HNCcorr

Release 2019

Quico Spaen, Robert Asin-Acha, Dorit S. Hochbaum

Jul 03, 2019

CONTENTS:

This implementation of the HNCcorr algorithm identifies cell bodies in two-photon calcium imaging movies. HNCcorr is described in detail in our eNeuro paper.

The code consists of modular components that can be configured to your liking.

If you use HNCcorr for academic purposes, please cite the following paper:

> Q Spaen, R Asín-Achá, SN Chettih, M Minderer, C Harvey, and DS Hochbaum (2019). HNCcorr: A novel combinatorial approach for cell identification in calcium-imaging movies. eNeuro, 6(2).

CHAPTER

ONE

QUICKSTART

1.1 Movies

It all starts from a calcium-imaging movie. If your movie is stored as a numpy array, you can directly construct a *Movie* object:

```
from hnccorr import Movie
from hnccorr.example import load_example_data

movie = Movie(
    "Example movie",  # Name of the movie
    load_example_data()  # Downloads sample Neurofinder dataset as a numpy array.
)
```

If the movie is stored in tiff files, you can construct the *Movie* object with *from_tiff_images()*. This method loads a set of tiff files, each containing one frame, from a folder. The filenames should contain the frame numbers with zero-padding: 00001.tiff, 00002.tiff, 00003.tiff, etc. With the `memmap` parameter you can specify whether the movie should be loaded into memory or a memory-mapped disk file should be created in the same folder. With the `subsample`, you can specify how many frames should be subsampled into a single frame. By default, every 10 frames are averaged into a single frame.

Caution: It is important that the tiff filenames are padded with zeros, such that they sort in the correct order.

1.2 Configuration

Before we construct the *HNCcorr* object, we need to configure the algorithm with an *HNCcorrConfig* object. The algorithm will perform better if some of the parameters are adjusted per dataset. For example, in the following example we adjust the minimum cell size for the postprocessor:

```
from hnccorr import HNCcorrConfig

config = HNCcorrConfig(postprocessor_min_cell_size=80)
```

The default value is used for any parameter that is not explicitly specified in the configuration.

The adjustable parameters and their default values are:

- **postprocessor_min_cell_size** = 40: Lower bound on pixel count of a cell.
- **postprocessor_preferred_cell_size** = 80: Pixel count of a typical cell.

- **postprocessor_max_cell_size** = 200: Upper bound on pixel count of a cell.
- **patch_size** = 31: Size in pixel of each dimension of the patch.
- **positive_seed_radius** = 0: Radius of the positive seed square / superpixel.
- **negative_seed_circle_radius** = 10: Radius in pixels of the circle with negative seeds.
- **seeder_mask_size** = 3: Width in pixels of the region used by the seeder to compute the average correlation between a pixel and its neighbors.
- **seeder_grid_size (int)**: Size of grid bloc per dimension. Seeder maintains only the best candidate pixel for each grid block.
- **seeder_exclusion_padding** = 4: Distance for excluding additional pixels surrounding segmented cells.
- **percentage_of_seeds** = 0.40: Fraction of candidate seeds to evaluate.
- **negative_seed_circle_count** = 10: Number of negative seeds.
- **gaussian_similarity_alpha** = 1: Decay factor in gaussian similarity function.
- **sparse_computation_grid_distance** = 1 / 35.0 : 1 / grid_resolution. Width of each block in sparse computation.
- **sparse_computation_dimension** = 3: Dimension of the low-dimensional space in sparse computation.

The parameters at the top of the list are more likely to need adjust than those at the bottom of the list.

1.3 Cell identification

Next, we construct the *HNCcorr* object from its configuration:

```
H = HNCcorr.from_config(config)
```

Note that the `config` parameter is optional. If no configuration is specified, the default values for *HNCcorr* are used.

We can then use *HNCcorr* to segment the movie and extract the resulting segmentations:

```
H.segment(movie)

H.segmentations  # List of identified cells
H.segmentations_to_list()  # Export list of cells (for Neurofinder)
```

CHAPTER

TWO

API DOCUMENTATION

Here you can find the details of the various HNCcorr components.

This HNCcorr implementation has the following components:

- **Candidate** - Contains the logic for segmenting a single cell.
- **Embedding** - Provides the feature vector of each pixel.
- **GraphConstructor** - Constructs the similarity graph.
- **HNC** - Solves Hochbaum's Normalized Cut (HNC) on a given similarity graph.
- **HNCcorr** - Provides the overal logic for segmenting all cells in a movie.
- **Movie** - Provides access to the data of a calcium imaging movie.
- **Patch** - Represents a square subregion of a movie (used for segmenting a cell).
- **Positive / negative seed selector** – Selects positive or negative seed pixels in a patch.
- **Post-processor** - Selects the best segmentation (if any) for a cell.
- **Seeder** - Generates candidate cell locations.
- **Segmentation** - Represents a candidate segmentation of a cell.

2.1 Submodules

2.1.1 hnccorr.base module

Base components of HNCcorr.

class hnccorr.base.**Candidate**(*center_seed*, *hnccorr*)

Bases: object

Encapsulates the logic for segmenting a single cell candidate / seed.

Variables

- **best_segmentation** (Segmentation) – Segmentation of a cell's spatial footprint as selected by the postprocessor.
- **center_seed** (*tuple*) – Seed pixel coordinates.
- **clean_segmentations** (*list[Segmentation]*) – List of segmentation after calling *clean()* on each segmentation.
- **segmentations** (*list[Segmentation]*) – List of segmentations returned by HNC.

- **_hnccorr** (HNCcorr) – HNCcorr object.

__eq__(*other*)
Compare Candidate object.

__init__(*center_seed*, *hnccorr*)
Initialize Candidate object.

segment()
Segment candidate cell and return footprint (if any).

Encapsulates the procedure for segmenting a single cell candidate. It determines the seeds, constructs the similarity graph, and solves the HNC clustering problem for all values of the trade-off parameter lambda. The postprocessor selects the best segmentation or determines that no cell is found.

Returns Best segmentation or None if no cell is found.

Return type *Segmentation* or None

class hnccorr.base.**HNCcorr**(*seeder*, *postprocessor*, *segmentor*, *positive_seed_selector*, *negative_seed_selector*, *graph_constructor*, *candidate_class*, *patch_class*, *embedding_class*, *patch_size*)
Bases: object

Implementation of the HNCcorr algorithm.

This class specifies all components of the algoritm and defines the procedure for segmenting the movie. How each candidate seed / location is evaluated is specified in the Candidate class.

References

Q Spaen, R Asín-Achá, SN Chettih, M Minderer, C Harvey, and DS Hochbaum (2019). HNCcorr: A Novel Combinatorial Approach for Cell Identification in Calcium-Imaging Movies. eNeuro, 6(2).

__init__(*seeder*, *postprocessor*, *segmentor*, *positive_seed_selector*, *negative_seed_selector*, *graph_constructor*, *candidate_class*, *patch_class*, *embedding_class*, *patch_size*)
Initalizes HNCcorr object.

classmethod from_config(*config=None*)
Initializes HNCcorr from an HNCcorrConfig object.

Provides a simple way to initialize an HNCcorr object from a configuration. Default components are used, and parameters are taken from the input configuration or inferred from the default configuration if not specified.

Parameters **config** (HNCcorrConfig) – HNCcorrConfig object with modified configuration. Parameters that are not explicitly specified in the *config* object are taken from the default configuration DEFAULT_CONFIGURATION as defined in the *hnccorr.config* module.

Returns Initialized HNCcorr object as parametrized by the configuration.

Return type *HNCcorr*

segment(*movie*)
Applies the HNCcorr algorithm to identify cells in a calcium-imaging movie.

Identifies cells the spatial footprints of cells in a calcium imaging movie. Cells are identified based on a set of candidate locations identified by the seeder. If a cell is found, the pixels in the spatial footprint are excluded as seeds for future segmentations. This prevents that a cell is segmented more than once. Although segmented pixels cannot seed a new segmentation, they may be segmented again.

Identified cells are accessible through the *segmentations* attribute.

Returns Reference to itself.

segmentations_to_list()
Exports segmentations to a list of dictionaries.

Each dictionary in the list corresponds to the footprint of a cell. Each dictionary contains the key *coordinates* containing a list of pixel coordinates. Each pixel coordinate is a tuple with the zero-indexed coordinates of the pixel. Pixels are indexed like matrix coordinates.

Returns list[dict[tuple]]: List of cell coordinates.

class hnccorr.base.**HNCcorrConfig**(***entries*)
Bases: object

Configuration class for HNCcorr algorithm.

Enables tweaking the parameters of HNCcorr when used with the default components. Configurations are modular and can be combined using the addition operation.

Each parameter is accessible as an attribute when specified.

Variables

- **seeder_mask_size** (*int*) – Width in pixels of the region used by the seeder to compute the average correlation between a pixel and its neighbors.
- **seeder_exclusion_padding** (*int*) – Distance for excluding additional pixels surrounding segmented cells.
- **seeder_grid_size** (*int*) – Size of grid bloc per dimension. Seeder maintains only the best candidate pixel for each grid block.
- **percentage_of_seeds** (*float[0, 1]*) – Fraction of candidate seeds to evaluate.
- **postprocessor_min_cell_size** (*int*) – Lower bound on pixel count of a cell.
- **postprocessor_max_cell_size** (*int*) – Upper bound on pixel count of a cell.
- **postprocessor_preferred_cell_size** (*int*) – Pixel count of a typical cell.
- **positive_seed_radius** (*int*) – Radius of the positive seed square / superpixel.
- **negative_seed_circle_radius** (*int*) – Radius in pixels of the circle with negative seeds.
- **negative_seed_circle_count** (*int*) – Number of negative seeds.
- **gaussian_similarity_alpha** (*alpha*) – Decay factor in gaussian similarity function.
- **sparse_computation_grid_distance** (*float*) – 1 / grid_resolution. Width of each block in sparse computation.
- **sparse_computation_dimension** (*int*) – Dimension of the low-dimensional space in sparse computation.
- **patch_size** (*int*) – Size in pixel of each dimension of the patch.
- **_entries** (*dict*) – Dict with parameter keys and values. Each parameter value (when defined) is also accessible as an attribute.

__add__(*other*)
Combines two configurations and returns a new one.

If parameters are defined in both configurations, then *other* takes precedence.

Parameters **other** (HNCcorrConfig) – Another configuration object.

Returns Configuration with combined parameter sets.

Return type *HNCcorrConfig*

Raises **TypeError** – When other is not an instance of HNCcorrConfig.

__init__(***entries*)
Initializes HNCcorrConfig object.

2.1.2 hnccorr.graph module

HNCcorr components related to the similarity graph.

class hnccorr.graph.**CorrelationEmbedding**(*patch*)
Bases: object

Computes correlation feature vector for each pixel.

Embedding provides a representation of a pixel in terms of feature vector. The feature vector for the CorrelationEmbedding is a vector of pairwise correlations to each (or some) pixel in the patch.

If the correlation is not defined due to a pixel with zero variance, then the corelation is set to zero.

Variables **embedding** (*np.array*) – (D, N_1, N_2, ..) array of pairwise correlations, where D is the dimension of the embedding and N_1, N_2, .. are the pixel shape of the patch.

__init__(*patch*)
Initializes a CorrelationEmbedding object.

See class description for details.

Parameters **patch** (Patch) – Subregion of movie for which the correlation embedding is computed.

get_vector(*pixel*)
Retrieve feature vector of pixel.

Parameters **pixel** (*tuple*) – Coordinate of pixel.

Returns Feature vector of pixel.

Return type np.array

class hnccorr.graph.**GraphConstructor**(*edge_selector*, *weight_function*)
Bases: object

Graph constructor over a set of pixels.

Constructs a similarity graph over the set of pixels in a patch. Edges are selected by an edge_selector and the similarity weight associated with each edge is computed with the weight_function. Edge weights are stored under the attribute weight.

A directed graph is used for efficiency. That is, arcs (i,j) and (j,i) are used to represent edge [i,j].

Variables

- **_edge_selector** (*EdgeSelector*) – Object that constructs the edge set of the graph.
- **_weight_function** (*function*) – Function that computes the edge weight between two pixels. The function should take as input two 1-dimensional numpy arrays, representing the feature vectors of the two pixels. The function should return a float between 0 and 1.

__init__(*edge_selector*, *weight_function*)
Initializes a graph constructor.

construct(*patch*, *embedding*)
Constructs similarity graph for a given patch.

See class description.

Parameters

- **patch** (Patch) – Defines subregion and pixel set for the graph.
- **embedding** (CorrelationEmbedding) – Provides feature vectors associated with each pixel in the patch.

Returns Similarity graph over pixels in patch.

Return type nx.DiGraph

class hnccorr.graph.**SparseComputationEmbeddingWrapper**(*dim_low*, *distance*, *dimension_reducer=None*)
Bases: object

Wrapper for SparseComputation that accepts an embedding.

Variables **_sc** (*SparseComputation*) – SparseComputation object.

__init__(*dim_low*, *distance*, *dimension_reducer=None*)
Initializes a SparseComputationEmbeddingWrapper instance.

Parameters

- **dim_low** (*int*) – Dimension of the low-dimensional space in sparse computation.
- **distance** (*float*) – 1 / grid_resolution. Defines the size of the grid blocks in sparse computation.
- **dimension_reducer** (*DimReducer*) – Provides dimension reduction for sparse computation. By default, approximate principle component analysis is used.

Returns SparseComputationEmbeddingWrapper

select_edges(*embedding*)
Selects relevant pairwise similarities with sparse computation.

Determines the set of relevant pairwise similarities based on the sparse computation algorithm. See sparse computation for details. Pixel coordinates are with respect to the index of the embedding.

Parameters **embedding** (CorrelationEmbedding) – Embedding of pixels into feature vectors.

Returns List of relevant pixel pairs.

Return type list(tuple)

hnccorr.graph.**exponential_distance_decay**(*feature_vec1*, *feature_vec2*, *alpha*)
Computes `exp(- alpha / n || x_1 - x_2 ||^2_2)` for x_1, x_2 in R^n.

2.1.3 hnccorr.movie module

Components for calcium-imaging movies in HNCcorr.

class hnccorr.movie.**Movie**(*name*, *data*)
Bases: object

Calcium imaging movie class.

Data is stored in an in-memory numpy array. Class supports both 2- and 3- dimensional movies.

Variables

- **name** (*str*) – Name of the experiment.
- **_data** (*np.array*) – Fluorescence data. Array has size T x N1 x N2. T is the number of frame (num_frames), N1 and N2 are the number of pixels in the first and second dimension respectively.
- **_data_size** (*tuple*) – Size of array _data.

__getitem__(*key*)

Provides direct access to the movie data.

Movie is stored in array with shape (T, N_1, N_2, ...), where T is the number of frames in the movie. N_1, N_2, ... are the number of pixels in the first dimension, second dimension, etc.

Parameters **key** (*tuple*) – Valid index for a numpy array.

Returns np.array

static _get_tiff_images_and_size(*image_dir*, *num_images*)

Provides a sorted list of images and computes the required array size.

Data is assumed to be stored in 16-bit unsigned integers. Frame numbers are assumed to be padded with zeros: 00000, 00001, 00002, etc. This is required such that Python sorts the images correctly. Frame numbers can start from 0, 1, or any other number. Files must have the extension `.tiff`.

Parameters

- **image_dir** (*str*) – Path of image folder.
- **num_images** (*int*) – Number of images in the folder.

Returns Tuple of the list of images and the array size.

Return type tuple[List[Str], tuple]

static _read_images(*images*, *output_array*, *subsampler*)

Loads images and copies them into the provided array.

Parameters

- **images** (*list[Str]*) – Sorted list image paths.
- **output_array** (*np.array like*) – T x N_1 x N_2 array-like object into which images should be loaded. T must equal the number of images in *images*. Each image should be of size N_1 x N_2.
- **subsampler** –

Returns The input array *array*.

Return type np.array like

extract_valid_pixels(*pixels*)

Returns subset of pixels that are valid coordinates for the movie.

classmethod from_tiff_images(*name*, *image_dir*, *num_images*, *memmap=False*, *subsample=10*)

Loads tiff images into a numpy array.

Data is assumed to be stored in 16-bit unsigned integers. Frame numbers are assumed to be padded with zeros: 00000, 00001, 00002, etc. This is required such that Python sorts the images correctly. Frame numbers can start from 0, 1, or any other number. Files must have the extension `.tiff`.

If memmap is True, the data is not loaded into memory bot a memory mapped file on disk is used. The file is named `$name.npy` and is placed in the *image_dir* folder.

Parameters

- **name** (*str*) – Movie name.
- **image_dir** (*str*) – Path of image folder.
- **num_images** (*int*) – Number of images in the folder.
- **memmap** (*bool*) – If True, a memory-mapped file is used. (*Default: False*)
- **subsample** (*int*) – Number of frames to average into a single frame.

Returns Movie created from image files.

Return type *Movie*

is_valid_pixel_coordinate(*coordinate*)
Checks if coordinate is a coordinate for a pixel in the movie.

property num_dimensions
Dimension of the movie (excludes time dimension).

property num_frames
Number of frames in the movie.

property num_pixels
Number of pixels in the movie.

property pixel_shape
Resolution of the movie in pixels.

class hnccorr.movie.**Patch**(*movie*, *center_seed*, *patch_size*)
Bases: object

Square subregion of Movie.

Patch limits the data used for the segmentation of a potential cell. Given a center seed pixel, Patch defines a square subregion centered on the seed pixel with width patch_size. If the square extends outside the movie boundaries, then the subregion is shifted such that it stays within the movie boundaries.

The patch also provides an alternative coordinate system with respect to the top left pixel of the patch. This pixel is the zero coordinate for the patch coordinate system. The coordinate offset is the coordinate of the top left pixel in the movie coordinate system.

Variables

- **_center_seed** (*tuple*) – Seed pixel that marks the potential cell. The pixel is represented as a tuple of coordinates. The coordinates are relative to the movie. The top left pixel of the movie represents zero.
- **_coordinate_offset** (*tuple*) – Movie coordinates of the pixel that represents the zero coordinate in the Patch object. Similar to the Movie, pixels in the Patch are indexed from the top left corner.
- **_data** (*np.array*) – Subset of the Movie data. Only data for the patch is stored.
- **_movie** (Movie) – Movie for which the Patch object is a subregion.
- **_num_dimensions** (*int*) – Dimension of the patch. It matches the dimension of the movie.
- **_patch_size** (*int*) – length of the patch in each dimension. Must be an odd number.

__getitem__(*key*)
Access data for pixels in the patch. Indexed in patch coordinates.

__init__(*movie*, *center_seed*, *patch_size*)
Initializes Patch object.

_compute_coordinate_offset()
Computes the coordinate offset of the patch.

Confirms that the patch falls within the movie boundaries and shifts the patch if necessary. The center seed pixel may not be in the center of the patch if a shift is necessary.

_movie_indices()
Computes the indices of the movie that correspond to the patch.

For a patch with top left pixel (5, 5) and bottom right pixel (9, 9), this method returns `(:, 5:10, 5:10)` which can be used to acccess the data corresponding to the patch in the movie.

enumerate_pixels()
Returns the movie coordinates of the pixels in the patch.

property num_frames
Number of frames in the Movie.

property pixel_shape
Shape of the patch in pixels. Does not not included the time dimension.

to_movie_coordinate(*patch_coordinate*)
Converts a movie coordinate into a patch coordinate.

Parameters **patch_coordinate** (*tuple*) – Coordinates of a pixel in patch coordinate system.

Returns Coordinate of pixel in movie coordinate system.

Return type tuple

to_patch_coordinate(*movie_coordinate*)
Converts a movie coordinate into a patch coordinate.

Parameters **movie_coordinate** (*tuple*) – Coordinates of a pixel in movie coordinate system.

Returns Coordinate of pixel in patch coordinate system.

Return type tuple

class hnccorr.movie.**Subsampler**(*movie_shape*, *subsample_frequency*, *buffer_size=10*)
Bases: object

Subsampler for averaging frames.

Averages *subsample_frequency* into a single frame. Stores averaged frames in a buffer and writes buffer to an output array.

Variables

- **_buffer** (*np.array*) – (b, N_1, N_2) array where the frame averages are compiled.
- **_buffer_frame_count** – (b,) array with the number of frames used in each averaged frame.
- **_buffer_size** (*int*) – Number of averaged frames to store in buffer. Short: b. Default is 10.
- **_buffer_start_index** (*int*) – Index of averaged movie corresponding with first frame in the buffer.
- **_current_index** (*int*) – Index of current frame in buffer.

- **_movie_shape** (*int*) – Shape of input movie.
- **_num_effective_frames** (*int*) – Number of frames in the averaged movie.
- **_subsample_frequency** (*int*) – Number of frames to average into a single frame.

__init__(*movie_shape*, *subsample_frequency*, *buffer_size=10*)
Initializes a subsampler object.

add_frame(*frame*)
Adds frame to average.

Frames should be provided in order of appearance in the movie.

Parameters **frame** (*np.array*) – (N_1, N_2) array with pixel intensities.

Returns None

Raises **ValueError** – If buffer is full.

advance_buffer()
Empties buffer and advances the buffer indices for new frames

property buffer
Provides access to data in buffer. Corrects last buffer for movie length.

property buffer_full
True if buffer is full.

property buffer_indices
Indices in average movie corresponding to current buffer

property output_shape
Shape of average movie array.

2.1.4 hnccorr.postprocessor module

Postprocesser component for selecting the best segmentation in HNCcorr.

class hnccorr.postprocessor.**SizePostprocessor**(*min_size*, *max_size*, *pref_size*)
Bases: object

Selects the best segmentation based on the number of selected pixels.

Discards all segmentations that contain more pixels than _max_size or less pixels then _min_size. If no segmentations remains, no cell was found and None is returned. Otherwise the segmentation is returned that minimizes |sqrt(x) - sqrt(_pref_size)| where x is the number of pixels in the segmentation.

Variables

- **_min_size** (*int*) – Lower bound for the cell size in pixels.
- **_max_size** (*int*) – Upper bound for the cell size in pixels.
- **_pref_size** (*int*) – Preferred cell size in pixels.

__init__(*min_size*, *max_size*, *pref_size*)
Initializes a SizePostprocessor object.

_filter(*segmentations*)
Returns a list of segmentations with size between min_size and max_size.

select (*segmentations*)
Selects the best segmentation based on the number of selected pixels.

See class description for details.

Parameters **segmentations** (*List[Segmentation]*) – List of candidate segmentations.

Returns Best segmentation or None if all are discarded.

Return type *Segmentation* or None

2.1.5 hnccorr.seeds module

Seed related components of HNCcorr.

class hnccorr.seeds.**LocalCorrelationSeeder**(*neighborhood_size*, *keep_fraction*, *padding*, *grid_size*)

Bases: object

Provide seeds based on the correlation of pixels to their local neighborhood.

Seed pixels are selected based on the average correlation of the pixel to its local neighborhood.For each block of *grid_size* by *grid_size* pixels, the pixel with the highest average local correlation is selected. The remaining pixels in each block are discarded. From the remaining pixels, a fraction of *_seed_fraction* pixels, those with the highest average local correlation, are kept and attempted for segmentation.

The local neighborhood of each pixel consist of the pixels in a square of width *_neighborhood_size* centered on the pixels. Pixel coordinates outside the boundary of the movie are ignored.

Variables

- **_current_index** (*int*) – Index of next seed in *_seeds* to return.
- **_excluded_pixels** (*set*) – Set of pixel coordinates to excluded as future seeds.
- **_grid_size** (*int*) – Number of pixels per dimension in a block.
- **_keep_fraction** (*float*) – Percentage of candidate seed pixels to attempt for segmentation. All other candidate seed pixels are discarded.
- **_movie** (Movie) – Movie to segment.
- **_neighborhood_size** (*int*) – Width in pixels of the local neighborhood of a pixel.
- **_padding** (*int*) – L-infinity distance for determining which pixels should be padded to the exclusion set in *exclude_pixels()*.
- **_seeds** (*list[tuple]*) – List of candidate seed coordinates to return.

__init__(*neighborhood_size*, *keep_fraction*, *padding*, *grid_size*)
Initializes a LocalCorrelationSeeder object.

_compute_average_local_correlation(*pixel*, *valid_neighbors*)
Compute average correlation between pixel and neighbors.

_select_best_per_grid_block(*scores*)
Selects pixel with highest score in a block of grid_size pixels per dim.

exclude_pixels(*pixels*)
Excludes pixels from being returned by *next()* method.

All pixels within in the set *pixels* as well as pixels that are within an L- infinity distance of *_padding* from any excluded pixel are excluded as seeds.

Method enables exclusion of pixels in previously segmented cells from serving as new seeds. This may help to prevent repeated segmentation of the cell.

Parameters **pixels** (*set*) – Set of pixel coordinates to exclude.

Returns None

next()
Provides next seed pixel for segmentation.

Returns the movie coordinates of the next available seed pixel for segmentation. Seed pixels that have previously been excluded will be ignored. Returns None when all seeds are exhausted.

Returns Coordinates of next seed pixel. None if no seeds remaining.

Return type tuple or None

reset()
Reinitialize the sequence of seed pixels and empties *_excluded_seeds*.

select_seeds(*movie*)
Identifies candidate seeds in movie.

Initializes list of candidate seeds in the movie. See class description for details. Seeds can be accessed via the *next()* method.

Parameters **movie** (Movie) – Movie object to segment.

Returns None

class hnccorr.seeds.**NegativeSeedSelector**(*radius*, *count*)
Bases: object

Selects negative seed pixels uniformly from a circle around center seed pixel.

Selects *_count* pixels from a circle centered on the center seed pixel with radius *_radius*. The selected pixels are spread uniformly over the circle. Non-integer pixel indices are rounded to the closest (integer) pixel. Currently only 2-dimensional movies are supported.

Variables

- **_radius** (*float*) – L2 distance to center seed.
- **_count** (*int*) – Number of negative seed pixels to select.

select(*center_seed*, *movie*)
Selects negative seed pixels.

Parameters

- **center_seed** (*tuple*) – Center seed pixels.
- **movie** (Movie) – Movie for segmentation.

Returns Set of negative seed pixels. Each pixel is denoted by a tuple.

Return type set

class hnccorr.seeds.**PositiveSeedSelector**(*max_distance*)
Bases: object

Selects positive seed pixels in a square centered on *center_seed*.

Selects all pixels in a square centered on *center_seed* as positive seeds. A pixel is selected if it is within a Chebyshev distance (L-Inf) of *_max_distance* from the center seed pixel.

Variables **_max_distance** (*int*) – Maximum L-Inf distance allowed.

select (*center_seed*, *movie*)
Selects positive seeds.

Parameters

- **center_seed** (*tuple*) – Center seed pixel.
- **movie** (Movie) – Movie for segmentation.

Returns Set of positive seed pixels. Each pixel is denoted by a tuple.

Return type set

2.1.6 hnccorr.segmentation module

HNC and segmentation related components in HNCcorr.

class hnccorr.segmentation.**HncParametricWrapper** (*lower_bound*, *upper_bound*)
Bases: object

Wrapper for solving the Hochbaum Normalized Cut (HNC) problem on a graph.

Given an undirected graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$ for $[i, j] \in E$, the linearized HNC problem is defined as:

$$\min_{\emptyset \subset S \subset V} \sum_{\substack{[i,j] \in E, \\ i \in S, \\ j \in V \setminus S}} w_{ij} - \lambda \sum_{i \in S} d_i,$$

where \$d_i\$ the degree of node $i \in V$ and $\lambda \geq 0$ provides the trade-off between the two objective terms.

See closure package for solution method.

__init__ (*lower_bound*, *upper_bound*)
Initializes HncParametricWrapper object.

static _construct_segmentations (*source_sets*, *breakpoints*)
Constructs a list of segmentations from output HNC.

Each source set and corresponding lambda upper bound is replaced with a Segmentation object where the selection matches the source set and the weight parameter matches the upper bound of the lambda range.

Parameters

- **source_sets** (*list[set]*) – List of source sets for each lambda range.
- **breakpoints** (*list[float]*) – List of upper bounds on the lambda range for which the corresponding source set is optimal.

Returns List of segmentations.

Return type list[*Segmentation*]

solve (*graph*, *pos_seeds*, *neg_seeds*)
Solves an instance of the HNC problem for all values of lambda.

Solves the HNC clustering problem on *graph* for all values of lambda simultaneously. See class description for a definition of HNC.

Parameters

- **graph** (*nx.Graph*) – Directed similarity graph with non-negative edge weights. Edge [i,j] is represented by two directed arcs (i,j) and (j,i). Edge weights must be defined via the attribute *weight*.

- **pos_seeds** (*set*) – Set of nodes in graph that must be part of the cluster.
- **neg_seeds** (*set*) – Set of nodes in graph that must be part of the complement.

Returns List of optimal clusters for each lambda range.

Return type list[*Segmentation*]

> **Caution:** Class modifies graph for performance. Pass a copy to prevent any issues.

class hnccorr.segmentation.**Segmentation**(*selection*, *weight*)

Bases: object

A set of pixels identified by HNC as a potential cell footprint.

Variables

- **selection** (*set*) – Pixels in the spatial footprint. Each pixel is represented as a tuple.
- **weight** (*float*) – Upper bound on the lambda coefficient for which this segmentation is optimal.

__eq__(*other*)

Compares two Segmentation objects.

__init__(*selection*, *weight*)

Initializes a Segmentation object.

clean(*positive_seeds*, *movie_pixel_shape*)

Cleans Segmentation by selecting a connected component and filling holes.

The Segmentation is decomposed into connected components by considering horizontal or vertical adjacent pixels as neighbors. The connected component with the most positive seeds is selected. Any holes in the selected component are added to the selection.

Parameters

- **positive_seeds** (*set*) – Pixels that are contained in the spatial footprint. Each pixel is represented by a tuple.
- **movie_pixel_shape** (*tuple*) – Pixel resolution of the movie.

Returns A new Segmentation object with the same weight.

Return type *Segmentation*

fill_holes(*movie_pixel_shape*)

Fills holes in the selection.

Parameters **movie_pixel_shape** (*tuple*) – Pixel resolution of the movie.

Returns A new Segmentation object with the same weight.

Return type *Segmentation*

select_max_seed_component(*positive_seeds*)

Selects the connected component of selection that contains the most seeds.

The Segmentation is decomposed into connected components by considering horizontal or vertical adjacent pixels as neighbors. The connected component with the most positive seeds is selected.

Parameters **positive_seeds** (*set*) – Pixels that are contained in the spatial footprint. Each pixel is represented by a tuple.

Returns A new Segmentation object with the same weight.

Return type *Segmentation*

2.1.7 hnccorr.utils module

Helper functions for HNCcorr.

hnccorr.utils.**add_offset_set_coordinates**(*iterable*, *offset*)
Adds a fixed offset to all pixel coordinates in a set.

Parameters

- **coordinates** (*set*) – Set of pixel coordinates. Each pixel coordinate is a tuple.
- **offset** (*tuple*) – Offset to add to each pixel coordinate. Tuple should be of the same length as the tuples in *coordinates*.

Returns Set of updated coordinates.

Return type set

Example

```
>>> add_offset_set_coordinates({(5, 2), (4, 7)}, (2, 2))
{(7, 4), (6, 9)}
```

hnccorr.utils.**add_offset_to_coordinate**(*coordinate*, *offset*)
Offsets pixel coordinate by another coordinate.

Parameters

- **coordinate** (*tuple*) – Pixel coordinate to offset.
- **offset** (*tuple*) – Offset to add to coordinate. Must be of the same length.

Example

```
>>> add_offset_to_coordinate((5, 3, 4), (1, -1, 3))
(6, 2, 7)
```

hnccorr.utils.**add_time_index**(*index*)
Inserts a full slice as the first dimension of an index for e.g. numpy.

Parameters **index** (*tuple*) – Index for e.g. numpy array.

Returns New index with additional dimension.

Return type tuple

Example

```
>>> add_time_index((5, :3))
(:, 5, :3)
```

hnccorr.utils.**eight_neighborhood**(*num_dims*, *max_radius*)
Returns all coordinates within a given L-infinity distance of zero.

Includes zero coordinate itself.

Parameters

- **num_dims** (*int*) – Number of dimensions for the coordinates.
- **max_radius** (*int*) – Largest L-infinity distance allowed.

Returns Set of pixel coordinates.

Return type set

Example

```
>>> eight_neighborhood(1, 1)
[(-1,), (0,), (1,)]
>>> eight_neighborhood(2, 1)
[
    (-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0),
    (0, 1), (1, -1), (1, 0), (1, 1)
]
```

hnccorr.utils.**four_neighborhood**(*num_dims*)

Returns all neighboring pixels of zero that differ in at most one coordinate.

Includes zero coordinate itself.

Parameters **num_dims** (*int*) – Number of dimensions for the coordinates.

Returns Set of pixel coordinates.

Return type set

Example

```
>>> four_neighborhood(1)
[(-1,), (0,), (1,)]
>>> eight_neighborhood(2)
[(-1, 0), (0, -1), (0, 0), (0, 1), (1, 0)]
```

hnccorr.utils.**generate_pixels**(*shape*)

Enumerate all pixel coordinates for a movie/patch.

Parameters **shape** (*tuple*) – Shape of movie. Number of pixels in each dimension.

Returns Iterates over all pixels.

Return type Iterator

Example

```
>>> generate_pixels((2,2))
[(0, 0), (0, 1), (1, 0), (1, 1)]
```

hnccorr.utils.**list_images**(*folder*)

Lists and sorts tiff images in a folder.

Images are sorted in ascending order based on filename.

> **Caution:** Filenames are sorted as strings. Note that `200.tiff` is sorted before `5.tiff`. Pad image filenames with zeros to prevent this: `005.tiff`.

Parameters **folder** – folder containing tiff image files.

Returns Sorted list of paths of tiff files in folder.

Return type list

2.2 Module contents

class hnccorr.base.**HNCcorr**(*seeder*, *postprocessor*, *segmentor*, *positive_seed_selector*, *negative_seed_selector*, *graph_constructor*, *candidate_class*, *patch_class*, *embedding_class*, *patch_size*)

Bases: `object`

Implementation of the HNCcorr algorithm.

This class specifies all components of the algoritm and defines the procedure for segmenting the movie. How each candidate seed / location is evaluated is specified in the Candidate class.

References

Q Spaen, R Asín-Achá, SN Chettih, M Minderer, C Harvey, and DS Hochbaum (2019). HNCcorr: A Novel Combinatorial Approach for Cell Identification in Calcium-Imaging Movies. eNeuro, 6(2).

__init__(*seeder*, *postprocessor*, *segmentor*, *positive_seed_selector*, *negative_seed_selector*, *graph_constructor*, *candidate_class*, *patch_class*, *embedding_class*, *patch_size*)

Initalizes HNCcorr object.

classmethod from_config(*config=None*)

Initializes HNCcorr from an HNCcorrConfig object.

Provides a simple way to initialize an HNCcorr object from a configuration. Default components are used, and parameters are taken from the input configuration or inferred from the default configuration if not specified.

Parameters **config** (HNCcorrConfig) – HNCcorrConfig object with modified configuration. Parameters that are not explicitly specified in the *config* object are taken from the default configuration `DEFAULT_CONFIGURATION` as defined in the *hnccorr.config* module.

Returns Initialized HNCcorr object as parametrized by the configuration.

Return type *HNCcorr*

segment(*movie*)

Applies the HNCcorr algorithm to identify cells in a calcium-imaging movie.

Identifies cells the spatial footprints of cells in a calcium imaging movie. Cells are identified based on a set of candidate locations identified by the seeder. If a cell is found, the pixels in the spatial footprint are excluded as seeds for future segmentations. This prevents that a cell is segmented more than once. Although segmented pixels cannot seed a new segmentation, they may be segmented again.

Identified cells are accessible through the *segmentations* attribute.

Returns Reference to itself.

segmentations_to_list()
Exports segmentations to a list of dictionaries.

Each dictionary in the list corresponds to the footprint of a cell. Each dictionary contains the key *coordinates* containing a list of pixel coordinates. Each pixel coordinate is a tuple with the zero-indexed coordinates of the pixel. Pixels are indexed like matrix coordinates.

Returns list[dict[tuple]]: List of cell coordinates.

class hnccorr.base.**HNCcorrConfig**(***entries*)
Bases: object

Configuration class for HNCcorr algorithm.

Enables tweaking the parameters of HNCcorr when used with the default components. Configurations are modular and can be combined using the addition operation.

Each parameter is accessible as an attribute when specified.

Variables

- **seeder_mask_size** (*int*) – Width in pixels of the region used by the seeder to compute the average correlation between a pixel and its neighbors.
- **seeder_exclusion_padding** (*int*) – Distance for excluding additional pixels surrounding segmented cells.
- **seeder_grid_size** (*int*) – Size of grid bloc per dimension. Seeder maintains only the best candidate pixel for each grid block.
- **percentage_of_seeds** (*float[0, 1]*) – Fraction of candidate seeds to evaluate.
- **postprocessor_min_cell_size** (*int*) – Lower bound on pixel count of a cell.
- **postprocessor_max_cell_size** (*int*) – Upper bound on pixel count of a cell.
- **postprocessor_preferred_cell_size** (*int*) – Pixel count of a typical cell.
- **positive_seed_radius** (*int*) – Radius of the positive seed square / superpixel.
- **negative_seed_circle_radius** (*int*) – Radius in pixels of the circle with negative seeds.
- **negative_seed_circle_count** (*int*) – Number of negative seeds.
- **gaussian_similarity_alpha** (*alpha*) – Decay factor in gaussian similarity function.
- **sparse_computation_grid_distance** (*float*) – 1 / grid_resolution. Width of each block in sparse computation.
- **sparse_computation_dimension** (*int*) – Dimension of the low-dimensional space in sparse computation.
- **patch_size** (*int*) – Size in pixel of each dimension of the patch.
- **_entries** (*dict*) – Dict with parameter keys and values. Each parameter value (when defined) is also accessible as an attribute.

__add__(*other*)
Combines two configurations and returns a new one.

If parameters are defined in both configurations, then *other* takes precedence.

Parameters **other** (HNCcorrConfig) – Another configuration object.

Returns Configuration with combined parameter sets.

Return type *HNCcorrConfig*

Raises **TypeError** – When other is not an instance of HNCcorrConfig.

__init__(***entries*)
Initializes HNCcorrConfig object.

class hnccorr.movie.**Movie**(*name*, *data*)
Bases: object

Calcium imaging movie class.

Data is stored in an in-memory numpy array. Class supports both 2- and 3- dimensional movies.

Variables

- **name** (*str*) – Name of the experiment.
- **_data** (*np.array*) – Fluorescence data. Array has size T x N1 x N2. T is the number of frame (num_frames), N1 and N2 are the number of pixels in the first and second dimension respectively.
- **_data_size** (*tuple*) – Size of array _data.

__getitem__(*key*)
Provides direct access to the movie data.

Movie is stored in array with shape (T, N_1, N_2, ...), where T is the number of frames in the movie. N_1, N_2, ... are the number of pixels in the first dimension, second dimension, etc.

Parameters **key** (*tuple*) – Valid index for a numpy array.

Returns np.array

static _get_tiff_images_and_size(*image_dir*, *num_images*)
Provides a sorted list of images and computes the required array size.

Data is assumed to be stored in 16-bit unsigned integers. Frame numbers are assumed to be padded with zeros: 00000, 00001, 00002, etc. This is required such that Python sorts the images correctly. Frame numbers can start from 0, 1, or any other number. Files must have the extension `.tiff`.

Parameters

- **image_dir** (*str*) – Path of image folder.
- **num_images** (*int*) – Number of images in the folder.

Returns Tuple of the list of images and the array size.

Return type tuple[List[Str], tuple]

static _read_images(*images*, *output_array*, *subsampler*)
Loads images and copies them into the provided array.

Parameters

- **images** (*list[Str]*) – Sorted list image paths.
- **output_array** (*np.array like*) – T x N_1 x N_2 array-like object into which images should be loaded. T must equal the number of images in *images*. Each image should be of size N_1 x N_2.
- **subsampler** –

Returns The input array *array*.

Return type np.array like

extract_valid_pixels(*pixels*)
Returns subset of pixels that are valid coordinates for the movie.

classmethod from_tiff_images(*name*, *image_dir*, *num_images*, *memmap=False*, *subsample=10*)
Loads tiff images into a numpy array.

Data is assumed to be stored in 16-bit unsigned integers. Frame numbers are assumed to be padded with zeros: 00000, 00001, 00002, etc. This is required such that Python sorts the images correctly. Frame numbers can start from 0, 1, or any other number. Files must have the extension `.tiff`.

If memmap is True, the data is not loaded into memory bot a memory mapped file on disk is used. The file is named `$name.npy` and is placed in the *image_dir* folder.

Parameters

- **name** (*str*) – Movie name.
- **image_dir** (*str*) – Path of image folder.
- **num_images** (*int*) – Number of images in the folder.
- **memmap** (*bool*) – If True, a memory-mapped file is used. (*Default: False*)
- **subsample** (*int*) – Number of frames to average into a single frame.

Returns Movie created from image files.

Return type *Movie*

is_valid_pixel_coordinate(*coordinate*)
Checks if coordinate is a coordinate for a pixel in the movie.

property num_dimensions
Dimension of the movie (excludes time dimension).

property num_frames
Number of frames in the movie.

property num_pixels
Number of pixels in the movie.

property pixel_shape
Resolution of the movie in pixels.

CHAPTER

THREE

INDICES AND TABLES

- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

E

F

G

H

I

L

M

N

O

P

R

S

T