pyAMNESIA

Release 0.0.1

Oct 15, 2020

Getting started

pyAMNESIA is a **py**thon pipeline for analysing the **A**ctivity and **M**orphology of **NE**urons using **S**keletonization and other **I**mage **A**nalysis techniques.

CHAPTER 1

Installation

We provide two ways to use pyAMNESIA.

- If you want a **lightweight tool** that runs in a command-line interface, you may want to install pyAMNESIA *as a standalone tool*;
- If you want to use CICADA to run the tool and have a **pretty GUI** to guide you, then you may want to install pyAMNESIA *as a CICADA module*.

We advise that you use conda to create a virtual environment for the tool.

```
conda create -n pyamnesia python==3.8  # or python==3.7
conda activate pyamnesia
```

1.1 Install pyAMNESIA

You only need to `git clone` pyAMNESIA:

```
# clone the repo
git clone https://gitlab.com/cossartlab/pyamnesia.git
cd pyamnesia
# install the dependencies
sh requirements.sh
```

Note: On Windows, you may not be able to use the `sh` command. To solve this, you may either download Cygwin or write the commands in `requirements.sh` yourself:

```
pip install -r requirements.txt
conda install -c conda-forge skan
conda install -c conda-forge hdbscan
```

The installation relies on conda to install skan, which is the package we use to skeletonize images. If you cannot use conda, please install skan and hdbscan with `pip`; we cannot guarantee that it will run smoothly though.

If you want to use pyAMNESIA as a standalone tool, no need for you to read the next section and you can read the *how-to page* to run the tool.

1.2 Install CICADA

In order to have a GUI to guide you during the pyAMNESIA analysis, you can install CICADA using `pip`, or from source (for more information, please visit CICADA's documentation):

```
# using pip
pip install -U pycicada
# from source
git clone https://gitlab.com/cossartlab/cicada.git
```

Warning: As for now, CICADA still is a work in progress, we advise that you clone the stable version we worked on and pip install some additional packages to make it run:

```
# install stable version
git clone -b stable-pyamnesia https://gitlab.com/theodumont/cicada.git
cd cicada
# pip install some packages
pip install -r requirements.txt
pip install neo elephant
```

CICADA and pyAMNESIA are now installed and you can go to the *how-to page* to run the tool.

CHAPTER 2

How to use

Table of contents

2.1 pyAMNESIA as a standalone tool

2.1.1 Use the CLI

If you want a **lightweight tool**, you may want to use pyAMNESIA **as a standalone tool**, using the **CLI**.

1. *(optional)* Change the `results_path` entry of the `pyamnesia/src/pyamnesia/config.yaml` file into the absolute path to the directory in which you want the results files to be. By default, results will be stored in the `pyamnesia/blob/res` folder.
2. Adapt the parameters in `pyamnesia/src/pyamnesia/config.yaml` if you want to. Fore more info on this, please read the **Parameter selection** section of the **OVERVIEW** page that corresponds to the analysis you are running. You can also use the **configuration templates** that are in the `pyamnesia/src/pyamnesia/config_templates/` directory, and read the *Configuration templates* page.
3. Launch the CLI by executing these commands in your terminal:

```
cd path/to/pyamnesia  # go to project
cd src
python pyamnesia.py   # execute CLI
```

4. The CLI is waiting for your input. Please write `skeleton`, `clustering` or `factorization` depending on the analysis you want to run.
5. When the *"Choose input TIF file(s)"* dialog window opens (you may need to search it in your open windows if it does not appear on top of your desktop), select the folder where your data is.

```
data/               # folder to select
    example1.tif
    example2.tif
    example3.tif
```

6. If you launched a `clustering` or a `factorization` analysis, a *"Choose skeleton results directory"* dialog window opens (see *here* for more info on the skeleton analysis, and *here* for more info on why a prior skeleton analysis is needed). Select the results folder of the corresponding skeleton analysis.

```
res/
    Skeleton_2020_01_01.00-00-01/    # folder to select
        example1/
            branch_validation/
            sequence_projection/
            skeleton_mask/
        example2/
            branch_validation/
            sequence_projection/
            skeleton_mask/
    Skeleton_2020_02_02.00-00-02/
```

   If the skeleton results of a `.tif` file cannot be found in the specified folder, the analysis of this file will be skipped. For instance, in the code snippets above, the `example3.tif` analysis will be skipped.

Here is a demo of the CLI tool.

Demo of the CLI tool.

2.1.2 Use pyAMNESIA's modules

If you want to **automate the analysis** (schedule the analysis, run them on all the files in a specific directory, add a Python preprocessing step...), you can use pyAMNESIA's modules.

This abstract class is inherited for each of the three modules `Skeleton`, `Clustering` and `Factorization`:

In order to use the inherited classes, you have to import them from their respective modules and instanciate them, using

```
from cli.pyamnesia_analysis_name import PyamnesiaAnalysisName
analysis = PyamnesiaAnalysisName()
```

where `name` is `'skeleton'`, `'clustering'` or `'factorization'`. For instance, if you want to apply the skeleton analysis to every file in the current working directory:

```
import os
from cli.pyamnesia_analysis_skeleton import PyamnesiaAnalysisSkeleton
analysis = PyamnesiaAnalysisSkeleton()

for file in os.listdir():  # for all files in cwd
    file_path = os.path.abspath(file)
    analysis.run_analysis(file_path)
```

2.2 pyAMNESIA as a CICADA module

If you want to use CICADA to run the tool and have a **pretty GUI** to guide you, then you may want to use pyAMNESIA **as a CICADA module**.

2.2.1 Setup

You have to add the pyAMNESIA analysis to CICADA. To do so:

1. Launch CICADA:

```
cd path/to/cicada  # go to CICADA
cd src
python -m cicada   # execute CICADA
```

2. Add the `tif_wrapper` (located in `pyamnesia/src/pyamnesia/gui/`) to CICADA by clicking on the first +.
3. Add the `ci_analyses_pyamnesia` folder (located in `pyamnesia/src/pyamnesia/gui/`) to CICADA by clicking on the second +.
4. Select the 3 analysis and click `RUN CICADA`.

Demo: CICADA setup for using pyAMNESIA.

2.2.2 Use

1. Launch CICADA:

```
cd path/to/cicada  # go to CICADA
cd src
python -m cicada   # execute CICADA
```

2. Select the 3 analysis and click `RUN CICADA`.
3. Go to `File > Open new dataset` or use `ctrl + O` and select the folder where your data is.

```
data/               # folder to select
    example1.tif
    example2.tif
    example3.tif
```

4. Select the `.tif` files to analyse, then click on the first arrow.

5. Select the analysis to run, then click on the second arrow.

Demo: how to select the `.tif` *files.*

6. If you chose a `Clustering` or a `Factorization` analysis, select the results folder of the skeleton analysis you run before (see *here* for more info on the skeleton analysis, and *here* for more info on why a prior skeleton analysis is needed).

```
res/
    Skeleton_2020_01_01.00-00-01/     # folder to select
        example1/
            branch_validation/
            sequence_projection/
            skeleton_mask/
        example2/
            branch_validation/
            sequence_projection/
            skeleton_mask/
    Skeleton_2020_02_02.00-00-02/
```

   If the skeleton results of a `.tif` file cannot be found in the specified folder, the analysis of this file will be skipped. For instance, in the code snippets above, the `example3.tif` analysis will be skipped.

7. Scroll down and choose the results directory.
8. Adapt the parameters if you want to. Fore more info on this, please read the **Parameter selection** section of the **OVERVIEW** page that corresponds to the analysis you are running.
9. Click `Run analysis`.

Demo: how to launch an analysis.

CHAPTER 3

Introduction

The Cossart lab aims at gaining an understanding of the hippocampal function by studying the development of its network dynamics.

3.1 Motivation

In the context of studying the mechanisms by which hippocampal assemblies evolve during the development of mice (5 to 12 days), we dispose of some two-photon calcium imaging videos, such as this one:

A two-photon calcium imaging movie.

The analysis of these calcium imaging videos may be split into two different parts:

- the evolution of the **morphology**, being the number of elements (somata, neurites…) and their visible connexions;
- the evolution of the **neuronal activity**, being the transients characteristics and the coactivity[1] of the aforementioned elements.

[1] that have similar and simultaneous neural activities over time.

In the state-of-the-art calcium imaging pipelines such as Suite2p and CaImAn, somata are segmented to subsequently study the evolution of their activity. In such cases, pixels in the region of a soma can reliably be considered as coactive, making the soma a *morphological* **and** *functional* entity.

Here, we want to study the **whole neural structure** of the hippocampus; not only the somata but also the neurites linking them. And there is **no obvious correlation** between the morphological "branches" that one can see on the videos, and the neurites, that are coactive functional entities. This is mainly due to the Z-axis projection - which "hides" vertical neurites and creates overlaps - but also to imaging hazards, or simply neurites that only activate partially during a neural transmission. Therefore, we do not know *a priori* whether a visible "branch" is a functional entity or not. We thus need to **separate** the morphological and the functional approaches, and try to build some **coherent structures** that could be considered as entities, in order to analyse their **coactivity** later.

Here is the problem we are trying to answer:

Problem

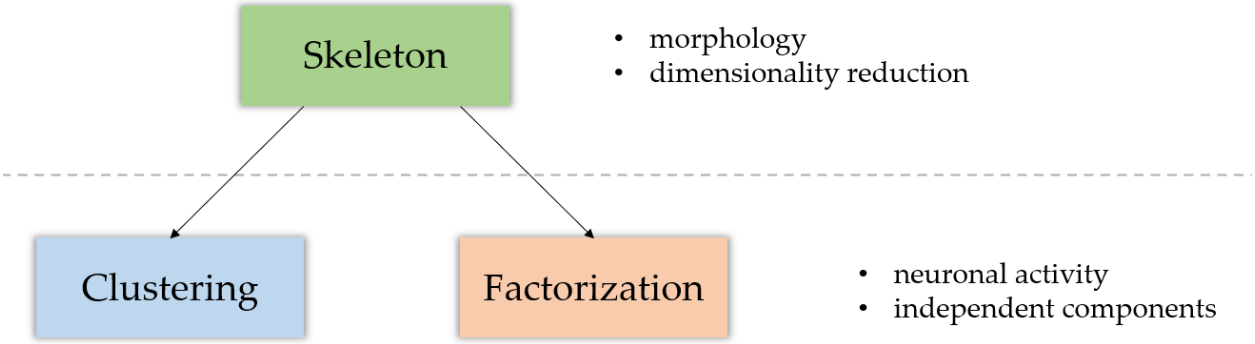
How can we use calcium imaging to get **statistics** on the evolution of interneurons in the hippocampus (morphology *and* activity) during **mice development**?

- **input:** a calcium imaging `.tif` sequence
- **output:** clusters of coactive pixels & morphological statistics

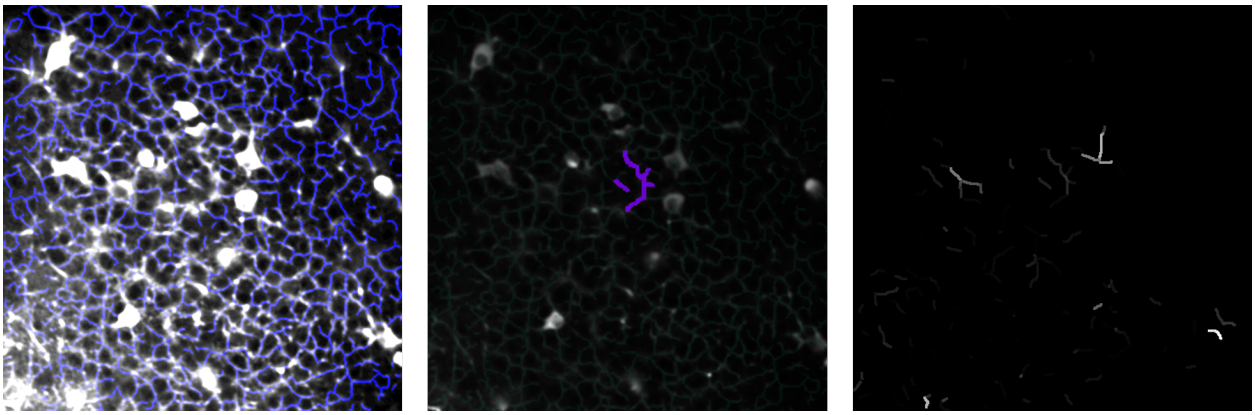
---

3.2 Functionalities

We split our approach in three parts:



1. The *skeletonization* module focuses on the **morphological** analysis of the data, by computing the underlying *morphological skeleton* of the sequence.
2. The *clustering* module performs an **activity analysis** of the elements in the sequence, whether they be pixels (no prior skeleton analysis) or branches (see the *module page* for more information about this). Its goal is to return clusters of coactive pixels.
3. The *factorization* module has a similar goal to the clustering module. It returns independent components of **coactive pixels**, but uses **matrix factorization techniques** to do so.



From left to right: a skeletonized image; a cluster; a factorization component.

Whereas the skeletonization part focuses on the **morphological** aspect of the analysis, both of the clustering and factorization modules tackle the **activity analysis** of it. As shown on the diagram above, the skeleton module is prior to the two others.

CHAPTER 4

Skeleton

Table of contents

In the next sections, the method illustrations (results, inputs...) focus on our study on Calcium Imaging data from the mice hippocampus (see *Motivation* for more info).

4.1 Methods

The `skeleton` module focuses on the **morphological analysis** of the data. It also reduces the dimensions of it, which is very useful to reduce the computational time of the activity analysis that can be run behind it.

A two-photon calcium imaging movie.

4.1.1 Sequence denoising

First of all, in order to remove the noise from the sequence, we use a **3D gaussian smoothing**. It allows to reduce both spatial and temporal noises.

4.1.2 Image skeletonization

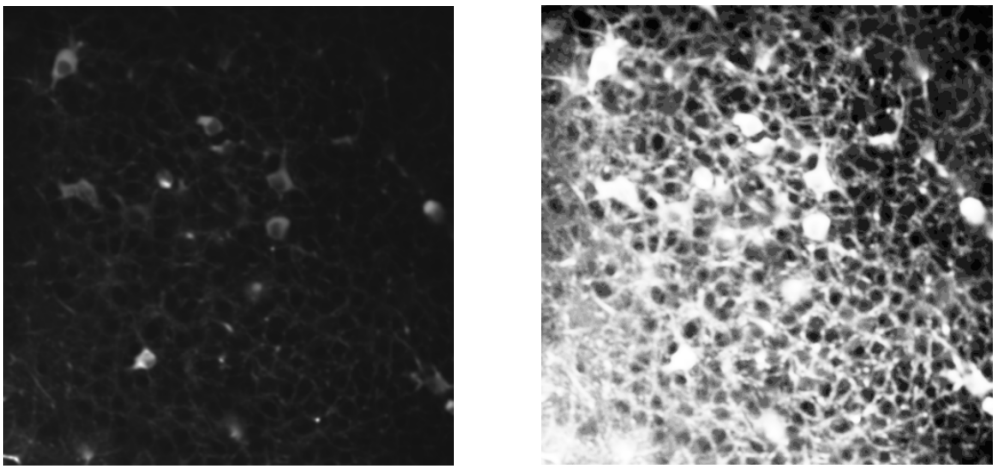
Assuming that the movies are well motion corrected and that not too much branches overlap, the filmed stack can be **"summed up"** with one image, representative of the real physical morphology of the imaged stack. An easy way of doing this is taking the temporal mean image of the movie, but other elementary methods were implemented.

Once we have this **summary image**, we apply some **image processing** methods to extract its skeleton. The following paragraphs detail the method used to skeletonize the summary image.

Note: Almost every sub-method is **flexible** to the input data. Several tips on how to adjust the method parameters are given in the corresponding *parameter selection* section.

Histogram equalization

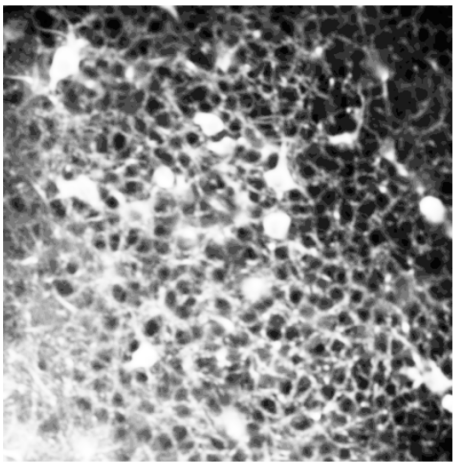
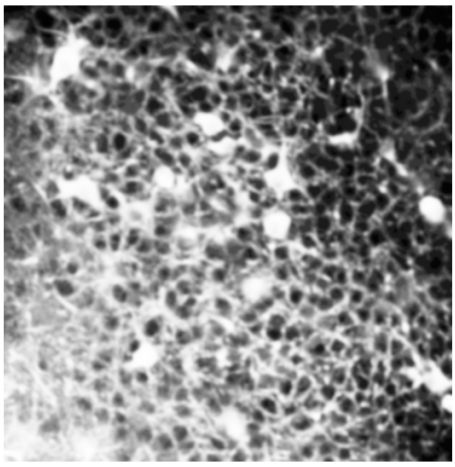
As the somata have a very high intensity throughout the movies, we have to be sure that they will not have a significant influence over the binarization method – which is based on a local intensity thresholding. To prevent this from happening, we firstly perform an **histogram equalization**. This image processing method consists in flattening the distribution of the image intensities in order to enhance the contrast.



Left: summary image; Right: histogram equalization.

Gaussian smoothing

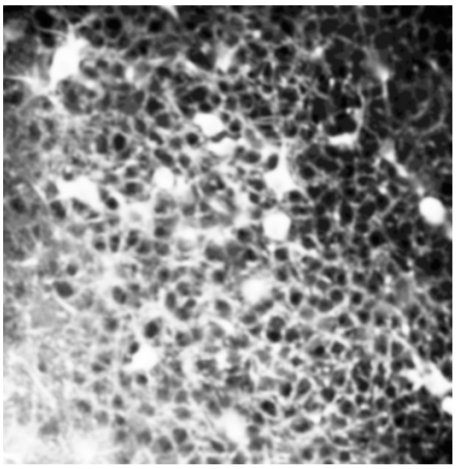
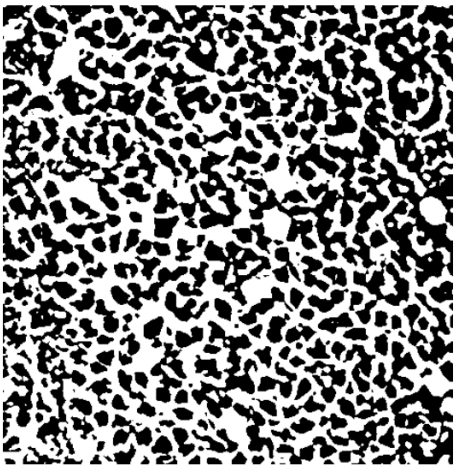
The image noise can be problematic during the binarization. Indeed, the consequent local intensity fluctuation makes the thresholding algorithm generate small **"holes"** on the binary structure. In order to limit their presence, we smooth the equalized image with a **Gaussian filter**.

Left: histogram equalization; Right: gaussian smoothing.

Adaptive thresholding

Once the image is well contrasted and smoothed, we binarize it with an **adaptive Gaussian thresholding**. Contrary to the classic thresholding methods, that set a global threshold for the image intensities, the adaptive thresholding changes the threshold depending on the local environment, which allows it to deal with eventual intensity shifts, like shadows zones for instance. For more details, see[1].

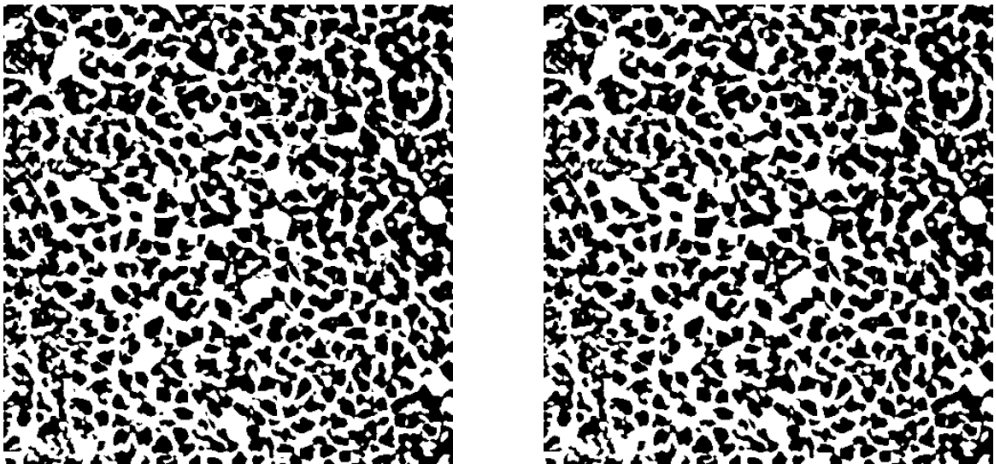



Left: gaussian smoothing; Right: adaptive thresholding.

Holes removing

Even with the preventive Gaussian smoothing, some small **"holes"** may still remain in the binary skeleton. It is important to get rid of them because they can cause the resulting skeleton to go around them, not only missing the heart of the structure, but also creating useless cycles around the holes. Thus, we developed a simple method that detects the small holes in the binary image and **"fills"** them.

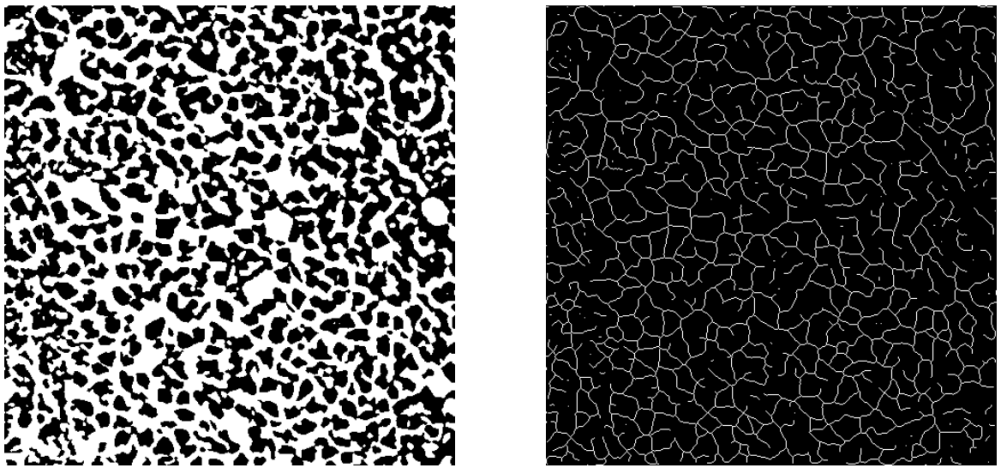
[1] *Image Thresholding*, OpenCV, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html



Left: adaptive thresholding; Right: holes removing.

Skeletonization (Lee)

Once the binary structure is clean, we are able to skeletonize it with *Lee's* method. For more details, see[2].

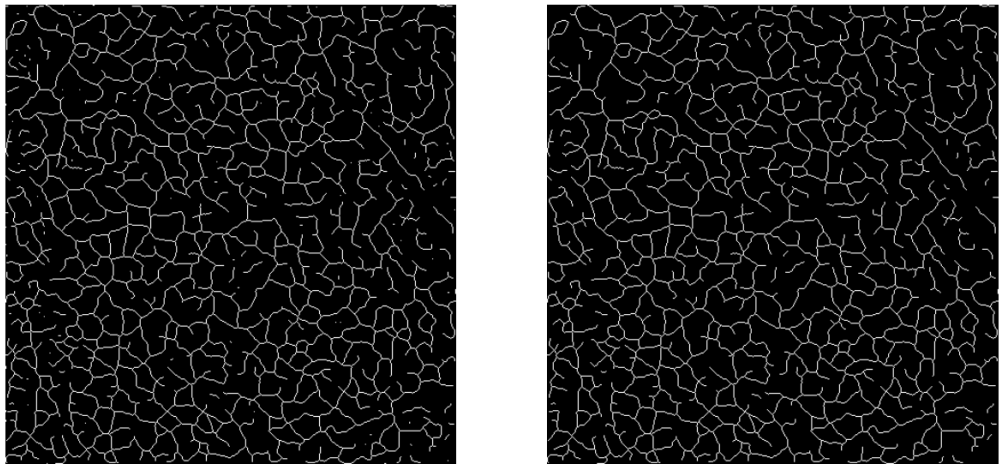


Left: holes removing; Right: skeletonization (Lee).

Parasitic skeletons removal

During skeletonization, the isolated spots in the binary image are converted to **small isolated skeletons**. They are not harmful, but we can still eliminate them to clean the skeleton structure.

[2] *Skeletonize*, Scikit-Image, https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html



Left: skeletonization (Lee); Right: parasitic skeletons removing.

Results

Finally, we can project the sequence on the computed skeleton and thus reduce its dimensionality.

Intensity projection on the skeleton, pixel by pixel.

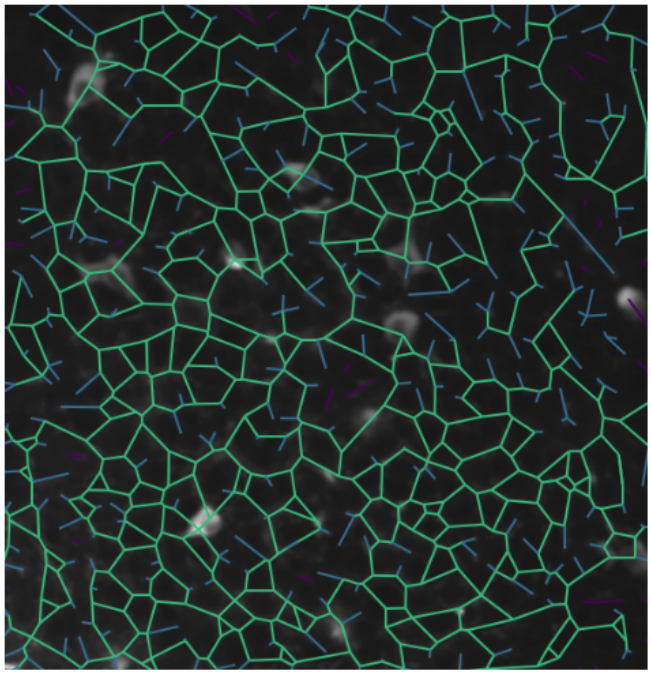
Note: This projection, pixel by pixel, is obtained by setting `projection_method` to `'pixel'`. For more details, see *Parameter selection*.

4.1.3 In-depth skeleton analysis

We can get statistics out of the skeleton mask using the python package **skan**.

By comparing the skeleton structure to a non-oriented graph, this package allows to decompose the skeleton into a set of **branches** and get some information about them: coordinates of their pixels, lengths, neighbors, *etc.*... By gathering all this data, we can get **morphological statistics** about the structure of the image and **clean it** by removing some specific branches. For more details about this package, see[3].

[3] *Skan's documentation*, Skan, https://jni.github.io/skan/index.html



	A	B	C	D	E	F	G	H	I	J	K	L
1		skeleton-id	node-id-src	node-id-dst	branch-distance	branch-type	mean-pixel-value	stdev-pixel-value	image-coord-src-0	image-coord-src-1	image-coord-dst-0	image-coord-dst-1
2	0	1	1	400	42.4550441227157	1	1.0	0.0	0.0	42.0	24.0	71.0
3	1	2	3	46	16.24264068711928	4	1.0	0.0	0.0	450.0	1.0	465.0
4	2	1	12	153	16.356527420689975	1	1.0	0.0	1.0	39.0	9.666666666666666	28.666666666666668
5	3	1	15	125	14.528100295943789	1	1.0	0.0	1.0	132.0	6.666666666666667	142.33333333333334
6	4	3	24	445	29.556349186104054	4	1.0	0.0	1.0	163.0	16.0	184.0
7	5	1	29	89	3.414213562373095	1	1.0	0.0	1.0	220.0	4.0	221.0
8	6	1	33	134	8.82842712474619	1	1.0	0.0	1.0	259.0	6.0	254.0
9	7	1	34	152	6.414213562373095	1	1.0	0.0	1.0	315.0	7.0	316.0
10	8	1	35	195	26.556349186104054	1	1.0	0.0	1.0	414.0	11.0	395.0
11	9	1	59	133	4.414213562373095	1	1.0	0.0	2.0	234.0	6.0	233.0
12	10	1	62	163	8.414213562373096	1	1.0	0.0	2.0	350.0	10.0	351.0
13	11	1	65	297	17.542472332656494	1	1.0	0.0	2.0	375.0	15.333333333333334	374.3333333333333
14	12	1	66	195	10.242640687119286	1	1.0	0.0	2.0	392.0	11.0	395.0
15	13	1	70	151	4.6996731711975945	1	1.0	0.0	3.0	118.0	7.666666666666667	118.33333333333333
16	14	1	77	260	11.828427124746192	1	1.0	0.0	3.0	276.0	14.0	274.0
17	15	1	83	300	16.899494936611667	1	1.0	0.0	3.0	436.0	16.0	420.0
18	16	1	89	367	10.656054249492381	1	1.0	0.0	4.0	221.0	13.0	217.0
19	17	1	89	133	12.828427124746191	2	1.0	0.0	4.0	221.0	6.0	233.0

Left: skeleton overlaying; Right: logo `.csv` *file containing branch DataFrame; Bottom: branch DataFrame extract.*

As explained in the *Motivation* section, *"there is no obvious correlation between the morphological 'branches' that one can see on the videos, and the neurites, that are coactive functional entities"*. That is why we are naturally tempted to uniformize the intensity along each branch, to conclude whether the averaging is biologically correct.

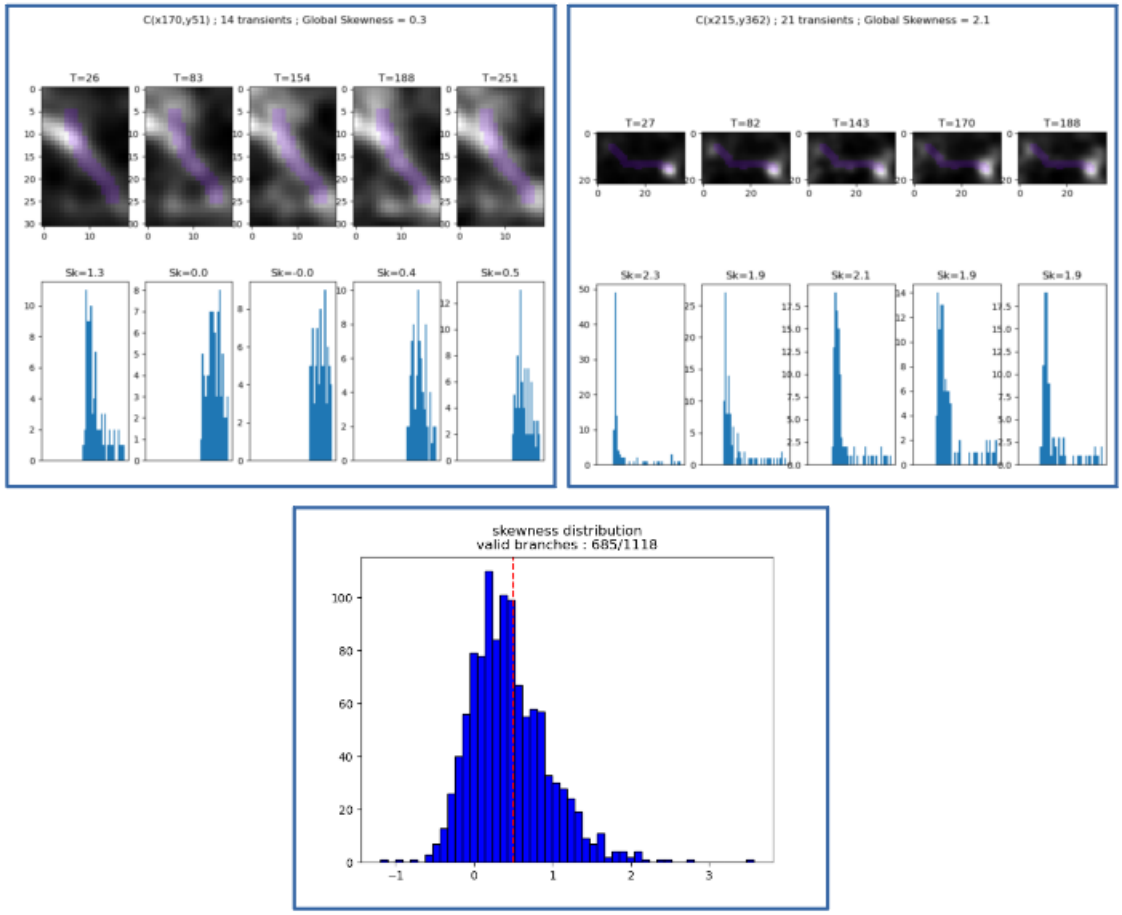
Here is an example of sequence projected on branches with intensity uniformization.

Intensity projection on the skeleton, branch by branch.

Note: This projection, branch by branch, is obtained by setting `projection_method` to `'branch'`. For more details, see *Parameter selection*.

4.1.4 Branch uniformity study

To check whether this branch intensity uniformization is legitimate, we drew from *Gauthier et al.* method[4], and developed **transient-by-transient visualization module** to classify the morphological branches, in two categories: valid and not valid. A branch is qualified as "valid" if its intensity during its transients is uniform. It confers the branch a functional interpretation, as a part of one and only one neurite, which allows to average its pixels intensities. To quantify the uniformity, we used the skewness metric (see[5]).



Top: screenshots of the visualization module; Bottom: distribution of skewness among the branches.

Depending on the movie, the number of remaining branches may vary.

Intensity projection on the valid branches.

4.2 Parameter selection

4.2.1 Sequence denoising

Here are the parameters that affect the sequence denoising.

[4] *"Detecting and Correcting False Transients in Calcium Imaging"*, Gauthier et al., https://www.biorxiv.org/content/10.1101/473470v1
[5] *Skewness*, Wikipedia, https://en.wikipedia.org/wiki/Skewness

Tip

The effects of these parameters can be seen by setting the parameter `save_denoised_sequence` to `True` and inspecting the `preprocessing/` folder generated when launching the `skeleton` module.

Selecting `denoising_method`

- **type:** `str`
- **default:** `'3D'`

Method used to denoise the sequence. The choices are:

- `'2D'`: Smooth each frame;
- `'3D'`: Smooth the whole sequence, as a 3D image.

Selecting `sigma_denoising`

- **type:** `float`
- **default:** `1.0`

Sigma in gaussian smoothing for denoising the sequence.

Warning: When `denoising_method` is set to `'3D'`, the smoothing is also temporal. In order to keep the transients locality, you must be careful of not setting it too high.

4.2.2 Image skeletonization

Here are the parameters used to perform *Image skeletonization*.

Tip

The effects of these parameters can be seen individually on the intermediate images by inspecting the `skeleton_mask/` folder generated when launching the `skeleton` module.

Important: These parameters are essentially morphological. Thus they need to be adapted to the scale and resolution of the input sequence. You might need several tests to find good parameter values.

Selecting `summary_method`

- **type:** `str`
- **default:** `'mean'`

The `summary_method` parameter selects which method to use for the generation of a summary image from the `.tif` movie. The choices are:

- `'mean'`: get the mean image along the time axis.

- `'median'`: get the median image along the time axis.
- `'standard_deviation'`: get the standard deviation image along the time axis.

Note: When `summary_method` is set to `'standard_deviation'`, it is needed to group some frames before computing the sequence standard deviation. That is the role of the next parameter, `std_grouping_ratio`.

Selecting `std_grouping_ratio`

- **type:** `int`
- **default:** `10`

The `std_grouping_ratio` parameter selects the number of sequence frames to group and average before computing the sequence standard deviation, in order to remove noise from the summary image. It should be large enough to remove the noise, but not too high, in order to preserve the intensity variations of the neurites.

Selecting `projection_substructure`

- **type:** `str`
- **default:** `'skeleton'`

The `projection_substructure` parameter selects which structure is used for the mask on which we project the intensities.

- `'skeleton'`: the mask is the skeletonized image.
- `'active'`: the mask is the binary image, just before the step of skeletonization in *Image skeletonization*.

The interest of this method is detailed in *Skip skeletonization*.

Selecting `sigma_smoothing`

- **type:** `float`
- **default:** `1.0`

The `sigma_smoothing` parameter selects the standard deviation for Gaussian kernel. The higher it is, the smoother is the image, and the fewer holes there are in the binary image.

Selecting `adaptive_threshold_block_size`

- **type:** `int`
- **default:** `43`

The `adaptive_threshold_block_size` parameter selects the size of a pixel neighborhood that is used to calculate a threshold value for the pixel. It must be odd. For more details, see[1].

Tip

Should be set so that the block captures a soma + a bit of its environment. For example, if in the movie a typical soma size is 30×30 pixels, then `43` should be a good value for `adaptive_threshold_block_size` to capture the environment.

Selecting `holes_max_size`

- **type:** `int`
- **default:** `20`

The `holes_max_size` parameter selects the maximum size (number of pixels) of the holes we want to fill in the binary image. As explained in *Image skeletonization*, some parasitic holes may appear during binarization. Most of the time, they appear inside the somas or inside the branches, due to defects in thresholding.

Important: These parasitic holes must not be confused with larger holes, representing rings, or simply void between branches.

Tip

A good advice to find the perfect value, is to identify the minimum size of the holes we want to keep (small rings for instance), and take a threshold just below.

Note: To remove the objects, we use the `scikit-image` method `remove_small_objects`, with `connectivity=1`, which means that two diagonal black pixels are **not** considered as part of the same hole. For more details, see[6].

Selecting `parasitic_skeletons_size`

- **type:** `int`
- **default:** `5`

The `parasitic_skeletons_size` parameter selects the maximum size (number of pixels) of isolated skeletons (see *Image skeletonization*) we want to remove.

Note: To remove the objects, we also used the `scikit-image` method `remove_small_objects`, but this time with `connectivity=2`, which means that two diagonal white pixels are considered as part of the same skeleton.

4.2.3 Projection method

As explained in *Image skeletonization*, two components are analyzed in our trace study: pixels of the skeleton individually, and branches of the skeleton. Respectively, two projection methods are developed: `pixel` and `branch`.

Selecting `projection_method`

- **type:** `str`
- **default:** `'pixel'`

[6] *Remove small objects* Scikit-Image, https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.remove_small_objects

The `projection_method` parameter selects the projection method. For `pixels`, we simply take the coordinates of the white pixels, and get their trace. For `branch`, we first get the coordinates of all the pixels in the branches, using the `skan` package, and then average their intensity, through time.

4.2.4 Trace processing

This parameter section is related to the various signal processing methods used to clean the traces.

Selecting `element_normalization_method`

- **type:** `str`
- **default:** `'z-score'`

The `element_normalization_method` parameter selects the method for normalizing the traces. The choices are:

- `'null'`: no normalization
- `'mean-substraction'`: $T = T - \overline{T}$
- `'z-score'`: $T = (T - \overline{T})/\sigma(T)$

Note: The common method for trace processing is `z-score`, because it allows to set en the same footing all the elements we consider. But we can imagine some cases where this method is not suitable, for instance if we want to give more importance to the somatas, which have high intensities.

Selecting `element_smoothing_window_len`

- **type:** `int`
- **default:** `11`

The `element_smoothing_window_len` parameter selects the dimension of the smoothing window used for convolution. It must be odd.

Warning: Set it below the minimum frame difference between two transients. Otherwise, 2 transients would be merged !

Selecting `branch_radius`

- **type:** `int`
- **default:** `1`

The `branch_radius` parameter selects the radius of the branches.

Note: In the case of `pixel`, it is only for display, but in the case of `branch`, all the pixels in the dilated branch are taken into account in the branch intensity averaging.

4.2.5 Branch validation

This parameter section is related to the *Branch uniformity study*.

Selecting `pixels_around`

- **type:** `int`
- **default:** `5`

The `pixels_around` parameter selects the number of pixels to display around the analyzed branch, for visualization. See *Branch uniformity study* for an example.

Selecting `peak_properties`

- **type:** `dict`
- **default:** `height=None`, `threshold=None`, `distance=None`, `prominence=3`, `width=None`

The `peak_properties` parameter selects the properties relative to the trace peaks, in order to consider an activity variation as a transient. It contains different sub-parameters:

- `height`: required height of peaks.
- `threshold`: required threshold of peaks, the vertical distance to its neighboring samples.
- `distance`: required minimal horizontal distance in samples between neighbouring peaks.
- `prominence`: required prominence of peaks.
- `width`: required width of peaks in samples.

(Source: scipy `find_peaks`[7]). For an example on the influence of each parameter, see[8].

Note: The values need to take into account the normalization method. The advantage of z-scoring is that these parameters can be kept for different inputs.

Selecting `transient_neighborhood`

- **type:** `int`
- **default:** `2`

The `transient_neighborhood` parameter selects the number of frames between the transients onsets and peaks.

Disclaimer

The assumption that the number of frames between transients onsets and peaks is constant is false. However we decided to keep it simple as is doesn't skew the method's results. But we could consider to implement a method that automatically detects the transients onsets and offsets, using Scipy's `find_peaks` function.

---

[7] *Find peaks* Scipy, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

[8] *Peak-finding algorithm for Python/SciPy* stack overflow, https://stackoverflow.com/questions/1713335/peak-finding-algorithm-for-python-scipy

Selecting `skewness_threshold`

- **type:** `float`
- **default:** `0.5`

The `skewness_threshold` parameter selects the skewness value above which a branch isn't considered as valid. For more details refer to *Branch uniformity study*.

CHAPTER 5

Clustering

Table of contents

In the next sections, the method illustrations (results, inputs...) focus on our study on Calcium Imaging data from the mice hippocampus (see *Motivation* for more info).

5.1 Methods

The `clustering` module provides a clustering pipeline to group coactive elements in a `.tif` sequence.

A transient, where several pixels coactivate.

It takes as an input an array of voltage traces, projects them in a 2D plane using **t-SNE** (t-distributed Stochastic Neighbor Embedding)[1] and find clusters using **HDBSCAN** (Hierarchical Density-Based Spatial Clustering of Applications with Noise)[2].

[1] *t-SNE*, Scikit-Learn, https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[2] *HDBSCAN Documentation*, HDBSCAN, https://hdbscan.readthedocs.io/en/latest/

5.1.1 Dimensionality reduction

Before clustering the traces, it is important to build a relevant metric in order to assess their proximity. This metric can then be used to compute the distance matrix between the traces, which enables their projection in a 2D space, to simplify the representation and increase the clustering computation speed.

PCA

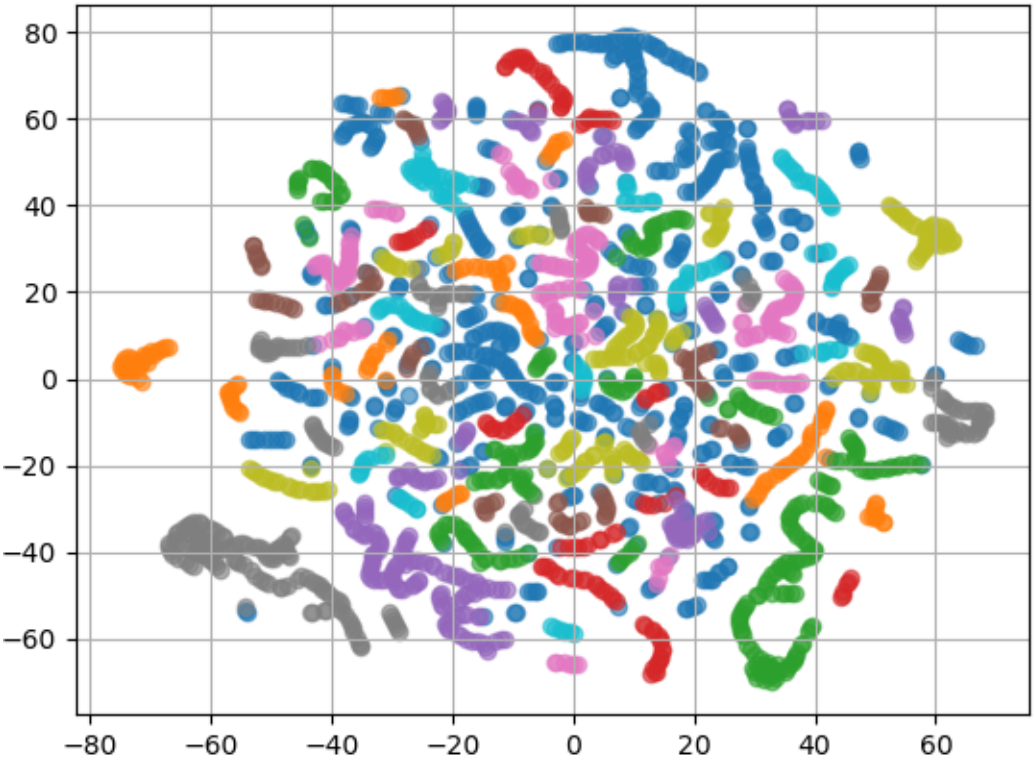
As t-SNE algorithm needs low dimension inputs to run fast, we first reduce the traces dimensionality using **PCA** (Principal Components Analysis).

t-SNE

Then we define relevant distance metric to assess traces correlation. In our case, the most famous one is **"Pearson correlation"**[5], which measures the linear correlation between two variables. **"Spearman's rank correlation"**[5] is also particularly adapted in our case. This allows to compute the distance matrix of the components traces, which **t-SNE** processes to project them into a 2D plane with a non-linear method.

5.1.2 Clustering

Once the traces are projected in a 2D plane, we apply **HDBSCAN**.



A plot of 2D projected traces.

[5] *Clearly explained: Pearson V/S Spearman Correlation Coefficient*, Juhi Ramzai on Medium, https://towardsdatascience.com/clearly-explained-pearson-v-s-spearman-correlation-coefficient-ada2f473b8

Each point represents one component. The components having "similar" traces (based on the distance matrix) are close, and the clustering algorithm forms colored groups.

5.2 Parameter selection

5.2.1 Dimensionality reduction

This parameter section is related to the *Dimensionality reduction*.

Selecting `normalization_method`

- **type:** `str`
- **default:** `'z-score'`

The `normalization_method` parameter selects the method for normalizing the traces. The choices are:

- `'null'`: no normalization
- `'mean-substraction'`: $T = T - \overline{T}$
- `'z-score'`: $T = (T - \overline{T})/\sigma(T)$

Note: The normalization is done again to make sure the dimensionality reduction is under the right format. If the traces were already *normalized during skeletonization*, there is no need to do it again.

Selecting `pca_variance_goal`

- **type:** `float`
- **default:** `0.90`

The `pca_variance_goal` parameter selects the **percentage** of variance to keep after PCA. It must be a `float` inferior to `1`. The higher it is, the more principal components are kept.

Tip

The advantage of setting a percentage goal is that you keep control on the level of information you want to keep on the traces. You can also input an `int`, if you know the exact number of components you want to give to the following t-SNE algorithm. But be aware of the fact that you might loose a lot of information if you set it too low.

Tip

The effects of this parameter can be seen on the "before/after" graphs by inspecting the `dimensionality_reduction/` folder generated when launching the `clustering` module.

Selecting `tsne_distance_metric`

- **type:** `str`
- **default:** `'spearman'`

The `tsne_distance_metric` parameter selects the distance metric for t-SNE distance matrix computation. It can be one of `pearson`, `spearman` and every metric in `sklearn.metrics.pairwise.distance_metrics`.

Tip

The effects of this parameter can be directly seen on the clusters plots by inspecting the `hdbscan_clustering/` folder generated when launching the `clustering` module. For instance, the clusters traces found using `spearman` will be largely different from the ones using `euclidean`.

Selecting `tsne_perplexity`

- **type:** `int`
- **default:** `30`

The `tsne_perplexity` parameter selects *"the number of nearest neighbors that is used in other manifold learning algorithms"*[1].

Important: The value needs to take into account the typical size of clusters we want. For instance, if we want to cluster skeleton pixel components, the perplexity needs to be much higher than if we want to cluster branches. An example is done in *Consecutive clusterings*.

Tip

The effects of this parameter can be directly seen on the point distribution in the scatter plot by inspecting the `hdbscan_clustering/` folder generated when launching the `clustering` module.

Selecting `tsne_random_state`

- **type:** `int`
- **default:** `42`

The `tsne_random_state` parameter selects the random number generator for t-SNE algorithm. For reproducible results, pass an `int`.

5.2.2 Clustering

This parameter section is related to the *Clustering*.

Selecting `min_cluster_size`

- **type:** `int`
- **default:** `5`

The `min_cluster_size` parameter selects the minimum size of clusters. For more details on how to set this parameter correctly, see[4].

Tip

The effects of this parameter can be directly seen on the colors of the scatter plot by inspecting the `hdbscan_clustering/` folder generated when launching the `clustering` module.

Selecting `min_samples`

- **type:** `int`
- **default:** `5`

The `min_samples` parameter selects *"the number of samples in a neighbourhood for a point to be considered a core point"*[3]. For more details on how to well couple this parameter with `min_cluster_size`, see[4].

Selecting `hdbscan_metric`

- **type:** `str`
- **default:** `'euclidean'`

The `hdbscan_metric` parameter selects *"the metric to use when calculating distance between instances in a feature array"*[3].

[4] *HDBSCAN Parameters*, HDBSCAN, https://hdbscan.readthedocs.io/en/latest/parameter_selection.html
[3] *HDBSCAN API*, HDBSCAN, https://hdbscan.readthedocs.io/en/latest/api.html

CHAPTER 6

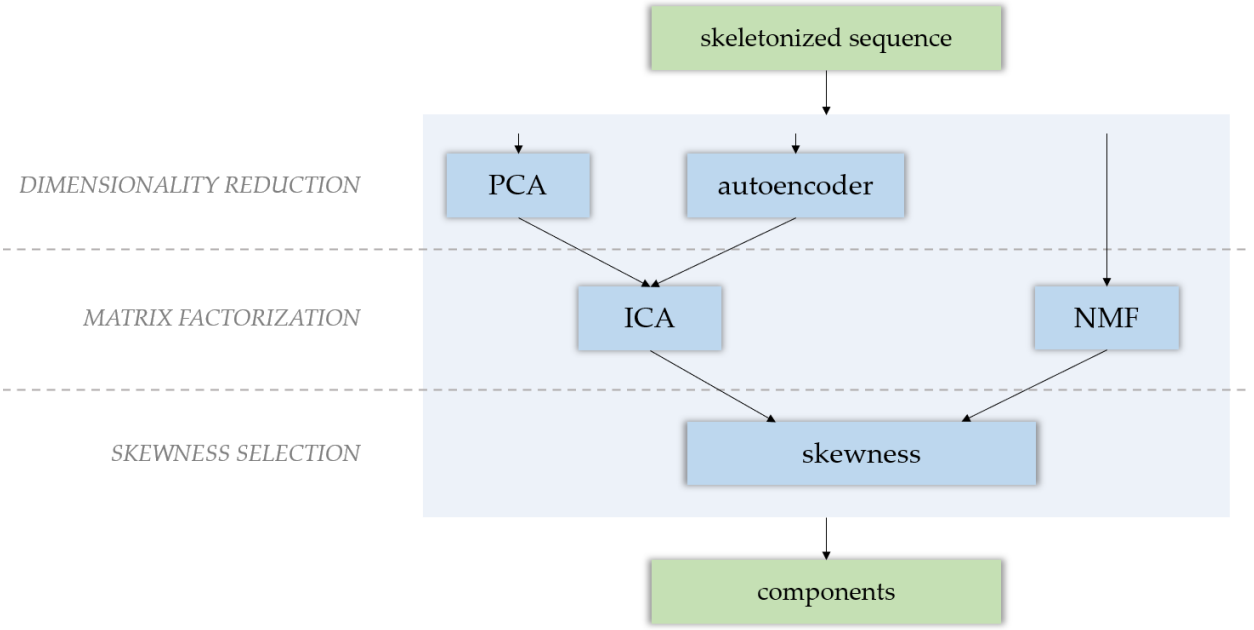
Factorization

Table of contents

In the next sections, the method illustrations (results, inputs...) focus on our study on calcium imaging data from the mice hippocampus (see *Motivation* for more info).

6.1 Methods

The `factorization` module provides several matrix factorization techniques to produce independent components from a `.tif` sequence, such as **PCA** (Principal Components Analysis), **ICA** (Independent Components Analysis) and **NMF** (Nonnegative Matrix Factorization). If you choose to perform an ICA, a dimensionality reduction is needed. For this, you can either perform a PCA, or use an **autoencoder**. After either a NMF or a PCA-ICA, components are selected by their **skewness**.



Let's consider a `.tif` sequence matrix, of shape (T, N_x, N_y), that is flattened into a matrix M of shape $(T, N_x \times N_y)$. Let's note $N := N_x \times N_y$. This matrix can be the raw calcium imaging `.tif` sequence, or an output of the skeletonization module.

From left to right: a frame of the input skeletonized sequence; a component; another component.

6.1.1 Dimensionality reduction

The goal of dimensionality reduction here is to transform the (T, N) matrix into a (T_1, N) matrix, with $T_1 \ll T$.

PCA

PCA (Principal Components Analysis) is a dimensionality reduction technique that analyses the p-dimensional points cloud data and finds the axis along which the variance is the highest, keeping them over the ones with a low variance. By omitting the axis with a low variance, we lose an equally small amount of information.

Theory

We want to find the vector u with the highest variance. Let's project our data matrix A onto this vector:

$$\pi_u(M) = Mu$$

The empirical variance v of $\pi_u(A)$ is $\frac{1}{K}\pi_u(M)^T\pi_u(M)$ (K being the number of pixels in A), and the covariance matrix $\text{Cov}(A)$ is $\frac{1}{K}A^TA$. We then have:

$$v = \frac{1}{K}\pi_u(A)^T\pi_u(A) = u^T\left(\frac{1}{K}A^TA\right)u = u^T\text{Cov}(A)u = u^TP^T\Delta Pu = \tilde{u}^T\Delta\tilde{u}$$

with $\Delta = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_K \end{pmatrix}$, matrix of the eigenvalues of $\text{Cov}(A)$.

We can then easily verify that v is then maximum when $\tilde{u}$ is an eigenvector of $\text{Cov}(A)$. In this case, we also have $v = \lambda_i$, which means that **the variance ratio is the same as the eigenvalues ratio**.

Thus, to get the axis of our newly created space, we sort the eigenvalues and successively take them one after another, starting from the highest one, until we reached a sufficient amount of variance.

We uses the sklearn implementation of PCA in our tool. It features the necessary pre-processing (data centering).

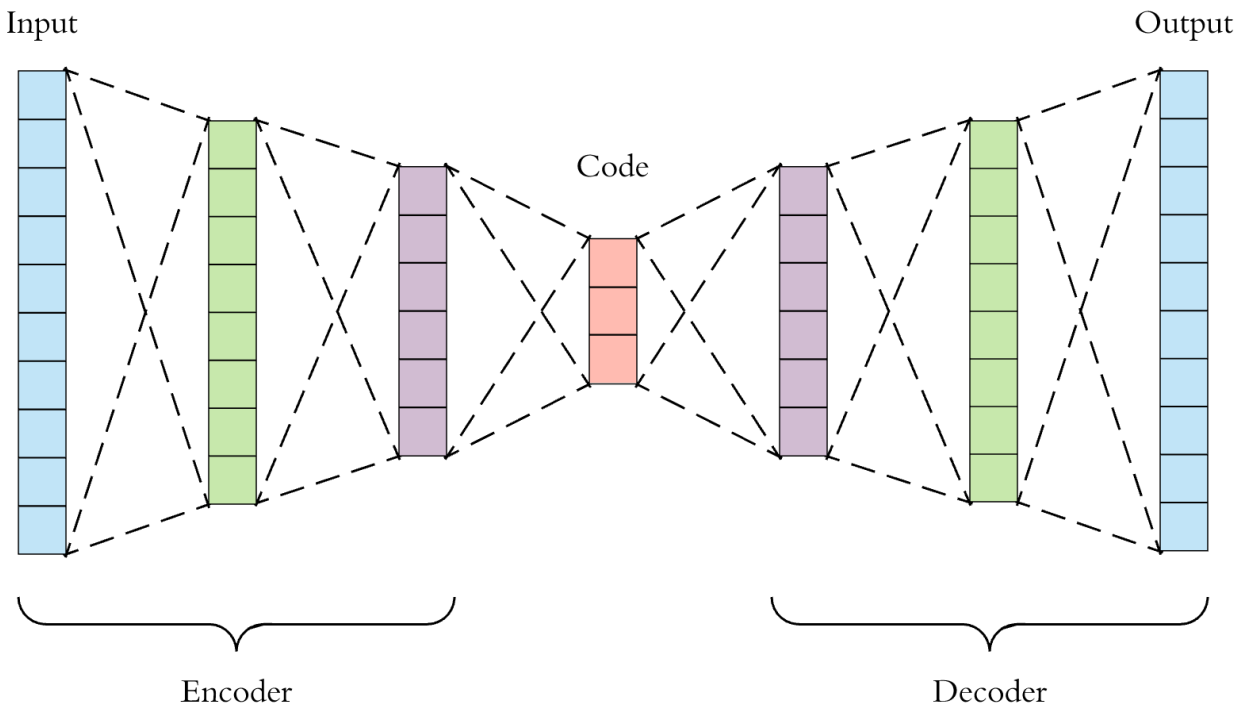
Note: In our tool, we apply PCA to M^T for the sake of simplicity. It doesn't change anything, except the **computation time**.

Indeed, PCA needs to compute the covariance matrix of the data, which is $\text{Cov}(A) = A^TA$, of dimensions (p, p) if A is of shape (n, p). This computational time increases with p.

Autoencoder

Warning: This functionality hasn't been implemented yet.

Autoencoding is a machine learning *self-supervised learning* technique, which can be used as an alternative to PCA in the context of dimensionality reduction. An autoencoder is a neural network composed of an encoder and a decoder, that can be trained to encode (project the data to a lower dimensional latent space) then reconstruct the data as precisely as possible.



In the context of dimensionality reduction, what is important is the **encoding operation**.

The interest of using an autoencoder rather than a PCA is that is is more powerful and can perform more complex operations to encode the data. A linear autoencoder that has 3 layers (encoding, hidden and decoding) with linear activations will simply simulate a PCA; but replacing the linear activations by sigmoids or ReLUs allows the model to perform **non-linear** transformations of the data.

For more information on autoencoders, see[1]; for more information on the difference between PCA and autoencoders, see[2].

6.1.2 Source separation

ICA

The goal of **ICA** (Independent Components Analysis) is to separate a multivariate signal into additive subcomponents, assuming they are statistically independent from each other. It requires a *dimensionality reduction* as a pre-processing: PCA, or autoencoder. Thus, we want to reduce the (T, N) once more, transforming the temporally reduced (T_1, N) shaped matrix into a (T_2, N) shaped matrix, with $T_2 \ll T_1$.

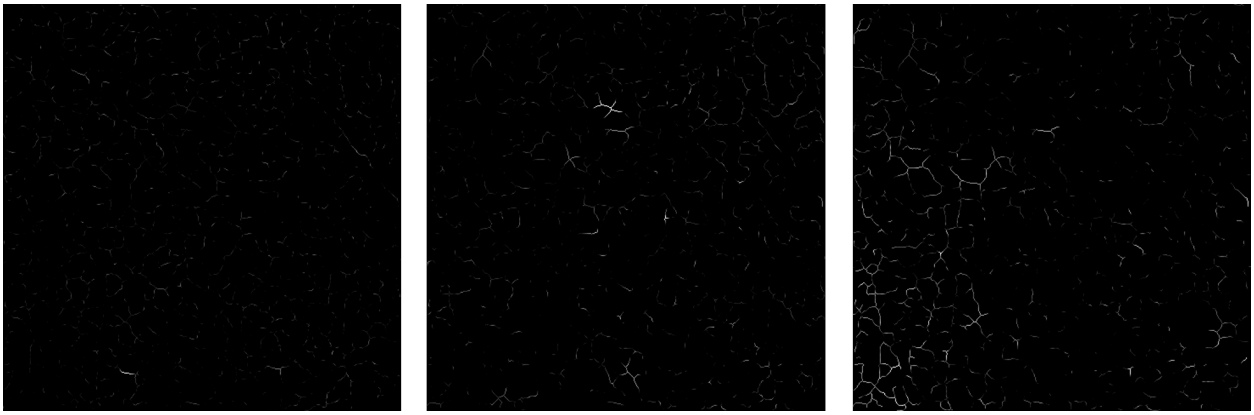
To do so, ICA finds the independent components by maximizing their statistical independence (*e.g.* their non-gaussianity)[3].

We use the sklearn implementation of FastICA in our tool. It features the necessary pre-processing (whitening).

[1] *Building Autoencoders in Keras*, The Keras Blog, https://blog.keras.io/building-autoencoders-in-keras.html

[2] *PCA & Autoencoders: Algorithms Everyone Can Understand*, Thomas Ciha, https://towardsdatascience.com/understanding-pca-autoencoders-algorithms-everyone-can-understand-28ee89b570e2

[3] *Independent Component Analysis: Algorithms and Applications*, Aapo Hyvärinen and Erkki Oja, https://doi.org/10.1016/S0893-6080(00)00026-5



Some ICA components.

Warning: Our implementation of ICA may have some flaws and shouldn't be taken as reference. It is adapted to pyAMNESIA and works well in this context but may have trouble to generalize.

NMF

The goal of **NMF** (Nonnegative Matrix Factorization) is to factorize a matrix V into two matrices H and W, with $V \approx H \times W$ and $W, H \geq 0$. The interest of NMF here is that as since W is nonnegative, its columns can be interpreted as images (the components), and H tells us how to sum up the components in order to reconstruct an approximation to a given frame of the `.tif` sequence. One matrix approximates the elements of V, the other infers some latent structure in the data. Moreover, as said in[4]:

> As the weights in the linear combinations are nonnegative ($H \geq 0$), these basis images can only be summed up to reconstruct each original image. Moreover, the large number of images in the dataset must be reconstructed approximately with only a few basis images, hence the latter should be **localized features** (hence **sparse**) found simultaneously in several images.

We use the sklearn implementation of NMF in our tool.

Some NMF components.

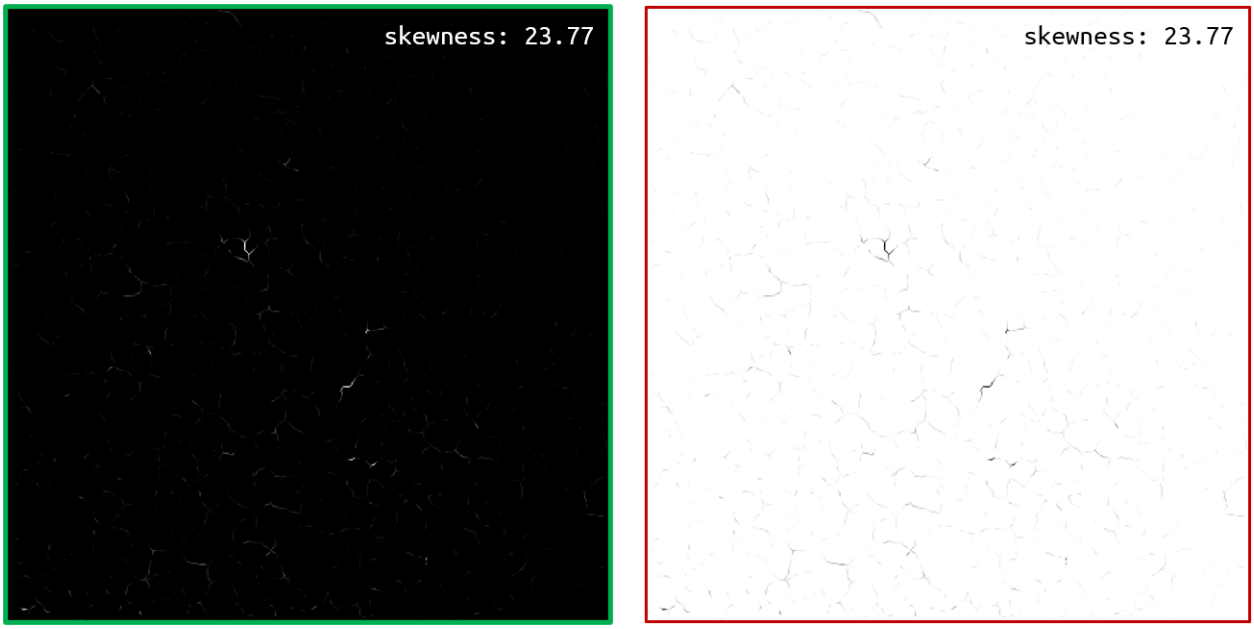
Warning: As the components **are not orthogonal**, some superpositions happen: many pixels belong to several components.

[4] *The Why and How of Nonnegative Matrix Factorization*, Nicolas Gillis, https://arxiv.org/pdf/1401.5226.pdf

6.1.3 Skewness selection

Once we have the components given by the methods described above, we can study their **skewness**, which is a measure of the asymmetry of the probability distribution of a real-valued random variable around its mean. Skewness is useful in two ways:

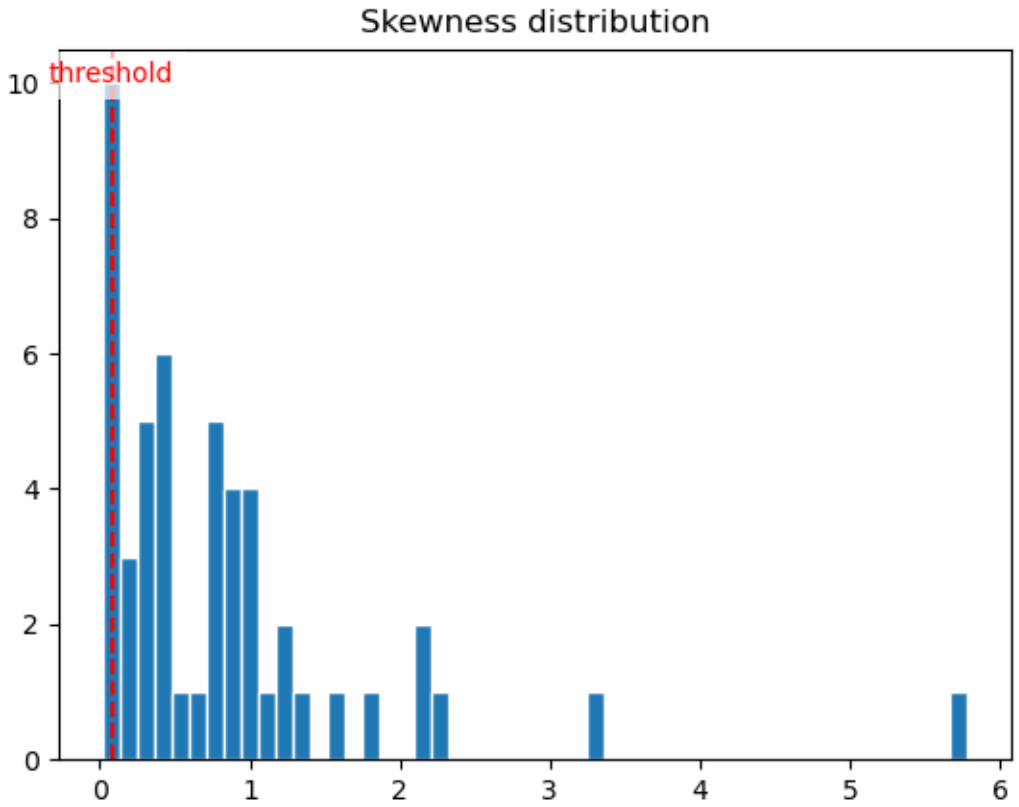
- As a factorization technique such as ICA leaves an ambiguity regarding the **sign of the components**[3], we want to be sure that all of them have the same orientation - a lot of dark pixels and some groups of bright ones. By studying the skewness of each component (*i.e.* of the repartition of the pixel values), one can identify the **negative-skewed** components and **invert them**.



An image and its inverted image, with their respective skewness.

- Among these processed components, some may be **noise**. To remove these components that do not reflect the latent structure of the `.tif` sequence, we can study their skewness. A skewness that is close to 0 means that the pixel intensity repartition is gaussian over the component, and that it is noise. We consequently introduce a `skewness_threshold`, removing every component that do not pass this threshold.

  By looking at the skewness distribution histogram of the components, one can see that there is a peak of low-skewed components near 0. Very often, setting the `skewness_threshold` to `0.1` is a good choice.



Components skewness distribution histogram.

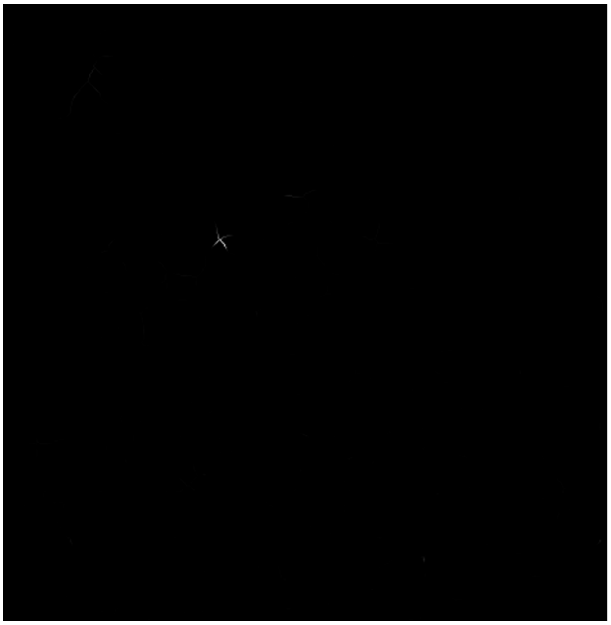
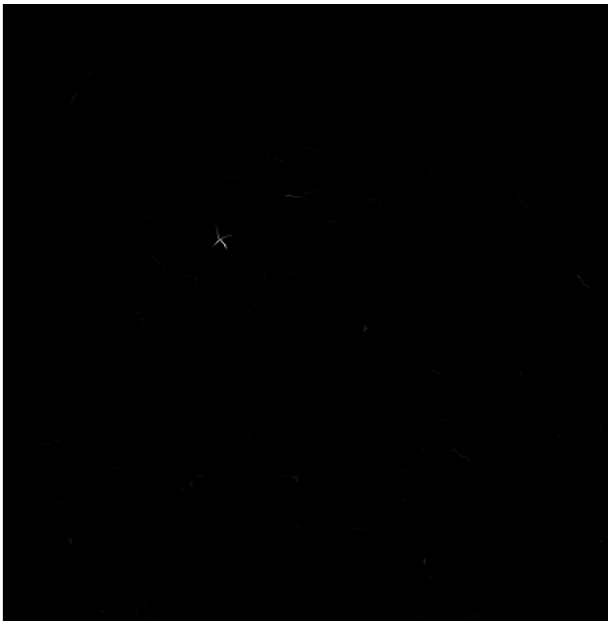
6.1.4 Clipping

Among the final components, some may have some negative pixels. As we want to obtain components that can be interpreted as **images** (*i.e.* nonnegative matrices), we could try **clipping them** by setting to 0 the negative pixels:

$$y = \max(0, x)$$

Note: As the outputs of NMF **are already positive**, clipping is not necessary after a NMF.

Even if there is no mathematical justification to this operation, we observe that a PCA followed by an ICA and a clipping gives nearly **the same results as a NMF**. We thus chose to keep this operation in our tool.

Left: a NMF component; Right: the same component, obtained by PCA, ICA and clipping (better zoomed).

6.2 Parameter selection

6.2.1 Selecting `factorization_method`

- **type:** `str`
- **default:** `'nmf'`

The `factorization_method` parameter selects which method is going to be used for the matrix factorization analysis. The choices are:

- `'nmf'`: perform a Non-negative Matrix Factorization (NMF) over the `.tif` sequence.

Note: When `factorization_method` is set to `'nmf'`, *element_normalization_method* must be set to `None` to make sure the input matrix is nonnegative.

- `'pca'`: perform a Principal Components Analysis (PCA) over the `.tif` sequence, followed by an Independent Components Analysis (ICA);
- `'ae'`: perform a dimensionality reduction of the `.tif` sequence using an autoencoder, followed by an Independent Components Analysis (ICA).

6.2.2 Selecting `nb_comp_intermed`

- **type:** `int`
- **default:** `100`

When choosing the `'ae'` or `'pca'` factorization method, you ask pyAMNESIA to perform a dimensionality reduction and then a source separation. The intermediate number of components specifies the number of dimensions you want the output of the dimensionality reduction to have. The dimensionality reduction can be seen as an operation that has the signature `f : (nb_frames, nb_pixels) -> (nb_comp, nb_pixels)`.

6.2.3 Selecting `nb_comp_final`

- **type:** `int`
- **default:** `50`

This parameter specifies the number of components you want in the final output. If `'ae'` or `'pca'` is chosen for `factorization_method`, it has to be less than `nb_comp_intermed` and will be set to `nb_comp_intermed` if it is not.

6.2.4 Selecting `skewness_threshold`

- **type:** `float`
- **default:** `0.08`

After having computed the final components, you can ask pyAMNESIA to check their skewness (measure of gaussianity). If the skewness of a component is too low, it means that it is noise and we remove it. All components of skewness inferior to `skewness_threshold` will be removed after the factorization.

CHAPTER 7

Results and computation

7.1 Results

Here you can find a detailed list of the content of the results folder of each analysis. You may want to read the **OVERVIEW** page that corresponds to the analysis you want to launch before reading this section (see *Skeleton*, *Clustering* and *Factorization*).

Skeleton (*page*)

- `sequence_projection/`
  - `projected_coordinates.npy`: a file containing the array of components coordinates
  - `projected_traces.npy`: a file containing the array of components traces
  - `projected_mask.png`: the binary skeleton (or 'active') mask
  - `summary_image.png`: the summary_image generated from the movie
- `skeleton_mask/`: a folder containing all the intermediate images obtained during *image skeletonization*:
  - `0_summary_image.png`
  - `1_histogram_equalization.png`
  - `2_smoothing.png`
  - `3_thresholding.png`
  - `4_holes_removing.png`
  - `5_skeletonization.png`
  - `6_skeleton_cleaning.png`
  - `7_skeleton_label.png`

Clustering (*page*)

- `dimensionality_reduction/`

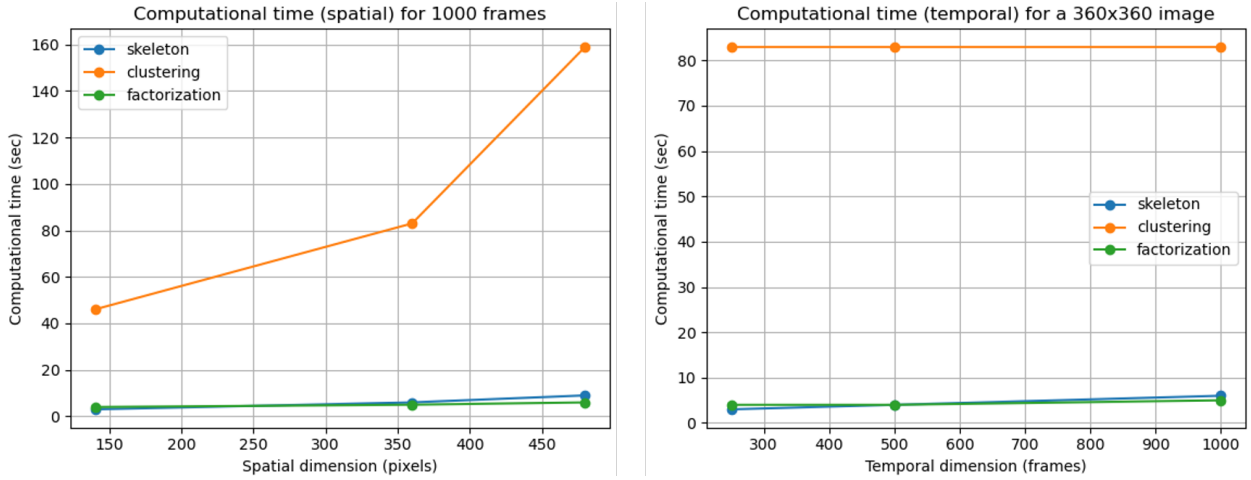
  - `pca_res_*.png`: a figure of a trace before and after PCA
- `hdbscan_clustering/`
  - `cluster_*.png`: a figure containing a cluster mask and traces
  - `hdbscan_*.png`: a scatter plot of traces, projected in a 2D plane
- `sequence_projection/`: the same folder than in skeleton, but whose components are the clusters (see *Consecutive clusterings*)

Factorization (*page*)

- `skewness/`
  - `comp_*_to_*.png`: each output component of the factorization, before the skewness selection, with a red or green outline depending on whether the skewness is high enough to be kept (green) or not (red)
  - `skewness_distribution.png`: histogram distribution of the components' skewnesses
- `components/`
  - `comp_*.png`: each final component of the analysis

7.2 Computational performance

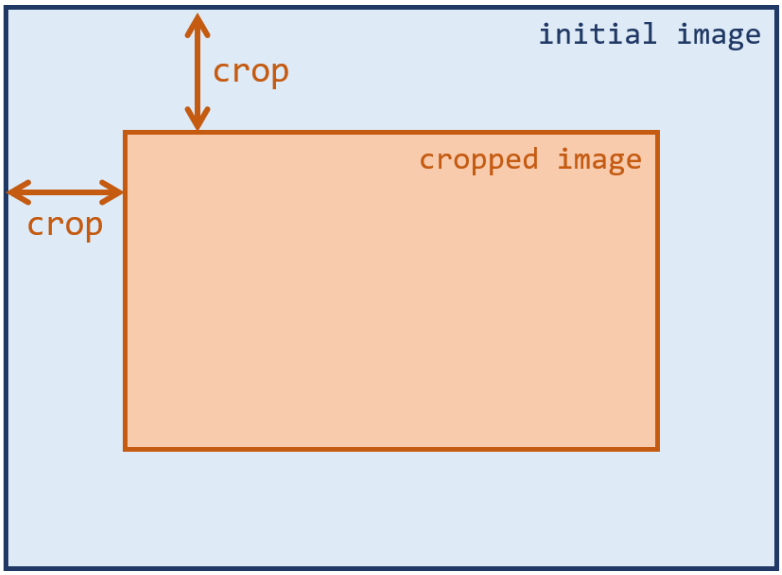
Here are plotted the execution durations of the different modules for some movies. The execution times may vary slightly with the methods and parameters selected. Experiments were conducted on a Intel(R) Core(TM) i7-8565U CPU @1.80GHz.



Computational time of the three modules.

Tip

Depending on which operation you want to perform on the sequence, you can adjust some parameters (*e.g.* `crop`) to increase computational speed. You can also cut your movie into smaller ones (*e.g.* $12500 = 5 \times 2500$ frames).



Cropping process diagram.

CHAPTER 8

Configuration templates

Here we present some **configuration templates** that we used during our experiments. Used on the `.tif` sequence above, they produce good results. In order for you to see what we call "good" results, they are shown above too for each configuration template.

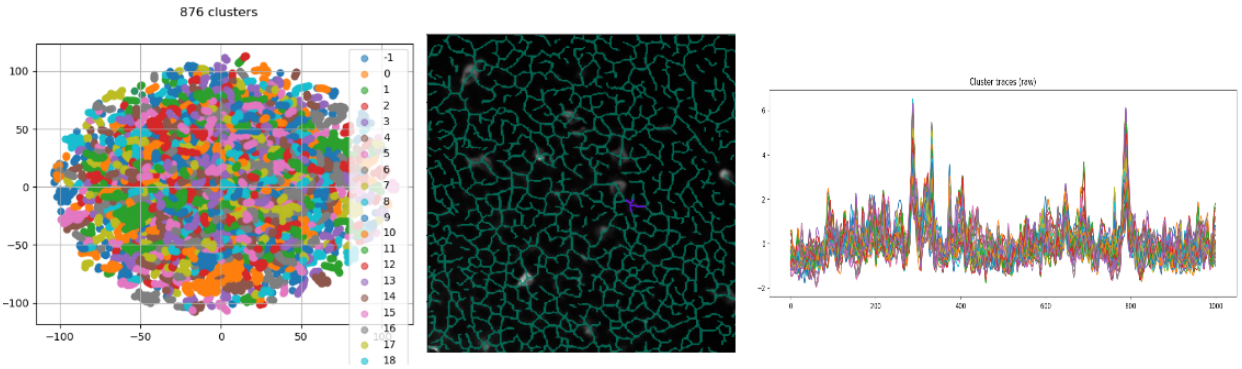
The initial sequence.

The configuration templates can be found in the `pyamnesia/config_templates/` directory.

8.1 Clustering templates

8.1.1 Skeleton pixels clustering

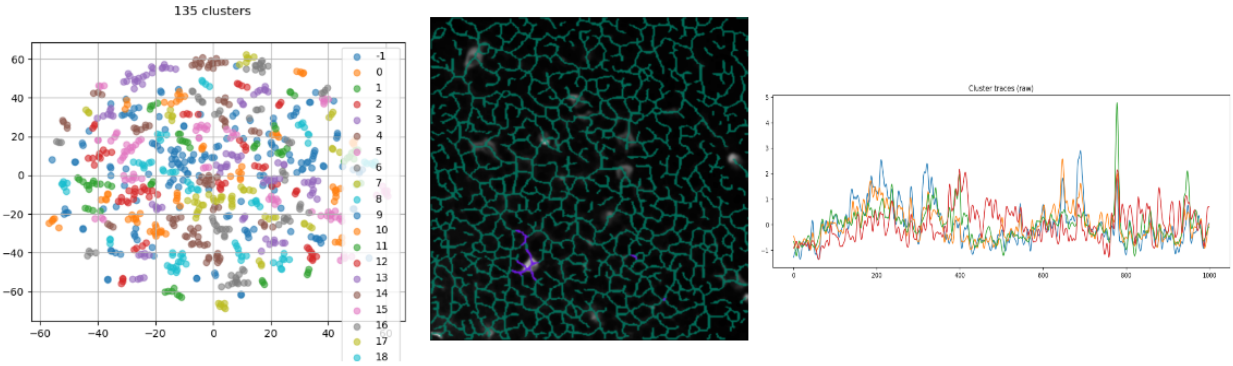
- **file:** `config_clustering_pixel.yaml`
- **context:** clustering the traces of skeleton pixels from the `.tif` sequence.
- **specs:** `projection_substructure` is set to `'skeleton'` and `projection_method` to `'pixel'`. t-SNE perplexity is relatively high, because we expect to find mesoscopic clusters composed of dozens of pixels.
- **results:**



Clustering plot, cluster overlay and cluster traces.

8.1.2 Skeleton branches clustering

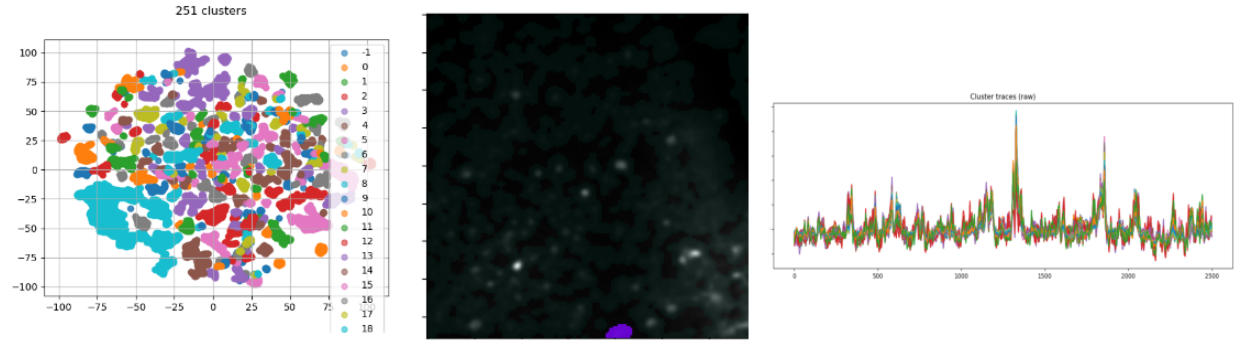
- **file:** `config_clustering_branch.yaml`
- **context:** clustering the traces of skeleton branches (or mesoscopic clusters from the pixel clustering) from the `.tif` sequence.
- **specs:** `projection_substructure` is set to `'skeleton'` and `projection_method` to `'branch'`. t-SNE perplexity is lower, because we expect to find macroscopic clusters composed of a few branches (or mesoscopic clusters).
- **results:**



Clustering plot, cluster overlay and cluster traces.

8.1.3 Active pixels clustering

- **file:** `config_clustering_active.yaml`
- **context:** clustering the traces of active pixels from the `.tif` sequence.
- **specs:** `projection_substructure` is set to `'active'` and `projection_method` to `'pixel'`. t-SNE perplexity is lower, because we expect to find macroscopic clusters composed of a few branches (or mesoscopic clusters).
- **results:**

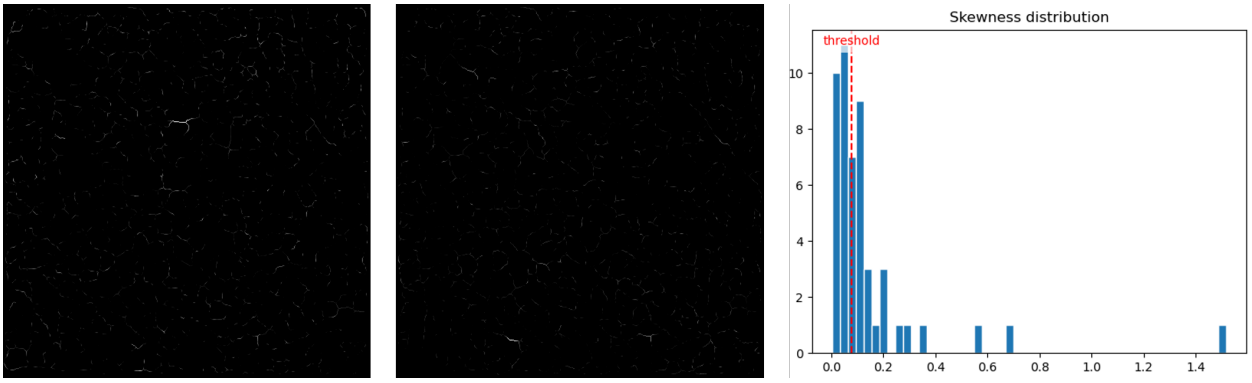


Clustering plot, cluster overlay and cluster traces.

8.2 Factorization templates

8.2.1 NMF

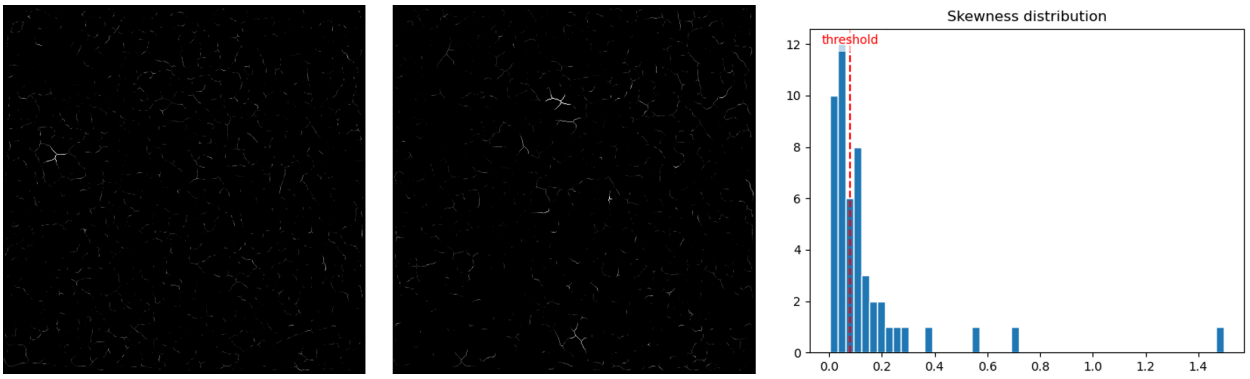
- **file:** `config_factorization_nmf.yaml`
- **context:** performing a NMF on the `.tif` sequence.
- **specs:** `element_normalization_method` is set to `null` so that the input of the NMF is nonnegative. The number of components is set to `50` but can be adapted.
- **results:**



Two components and the skewness histogram (open image in new tab for a better view).

8.2.2 PCA and ICA

- **file:** `config_factorization_pca_ica.yaml`
- **context:** performing a PCA and an ICA on the `.tif` sequence.
- **specs:** `element_normalization_method` is set to `'z-score'`. The number of components is set to `50` but can be adapted.
- **results:**



Two components and the skewness histogram (open image in new tab for a better view).

CHAPTER 9

Robustness

pyAMNESIA aims at analyzing many calcium images to produce statistics. Among all the criteria that could quantify its relevancy, the most important is without any doubt:

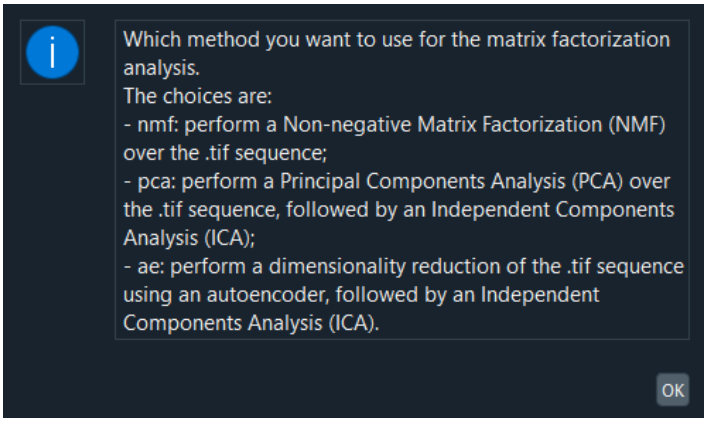
Important: Does pyAMNESIA perform great on all calcium imaging sequences?

The following sections explain how one can get the best result possible while using the tool.

9.1 Parameter tuning

In order for you to choose the best set of parameters for each analysis scenario, we provide in this documentation a description of every tunable parameter of the tool. To read these descriptions, please see the **Parameter selection** section of the **OVERVIEW** page that corresponds to the analysis you are running.

These descriptions can also be found in CICADA, when clicking on the ? icons.



A help window in CICADA.

We also provide some *configuration templates* that you can use for your analysis.

9.2 Consecutive clusterings

There are different scales of components, that we could mistakenly call micro, meso and macroscopic.

- **microscopic:** a pixel;
- **mesoscopic:** a neurite, a morphological branch;
- **macroscopic:** a set of several branches, the whole sequence.

The clustering algorithm finds groups of coactive components. When the `projection_method` parameter is set to `'pixel'`, the algorithm forms groups of pixels, and thus forms mesoscopic components from microscopic components. When the `projection_method` parameter is set to `'branch'`, the algorithm forms groups of branches, and thus forms macroscopic components from mesoscopic components.

In the willingness to compare both methods, we developed a functionality of **consecutive clusterings**. It allows the tool to run several clusterings in a row, every time increasing the clusters scale. It simply works by taking the average trace of each cluster, and considering the clusters as input components of the next clustering.

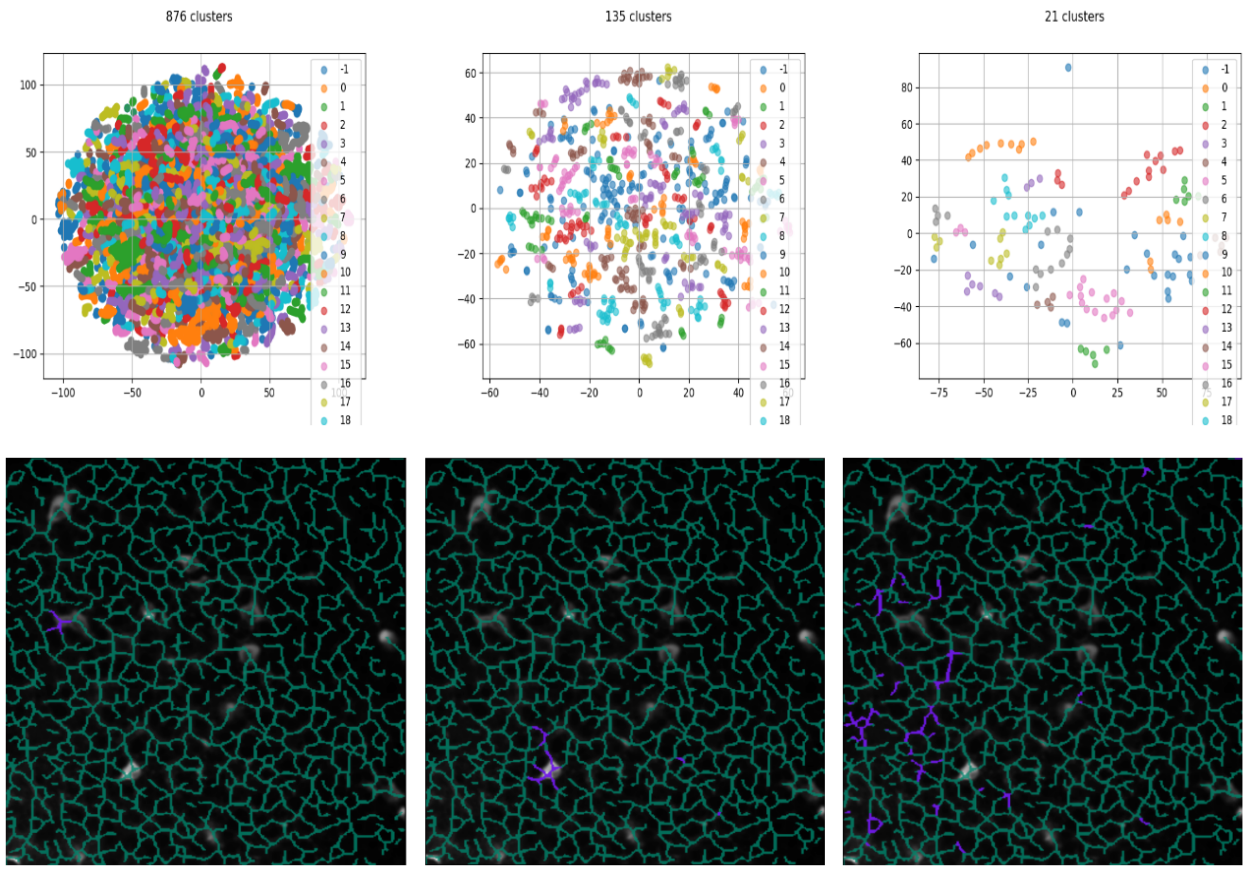
In the CLI and the GUI, we only need to pass the `Clustering_*/` folder as an input to the `clustering` module.

Warning: As the current module folder is created before opening the file dialog box, you must select the **penultimate** `Clustering_*/` folder, the last one being empty at this time.

Note: Depending on the scale, you are expected to change the t-SNE and HDBSCAN parameters values from one clustering to another.

For example, if your first clustering concerns skeleton pixels, you might want to find mesoscopic clusters, of a dozen of pixels. Then for your second clustering, you might want to find macroscopic clusters. The t-SNE and HDBSCAN parameters will then be expected to be lower (cf *Clustering parameter selection*).

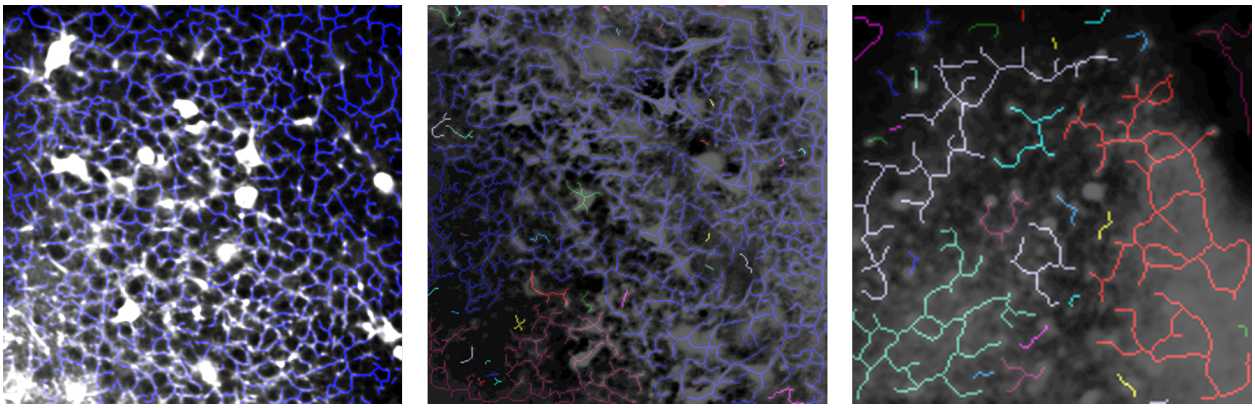
Here is an example of 3 consecutive clusterings:



Consecutive clusters: from microscopic to macroscopic. Top: The 2D projected traces. Bottom: A cluster overlay (purple) on the skeleton (green).

9.3 Skip skeletonization

There *are* some sequences that cannot be well processed by the tool. For instance:



Some skeletonization results.

- The image at the left comes from a sequence that is quite optimal and therefore has been well-skeletonized.

- The sequence from which the image in the middle comes presents some **flickering** and has a lower resolution; yet its skeletonization is good.
- The image at the right, however, has a resolution that is **too low**. The tool has trouble determining the morphological structure of the data and thus produces an irrelevant skeletonization.

From this and other experiments, we can extract two main cases that could cause a **bad skeletonization**:

1. a **flickering** of the sequence;
2. a **low resolution** of the sequence.

In any of these cases, you can skip the skeletonization, directly performing a clustering or a matrix factorization.

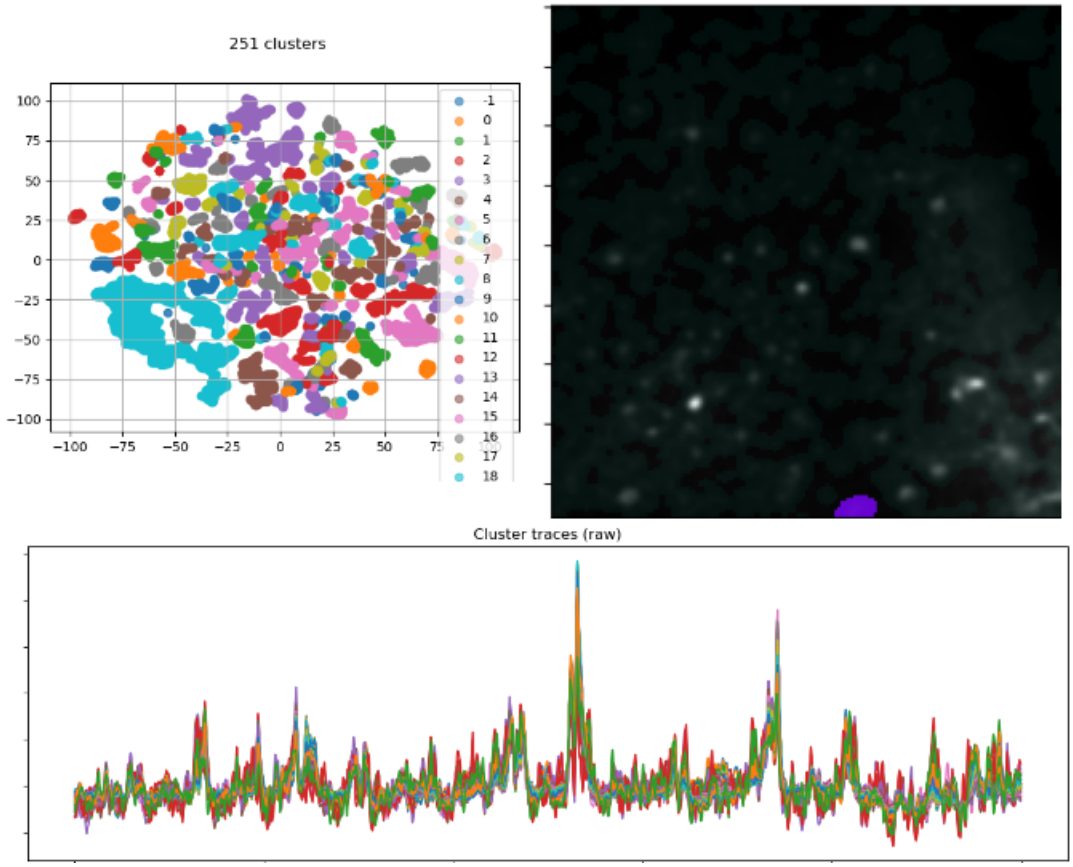
Warning: This may increase the **computational time** of the analysis, as the skeletonization module also conducts a **dimensionality reduction** of the input data.

Note: Actually, we do not "skip" the skeletonization module. For computational reasons, we decide to keep the beginning of the *Image skeletonization* pipeline, only skipping the two last steps (**skeletonization** and **skeleton cleaning**). This allows to extract the binary image that represents the pixels that are active during the movie. As a consequence, the background and noise pixels are removed from the components to analyze, which reduces the dimension.

In order to do so, we set the `projection_substructure` parameter to `'active'` (cf *Skeleton parameter selection*).

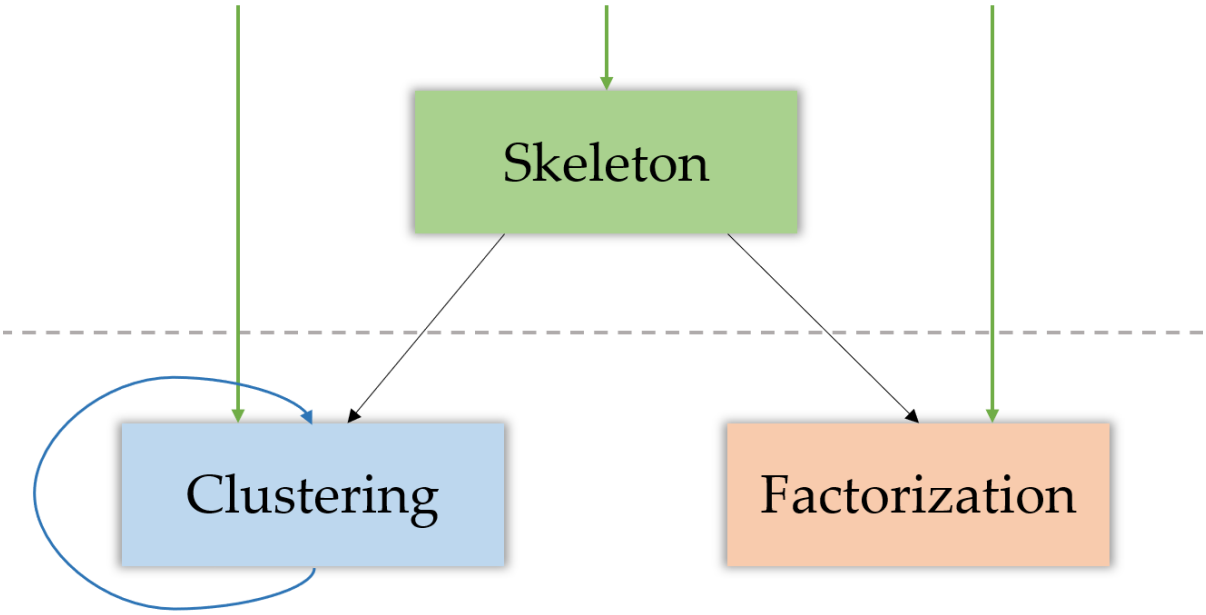
A movie projected on the active pixels.

The results are in the `Skeleton_*/` folder, in the same format as when we don't "skip" the skeletonization. Then, they can be used by the clustering or the factorization module.



Results of clustering on the active sequence. Left: The 2D projected traces. Right: A cluster overlay (purple) on the summary image. Bottom: The traces of the pixels belonging to the overlaid cluster.

Finally, here is the structure of the approach, including the optional consecutive clusterings and skipping of skeletonization.



CHAPTER 10

Support

10.1 Contributing

You can read the contributing guidelines here.

10.2 License

10.3 Contact

If you have any question concerning the tool, feel free to contact us:

Cossart interns

- Théo Dumont: *theo.dumont[at]mines-paristech.fr*
- Tom Szwagier: *tom.szwagier[at]mines-paristech.fr*

Cossart team

- Robin Dard: *robin.dard[at]inserm.fr*