In-Vivo-Imaging-Pipeline

Release 0.0.0

Darik A. O'Neil

Jan 01, 2023

CONTENTS:

CHAPTER

ONE

EXPERIMENTMANAGEMENT

1.1 BrukerMetaModule module

class BrukerMetaModule.**BrukerMeta**(*ImagingMetaFile*, **args*)

Bases: `object`

Module for bruker meta data

import_meta_data()

static load_meta_data(*ImagingMetaFile*, **args*)

1.2 ExperimentHierarchy module

class ExperimentHierarchy.**BehavioralStage**(*Meta: Tuple[str, str]*, *Stage: str*)

Bases: `ExperimentStage`

Data Class for a generic day of a behavioral task

Required Inputs

Meta : Passed meta from experimental hierarchy (directory, mouse_id)
Stage : Title of Stage

Properties

mouse_id : Identifies which mouse this data belongs to
instance_data : Identifies when this behavioral stage was created

Attributes

data : Pandas dataframe of synced data
folder_dictionary : A dictionary of relevant folders for this behavioral stage
modifications : List of modifications made to this behavioral stage
meta : bruker metadata
multi_index: Pandas multi-index of behavioral components
state_index : look-up table / index relating states to integers
trial_parameters : behavioral parameters

Methods

add_image_sampling_folder : Generates a folder for containing imaging data of a specific sampling rate
load_data : Loads all data
record_mod : Records a modification made to the behavioral stage (Date & Time)
update_folder_dictionary : This function reindexes all folders in the folder dictionary

load_data(*ImagingParameters: Optional[Union[dict, list[dict]]] = None*, **args: Optional[Tuple[str, str]]*, ***kwargs*) → Self

Loads all data

Parameters

- **ImagingParameters** (*Optional[dict]*) – Parameters for some imaging dataset or list of datasets (e.g., for two different sampling rates)
- **args** (*Tuple[str, str]*) – Optionally pass Sync Key to synchronize bruker recordings
- **kwargs** – passed to internal functions taking kwargs

Return type
Any

class ExperimentHierarchy.**CollectedDataFolder**(*Path: str*)

Bases: `object`

This is a class for managing a folder of unorganized data files

Required Inputs

Path : path to folder

Self Methods

find_matching_files : Finds all matching files
reindex : Function that indexed the files within folder again
find_all_ext : Finds all files with specific extension

Properties

instance_data : Data created
path : path to folder
files : List of files in folder

property files: List[str]

find_all_ext(*Ext: str*) → Optional[List[str]]

Finds all files with specific extension

Parameters
Ext (*str*) – Filename extension

Returns
List of files

Return type
List[str]

find_matching_files(*Filename: str*, *Folder: Optional[str] = None*) → Optional[Tuple[str]]

Finds all matching files

Parameters

- **Filename** (*str*) – Filename or ID to search for
- **Folder** (*Any*) – Specify folder filename in

Returns
Matching file/s

Return type
Any

property folders: dict

Dictionary of folders in path

Return type
dict

property instance_date: str

Date Created

Return type
str

property path: str

Path to folder

Return type
str

reindex() → Self

Function that indexes the files within folder again

class ExperimentHierarchy.**CollectedFiguresFolder**(*Path: str*)

Bases: *CollectedDataFolder*

A class for storing figures, inherits collected data folder

view_figure(*Name: str*) → plt.Figure

Function identifies and views a figure based on supplied name

Parameters
Name (*str*) – Name of figure (can be partial)

Returns
the plotted figure

Return type
Any

class ExperimentHierarchy.**CollectedImagingAnalysisFolder**(*Path: str*)

Bases: *CollectedDataFolder*

Class specifically for imaging analysis folders, inherits collected data folder

Self Methods

load_fissa_exports : loads fissa exported files

load_cascade_exports : loads cascade exported files

load_suite2p : loads suite2p exported files

export_registration_to_denoised : moves registration to new folder for namespace compatibility when skipping denoising step

clean_up_motion_correction : This function removes the reg_tif folder and registered.bin generated during motion correction.

clean_up_compilation : This function removes the compiled tif files

add_notes : Function adds notes

`add_notes`(*Step: str*, *KeyOrDict: Union[str, dict]*, *Notes: Optional[Any] = None*) → Self

Function adds notes indicating steps

Parameters

- `Step` – Step of Analysis
- `Step` – str
- `KeyOrDict` (*Union[str, dict]*) – Either a Key or a dictionary containing multiple key-value (note) pairs
- `Notes` (*Optional[Any]*) – If using key, then notes is the paired value

Return type

Any

`clean_up_compilation`() → Self

This function removes the compiled tif files generated inside CompiledImagingData (You can avoid the creation of these in the first place by changing suite2p parameters)

Return type

Any

`clean_up_motion_correction`() → Self

This function removes the reg_tif folder and registered.bin generated during motion correction.

(You can avoid the creation of these in the first place by changing suite2p parameters)

Return type

Any

`property current_stage: str`

Stage of Analysis

Return type

str

`default_folders`()

`export_registration_to_denoised`()

moves registration to new folder for namespace compatibility

Returns

`load_cascade_exports`() → Tuple[ndarray, ndarray, ndarray, dict]

This function loads the Spike Times, Spike Prob, Discrete Approximation and ProcessedInferences files exported from Cascade

Returns
SpikeTimes, SpikeProb, DiscreteApproximation, Processed Inferences

Return type
tuple[Any, Any, Any, dict]

load_fissa_exports() → Tuple[dict, dict, dict]

This function loads the prepared and separated files exported from Fissa

Returns
Prepared, Separated, ProcessedTraces

Return type
tuple[dict, dict, dict]

load_suite2p(**args: str*)

class ExperimentHierarchy.**CollectedImagingFolder**(*Path: str*)

Bases: `CollectedDataFolder`

Class specifically for folders containing raw images, inherits collected data folder

property channels

property file_format

property frames

property height

property imaging_files

property meta_files

property num_imaging_files

property num_meta_files

property planes

reorganize_bruker_files() → None

This function extracts out the meta files and saves in a new directory

Return type
None

property width

class ExperimentHierarchy.**ExperimentData**(***kwargs*)

Bases: `object`

Class for Organizing & Managing Experimental Data Across Sessions

Keyword Arguments

Logfile : Path to existing log file (str, default None)
Mouse : Mouse ID (str, default None)
Condition : Experimental Condition (str, default None)
Directory : Directory for hierarchy (str, default None)
Study : Study (str, default None)

StudyMouse : Study ID (str, default None)

Class Methods

load_experiments : Function that loads the entire experimental hierarchy
get_date: Function returns date
get_time : Function returns time
check_path : Checks Path
_generate_directory_structure : Generates the Directory Structure (The structured folders where data stored)
_generate_histology_directory : Generates Histology Folder
_generate_roi_matching_index_directory : Generate ROI Matching Folder
_generate_experiment_stage_directory : Generate Behavioral Stage Folder
_generate_behavior_subdirectory : Generate Behavioral Folder
_generate_imaging_subdirectory : Generate Imaging Folder
_generate_computation_subdirectory : Generate Computation Folder
_generate_analysis_technique_subdirectory : Generate Analysis Technique

Static Methods

_generate_read_me : Generate a read me file

Self Methods

pass_meta : Passes directory/mouse id
record_mod : Record modification of experiment
record_stage_mod : Record modification of experiment and behavioral stage
save_experiments : Saves Hierarchy to pickle
create_log_file : Creates log file
start_log : Starts Log
check_log : Checks Log Status
create : This function generates the directory hierarchy in one step

Properties

mouse_id : ID of Mouse
log_file : Log Filename Path
experimental_condition : Experiment condition of the mouse
instance_data : Date when this experimental hierarchy was created

Attributes

directory : Experimental Hierarchy Directory
study : Study
study_mouse : ID of mouse in study
modifications : modifications made to this file

`check_log()` → Self

Checks log status

Return type
Any

classmethod check_path(*Path: str*) → bool

create() → Self

This function generates the directory hierarchy in one step

Return type
Any

create_experimental_stage(*Stage: str, Type: Optional[str, object] = 'ExperimentStage', **kwargs*) → Self

Generates an experiment stage folder and attribute

Kwargs are passed to underlying functions

Parameters

- **Stage** (*str*) – Name of experimental stage
- **Type** (*Optional[str, object]*) – Type of experimental stage (Optional, default = ExperimentStage)

Return type
Any

create_log_file() → Self

Creates log file

Return type
Any

end_log() → Self

Ends Logging

Return type
Any

property experimental_condition: str

Experiment condition of the mouse

Return type
str

classmethod get_date()

classmethod get_time()

property instance_date: str

Date when this experimental hierarchy was created

Return type
str

classmethod load_experiments(*ExperimentDirectory: str*) → *ExperimentData*

Function that loads the entire experimental hierarchy

Parameters
ExperimentDirectory (*str*) – Directory containing the experimental hierarchy pickle file

Returns
ExperimentData

Return type
ExperimentManagement.ExperimentHierarchy.ExperimentData

property log_file: str

Log Filename Path

Return type
str

property mouse_id: str

ID of Mouse

Return type
str

pass_meta() → Tuple[str, str]

Passes directory/mouse id

Returns
directory/mouse id

Return type
tuple[str, str]

record_mod(**args: str*) → Self

Record modification of experiment (Data, Time, *args)

Parameters
args (*str*) – A string explaining the modification

Return type
Any

record_stage_mod(*StageKey: str*, **args*) → Self

Record modification of experiment (Data, Time, *args)

Parameters

- **StageKey** (*str*) – The key name for the stage
- **args** (*str*) – A string explaining the modification

Return type
Any

save_experiments() → Self

Saves Hierarchy to pickle

Return type
Any

start_log() → Self

Starts Log

Return type
Any

update_all_folder_dictionaries() → Self

This function iterates through all behavioral stages to update their folder dictionaries

Return type
Any

class ExperimentHierarchy.**ExperimentStage**(*Meta: Tuple[str, str]*, *Stage: str*)

Bases: `object`

Data Class for a generic experiment stage **Required Inputs |** *Meta* : Passed meta from experimental hierarchy (directory, mouse_id) **|** *Stage* : Title of Stage

Properties | *mouse_id* : Identifies which mouse this data belongs to **|** *instance_data* : Identifies when this behavioral stage was created

Attributes | *folder_dictionary* : A dictionary of relevant folders for this behavioral stage **|** *modifications* : List of modifications made to this behavioral stage **|** *meta* : bruker metadata

Methods | *add_image_sampling_folder* : Generates a folder for containing imaging data of a specific sampling rate **|** *load_data* : Loads all data **|** *_generate_imaging_sampling_rate_subdirectory* : Generate Sample Frequency Folder Innards **|** *_generate_read_me* : Generate a read me file **|** *record_mod* : Records a modification made to the behavioral stage (Date & Time) **|** *update_folder_dictionary* : This function reindexes all folders in the folder dictionary

add_image_sampling_folder(*SamplingRate: int*) → Self

Generates a folder for containing imaging data of a specific sampling rate

Parameters
SamplingRate (*int*) – Sampling Rate of Dataset in Hz

Return type
Any

property instance_date: str

Date created

Return type
str

load_data(*ImagingParameters: Optional[Union[dict, list[dict]]] = None*, **args: Optional[Tuple[str, str]]*, ***kwargs*) → Self

Loads all data

Parameters

- **ImagingParameters** (*Optional[dict]*) – Parameters for some imaging dataset or list of datasets (e.g., for two different sampling rates)
- **args** (*Tuple[str, str]*) – Optionally pass Sync Key to synchronize bruker recordings
- **kwargs** – passed to internal functions taking kwargs

Return type
Any

Return type
Any

property mouse_id: str

ID of mouse

Return type
str

record_mod() → Self

Records a modification made to the behavioral stage (Date & Time)

Return type
Any

property stage_id: str

update_folder_dictionary() → Self

This function reindexes all folders in the folder dictionary

Return type

Any

CHAPTER

TWO

BEHAVIOR

2.1 BehaviorUtilities module

`BehaviorUtilities.`**`extract_specific_data`**(*DataFrame: DataFrame*, *KeyValuePairs: Union[Tuple[Tuple[str, Union[str, int, float, list]]], Tuple[str, Union[str, int, float, list]]]*, ***kwargs: bool*) → DataFrame

This Function extracts some specific portion of the behavior

Parameters

- **`DataFrame`** (*`Any`*) – synced behavioral data
- **`KeyValuePairs`** (*`Union[tuple[str, Union[str, int, float]], tuple[str, Union[str, int, float]]]`*) – A tuple containing a column name in the data and expression for pattern matching. Can use tuple of tuples for multiple extracts. ORDER MATTERS.
- **`keep_index`** – whether to keep original index on export (bool, default True)

Returns
some subset of the dataset

Return type
pd.DataFrame

`BehaviorUtilities.`**`lowpass_filter`**(*Data: ndarray*, *SamplingFrequency: float*, *Cutoff: float*, *Order: Optional[int] = None*) → ndarray

Low pass filter (butter)

Parameters

- **`Data`** (*`Any`*) – Data to be filtered
- **`SamplingFrequency`** (*`float`*) – Sampling frequency of data
- **`Cutoff`** (*`float`*) – Cutoff Frequency for filter
- **`Order`** (*`Optional[int]`*) – Optional Order of Filter

Returns
Filtered Data

Return type
Any

`BehaviorUtilities.`**`time_spent_in_burrow`**(*BehavioralObject:* FearConditioning, **args: int*) → Tuple[float]

Calculates time spent in burrow via the gate signal

Parameters

- **BehavioralObject** (*Any*) – FearConditioning Behavioral Stage Object
- **args** (*int*) – Number of trials per stimulus to drop due to forced retraction

Returns
Time spent in burrow (%) per stage

Return type
Tuple[float]

2.2 BurrowFearConditioning module

class BurrowFearConditioning.**DeepLabData**(*DataFolderDLC: CollectedDataFolder*, *DataFolderBehavioralExports: CollectedDataFolder*)

Bases: object

Module for importing deeplabcut data

static calculate_distance(*X: ndarray*, *Y: ndarray*) → ndarray

Function calculates the euclidean distance for each pair of points in (X, Y) :param X: A numpy array of X positions :param Y: A numpy array of Y positions :return: A numpy array containing the distance between each sequential pair of points :rtype: Any

classmethod convert_dataframe_to_physical_units(*DataFrame: DataFrame*, *oldMin: int*, *oldMax: int*, *idx: Union[str, Tuple[str]]*, ***kwargs: int*) → DataFrame

Converts data range to physical range

Parameters

- **DataFrame** (*pd.DataFrame*) – dlc data
- **oldMin** (*int*) – value representing left-side
- **oldMax** (*int*) – value representing right-side
- **idx** – Which columns to rescale
- **idx** – Union[str, Tuple[str]]
- **new_min** – value representing new left-side (int, default 0)
- **new_max** – value representing new right-side (int, default 140)

Returns
DataFrame with rescaled data

Return type
pd.DataFrame

static convert_to_mean_zero(*DataFrame: DataFrame*, *idx: Union[str, Tuple[str]]*) → DataFrame

Converts data range to mean zero

Parameters

- **DataFrame** (*pd.DataFrame*) – dlc data
- **idx** – Which columns to rescale
- **idx** – Union[str, Tuple[str]]

Returns
DataFrame with rescaled data

Return type
pd.DataFrame

classmethod load_data(*DataFolderDLC: CollectedDataFolder*, *DataFolderBehavioralExports: CollectedDataFolder*) → Tuple[DataFrame, DataFrame]

Load DeepLabCut Data

Parameters

- **DataFolderDLC** (*object*) – Collected Data Folder object for deep lab cut folder
- **DataFolderBehavioralExports** (*object*) – Collected Data Folder object for behavioral exports folder

Returns
pre_trial_data, trial_data

Return type
tuple[pd.DataFrame, pd.DataFrame]

classmethod merge_dlc_data(*DataFrame: DataFrame*, *DLC: DeepLabData*, *StateCastDict: dict*) → DataFrame

Function to merge DLC data with some DataFrame

Parameters

- **DataFrame** (*pd.DataFrame*) – Data to merge with
- **DLC** (DeepLabData) – Data to merge
- **StateCastDict** (*dict*) – dictionary relating the state integers with pre-trial and trial states

Returns
the DataFrame with DLC data merged and time-matched

Return type
pd.DataFrame

class BurrowFearConditioning.**FearConditioning**(*Meta: Tuple[str, str]*, *Stage: str*)

Bases: `BehavioralStage`

Instance Factory for Fear Conditioning Data

See BehavioralStage for more information

static check_sync_plot(*DataFrame: DataFrame*) → None

Visualized syncing of the data

Parameters
DataFrame – The data

Returns
Plots in matplotlib

Return type
None

load_data(*ImagingParameters: Optional[Union[dict, list[dict]]] = None*, **args: Optional[Tuple[int, int]]*, ***kwargs*) → Self

Loads all data (Convenience Function)

Parameters

- **ImagingParameters** – Parameters for some imaging dataset
- **args** (*Tuple[int, int]*) – Optional input indicating min/max of video actuator range
- **kwargs** – passed to internal functions taking kwargs

Return type
Any

property num_stim: int

property num_trials: int

property trial_groups: Optional[Tuple[Tuple]]

property trials_per_stim: int

property unique_stim: List[Any]

BurrowFearConditioning.**plot_burrow_coordinates**(*Coordinates*)

BurrowFearConditioning.**plot_column_by_trial_type**(*BehavioralObject: FearConditioning*, *ColumnName: str*, **args: Tuple[str, Union[str, int, float, list]]*, ***kwargs: str*) → Figure

This function plots some column organized by trial type

Parameters

- **BehavioralObject** (*Any*) – The FearConditioning object
- **ColumnName** (*str*) – Name of the column to be plotted
- **args** – Second tuple for data extraction
- **cmap** – string identifying desired colormap

Returns
figure

Return type
Any

BurrowFearConditioning.**plot_trial**(*BehavioralObject: FearConditioning*, *ColumnNames: list[str]*, *Trials: list[int]*, ***kwargs: str*) → plt.Figure

CHAPTER

THREE

IMAGING

3.1 Coloring module

class Coloring.**ColorImages**(*Images: ndarray*, *Stats: ndarray*, *Cells: ndarray*)

Bases: `object`

property background

property background_cutoffs

property background_style

property color_video

property neuron_subsets

property neuronal_ids

property num_frames

property num_neurons

property overlays

preview_background() → Figure

preview_color(*idx*)

property total_rois

property xpix

property ypix

Coloring.**colorize_complete_image**(*Images: ndarray*, *cmap: Union[Colormap, str]*) → ndarray

Colorizes an Image

Parameters

- **Images** (*Any*) – Image to be colorized
- **cmap** – Matplotlib colormap [Object or str]

Type

Any

Returns
Colorized Image

Return type
Any

`Coloring.`**`colorize_rois`**(*Images: ndarray*, *Stats: ndarray*, *ROIs: Optional[List[int]] = None*, **args: Optional[Colormap]*) → ndarray

Generates a colorized roi overlay video

Parameters

- **`Images`** (*Any*) – Images To Extract ROI Overlay
- **`Stats`** (*Any*) – Suite2P Stats
- **`ROIs`** (*list[int] | None*) – Subset of ROIs

Returns
Colorized ROIs

Return type
Any

`Coloring.`**`convert_grayscale_to_color`**(*Image: ndarray*) → ndarray

Converts Image to Grayscale

Parameters
`Image` (*Any*) – Image to be converted

Returns
Color-Grayscale Image

Return type
Any

`Coloring.`**`generate_background`**(*Images: ndarray*, *Option: str = 'True'*, *Cutoffs: Tuple[float, float] = (0, 100)*) → ndarray

`Coloring.`**`generate_custom_map`**(*Colors: List[str]*) → Colormap

Generates a custom linearized colormap

Parameters
`Colors` (*list[str]*) – List of colors included

Returns
Colormap

Return type
Any

`Coloring.`**`generate_pixel_pairs`**(*Stats: ndarray*, *ROIs: List[int]*) → Tuple[Tuple[int, int]]

Generates a tuple containing a list of each pixel pair from every ROI

Parameters

- **`Stats`** (*Any*) – Suite2P Stats
- **`ROIs`** (*list[int]*) – List of ROIs

Returns
List of each pixel for every ROI

Return type
tuple[tuple[int, int]]

`Coloring.`**`merge_background`**(*Background: ndarray*, *NewVideo: ndarray*, *PixelPairs: Tuple[Tuple[int, int]]*) → ndarray

Merges background video and new video at each specified pixel pair

Parameters

- **`Background`** (*`Any`*) – Background video
- **`NewVideo`** (*`Any`*) – Images to merge with
- **`PixelPairs`** (*`tuple[tuple[int,int]]`*) – Pairs of pixels at which merging will occur

Returns
Merged Image

Return type
Any

`Coloring.`**`normalize_image`**(*Image: ndarray*) → ndarray

Normalizes an image for color-mapping

Parameters
`Image` (*`Any`*) – Image to be normalized

Returns
Normalized Image

Return type
Any

`Coloring.`**`overlay_colorized_rois`**(*Background: ndarray*, *ColorizedVideo: ndarray*, **args: Optional[float]*) → ndarray

This function overlays colorized videos onto background video

param Background
Background Images in Grayscale

type Background
Any

param ColorizedVideo
Colorized Overlays In Colormap Space + Alpha Channel

type ColorizedVideo
Any

param args
Alpha for Background

type args
float

return
Merged Images

rtype
Any

`Coloring.`**`rescale_images`**(*Images: ndarray*, *LowCut: float*, *HighCut: float*) → ndarray

Rescale Images within percentiles

Parameters

- **`Images`** (*`Any`*) – Images to be rescaled
- **`LowCut`** (*`float`*) – Low Percentile Cutoff
- **`HighCut`** (*`float`*) – High Percentile Cutoff

Returns

Rescaled Images

Return type

Any

3.2 Colorizer module

`Colorizer.`**`colorize_complete_image`**(*Images: ndarray*, *cmap: Union[Colormap, str]*) → ndarray

Colorizes an Image

Parameters

- **`Images`** (*`Any`*) – Image to be colorized
- **`cmap`** – Matplotlib colormap [Object or str]

Type

Any

Returns

Colorized Image

Return type

Any

`Colorizer.`**`colorize_rois`**(*Images: ndarray*, *Stats: ndarray*, *ROIs: Optional[List[int]] = None*, **args: Optional[Colormap]*) → ndarray

Generates a colorized roi overlay video

Parameters

- **`Images`** (*`Any`*) – Images To Extract ROI Overlay
- **`Stats`** (*`Any`*) – Suite2P Stats
- **`ROIs`** (*`list[int]|None`*) – Subset of ROIs

Returns

Colorized ROIs

Return type

Any

`Colorizer.`**`colorize_video`**(*Images: ndarray*, *Stats: ndarray*, *ROIs: Optional[List[int]] = None*, *Cutoffs: Optional[Tuple[float, float, float, float]] = None*, ***kwargs*) → ndarray

This function generates a video (i.e., numpy array [Z x Y x X]) in which the ROIs or subsets of ROIs utilize a different colormap

Keyword Arguments

cmap : Colormap to use on ROIs (str, default None)
colors : colors which will be used to generate custom colormap
(tuple[tuple[float]], default None)
Example -> ((0, 0, 0), (0.074, 0.624, 1.000), (0.074, 0.624, 1.000))
background : boolean indicating whether to overlay on a blank image
or the background of input image (bool , default True)
white_background : boolean indicating whether to use a white or black background
(bool, default False)
write : boolean indicating whether to write video to file
(bool, default False)
filename : file path for saving video (str, default None)

Parameters

- `Images` (*Any*) – The images to be colorized
- `Stats` (*Any*) – Suite2P Stats file
- `ROIs` (*list[int]*) – A List of ROIs
- `Cutoffs` (*tuple[float]*) – Percentile cutoffs for rescaling data. Data below or above these cutoffs will be replaced by the smallest or largest value in the data type
- `cmap` – Colormap to use on ROIs (str, default None)
- `colors` – colors which will be used to generate custom colormap (default None, overrides cmap is not None) (Tuple of Tuples of Floats, RGB, ranged 0.0-1.0)(default None)
- `background` – boolean indicating whether to overlay on a blank image or the background of input image (default True)
- `white_background` – Boolean indicating whether to use a white or black background (default False, requires background = False)
- `write` – boolean indicating whether to write video to file (default False)
- `filename` – str file path for saving video(default None, which saves to current directory)

Returns
Colorized Images

Return type
Any

`Colorizer.generate_custom_map`(*Colors: List[str]*) → Colormap

Generates a custom linearized colormap

Parameters
`Colors` (*list[str]*) – List of colors included

Returns
Colormap

Return type
Any

`Colorizer.generate_pixel_pairs`(*Stats: ndarray*, *ROIs: List[int]*) → Tuple[Tuple[int, int]]

Generates a tuple containing a list of each pixel pair from every ROI

Parameters

- `Stats` (*Any*) – Suite2P Stats
- `ROIs` (*list[int]*) – List of ROIs

Returns
List of each pixel for every ROI

Return type
tuple[tuple[int, int]]

`Colorizer.`**`merge_background`**(*Background: ndarray*, *NewVideo: ndarray*, *PixelPairs: Tuple[Tuple[int, int]]*) → ndarray

Merges background video and new video at each specified pixel pair

Parameters

- `Background` (*Any*) – Background video
- `NewVideo` (*Any*) – Images to merge with
- `PixelPairs` (*tuple[tuple[int,int]]*) – Pairs of pixels at which merging will occur

Returns
Merged Image

Return type
Any

`Colorizer.`**`overlay_colorized_rois`**(*Background: ndarray*, *ColorizedVideo: ndarray*, **args: Optional[float]*) → ndarray

This function overlays colorized videos onto background video

param Background
Background Images in Grayscale

type Background
Any

param ColorizedVideo
Colorized Overlays In Colormap Space + Alpha Channel

type ColorizedVideo
Any

param args
Alpha for Background

type args
float

return
Merged Images

rtype
Any

3.3 IO module

IO.**determine_bruker_folder_contents**(*ImageDirectory: str*) → Tuple[int, int, int, int, int]

Function determine contents of the bruker folder

Parameters
ImageDirectory (*str*) – Directory containing bruker imaging data

Return type
tuple

IO.**load_all_tiffs**(*ImageDirectory: str*) → ndarray

Load a sequence of tiff stacks

Parameters
ImageDirectory (*str*) – Directory containing a sequence of tiff stacks

Returns
complete_image numpy array [Z x Y x X] as int16

Return type
Any

IO.**load_binary_meta**(*Filename: str*) → Tuple[int, int, int, str]

Loads meta file for binary video

Parameters
Filename (*str*) – The meta file (.txt ext)

Returns
A tuple containing the number of frames, y pixels, and x pixels [Z x Y x X]

Return type
tuple[int, int, int, str]

IO.**load_bruker_tiffs**(*ImageDirectory: str*) → Union[ndarray, Tuple[ndarray]]

Load a sequence of tiff files from a directory.

Designed to compile the outputs of a certain imaging utility that exports recordings such that each frame is saved as a single tiff.

Parameters
ImageDirectory (*str*) – Directory containing a sequence of single frame tiff files

Returns
complete_image: All tiff files in the directory compiled into a single array (Z x Y x X, uint16)

Return type
Any

IO.**load_mapped_binary**(*Filename: str*, *MetaFile: str*, **args: Optional[str]*, ***kwargs: str*) → memmap

Loads a raw binary file in the workspace without loading into memory

Enter the path to autofill (assumes Filename & meta are path + binary_video, video_meta.txt)

Parameters

- **Filename** (*str*) – filename for binary video
- **MetaFile** (*str*) – filename for meta file
- **args** (*str*) – Path

- **mode** – pass mode to numpy.memmap (str, default = "r")

Returns
memmap(numpy) array [Z x Y x X]

Return type
Any

IO.**load_raw_binary**(*Filename: Optional[str]*, *MetaFile: Optional[str]*, **args: Optional[str]*) → ndarray

Loads a raw binary file

Enter the path to autofill (assumes Filename & meta are path + binary_video, video_meta.txt)

Parameters

- **Filename** (*str*) – filename for binary video
- **MetaFile** (*str*) – filename for meta file
- **args** (*str*) – path to a directory containing Filename and MetaFile

Returns
numpy array [Z x Y x X]

Return type
Any

IO.**load_single_tiff**(*Filename: str*, *NumFrames: int*) → ndarray

Load a single tiff file

Parameters

- **Filename** (*str*) – filename
- **NumFrames** (*int*) – number of frames

Returns
numpy array [Z x Y x X]

Return type
Any

IO.**repackage_bruker_tiffs**(*ImageDirectory: str*, *OutputDirectory: str*, **args: Union[int, tuple[int]]*) → None

Repackages a sequence of tiff files within a directory to a smaller sequence of tiff stacks. Designed to compile the outputs of a certain imaging utility that exports recordings such that each frame is saved as a single tiff. :param ImageDirectory: Directory containing a sequence of single frame tiff files :type ImageDirectory: str :param OutputDirectory: Empty directory where tiff stacks will be saved :type OutputDirectory: str :param args: optional argument to indicate the repackaging of a specific channel and/or plane :type args: int :rtype: None

IO.**save_raw_binary**(*Images: ndarray*, *ImageDirectory: str*) → None

This function saves a tiff stack as a binary file

Parameters

- **Images** (*np.ndarray*) – Images to be saved [Z x Y x X]
- **ImageDirectory** (*str*) – Directory to save images in

Return type
None

IO.**save_single_tiff**(*Images: ~numpy.ndarray*, *Filename: str*, *Type: ~typing.Optional[~numpy.dtype] = <class 'numpy.int16'>*) → None

Save a numpy array to a single tiff file as type int16

Parameters

- `Images` (*Any*) – numpy array [frames, y pixels, x pixels]
- `Filename` (*str*) – filename
- `Type` (*Optional[Any]*) – type for saving

Return type
None

`IO.save_tiff_stack`(*Images: str*, *OutputDirectory: str*, *Type: ~typing.Optional[~numpy.dtype] = <class 'numpy.int16'>*) → None

Save a numpy array to a sequence of tiff stacks

Parameters

- `Images` (*Any*) – A numpy array containing a tiff stack [Z x Y x X]
- `OutputDirectory` (*str*) – A directory to save the sequence of tiff stacks in int16
- `Type` (*Optional[Any]*) – type for saving

Return type
None

`IO.save_video`(*Images: ndarray*, *Filename: str*, *fps: Union[float, int] = 30*) → None

Function writes video to .mp4

Parameters

- `Images` (*Any*) – Images to be written
- `Filename` (*str*) – Filename (Or Complete Filename Path)
- `fps` (*Union[float, int]*) – frame rate

Return type
None

3.4 ImageProcessing module

`ImageProcessing.blockwise_fast_filter_tiff`(*Images: ndarray*, *Footprint: Optional[ndarray] = None*, ***kwargs: int*) → ndarray

GPU-parallelized multidimensional median filter performed in overlapping blocks.

Designed for use on arrays larger than the available memory capacity.

Footprint is of the form np.ones((Z pixels, Y pixels, X pixels)) with the origin in the center

Requires CuPy

Parameters

- `Images` (*Any*) – Images stack to be filtered
- `Footprint` (*Any*) – Mask of the median filter (Optional, Default 3 x 3 x 3)
- `block_size` – Integer indicating the size of each block. Must fit within memory. (int, default 21000)
- `block_buffer_region` – Integer indicating the size of the overlapping region between blocks (int, default 500)

Returns
Images: numpy array [Z x Y x X]

Return type
Any

`ImageProcessing.fast_filter_images(`*Images: ndarray*, *Footprint: Optional[ndarray] = None*`)` → ndarray

GPU-parallelized multidimensional median filter

Footprint is of the form np.ones((Z pixels, Y pixels, X pixels)) with the origin in the center

Requires CuPy

Parameters

- **Images** (*Any*) – Image stack to be filtered [Z x Y x X]
- **Footprint** (*Any*) – Mask of the median filter (Optional, Default 3 x 3 x 3)

Returns
filtered_image [Z x Y x X]

Return type
Any

`ImageProcessing.filter_images(`*Images: ndarray*, *Footprint: Optional[ndarray] = None*`)` → ndarray

Denoise a tiff stack using a multidimensional median filter

This function simply calls scipy.ndimage.median_filter

Footprint is of the form np.ones((Z pixels, Y pixels, X pixels)) with the origin in the center

Parameters

- **Images** (*Any*) – Images stack to be filtered [Z x Y x X]
- **Footprint** (*Any*) – Mask of the median filter (Optional, Default 3 x 3 x 3)

Returns
filtered images [Z x Y x X]

Return type
Any

`ImageProcessing.grouped_z_project(`*Images: ndarray*, *BinSize: Union[Tuple[int, int, int], int]*, *DownsampleFunction: Callable[[ndarray], ndarray]*`)` → ndarray

Utilize grouped z-project to downsample data

Downsample example function -> np.mean

Parameters

- **Images** (*Any*) – A numpy array containing a tiff stack [Z x Y x X]
- **BinSize** (*Union[tuple, int]*) – Size of each bin passed to downsampling function
- **DownsampleFunction** (*Any*) – Downsampling function

Returns
downsampled image [Z x Y x X]

Return type
Any

`ImageProcessing.`**`remove_shuttle_artifact`**`(`*Images: ndarray*, ***kwargs: int*`)` → ndarray

Function to remove the shuttle artifacts present at the initial imaging frames

Parameters

- **`Images`** (*`Any`*) – Images array with shape Z x Y x X
- **`kwargs`** –
- **`artifact_length`** – number of frames considered artifact (int)
- **`chunk_size`** – number of frames per chunk_size (makes divisible by value) (int)

Returns

Images

Return type

Any

3.5 PowerSpectrum module

3.6 SignalProcessing module

`SignalProcessing.`**`anisotropicDiffusion`**`(`*Trace*, ***kwargs*`)`

`SignalProcessing.`**`anisotropic_diffusion`**`(`*img*, *niter=1*, *kappa=50*, *gamma=0.1*, *voxelspacing=None*, *option=1*`)`

`SignalProcessing.`**`bin_data`**`(`*NeuralDataTensorForm*, *BinSizeInFrames*`)`

`SignalProcessing.`**`bind_data`**`(`*NeuralData*, *BinSize*`)`

`SignalProcessing.`**`calculateFiringRate`**`(`*SpikeProb*, *FrameRate*`)`

`SignalProcessing.`**`calculate_dFoF`**`(`*Traces: ndarray*, *FrameRate: float*, ***kwargs: Union[bool, float]*`)`

`SignalProcessing.`**`calculate_mean_firing_rate`**`(`*NeuralData*`)`

`SignalProcessing.`**`calculate_standardized_noise`**`(`*DFF: ndarray*, *FrameRate: float*`)` → Union[float, ndarray]

Calculates standardized noise, see: https://www.nature.com/articles/s41593-021-00895-5

Parameters

- **`DFF`** (*`Any`*) – Fluorescence over Baseline (DF/F)
- **`FrameRate`** (*`float`*) – Imaging framerate

Returns

standardized noise

Return type

Any

`SignalProcessing.`**`detrendTraces`**`(`*Traces*, ***kwargs*`)`

`SignalProcessing.`**`detrendTraces_TiffOrg`**`(`*Traces*, ***kwargs*`)`

`SignalProcessing.`**`normalizeSmoothFiringRates`**`(`*FiringRates*, *Sigma*`)`

SignalProcessing.**smoothTraces**(*Traces*, ***kwargs*)

SignalProcessing.**smoothTraces_TiffOrg**(*Traces*, ***kwargs*)

3.7 Utilities module

Utilities.**generateCovarianceMatrix**(*NeuralActivity*, *ActivityMeasure*, ***kwargs*)

Utilities.**generateSpikeMatrix**(*SpikeTimes*, *NumFrames*)

Utilities.**generate_features**(*FramesPerTrial*, *NumTrials*, *TrialParameters*)

Utilities.**mergeTraces**(*Traces*, ***kwargs*)

Utilities.**pruneNaN**(*NeuralActivity*, ***kwargs*)

Utilities.**pruneTracesByNeuronalIndex**(*Traces*, *NeuronalIndex*)

Utilities.**trial_matrix_org**(*DataFrame*, *NeuralData*)

3.8 Visualization module

Visualization.**assessSpikeInference**(*SpikeProb*, *SpikeTimes*, *Traces*, *FrameRate*)

Visualization.**compareTraces**(*RawTraces: ndarray*, *SmoothTraces: ndarray*, *FrameRate: float*, *Frames: int*) → None

Compare two sets of traces interactively

Parameters

- **RawTraces** (*Any*) – Trace Set 1
- **SmoothTraces** (*Any*) – Trace Set 2
- **FrameRate** (*float*) – FrameRate
- **Frames** (*int*) – Number of Frames

Return type
None

Visualization.**compareTraces2**(*RawTraces: ndarray*, *SmoothTraces: ndarray*, *FrameRate: float*, *Frames: int*) → None

Compare two sets of traces interactively

Parameters

- **RawTraces** (*Any*) – Trace Set 1
- **SmoothTraces** (*Any*) – Trace Set 2
- **FrameRate** (*float*) – FrameRate
- **Frames** (*int*) – Number of Frames

Return type
None

`Visualization.compareTraces3`(*RawTraces: ndarray*, *SmoothTraces: ndarray*, *FrameRate: float*, *Frames: int*) → None

Compare two sets of traces interactively

Parameters

- **RawTraces** (*Any*) – Trace Set 1
- **SmoothTraces** (*Any*) – Trace Set 2
- **FrameRate** (*float*) – FrameRate
- **Frames** (*int*) – Number of Frames

Return type
None

`Visualization.interactive_traces`(*Traces: ndarray*, *FrameRate: float*, ***kwargs*) → None

`Visualization.plotFiringRateMatrix`(*FiringRates*, *FrameRate*, ***kwargs*)

`Visualization.plotNeuralHeatMap`(*NeuralActivity: ndarray*, *FrameRate: float*, **args: Optional[Tuple[Union[List[int], Tuple[int, int]]]]*) → None

`Visualization.plotNoise`(*Traces*, *FrameRate*)

`Visualization.plotROC`(*TPR*, *FPR*, ***kwargs*)

`Visualization.plotSpikeInference`(*SpikeProb*, *SpikeTimes*, *Traces*, *FrameRate*, ***kwargs*)

`Visualization.view_image`(*Images: ndarray*, *FPS: float*, ***kwargs: Union[str, int]*) → List[object]

Visualize a numpy array [Z x Y x X] as a video

Parameters

- **Images** (*Any*) – A numpy array [Z x Y x X]
- **FPS** (*float*) – Frames Per Second
- **cmap** – colormap (str, default binary_r)
- **interpolation** – interpolation method (str, default none)
- **SpeedUp** – FPS multiplier (int, default 1)
- **Vmin** – minimum value of colormap (int, default 0)
- **Vmax** – maximum value of colormap (int, default 32000)

Returns
Figure Animation

Return type
list[matplotlib.pyplot.figure, matplotlib.pyplot.axes, matplotlib.pyplot.axes,

matplotlib.pyplot.axes, Any, Any]

CHAPTER

FOUR

THEORY

4.1 DecodingAnalysis module

`class DecodingAnalysis.DecodingModule(**kwargs)`

Bases: `object`

This a super class passing conserved functions for decoding modules

Properties

imported_neural_organization : the structure of the passed neural data
imported_feature_organization : the structure of the passed feature data

`assessFit(**kwargs)`

`cleanKwargs()`

`classmethod collapseFeatures(Features, **kwargs)`

`commonAssessment(**kwargs)`

`static createTrialIndicator(NumTrials, FramesPerTrial)`

`dep_shuffle_trials()`

`property feature_matrix`

`property feature_tensor`

`fitModel(**kwargs)`

`fullAssessment(**kwargs)`

`property imported_feature_organization`

Return type
str

`property imported_neural_organization`

Return type
str

`classmethod loadFeatures(FeatureFile)`

loadFeaturesFile(*_feature_data_file*)

makePrediction(***kwargs*)

property **neural_matrix**

property **neural_tensor**

property **num_frames**

property **num_neurons**

property **num_trials**

Return type
int

plotROCs(***kwargs*)

printAssessment()

saveModel(*OutputFolder*)

static **shuffleByTrialIndex**(*NeuralActivityInTrialForm*, *TrialIndex*)

classmethod **shuffleEachNeuron**(*NeuralActivityInMatrixForm*)

classmethod **shuffleFrames**(*DataInMatrixForm*, ***kwargs*)

classmethod **shuffleLabels**(*Labels*)

shuffle_trial_labels()

shuffle_trials()

static **shuffled_trials_by_group**(*NumTrials*, *TrialIndex*)

splitData(***kwargs*)

classmethod **split_by_trials**(*NeuralDataTensor*, *FeatureDataTensor*, *DataSplits*, *TrialGroups*)

structural_report()

validate_data_sets()

DecodingAnalysis.**PerformanceMetrics**(*Type*, *NumberOfSplits*)

4.2 LSTMRegression module

class LSTMRegression.**LongShortTermMemoryRegression**(***kwargs*)

Bases: DecodingModule

fitModel(***kwargs*)

makeAllPredictions(***kwargs*)

makePrediction(***kwargs*)

4.3 LinearNonLinearRegression module

class `LinearNonLinearRegression.`**`LinearNonLinearRegression`**`(**kwargs)`

Bases: `DecodingModule`

`assessFit``(**kwargs)`

`fitModel``(**kwargs)`

`makePrediction``(**kwargs)`

4.4 LinearRegression module

class `LinearRegression.`**`LinearRegression`**`(**kwargs)`

Bases: `DecodingModule`

`assessFit``(**kwargs)`

`commonAssessment``(**kwargs)`

`fitModel``(**kwargs)`

`fullAssessment``(**kwargs)`

`makeAllPredictions``()`

`makePrediction``(**kwargs)`

class `LinearRegression.`**`WienerFilterDecoder`**

Bases: `object`

Class for the Wiener Filter Decoder

There are no parameters to set.

This simply leverages the scikit-learn linear regression.

`fit``(X_flat_train, y_train, **kwargs)`

Train Wiener Filter Decoder

X_flat_train: numpy 2d array of shape [n_samples,n_features]
This is the neural data. See example file for an example of how to format the neural data correctly

y_train: numpy 2d array of shape [n_samples, n_outputs]
This is the outputs that are being predicted

`predict``(X_flat_test)`

Predict outcomes using trained Wiener Cascade Decoder

X_flat_test: numpy 2d array of shape [n_samples,n_features]
This is the neural data being used to predict outputs.

y_test_predicted: numpy 2d array of shape [n_samples,n_outputs]
The predicted outputs

4.5 LogisticRegression module

`class LogisticRegression.`**`LogisticRegression`**`(**kwargs)`

Bases: `DecodingModule`

`assessFit``(**kwargs)`

`commonAssessment``(**kwargs)`

`fitModel``(**kwargs)`

`makeAllPredictions``()`

`makePrediction``(**kwargs)`

`class LogisticRegression.`**`LogisticRegressionDecoder`**

Bases: `object`

`fit``(x, y, **kwargs)`

`predict``(x)`

4.6 SupportVectorMachine module

`class SupportVectorMachine.`**`SVM`**`(**kwargs)`

Bases: `DecodingModule`

`assessFit``(**kwargs)`

`commonAssessment``(**kwargs)`

`fitModel``()`

`makeAllPredictions``()`

`makePrediction``(**kwargs)`

CHAPTER FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX