_augmentation_for_training_data** (-) – (bool, default is True): is True, then geometric transformations are
- **dataset** (*precise the proportion in the final training*) –
- **buffer** (-) – (int, default is 1): indicated of how many pixels to inflate the mask of the cell.
- **split_values** (-) – (tuple of 3 floats, default is (0.8, 0.2, 0)), tuple of 3 floats, the sum should be equal
- **training** (*to 1. Give the distribution of the dataset into*) –
- **test** (*validation and*) –

- **loss_fct** (-) – (str, default is 'binary_crossentropy' is using_multi_class is 1, 'categorical_crossentropy'
- **else)** –
- **classifier.** –
- **with_learning_rate_reduction** (-) –
- **parameters** (*according to the following*) –
- **learning_rate_reduction_patience** (-) – (int, default is 2) number of epochs before reducing the learning
- **improved** (*rate if the validation accuracy has not*) –
- **True** (*with_learning_rate_reduction needs to be*) –
- **learning_rate_start** (-) – (float, default is 0.001) default learning rate to start with
- **with_early_stopping** (-) – (bool, True) if True, then early stopping is activated, if the classifier doesn't
- **early_stop_patience** (-) –
- **early_stop_patience** – (int, default if 15): number of epochs before the training stops if no progress is
- **made** (*based on validation accuracy values*) –
- **model_descr** (-) –
- **model** (*file used to save the*) –
- **with_shuffling** (-) –
- **dataset** –
- **seed_value** (-) –
- **dataset.** (*means the shuffle will always be the same for a given*) –
- **main_ratio_balance** (-) – (tuple of 3 floats, default is (0.6, 0.2, 0.2)), sequence of 3 float, the sum should
- **classifier** (*be equal to 1. Used for activity*) –
- **dataset** –
- **transient** (*between sequence according to real*) –
- **"neuropil"** (*fake transient and*) –
- **crop_non_crop_ratio_balance** (-) – (tuple of 2 floats, default is (-1, -1)), use for stratification in activity
- **classifier** –
- **-1** (*if values are*) –
- **segment** (*we don't use it. Otherwise allows to balance segments between*) –
- **cropped.** (*with transient cropped versus not*) –
- **non_crop_ratio_balance** (-) – (tuple of 2 floats, default is (-1, -1)), use for stratification in activity
- **classifier** –

- **-1** –
- **segment** –
- **transient.** (*with one transient vers more than one*) –
- **with_model_check_point** (-) – (bool, default is True) allows to save weights of the classifier at each epoch
- **verbose** (-) – (int, default is 2), 0 no verbose, 1 main outputs printed, 2 all outputs printed
- **dropout_value** (-) – (float, default 0.5), dropout between CNN layers
- **dropout_value_rnn** (-) – (float, default 0.5), dropout between RNN layers
- **dropout_at_the_end** (-) – (float, default 0), dropout value at the end of the model
- **with_batch_normalization** (-) – (bool, default False), if True use batch normalization
- **optimizer_choice** (-) – (str, default is "RMSprop"), optimizer to use, choices "SGD", "RMSprop","Adam"
- **activation_fct** (-) – (str, default is "swish") , activation function to use, you can choose any activation
- **TensorFlow** (*function available in*) –
- **available.** (*swish is also*) –
- **without_bidirectional** (-) – (bool, default is False), if True, LSTM is not bidirectionnal
- **conv_filters** (-) – (tuple of 4 int, default is (64, 64, 128, 128)), the dimensionality of the output space
- **integers** (*(i.e. the number of filters in the convolution) sequence of 4*) –

:param : :param representing the filters of the 4 convolution layers: :param - lstm_layers_size: :type - lstm_layers_size: sequence of int, default is (128, 256) :param - bin_lstm_size: :type - bin_lstm_size: int, default is 256 :param - use_bin_at_al_version: :type - use_bin_at_al_version: bool, default is True :param - apply_attention: (bool, default is True), if True, attention mechanism is used :param - apply_attention_before_lstm: (bool, default is True), if True it means attention mechanism will be apply :param before LSTM: :param - use_single_attention_vector: :type - use_single_attention_vector: bool, default is False :param - cell_type_categories_yaml_file: (str, default is None) path and filename of the yaml file used to c :param configure the classifier for cell type: :type configure the classifier for cell type: types of cell, number of classes :param cell_type_categories_default.yaml will be used with pyramidal cells and interneuron and 2 classes.: :param - n_windows_len_to_keep_by_cell: (int, default 2), used only for cell type classifier, indicate how many :param segment of length window_len to keep for training. If too many segment are given then the classifier might: :param not be able to generalize well: :param - frames_to_avoid_for_cell_type: (sequence of int, default []), list of frame indices. If given, then :param segment than contains one of those indices won't be add to the training. Useful for example if movies are: :param concatenated.:

load_cell_type_categories_from_yaml_file(*yaml_file*)

Load cell type names from a yaml file. If more than 2 type cells are given, then a multi-class classifier will be used. If 2 type cells are given, then either it could be multi-class or binary classifier, then this choice should be given in the parameters of CinacModel. If 2 cell-type are given, for binary classifier, it should be precised which cell type should be predicted if we get more than 0.5 probability. :param yaml_file:

Returns:

add_input_data_from_dir(*dir_name*, *verbose=0*, *display_cells_count=False*)

Add input data loading all .cinac file in dir_name If a (UTF-8 encoded) txt file is in the dir, it is parsed in order to give to each cinac_file an id the format is for each line: cinac_file_name: id :param dir_name:

str, path + directory from which to load .cinac files :param verbose: 0 no print, 1 informations are printed :param display_cells_count: if True, print the number of cells added in training and the number of sessions associated, :param common field of view are identified using the basename of the ci movie in the cinac file:

Returns:

add_input_data(*cinac_file_names*, *session_ids=None*, *verbose=0*, *cinac_files_only_for_training=None*, *cinac_files_to_keep_absolutely=None*, *cinac_files_to_exclude=None*, *display_cells_count=False*)

Add input data. :param cinac_file_name: str or list of str, represents the files .cinac :param session_ids: None if no session_id otherwise a tuple, list or just a string (or int) representing the :param id of each cinac_file: :type id of each cinac_file: one id can be common to several cinac_file :param verbose: 0 no print, 1 informations are printed :param cinac_files_to_exclude: list of String or None, if the cinac_file_name is in the list, then we don't add it :param display_cells_count: if True, print the number of cells added in training and the number of sessions associated, :param common field of view are identified using the basename of the ci movie in the cinac file:

Returns:

__build_model()

Returns:

_split_and_stratify_cell_type_mode_data(*verbose=0*)

Split (between validation and training) and stratify the data, should be used when self.cell_type_classifier_mode is True :param verbose:

Returns:

_split_and_stratify_data_for_window_len_h5(*verbose=0*)

Split (between validation and training) and stratify the data

Returns:

_split_and_stratify_data(*verbose=0*)

Split (between validation and training) and stratify the data

Returns:

prepare_model(*verbose=0*)

Will build the model that will be use to fit the data. Should be called only after the data has been set. Returns:

fit()

deepcinac.cinac_movie_patch

Module Contents

Classes

MoviePatchGenerator	Used to generate movie patches, that will be produce for training data during each mini-batch.
MoviePatchGeneratorForCellType	Used to generate movie patches, that will be produce for training data during each mini-batch.
MoviePatchGeneratorMaskedVersions	Will generate one input being the masked cell (the one we focus on), the second input
MoviePatchData	
DataGenerator	Based on an exemple found in https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

Attributes

TF_VERSION

deepcinac.cinac_movie_patch.**TF_VERSION**

class deepcinac.cinac_movie_patch.**MoviePatchGenerator**(*window_len*, *max_width*, *max_height*, *using_multi_class*, *cell_type_classifier_mode*)

Used to generate movie patches, that will be produce for training data during each mini-batch. This is an abstract classes that need to have heritage. The function generate_movies_from_metadata will be used to produced those movie patches, the number vary depending on the class instantiated

get_nb_inputs()

generate_movies_from_metadata(*movie_data_list*, *memory_dict*, *with_labels=True*)

class deepcinac.cinac_movie_patch.**MoviePatchGeneratorForCellType**(*window_len*, *max_width*, *max_height*, *pixels_around*, *buffer*, *using_multi_class*, *cell_type_classifier_mode*, *with_all_pixels=False*)

Bases: *MoviePatchGenerator*

Used to generate movie patches, that will be produce for training data during each mini-batch. This is an abstract classes that need to have heritage. The function generate_movies_from_metadata will be used to produced those movie patches, the number vary depending on the class instantiated

generate_movies_from_metadata(*movie_data_list*, *memory_dict=None*, *with_labels=True*)

Parameters

- **movie_data_list** – list of MoviePatchData instances
- **memory_dict** –
- **with_labels** –

Returns:

__str__()

Return str(self).

class deepcinac.cinac_movie_patch.**MoviePatchGeneratorMaskedVersions**(*window_len*, *max_width*, *max_height*, *pixels_around*, *buffer*, *with_neuropil_mask*, *using_multi_class*, *cell_type_classifier_mode*)

Bases: *MoviePatchGenerator*

Will generate one input being the masked cell (the one we focus on), the second input would be the whole patch without neuorpil and the main cell, the last inpu if with_neuropil_mask is True would be just the neuropil without the pixels in the cells

generate_movies_from_metadata(*movie_data_list*, *memory_dict=None*, *with_labels=True*)

Parameters

- **movie_data_list** – list of MoviePatchData instances
- **memory_dict** –
- **with_labels** –

Returns:

__str__()

Return str(self).

class deepcinac.cinac_movie_patch.**MoviePatchData**(*cinac_recording*, *cell*, *index_movie*, *max_n_transformations*, *encoded_frames*, *decoding_frame_dict*, *window_len*, *cell_type_classifier_mode=False*, *session_id=None*, *with_info=False*, *to_keep_absolutely=False*, *ground_truth=None*)

n_available_augmentation_fct

Keys so far for self.movie_info (with value type) -> comments:

n_transient (int) transients_lengths (list of int) transients_amplitudes (list of float) n_cropped_transient (int) -> max value should be 2 cropped_transients_lengths (list of int) n_fake_transient (int) n_cropped_fake_transient (int) > max value should be 2 fake_transients_lengths (list of int) fake_transients_amplitudes (list of float)

get_labels(*using_multi_class*)

Return the labels for this data, could be if the cell is active for any given frame or the cell type depending on the classifier mode :param using_multi_class:

Returns:

__eq__(*other*)

Return self==value.

copy()

add_n_augmentation(*n_augmentation*)

pick_a_transformation_fct()

is_only_neuropil()

Returns
True if there is only neuropil (no transients), False otherwise

class deepcinac.cinac_movie_patch.**DataGenerator**(*data_list*, *movie_patch_generator*, *batch_size*, *window_len*, *with_augmentation*, *pixels_around*, *buffer*, *max_width*, *max_height*, *is_shuffle=True*)

Bases: tensorflow.keras.utils.Sequence

Based on an exemple found in https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly Feed to keras to generate data

prepare_augmentation()

__len__()

__getitem__(*index*)

on_epoch_end()

__data_generation(*data_list_tmp*)

deepcinac.cinac_predictor

Module Contents

Classes

CinacPredictor

Functions

fusion_cell_type_predictions_by_type(...[, ...])	Allows to associate a cell type to a prediction (a prediction being a 2d array of n_cells * n_classes)
fusion_cell_type_predictions(cell_type_pred_1, ...[, ...])	Take two cell type predictions (should be for the same movie, so the same number of cells in both),
select_activity_classifier_on_cell_type_output(...[, ...])	Allows to create a dictionary that will be passed to CinacPredictor.add_recording() as the model_files_dict
predict_cell_type_from_model(cinac_recording, cell, ...)	**param cinac_recording**
load_data_for_prediction(cinac_recording, cell, ...)	Create data that will be used to predict neural activity. The data will be representing by instances of
load_data_for_cell_type_prediction(cinac_recording, ...)	Create data that will be used to predict cell type. The data will be representing by instances of
predict_transient_from_model(cinac_recording, cell, ...)	**param cinac_recording**
activity_predictions_from_cinac_files(cinac_dir_name, ...)	Evaluate activity prediction on cinac file with ground truth.
evaluate_cell_type_predictions(cinac_dir_name, ...[, ...])	Evaluation the cell type prediction on cinac file with ground truth.

Attributes

TF_VERSION

deepcinac.cinac_predictor.**TF_VERSION**

deepcinac.cinac_predictor.**fusion_cell_type_predictions_by_type**(*cell_type_preds_dict*, *default_cell_type_pred*, *cell_type_to_skip_if_conflict=None*, *cell_type_config_file=None*, *filename_to_save=None*)

Allows to associate a cell type to a prediction (a prediction being a 2d array of n_cells * n_classes) :param cell_type_preds_dict: key: code of the cell type, value 2d array of n_cells * n_classes representing :param the predictions to associate to this cell type: :type the predictions to associate to this cell type: for cell predicted as so :param default_cell_type_pred: 2d array of n_cells * n_classes :param Same length as cell_type_preds_dict: :param cell_type_config_file: :param filename_to_save: :param cell_type_to_skip_if_conflict: int representing the cell type code, if given and there is a conflict between :param two classifier including this cell type: :param then the other one in conflict will be chosen. If a conflict with more: :param than 2 cell types: :param then the default one is chosen.:

Returns:

deepcinac.cinac_predictor.**fusion_cell_type_predictions**(*cell_type_pred_1*, *cell_type_pred_2*, *with_new_category=None*, *cell_type_config_file=None*, *filename_to_save=None*, *cell_types_to_not_fusion_with_gt=None*)

Take two cell type predictions (should be for the same movie, so the same number of cells in both), and fusion it so that cells that have a cell type probabilities in cell_type_pred_2 (sum of probablities equals to 1) will be transfered to cell_type_pred_1, so that the cell type for those cells in cell_type_pred_1 might change. If a yaml cell_type_config_file is provided, then verbose mode will be on and the changes will be print. It is possible to give filename_to_save and the new prediction will be saved in npy format :param cell_type_pred_1: 2d array (n_cells, n_classes), for each cell give the probability for the cell to be one of :param the cell type (n_classes represent the number of cell type possible.: :param cell_type_pred_2: 2d array (n_cells, n_classes), for each cell give the probability for the cell to be one of :param the cell type (n_classes represent the number of cell type possible.: :param cell_type_config_file: (str) yaml cell type that associate a cell type name to a code (one of the column :param in cell_type_pred): :param filename_to_save: (str) should have npy extension or it will be added, should include the full path. :param cell_types_to_not_fusion_with_gt: list of int, representing the cell types that should be replaced by the :param GT. For example if a cell is classified as Noise in pred_1 but INs in pred_2: :param then it stayed noise.: :param In that case cell_types_to_not_fusion_with_gt=[2]: :param with_new_category: list of tuple of 2 ints, if not None, then change the category in cell_type_pred_2 that :param are given to cell_type_pred_1: :param by either swaping the 2 numbers: :param or creating a new one if non existent. The first: :param number category will then be set to zero and the 2nd one will take the probability of the first one:

Returns: a 2d array (n_cells, n_classes) representing the new cell type predictions will be returned

`deepcinac.cinac_predictor.`**`select_activity_classifier_on_cell_type_outputs`**(*cell_type_config_file*, *cell_type_predictions*, *cell_type_to_classifier*, *default_classifier*, *unknown_cell_type_action='ignore'*, *cell_type_threshold=0.5*, *verbose=1*)

Allows to create a dictionary that will be passed to CinacPredictor.add_recording() as the model_files_dict argument. Each cell will be given a specific activity classifier regarding its cell type. :param cell_type_config_file: yaml file containing the description of the cell classifier, allows :param to read the cell type predictions: :param cell_type_predictions: np array of 2 dimensions, (n_cells, n_cell_type), value between 0 and 1 predicting :param if the cell is this cell type: :param cell_type_threshold: use only if binary classifier, if multi_class not used; the cell_type with the maximum :param prediction is picked.: :param cell_type_to_classifier: (dict), key is the name of a cell type (ex: 'interneuron') it should be in the :param yaml file: :type yaml file: model file, weights file, classifier id :param value is tuple of 3 strings: :type value is tuple of 3 strings: model file, weights file, classifier id :param specific activity classifier to a cell type: :param default_classifier: (tuple of 3 strings) (model file, weights file, classifier id) that attribute a default

activity classifier in a cell has no predominant cell type

Parameters

- **unknown_cell_type_action** – (str) Decide what to do when cell type is not in cell_type_to_classifier. Either
- **predicted** (*'ignore' meaning the cell with that cell type won't be*) –
- **classifier** (*"default" we use the default*) –
- **aborded** (*on it or "exception" then an exception would be raise and the excecution will be*) –
- **verbose** – (int) if > 0, then the number of cells by cell type will be printed

Returns: return a dict compatible with CinacPredictor.add_recording() model_files_dict argument. key is a tuple of 3 strings (model file, weights file, classifier id), and value is the cells whose activity should be classify using

this model.

class deepcinac.cinac_predictor.**CinacPredictor**(*verbose=0*)

add_recording(*cinac_recording*, *model_files_dict=None*, *removed_cells_mapping=None*)

Add a recording as a set of a calcium imaging movie, ROIs, and a list of cell to predict :param cinac_recording: an instance of CinacRecording :param model_files_dict:

dictionary. Key is a tuple of three string representing the file_name of the json file,
the filename of the weights file and network_identifier and an identifier for the model

and weights used (will be added to the name of files containing the predictions in addition of recording identifier.) the value of the dict being a list or array of integers that represents the indices of the cell to be predicted by the given model and set of weights. If None, means all cells activity will be predicted. Cells not including, will have their prediction set to 0 for all the frames

Parameters

- **removed_cells_mapping** – integers array of length the original numbers of cells
- **CinacRecording)** (*(such as defined in*) –
- **been** (*and as value either of positive int representing the new index of the cell or -1 if the cell has*) –
- **removed** –

Returns:

static get_new_cell_indices_if_cells_removed(*cell_indices_array*, *removed_cells_mapping*)

Take an array of int, and return another one with new index in case some cells would have been removed and some cells ROIs are not matching anymore :param cell_indices_array: np.array of integers, containing the cells indices to remap :param removed_cells_mapping: integers array of length the original numbers of cells :param (such as defined in CinacRecording): :param and as value either of positive int representing the new index of the cell or -1 if the cell has been: :param removed:

Returns: new_cell_indices_array an np.array that contains integers representing the new indices of cells that have not been removed original_cell_indices_mapping, np.array, for each new cell index, contains the corresponding original index

predict(*results_path*, *overlap_value=0.5*, *output_file_formats='npy'*, *cell_type_classifier_mode=False*, *n_segments_to_use_for_prediction=2*, *cell_type_pred_fct=np.mean*, *create_dir_for_results=True*, *cell_buffer=None*, *all_pixels=True*, *time_verbose=True*, ***kwargs*)

Will predict the neural activity state of the cell for each frame of the calcium imaging movies. Recordings have to be added previously though the add_recording method. :param results_path: if None, predictions are not saved in a file :param overlap_value: :param cell_type_classifier_mode: means we want to classify each cell to know its type. So far return a prediction :param value between 0 and 1: :type value between 0 and 1: should be interneuron then :param 1 meaning 100% a pyramidal cell: :type 1 meaning 100% a pyramidal cell: should be interneuron then :param 0 100% sure not: :type 0 100% sure not: should be interneuron then :param output_file_formats: a string or list of string, representing the format in which saving the 2d array :param representing the prediction for each movie. The choices are: "mat" or "npy" or "npz". npy and npz will both :param comes to a npz format. Two keys will be in the .mat or .npz: :param one is "predictions" containing a 1d or 2 array: :param with predictions inside (1d array for cell activity predictions: :param 2d array for cell type activity predicctions: :param (n_cells: :param n_cells_type): :param and the second key is "cells" which is a 1d array containing the indices of: :param the cells that has been predicted: :type the cells that has been predicted: other cells will have a 0 value prediction in the predictions array :param n_segments_to_use_for_prediction: used when cell_type_classifier_mode is at True. Indicate how many segment :param of window_len should be used to predict the cell_type. For

example if the value is 2 and window_len is 200: :param : :param then we take the mean of the prediction over 2 segments of 200 frames. Those segments are selected based: :param on their activity: :type on their activity: the more a cell is active (amplitudes) the best chance the segment is to be selected :param cell_type_pred_fct: fct to use to average the predictons if it is done in more than one segment :param create_dir_for_results: if True, create a dir with timestamps to store the results. The directory is created :param in results_path. Else predictions are stored directly in results_path: :param cell_buffer: None or int, distance in order to transform the cell contour giving an approximate :param representation of all points within a given distance of the this geometric object: :param all_pixels: bool, use for cell type prediction, add input with all pixels :param time_verbose: if True, print the time to predict each cell :param **kwargs:

Returns: a dict with key being a tuple with 2 string (the id of the cinac recording and the id of the classifier used) and value:

numpy array of n_cells containing the predictions (1d for cell activity, and 2d for cell type (n_cells, n_cell_types),.

`deepcinac.cinac_predictor.`**`predict_cell_type_from_model`**(*cinac_recording*, *cell*, *model*, *n_frames*, *n_segments_to_use_for_prediction=2*, *overlap_value=0*, *pixels_around=0*, *all_pixels=False*, *use_data_augmentation=False*, *buffer=None*, *verbose=0*)

Parameters

- **cinac_recording** –
- **cell** –
- **model** –
- **n_frames** –
- **n_segments_to_use_for_prediction** –
- **overlap_value** –
- **pixels_around** –
- **all_pixels** –
- **use_data_augmentation** –
- **buffer** –
- **verbose** –

Returns:

`deepcinac.cinac_predictor.`**`load_data_for_prediction`**(*cinac_recording*, *cell*, *sliding_window_len*, *overlap_value*, *augmentation_functions*, *n_frames*)

Create data that will be used to predict neural activity. The data will be representing by instances of MoviePatchData that will contains information concerning this movie segment for a given cell to give to the neuronal network. :param cinac_recording: instance of CinacRecording, contains the movie frames and the ROIs :param cell: integer, the cell index :param sliding_window_len: integer, length in frames of the window used by the neuronal network. Predictions will be :param made for each frame of this segment: :param overlap_value: float value between 0 and 1, representing by how much 2 movie segments will overlap. :param 0.5 is equivalent to a 50% overlap. It allows the network to avoid edge effect in order to get a full temporal: :param vision of all transient. A good default value is 0.5: :param augmentation_functions: list of function that takes an image (np array) as input and return a copy of the image :param transformed.: :param n_frames: number of frames in the movie

Returns: a list of MoviePatchData instance and an integer representing the index of the first frame of the patch

deepcinac.cinac_predictor.**load_data_for_cell_type_prediction**(*cinac_recording*, *cell*, *sliding_window_len*, *overlap_value*, *augmentation_functions*, *n_frames*, *n_windows_len_to_keep_by_cell=2*)

Create data that will be used to predict cell type. The data will be representing by instances of MoviePatchData that will contains information concerning this movie segment for a given cell to give to the neuronal network. :param cinac_recording: instance of CinacRecording, contains the movie frames and the ROIs :param cell: integer, the cell index :param sliding_window_len: integer, length in frames of the window used by the neuronal network. Predictions will be :param made for each frame of this segment: :param overlap_value: float value between 0 and 1, representing by how much 2 movie segments will overlap. :param 0.5 is equivalent to a 50% overlap. It allows the network to avoid edge effect in order to get a full temporal: :param vision of all transient. A good default value is 0.5: :param augmentation_functions: list of function that takes an image (np array) as input and return a copy of the image :param transformed.: :param n_frames: number of frames in the movie :param n_windows_len_to_keep_by_cell: how many segment of window_len frames should be used to predict cell_type, :param so the length of the returned list will be the same as n_windows_len_to_keep_by_cell:

Returns: a list of MoviePatchData instance and an integer representing the index of the first frame of the patch

deepcinac.cinac_predictor.**predict_transient_from_model**(*cinac_recording*, *cell*, *model*, *n_frames*, *overlap_value=0.8*, *pixels_around=0*, *use_data_augmentation=False*, *buffer=None*, *time_verbose=True*)

Parameters

- **cinac_recording** –
- **cell** –
- **model** –
- **n_frames** –
- **overlap_value** –
- **pixels_around** –
- **use_data_augmentation** –
- **buffer** –
- **time_verbose** – if True, print the time to predict the cell

Returns:

deepcinac.cinac_predictor.**activity_predictions_from_cinac_files**(*cinac_dir_name*, *results_path*, *json_file_name*, *weights_file_name*, *classifier_id=None*, *output_file_formats='npy'*)

Evaluate activity prediction on cinac file with ground truth. Load all cinac files from a folder and compare the predictions from the classifier to the activity in the cinac file and then display the metrics and can save the plot of the distribution of the probability by cell type. :param cinac_dir_name: Directory in which to find the .cinac file to use :param results_path: Directory in which to save the plots, can be None if save_activity_distribution is False :param json_file_name: .json file containing the model to use to classify the cell types :param weights_file_name: .h5 file containing the weights of the network to classify the cell types :param save_activity_distribution: (bool) if True, a figure representing the distribution of the predictions score :param for activity will be save in results_path:

Returns:

`deepcinac.cinac_predictor.evaluate_cell_type_predictions(cinac_dir_name, cell_type_yaml_file, results_path, json_file_name, weights_file_name, save_cell_type_distribution, all_pixels=False)`

Evaluation the cell type prediction on cinac file with ground truth. Load all cinac files from a folder and compare the predictions from the classifier to the cell type in the cinac file and then display the metrics and can save the plot of the distribution of the probability by cell type. :param cinac_dir_name: Directory in which to find the .cinac file to use :param cell_type_yaml_file: ymal file containing the configuration to use by the classifier :param (cell type: :param number of classes): :param results_path: Directory in which to save the plots, can be None if save_cell_type_distribution is False :param json_file_name: .json file containing the model to use to classify the cell types :param weights_file_name: .h5 file containing the weights of the network to classify the cell types :param save_cell_type_distribution: (bool) if True, a figure representing the distribution of the predictions score :param of each cell type will be save in results_path: :param all_pixels: if True, add an input with all pixels in the frame

Returns:

deepcinac.cinac_simulated_movie_generator

Module Contents

Classes

CellPiece	
MovieConstructor	Used to construct the movie, will build the frames, deal with overlapping and pixels intensity
SimulatedMovieGenerator	Class that will handle the generation of simulated calcium imaging movie

Functions

produce_cell_coord_from_cnn_validated_cells(param)	
shift_cell_coord_to_centroid(centroid, cell_coord[, ...])	
change_polygon_centroid(new_centroid, poly_cell)	
make_video(images[, outvid, fps, size, is_color, format])	Create a video from a list of images.
fig2data(fig)	http://www.icare.univ-lille1.fr/tutorials/convert_a_matplotlib_figure
fig2img(fig)	@brief Convert a Matplotlib figure to a PIL Image in RGBA format and return it
normalize_array_0_255(img_array)	
noisy(noise_typ, image)	**param image** Input image data. Will be converted to float.
get_mask(dimensions, poly_gon)	
get_weighted_activity_mask_for_a_cell(mask, soma_mask, ...)	
construct_movie_images(coord_obj, traces, dimensions, ...)	
build_somas(coord_obj, dimensions)	
give_values_on_linear_line_between_2_points(x_coords, ...)	
exponential_decay_formula(t, a, k, c)	Exponential decay formula
finding_growth_rate(t, a, end_value)	Find the growth rate.
produce_vessels(vessels_imgs_dir[, path_results])	Produce vessels polygons based on images in vessels_imgs_dir
plot_all_cells_on_map(coord_obj, path_results[, ...])	Plot all cells contour on a map using welsh powell algorithm to color cell that intersect

deepcinac.cinac_simulated_movie_generator.**produce_cell_coord_from_cnn_validated_cells**(*param*)

deepcinac.cinac_simulated_movie_generator.**shift_cell_coord_to_centroid**(*centroid*, *cell_coord*, *from_matlab=False*)

deepcinac.cinac_simulated_movie_generator.**change_polygon_centroid**(*new_centroid*, *poly_cell*)

deepcinac.cinac_simulated_movie_generator.**make_video**(*images*, *outvid=None*, *fps=5*, *size=None*, *is_color=True*, *format='XVID'*)

Create a video from a list of images.

@param outvid output video file_name @param images list of images to use in the video @param fps frame per second @param size size of each frame @param is_color color @param format see http://www.fourcc.org/codecs.php @return see http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

The function relies on http://opencv-python-tutroals.readthedocs.org/en/latest/. By default, the video will have the size of the first image. It will resize every image to this size before adding them to the video.

`deepcinac.cinac_simulated_movie_generator.fig2data(fig)`

http://www.icare.univ-lille1.fr/tutorials/convert_a_matplotlib_figure @brief Convert a Matplotlib figure to a 4D numpy array with RGBA channels and return it @param fig a matplotlib figure @return a numpy 3D array of RGBA values

`deepcinac.cinac_simulated_movie_generator.fig2img(fig)`

@brief Convert a Matplotlib figure to a PIL Image in RGBA format and return it @param fig a matplotlib figure @return a Python Imaging Library (PIL) image

`deepcinac.cinac_simulated_movie_generator.normalize_array_0_255(img_array)`

`deepcinac.cinac_simulated_movie_generator.noisy(noise_typ, image)`

Parameters

- **image** (*ndarray*) – Input image data. Will be converted to float.
- **mode** (*str*) –
- **strings** (*One of the following*) –
- **add** (*selecting the type of noise to*) –
- **noise.** (*'gauss' Gaussian-distributed additive*) –
- **data.** (*'poisson' Poisson-distributed noise generated from the*) –
- **1.** (*'s&p' Replaces random pixels with 0 or*) –
- **n*image** (*'speckle' Multiplicative noise using out = image +*) – n is uniform noise with specified mean & variance.
- **where** – n is uniform noise with specified mean & variance.

:param : :type : param img_array: :param : :type : return:

class `deepcinac.cinac_simulated_movie_generator.CellPiece(id, poly_gon, dimensions, activity_mask=None, frame_mode=False)`

fill_movie_images(*images*)

set_activity_mask_from_other(*other_activity_mask*)

set_activity_mask_from_two_other(*other_1*, *other_2*)

get_mask()

split(*other*)

update_activity_mask(*activity_mask=None*)

__eq__(*other*)

Return self==value.

__hash__()

Return hash(self).

`deepcinac.cinac_simulated_movie_generator.get_mask(dimensions, poly_gon)`

deepcinac.cinac_simulated_movie_generator.**get_weighted_activity_mask_for_a_cell**(*mask*, *soma_mask*, *n_pixels*, *n_pixels_soma*)

class deepcinac.cinac_simulated_movie_generator.**MovieConstructor**(*coord_obj*, *traces*, *dimensions*, *baseline*, *soma_geoms*, *vessels*)

Used to construct the movie, will build the frames, deal with overlapping and pixels intensity

get_frame(*frame*)

Parameters
frame –

Returns:

deepcinac.cinac_simulated_movie_generator.**construct_movie_images**(*coord_obj*, *traces*, *dimensions*, *baseline*, *soma_geoms*, *vessels*, *param*, *n_pixels_by_cell=None*)

deepcinac.cinac_simulated_movie_generator.**build_somas**(*coord_obj*, *dimensions*)

deepcinac.cinac_simulated_movie_generator.**give_values_on_linear_line_between_2_points**(*x_coords*, *y_coords*)

deepcinac.cinac_simulated_movie_generator.**exponential_decay_formula**(*t*, *a*, *k*, *c*)

Exponential decay formula Code probably copy from an online source. Sorry i don't have the reference :param t: time that has passed :param a: initial value (amount before measuring growth or decay) :param k: continuous growth rate (also called constant of proportionality) (k > 0, the amount is increasing (growing); k < 0, the amount is decreasing (decaying)) :param c: lowest value :return:

deepcinac.cinac_simulated_movie_generator.**finding_growth_rate**(*t*, *a*, *end_value*)

Find the growth rate. Code probably copy from an online source. Sorry i don't have the reference :param t: time :param a: :param end_value:

Returns:

deepcinac.cinac_simulated_movie_generator.**produce_vessels**(*vessels_imgs_dir*, *path_results=None*)

Produce vessels polygons based on images in vessels_imgs_dir :param vessels_imgs_dir: directory from which load the images containing the vessels :param path_results: If not None, the path where to save to vessels produced

Returns: coord_list a list of 2D array (n_coord x 2): each column represents the x and y coordinates of each point
of the polygon (vessel) dimensions_list a List of 1D array with 2 values, integer, representing the height and width of the movie

deepcinac.cinac_simulated_movie_generator.**plot_all_cells_on_map**(*coord_obj*, *path_results*, *save_formats='pdf'*)

Plot all cells contour on a map using welsh powell algorithm to color cell that intersect with a different color :param coord_obj: instance of CellsCoord object :param path_results: (string), directory where to save the figure :param save_formats: string or list of string, formats in which to save the figure

Returns:

class deepcinac.cinac_simulated_movie_generator.**SimulatedMovieGenerator**(*n_frames*, *path_results*, *dimensions=(120, 120)*, *with_mvt=False*, *n_cells_of_interest=16*, *n_overlap_by_cell_range=(1, 4)*, *non_overlap_by_cell_range=(2, 10)*, *range_n_transient_cells_of_interest=(2, 4)*, *range_n_transient_overlapping_cells=(8 16)*, *range_n_transient_other_cells=(2, 16)*, *range_duration_transient=(1, 8)*, *decay_factor=10*, *max_decay=12*, *time_str=None*, *use_only_valid_cells=True*)

Class that will handle the generation of simulated calcium imaging movie

generate_movie()

Generate the movie by buimding first the cells map, then generate ting the raster then the traces and then the movie. Before calling this method vessels must have been created or loaded if necessary and the cells model must be loaded as well. Returns:

save_cell_coords()

Save the cell coordinate so far in the same output as CaImAn Returns:

__generate_artificial_map()

Use the cells models to generate a map of the dimensions given when instanciating SimulatedMovieGenerator. It will create a "square" for each cell of interest that will be in the center of this square, this square will be filled with a certain number of cells overlapping the cell of interest and other cell that will not overlapping it but might overlap other cells. All the parameters can be change in the __init__ of the class Returns: None

save_raster_dur_for_gui()

Save the raster dur that was generated in the path_results defined in the constructor Returns: None

__build_raster_dur()

Build the raster that will be used to produce the simulated movie Returns: a 2d-array (int8) of n_cells x n_frames

build_traces(*n_pixels_by_cell*, *baseline*, *use_traces_for_amplitude=None*)

__produce_movie(*use_traces_for_amplitude=None*, *file_name=None*)

Generate the movie based on the cell contour map, the raster and the traces generated previously :param use_traces_for_amplitude: :param file_name:

Returns:

save_traces(*output_formats='npy'*)

Save traces :param output_formats: (string or list of string) npy or mat (if matlab, the traces variable name will be raw_traces

Returns:

produce_and_load_vessels(*vessels_imgs_dir*, *n_vessels_max=None*, *path_results=None*)

Parameters

- **path_results** – indicate the directory where to save the vessels produced. If None, the vessels won't be
- **saved.** –
- **vessels_imgs_dir** –

Returns:

load_vessels(*vessels_dir*, *n_vessels_max*)

Load vessels from data generated previously :param vessels_dir: Directory that contain the data file allowing to load vessels to the movie. :param n_vessels_max: max number of vessels that can be loaded. If None, no limit

Returns:

load_cell_coords(*data_file*, *from_matlab*)

Load cell coords based on model from the data_file path :param data_file: path and file name of the file containing model of cell contours. :param from_matlab: means the cells coordinates have been computed with matlab and starts at 1

Returns:

__add_vessels(*coord_list*, *dimensions_list*, *n_vessels_max=None*)

Add vessels in the movie based on the the coordinates and dimensions list given :param coord_list: List of 2D array (n_coord x 2): each column represents the x and y coordinates of each point :param of the polygon: :type of the polygon: vessel :param dimensions_list: List of 1D array with 2 values, integer, representing the height and width of the movie :param n_vessels_max: max number of vessels that can be loaded. If None, no limit

Returns:

deepcinac.cinac_stratification

Module Contents

Classes

MovieEvent	Class that represent an event in a movie, for exemple a transient, neuropil etc...
NeuropilEvent	Class that represent an event in a movie, for exemple a transient, neuropil etc...
RealTransientEvent	Class that represent an event in a movie, for exemple a transient, neuropil etc...
FakeTransientEvent	Class that represent an event in a movie, for exemple a transient, neuropil etc...
MovementEvent	Class that represent an event in a movie, for exemple a transient, neuropil etc...
StratificationCellTypeCamembert	
StratificationCamembert	
StratificationDataProcessor	
StratificationCellTypeDataProcessor	Class used to stratified data for cell type

Functions

neuronal_activity_encoding(raw_traces, smooth_traces, ...)	Give for each frame of the cell what kind of activity is going on (real transient, fake etc...)

deepcinac.cinac_stratification.**neuronal_activity_encoding**(*raw_traces*, *smooth_traces*, *raster_dur*, *identifier=None*)

Give for each frame of the cell what kind of activity is going on (real transient, fake etc…) :param raw_traces: 1d float representing the raw fluorescence signal (should be normalized using z-score) :param smooth_traces: 1d float representing the smoothed fluorescence signal (should be normalized using z-score) :param raster_dur: 1d int (or bool), representing the frames during which a given cell is active :param (corresponding to the traces). It represents the "ground truth" used to know real transients.:

Returns:

class deepcinac.cinac_stratification.**MovieEvent**

Class that represent an event in a movie, for exemple a transient, neuropil etc…

class deepcinac.cinac_stratification.**NeuropilEvent**(*frame_index*)

Bases: *MovieEvent*

Class that represent an event in a movie, for exemple a transient, neuropil etc…

class deepcinac.cinac_stratification.**RealTransientEvent**(*frames_period*, *amplitude*)

Bases: *MovieEvent*

Class that represent an event in a movie, for exemple a transient, neuropil etc…

class deepcinac.cinac_stratification.**FakeTransientEvent**(*frames_period*, *amplitude*)

Bases: *MovieEvent*

Class that represent an event in a movie, for exemple a transient, neuropil etc…

class deepcinac.cinac_stratification.**MovementEvent**(*frames_period*)

Bases: *MovieEvent*

Class that represent an event in a movie, for exemple a transient, neuropil etc…

class deepcinac.cinac_stratification.**StratificationCellTypeCamembert**(*data_list*, *description*, *n_max_transformations*, *debug_mode=False*)

augment_them_all()

Add to all movie patches in the camembert a given number of augmentation self.n_max_transformations :return:

print_data_description()

class deepcinac.cinac_stratification.**StratificationCamembert**(*data_list*, *description*, *n_max_transformations*, *debug_mode=False*)

compute_slices()

Compute the slices of the camembert :return:

add_augmentation_to_all_patches(*n_augmentation*)

Add to all movie patches in the camember a given number of augmentation, except neuropil :param n_augmentation: :return:

set_weights()

balance_all(*main_ratio_balance*, *first_round*)

balance_transients(*which_ones*, *crop_non_crop_ratio_balance*, *non_crop_ratio_balance*)

class deepcinac.cinac_stratification.**StratificationDataProcessor**(*data_list*, *description*, *n_max_transformations*, *main_ratio_balance=(0.6, 0.25, 0.15)*, *crop_non_crop_ratio_balance=(0.9, 0.1)*, *non_crop_ratio_balance=(0.6, 0.4)*, *debug_mode=False*)

get_new_data_list()

class deepcinac.cinac_stratification.**StratificationCellTypeDataProcessor**(*data_list*, *description*, *n_max_transformations*, *debug_mode=False*)

Class used to stratified data for cell type So far we make it simple, the postulate being that the data has already been stratify by the user (given the same amount of pramidal and interneuron cells for exemple). So for now we just used it to do data augmentation on each movie patch and create a new dataset thus

get_new_data_list()

deepcinac.cinac_structures

Module Contents

Classes

CinacMovie	Helper class that provides a standard way to create an ABC using
CinacDataMovie	Take the movie as a 2d array directly
CinacTiffMovie	Helper class that provides a standard way to create an ABC using
CinacFileReaderMovie	Helper class that provides a standard way to create an ABC using
CinacSplitedTiffMovie	Used if tiff movie has been splitted as as many tiff files as frames in the movie
CinacSplitedNpyMovie	Used if tiff movie has been splitted as as many tiff files as frames in the movie
CinacRecording	

Functions

get_cinac_movie_from_cinac_file_reader(cinac_file_reader)	**param cinac_file_reader** CinacFileReader instance
create_cinac_recording_from_cinac_file_segment(...)	**param identifier** cinac_recording identifier

class deepcinac.cinac_structures.**CinacMovie**

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

abstract get_frames_section(*frames*, *minx*, *maxx*, *miny*, *maxy*)

get_dimensions()

Get x and y dimensions of the movie Returns: a 1d array of integers

get_n_frames()

The number of frames in the movie Returns: integer

abstract get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

class deepcinac.cinac_structures.**CinacDataMovie**(*movie*, *already_normalized=False*)

Bases: *CinacMovie*

Take the movie as a 2d array directly

get_frames_section(*frames*, *minx*, *maxx*, *miny*, *maxy*)

get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

class deepcinac.cinac_structures.**CinacTiffMovie**(*tiff_file_name=None*, *tiff_movie=None*)

Bases: *CinacMovie*

Helper class that provides a standard way to create an ABC using inheritance.

get_frames_section(*frames*, *minx*, *maxx*, *miny*, *maxy*)

get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

class deepcinac.cinac_structures.**CinacFileReaderMovie**(*cinac_file_reader*, *segment*)

Bases: *CinacMovie*

Helper class that provides a standard way to create an ABC using inheritance.

get_frames_section(*frames*, *minx*, *maxx*, *miny*, *maxy*)

get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

class deepcinac.cinac_structures.**CinacSplitedTiffMovie**(*identifier*, *tiffs_dirname*, *already_normalized=False*, *tiff_file_name=None*, *tiff_movie=None*)

Bases: *CinacMovie*

Used if tiff movie has been splitted as as many tiff files as frames in the movie

get_frames_section(*frames_indices*, *minx*, *maxx*, *miny*, *maxy*)

get section of given frames from the calcium imaging movie :param frames_indices: numpy array of integers, representing the frame's indices to select :param minx: integer, min x coordinate :param maxx: integer, max x coordinate :param miny: integer, min y coordinate :param maxy: integer, max y coordinate

Returns:

get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

class deepcinac.cinac_structures.**CinacSplitedNpyMovie**(*identifier*, *tiffs_dirname*, *already_normalized=False*, *tiff_file_name=None*, *tiff_movie=None*)

Bases: *CinacMovie*

Used if tiff movie has been splitted as as many tiff files as frames in the movie

get_frames_section(*frames_indices*, *minx*, *maxx*, *miny*, *maxy*)

get section of given frames from the calcium imaging movie :param frames_indices: numpy array of integers, representing the frame's indices to select :param minx: integer, min x coordinate :param maxx: integer, max x coordinate :param miny: integer, min y coordinate :param maxy: integer, max y coordinate

Returns:

get_full_movie(*normalized*)

Full movie, if available a 3d array n_frames x len_y x len_x :param normalized: bool, if True return normalized movie, False original movie if available

Returns:

deepcinac.cinac_structures.**get_cinac_movie_from_cinac_file_reader**(*cinac_file_reader*)

Parameters
cinac_file_reader – CinacFileReader instance

Returns:

deepcinac.cinac_structures.**create_cinac_recording_from_cinac_file_segment**(*identifier*, *cinac_file_reader*, *segment*)

Parameters

- **identifier** – cinac_recording identifier
- **cinac_file_reader** – CinacFileReaderInstance
- **cinac_movie** – CinacMovie instance
- **segment** – tuple of 3 int (cell, first_frame, last_frame

Returns:

class deepcinac.cinac_structures.**CinacRecording**(*identifier*)

_build_traces()

get_raw_traces(*normalized*)

Parameters
normalized – if True, get the raw_traces normalized

Returns:

get_smooth_traces(*normalized*)

Parameters
normalized – if True, get the raw_traces normalized

Returns:

set_movie(*cinac_movie*)

Set the instance of CinacMovie, that will be used to get the frames given to the network :param cinac_movie:

Returns:

set_rois_from_suite_2p(*is_cell_file_name*, *stat_file_name*)

Parameters

- **is_cell_file_name** – path and file_name of the file iscell.npy produce by suite2p segmentation process
- **stat_file_name** – path and file_name of the file stat.npy produce by suite2p segmentation process

Returns:

set_rois_2d_array(*coord*, *from_matlab*)

Parameters

- **coord** – numpy array of 2d, first dimension of length 2 (x and y) and 2nd dimension of length the number of
- **integers** (*cells. Could also be a list of lists or tuples of 2*) –
- **from_matlab** – Indicate if the data has been computed by matlab, then 1 will be removed to the coordinates
- **zero.** (*so it starts at*) –

Returns:

set_rois_from_nwb(*nwb_data*, *name_module*, *name_segmentation*, *name_seg_plane*)

Parameters

- **nwb_data** – nwb object instance
- **name_module** – Name of the module to find segmentation. Will be used this way: nwb_data.modules[name_module] Ex: name_module = 'ophys'
- **name_segmentation** – Name of the segmentation in which find the plane segmentation. Used this way:get_plane_segmentation(name_segmentation) Ex: name_segmentation = 'segmentation_suite2p'
- **name_seg_plane** – Name of the segmentation plane in which to find the ROIs data
- **way** (*Used this*) – mod[name_segmentation]get_plane_segmentation(name_seq_plane) Ex: name_segmentation = 'my_plane_seg'

Returns:

set_rois_using_pixel_mask(*pixel_masks*)

Parameters

pixel_masks – list of list of 2 integers representing for each cell all the pixels that belongs to the cell

Returns:

get_n_frames()

Return the number of frames in the movie Returns:

get_n_cells()

get_source_profile_frames(*frames_indices*, *coords*)

Return frames section based on the indices of the frames and the coordinates of the corners of the section :param frames_indices: array of integers :param coords: tuple of 4 integers: (minx, maxx, miny, maxy)

Returns: A numpy array of dimensions len(frames_indices) * (maxy - miny + 1) * (maxx - minx + 1)

Package Contents

deepcinac.**__version__**

CHAPTER

TWO

INDICES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

E

F

G

N

O

P

R

S

T