Package ‘SMARTTR’

January 15, 2025

Title Mapping, Analysis, and Visualization Package for Brainwide Dual-Ensemble Coronal Datasets

Version 1.0.1

Maintainer Michelle Jin <mj2947@cumc.columbia.edu>

Description A workflow designed for the high-throughput mapping and analysis of dual-labelled ensemble datasets.
SMARTTR stands for simple multi-ensemble atlas registration and statistical testing in R.
Genetic activity-tagging strategies in mice allow for investigation of neural ensembles underlying behavior during multiple time points. SMARTTR provides a streamlined API for storing metadata related to imaging, animal subject, and experimental parameters and groupings.
Data management is intrinsically built into the package, which helps automate the process of combining ensemble datasets across multiple images, animals, and experimental groupings.
SMARTTR is compatible with importing external datasets, and provides a set of built-in analysis and visualization functions for network analysis for each ensemble dataset.

License GPL (>= 3)

URL https://mjin1812.github.io/SMARTTR/, https://github.com/mjin1812/SMARTTR

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 3.6.0)

Imports rlang (>= 1.1.0),
dplyr (>= 1.0.0),
ggplot2 (>= 3.3.3),
ggrepel (>= 0.9.1),
tidyr (>= 1.1.3),
tidyselect (>= 1.1.1),
tibble (>= 3.2.1),
tidygraph (>= 1.2.0),
igraph (>= 1.2.6),
ggraph (>= 2.0.5),
stringdist (>= 0.9.6),
stringr (>= 1.4.0),
readr (>= 1.4.0),
readxl (>= 1.3.1),
Hmisc (>= 4.5.0),
purrr (>= 1.0.0),

forcats (>= 0.5.0),
magrittr

Suggests SMART (>= 0.1.0),
wholebrain (>= 0.1.1),
ggpattern (>= 0.2.0),
grImport (>= 0.9.3),
rmarkdown (>= 2.8.0),
testthat (>= 3.1.1),
ggh4x (>= 0.2.8),
knitr (>= 1.33.0),
rstatix

Additional_repositories `https://mjin1812.github.io/drat/`

Config/testthat/edition 3

VignetteBuilder knitr

Contents

`add_mouse` *Add mouse object to an experiment*

Description

This function takes an experiment object and mouse object and adds the mouse object to the experiment. The mouse's unprocessed data, including all of it's individual slice information and raw imported segmentation and registration data will be not be added from the mouse object to save space. Any desire to modify this data must be done at the mouse object level before continuing further.

This function will also read the individual mouse attributes and automatically populate the experimental attributes that are relevant. For example, the 'group' attribute of a mouse will be read and automatically added to the experiment object's 'experimental_groups' attribute if it is a new unique experimental group name.

Usage

```
add_mouse(e, m, replace = FALSE)
```

Arguments

`e`	experiment object
`m`	mouse object
`replace`	(bool, default = FALSE) Replace a mouse already contained in an experiment object.

Value

an experiment object

Examples

```
e <- experiment(experiment_name = "example")
m <- mouse(mouse_ID = "Mouse1")
e <- add_mouse(e, m)
e <- add_mouse(e, m, replace = TRUE)
```

`add_slice` *Add slice to a mouse object*

Description

Add slice to a mouse object

Usage

```
add_slice(m, s, replace = FALSE)
```

Arguments

`m`	mouse object
`s`	slice object
`replace`	(bool, default = FALSE) Replace a slice already contained in a mouse object.

Examples

```
m <- mouse()
s <- slice()
m <- add_slice(m, s, replace = FALSE)
m <- add_slice(m, s, replace = TRUE)
```

`adjust_brain_outline` *Adjust brain outline.*

Description

This function takes a slice object and first applies a filter with default settings to the image set as the slice registration path. An interative user loop allows for easy adjustment of the brain threshold since the wholebrain GUI tends to be a bit buggy. This function then returns a filter with the adjusted brain threshold.

Usage

```
adjust_brain_outline(s, filter = NULL)
```

Arguments

`s`	slice object
`filter`	(list, default = NULL) If the user passes their own filter list, it will use that instead of the presaved filter list from SMARTTR.

Value

filter (list) wholebrain compatible filter

Examples

```
## Not run:
# Adjust the brain threshold, then run register on the slice object
filter <- adjust_brain_outline(s); s <- register(s, filter = filter)

## End(Not run)
```

`attr2match` *attr2match*

Description

A custom list to match attributes of a mouse and experiment object respectively.

Usage

```
attr2match
```

Format

A list

`check_ontology_coding` *Checks the acronyms and full length region names to match with internally stored ontology*

Description

Run this as a quality check after importing an external dataset using `import_mapped_datasets()`. This goes through all dataframes for all channels imported and replaces any non-matching acronyms and region names with those as they are exactly coded in SMARTTR.

Usage

```
check_ontology_coding(e, ontology = "allen")
```

Arguments

`e`	experiment object
`ontology`	(str, default = "allen") Region ontology to check against. Options = "allen" or "unified"

Details

Note: Processing times scales with the size of your dataset so it may take a few minutes if your dataset is large...

Value

e experiment object

Examples

```
## Not run:
e <- check_ontology_coding(e, "allen")

## End(Not run)
```

`check_redundant_parents` *Check for redundant parent regions included in a list of acronyms in a plate. For example, if all the the subregions for the hypothalamus are represented, the HY should not be included in the list.*

Description

Check for redundant parent regions included in a list of acronyms in a plate. For example, if all the the subregions for the hypothalamus are represented, the HY should not be included in the list.

Usage

```
check_redundant_parents(acronyms, ontology = "allen")
```

Arguments

`acronyms`	(vec) a vector of acronyms to check for possible parents that are redundantly included in the vector.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"

Value

A list containing two elements: one vector of unique child acronyms, a vector of the parent regions considered redundant

Examples

```
check_redundant_parents(acronyms = c("ACA5", "ACA1", "ACA"))
check_redundant_parents(acronyms = c("VMHDM", "VMHSh", "VMH"), ontology = "unified")
```

`combine_cell_counts` *Combine cell counts across all mice in an experiment into a single dataframe.*

Description

This function also stores the mouse attribute names (not experiment attributes) as columns that will be used as categorical variables to make analysis subgroups. The values of these attributes (`group`, `drug`, `age`) will automatically be converted to a string values for consistency.

Usage

```
combine_cell_counts(e, by)
```

Arguments

`e`	experiment object
`by`	(str) names of the experiment attributes (categorical variables) that will be used to create analysis subgroups.

Value

e en experiment object

Examples

```
## Not run:
e <- combine_cell_counts(e, by = c('groups', 'sex'))

## End(Not run)
```

correlation_diff_permutation

This function performs a permutation analysis to compare the region pairwise correlation coefficients between two different analysis groups.

Description

The data from two different analysis groups are compared by specifying the `correlation_list_name_1` and `correlation_list_name_2` parameters. Note that both of these analysis groups must have the same number of channels to compare. The functions `get_correlations()` needs to have been run for each of these analysis groups prior to running this function. The test statistics used is the pearson values of those in correlation_list_name_2 subtracted from corresponding Pearson values in correlation_list_name_1.

Usage

```
correlation_diff_permutation(
  e,
  correlation_list_name_1,
  correlation_list_name_2,
  channels = c("cfos", "eyfp", "colabel"),
  n_shuffle = 1000,
  method = "pearson",
  seed = 5,
  p_adjust_method = "none",
  alpha = 0.05,
  ...
)
```

Arguments

`e`	experiment object
`correlation_list_name_1`	(str) The name of the correlation list object used as the first group for comparison.
`correlation_list_name_2`	(str) The name of the correlation list object used as the second group for comparison.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) The channels to process.
`n_shuffle`	(int, default = 1000) The number of permutation shuffles.
`method`	(str, default = "pearson", options = c("pearson", "spearman")) Specifies the type of correlations to compute. Spearman correlations are the Pearson linear correlations computed on the ranks of non-missing elements, using midranks for ties. See also `Hmisc::rcorr()`
`seed`	(int, default = 5) Random seed for future replication.
`p_adjust_method`	(bool or str, default = "none") Apply the named method to control for the inflated false discovery rate or FWER. Set to FALSE or "none" to keep "raw" p values. See also `stats::p.adjust()` for the correction options.

`alpha`	(float, default = 0.05) The alpha cutoff for significance between region pairwise correlation differences
`...`	additional parameters to pass to `permute_corr_diff_distrib()`

Value

e experiment object. The experiment object now has a list called `permutation_p_matrix` stored in it. Elements of this `permutation_p_matrix` list are the outputs of different permutation comparison analyses. These elements are named by the groups that were compared.

See Also

`get_correlations()`

Examples

```
## Not run:
e <- correlation_diff_permutation(sundowning,
                                            correlation_list_name_1 = "female_AD",
                                            correlation_list_name_2 = "female_control",
                                            channels = c("cfos", "eyfp", "colabel"),
                                            n_shuffles = 1000,
                                            seed = 5,
                                            p_adjust_method = "none"
                                            alpha = 0.001
                                            )

## End(Not run)
```

`create_joined_networks` *Create a joined network to visualize overlapping connections with the same outer joined node set.*

Description

Create a joined network to visualize overlapping connections with the same outer joined node set.

Usage

```
create_joined_networks(
  e,
  correlation_list_names = c("male_agg", "female_non"),
  channels = "cfos",
  ontology = "unified",
  alpha = 0.001,
  pearson_thresh = 0.9,
  proportional_thresh = NULL,
  alpha2 = NULL,
  pearson_thresh2 = NULL,
  proportional_thresh2 = NULL,
  filter_isolates = TRUE,
 anatomical.order = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CB"),
  export_overlapping_edges = TRUE
)
```

Arguments

`e`	experiment object
`correlation_list_names`	(str vec) character vector of the two correlation lists used to include in a joined network
`channels`	(str, default = c("cfos", "eyfp", "colabel")) The channels to process.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`alpha`	(float, default = 0.05) The significance threshold for including brain regions in the network. if NULL or NA, this threshold is not applied.
`pearson_thresh`	(float, default = 0.8) The pearson correlation coefficient threshold to apply for filtering out
`proportional_thresh`	(float, default = NULL) Takes precedent over the `alpha` and the `pearson_thresh` parameters. Input the desired edge proportion (i.e., edge density) as your desired sparsity constraint.
`alpha2`	(NULL) If not NULL, this gives the option of filtering the second network by a different alpha from the first. The `alpha` parameter will then be used as the threshold for network 1.
`pearson_thresh2`	(NULL) If not NULL, this gives the option of filtering the second network by a different pearson threshold from the first network. The `pearson_thresh` parameter will then be used as the threshold for network 1.
`proportional_thresh2`	(NULL) If not NULL, this gives the option of filtering the second network by a different proportional threshold from the first.
`filter_isolates`	(logical, default = TRUE) Whether to filter out the number of isolated (zero degree) nodes from the network. Default is to retain them.
`anatomical.order`	(vec, c("Isocortex","OLF","HPF","CTXsp","CNU","TH","HY","MB","HB","CB")) The default super region acronym list that groups all subregions in the dataset.
`export_overlapping_edges`	(bool, default = TRUE) Whether to export the overlapping edges between the two networks as a csv into the `table` directory.

Value

e experiment object. This object now has a new added element called `networks`. This is a list storing a graph object per channel for each network analysis run. The name of each network (`network_name`) is the same as the `correlation_list_name` used to generate the network. This `network_name` is fed as a parameter into the `plot_networks()` function.

See Also

`plot_networks()`

Examples

```
## Not run:
e sundowning <- create_networks(sundowning, correlation_list_name = "female_control", alpha = 0.05)

## End(Not run)
```

`create_networks` *Create graph objects for plotting different analysis subgroups.*

Description

Create graph objects for plotting different analysis subgroups.

Usage

```
create_networks(
  e,
  correlation_list_name,
  channels = c("cfos", "eyfp", "colabel"),
  proportional_thresh = NULL,
  alpha = 0.05,
  pearson_thresh = 0,
  ontology = "allen",
 anatomical.order = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CB"),
  filter_isolates = FALSE
)
```

Arguments

`e`	experiment object
`correlation_list_name`	(str) Name of the correlation list object used to generate the networks.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) The channels to process.
`proportional_thresh`	(float, default = NULL) Takes precedent over the `alpha` and the `pearson_thresh` parameters. Input the desired edge proportion (i.e., edge density) as your desired sparsity constraint.
`alpha`	(float, default = 0.05) The significance threshold for including brain regions in the network. if NULL or NA, this threshold is not applied.
`pearson_thresh`	(float, default = 0) The pearson correlation coefficient threshold to apply for filtering out
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`anatomical.order`	(vec, c("Isocortex","OLF","HPF","CTXsp","CNU","TH","HY","MB","HB","CB")) The default super region acronym list that groups all subregions in the dataset.
`filter_isolates`	(logical, default = FALSE) Whether to filter out the number of isolated (zero degree) nodes from the network. Default is to retain them.

Value

e experiment object. This object now has a new added element called `networks`. This is a list storing a graph object per channel for each network analysis run. The name of each network (`network_name`) is the same as the `correlation_list_name` used to generate the network. This `network_name` is fed as a parameter into the `plot_networks()` function.

See Also

`plot_networks()`

Examples

```
## Not run:
e sundowning <- create_networks(sundowning, correlation_list_name = "female_control", alpha = 0.05)

## End(Not run)
```

---

detect_single_slice_regions

Detect atlas regions that only show up in a single slice object within a mouse.

Description

Quality check function to make sure that the regions included for analysis show up in more than 1 slice object, otherwise the user can remove exceptions from the mouse object.

Regions counts derived from only one image may be less accurate. This function can generate a log of these regions so the user can qualitatively evaluate the raw data. Users also have to option of removing these regions automatically from normalized_counts dataframe.

The user should run `normalize_cell_counts()` and `get_cell_table()` functions prior to using this function.

Usage

```
detect_single_slice_regions(m, remove = FALSE, log = TRUE)
```

Arguments

`m`	mouse object
`remove`	(bool, FALSE) Remove any regions in the normalized counts table that
`log`	(bool, TRUE) Save the regions that don't have enough n into a .csv file in the output folder.

Value

mouse object

enough_mice_per_group

Check if there are enough mice per analysis subgroup across all regions. if the normalized counts data sets are split by specified grouping variables. This function also automatically keeps only the common regions that are found across all comparison groups.

Description

Check if there are enough mice per analysis subgroup across all regions. if the normalized counts data sets are split by specified grouping variables. This function also automatically keeps only the common regions that are found across all comparison groups.

Usage

```
enough_mice_per_group(
  e,
  by = c("group", "sex"),
  min_n = 5,
  remove = TRUE,
  log = TRUE
)
```

Arguments

`e`	experiment object
`by`	(str, default = c("group", "sex")) The mice attributes used to group the datasets into comparison groups.
`min_n`	(int, default = 5) The minimum number of mice in each group for region comparisons.
`remove`	(bool, TRUE) Remove any regions in the combined normalized count dataframes that don't have enough n to do a comparison on. These regions are removed across all comparison groups.
`log`	(bool, TRUE) Save the regions that don't have enough n into a '.csv' file in the output folder.

Value

e experiment object

Examples

```
## Not run:
e <- enough_mice_per_group(e, by = c("group", "sex"), min_n = 4, remove = TRUE, log = TRUE)

## End(Not run)
```

exclude_anatomy *exclude_anatomy (generic function)*

Description

Method for excluding user specified regions, layer 1, and contralateral hemisphere per slice. This function automatically excludes the default regions included in the attribute "regions_excluded" in each slice IN ADDITION to the regions added to the 'exclude_regions' parameter. Regions added to the 'exclude_regions' parameter will then be updated in the slice attribute to keep track of what was excluded. Note: Please see the `simplify_regions` and `simplify_keywords` parameters. By default, if a subregion can be folded into a parent region based on a certain keywords, then this function will automatically exclude the entire parent region as a conservative exclusion approach. Keep `simplify_regions=TRUE` if the final analysis will contain simplified regions.

Method for excluding cell counts in specified regions, layer 1, out-of-bounds cells counts, and hemispheres per mouse. This function automatically excludes the default regions included in the attribute "regions_excluded" in each slice IN ADDITION to the regions added to the 'exclude_regions' parameter. Regions added to the 'exclude_regions' parameter will then be updated in the slice attribute to keep track of what was excluded.

Note: Please see the `simplify_regions` and `simplify_keywords` parameters. By default, if a subregion can be folded into a parent region based on a certain keywords, then this function will automatically exclude the entire parent region as a conservative exclusion approach. Keep `simplify_regions=TRUE` if the final analysis will contain simplified regions.

Usage

```
exclude_anatomy(x, ...)

## S3 method for class 'slice'
exclude_anatomy(
  x,
  channels = NULL,
  clean = TRUE,
  exclude_right_regions = NULL,
  exclude_left_regions = NULL,
  exclude_hemisphere = TRUE,
  exclude_layer_1 = TRUE,
  include_right_regions = NULL,
  include_left_regions = NULL,
  simplify_regions = TRUE,
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  plot_filtered = TRUE,
  ...
)

## S3 method for class 'mouse'
exclude_anatomy(
  x,
  slice_ID = NA,
  hemisphere = NULL,
  channels = NULL,
  clean = TRUE,
  exclude_right_regions = NULL,
  exclude_left_regions = NULL,
  exclude_hemisphere = FALSE,
  exclude_layer_1 = TRUE,
  include_right_regions = NULL,
  include_left_regions = NULL,
  simplify_regions = TRUE,
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  plot_filtered = TRUE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	further arguments passed to or from other methods.
`channels`	(str vector, default = NULL) Channels to process. If NULL, defaults to the channels stored in the slice object attributes.
`clean`	(bool, default = TRUE). Remove cells that don't map to any regions.
`exclude_right_regions`	(str vector, default = NULL); acronyms of regions you want to exclude from right hemi,in addition to regions that will by default be excluded in the slice attribute 'right_regions_excluded'
`exclude_left_regions`	(str vector, default = NULL); acronyms of regions you want to exclude from left hemi, in addition to regions that will by default be excluded in the slice attribute 'left_regions_excluded'
`exclude_hemisphere`	(bool, default = TRUE); excludes the contralateral hemisphere from one indicated in slice attribute
`exclude_layer_1`	(bool, default = TRUE); excludes all counts from layer 1 (TEMPORARY, may not be hardcoded in later)
`include_right_regions`	(str vector, default = NULL) Acronyms of regions to include from the right hemi; if not NULL, takes precedence over `exclude_right_regions` & all other regions will be excluded. Typically, this is used for slices with poor quality/lots of tears.
`include_left_regions`	(str vector, default = NULL) Acronyms of regions to include from the light hemi; if not NULL, takes precedence over `exclude_left_regions` & all other regions will be excluded. Typically, this is used for slices with poor quality/lots of tears.
`simplify_regions`	(bool, default = TRUE) simplify the normalized region counts based on keywords in the internal function, `simplify_keywords`

`simplify_keywords`	(str vec, default = c("layer","part","stratum","division")). Keywords to search through region names and simplify to parent structure. This means the parent structure is also excluded if the list of excluded right and left regions can be further
`plot_filtered`	(bool, default = TRUE) plot the segmented cells following the filtering process
`slice_ID`	(str) ID of slice
`hemisphere`	(str) 'left', 'right' or NULL (both)

Value

a mouse or slice object

m mouse object

Examples

```
## Not run:
 s <-  exclude_anatomy(s, channels = c('cfos', 'eyfp', 'colabel'), clean = TRUE,
 exclude_regions = NULL, exclude_hemisphere = TRUE, exclude_layer_1 = TRUE, plot_filtered = TRUE)

## End(Not run)
## Not run:
m <- exclude_anatomy(m, slice_ID = "1_10",
hemisphere = NULL, channels = c('cfos', 'eyfp', 'colabel'), clean = TRUE,
exclude_regions = NULL, exclude_hemisphere = TRUE, exclude_layer_1 = TRUE

## End(Not run)
```

`exclude_by_acronym` *Excluded user chosen regions by entering acronyms*

Description

Excluded user chosen regions by entering acronyms

Usage

```
exclude_by_acronym(
  e,
  acronyms = "fiber_tracts",
  ontology = "allen",
  channels = NULL
)
```

Arguments

`e`	experiment object
`acronyms`	(str, default = "fiber_tracts") vector of region/structure acronyms to exclude from the datasets, e.g. c("CBL", "sox", "3V")
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`channels`	(str, default = NULL) NULL option processes all channels. `channels = c("cfos", "eyfp")` specifies exact channels.

Value

e experiment object

Examples

```
## Not run:
e <- exclude_by_acronym(e, acronyms = "CBL") # exclude the cerebellum

## End(Not run)
```

`exclude_by_keyword` *Excluded user chosen regions by keywords found in long-form name*

Description

Excluded user chosen regions by keywords found in long-form name

Usage

```
exclude_by_keyword(e, keywords, channels = NULL)
```

Arguments

`e`	experiment object
`keywords`	(str) vector of region/structure keywords to match and exclude from the datasets, e.g. c("nerve", "tract")
`channels`	(str, default = NULL) NULL option processes all channels. `channels = c("cfos", "eyfp")` specifies exact channels.

Value

e experiment object

Examples

```
## Not run:
e <- exclude_by_keyword(e, keywords = "tract")

## End(Not run)
```

`exclude_redundant_regions` *Exclude redundant regions*

Description

Your dataset may contain redundant information because it includes counts at different "ontological" resolutions. SMARTTR initially operates at the highest "resolution" and later on, can fold subregions into parent regions later. Therefore redundant counts from parent regions should be removed from datasets using this function prior to analysis and plotting functions.

Usage

```
exclude_redundant_regions(e, ontology = "allen", channels = NULL)
```

Arguments

`e`	exoeriment object
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`channels`	(str, default = NULL) NULL option processes all channels. `channels = c("cfos", "eyfp")` specifies exact channels.

Details

To check and see redundant regions contained in a list of acronyms, see the function `check_redundant_parents()`.

Value

e experiment object

Examples

```
## Not run:
e <- exclude_redundant_regions(e)

## End(Not run)
```

`experiment` *Create an experiment object*

Description

experiment() constructs an S3 object of class 'wb_experiment'. An experiment object consists of a list of processed mouse objects with raw data from slices omitted, and experimental attributes stored as a list.

Usage

```
experiment(
  experiment_name = NULL,
  experimenters = NULL,
  channels = NULL,
  experiment_groups = NULL,
  drug_groups = NULL,
  sex_groups = NULL,
  cohorts = NULL,
  strains = NULL,
  genotypes = NULL,
  reporters = NULL,
  ages = NULL,
  output_path = "set output path for your experiment",
  ...
)
```

Arguments

`experiment_name`	(str, default = NULL)
`experimenters`	(str, default = NULL)
`channels`	(str, default = NULL) Autogenerated with `add_mouse()` function. Will detect all unique channels stored in a mouse object.
`experiment_groups`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`drug_groups`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`sex_groups`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`cohorts`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`strains`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`genotypes`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`reporters`	(str, default = NULL) Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`ages`	(str, default = NULL). Autogenerated with `add_mouse()` function. Must exactly match the string values from the mouse objects.
`output_path`	(str, default = 'set output path for your experiment') Where to save the RData file for your experiment object
`...`	additional custom keyword pair attributes you'd like to store

Details

The experimental attributes can be assigned as arguments to the experiment constructor function. See the parameters listed for the default values for these attributes Note that you are able to add custom attributes as keyword pairs, if you would like to keep track of an additional piece of information. However, this will only serve a descriptive purpose and will not be used for analysis. You may not need to use all experimental attributes but fill out as many are applicable to your experiment.

Value

An experiment, a colloquial term for an object of class 'wb_experiment'. An 'experiment' object is also a list, with class list.

See Also

See also `mouse()` for the description of a mouse object and it's attributes.

Examples

```
my_experiment <- experiment() # constructs an experiment object
```

---

export_permutation_results

Export the permutation results as a csv file. This automatically saves into the tables folder.

Description

Export the permutation results as a csv file. This automatically saves into the tables folder.

Usage

```
export_permutation_results(
  e,
  permutation_groups = "all",
  channels = c("cfos"),
  ontology = "allen",
  filter_significant = TRUE
)
```

Arguments

`e`	experiment object
`permutation_groups`	(str vec, default = "all") Default is to export all analyses run. Supply names of the specific analyses found under `e$permutation_p_matrix %>% names()` if you want to export specific groups.
`channels`	(str, channels = c("cfos", "eyfp", "colabel") The channels to process.
`ontology`	(str, default = "allen") Set to "unified" for Kim Lab unified ontology
`filter_significant`	(bool, default = TRUE) If FALSE, this keeps all comparisons. Otherwise exports only the most different and significant permutations.

Examples

```
## Not run:
e <- export_permutation_results(e, permutation_groups = "all", filter_significant = TRUE)

## End(Not run)
```

`filter` *A filter with parameters for registration used in wholebrain functions.*

Description

A filter with parameters for registration used in wholebrain functions.

Usage

```
filter
```

Format

A list of parameters to filter features of interest in an image

alim price, in US dollars

threshold.range weight of the diamond, in carats

eccentricity eccentricity (elongation) of contours sets how round you want cell bodies to be. Default is 500 and smaller values equal to more round.

Max Maximum value to display in the 8-bit rendered (sets sort of brightness contrast)

Min Minimum value to display in the 8-bit rendered (sets sort of brightness contrast)

brain.threshold the exact value where you want to start segmeting the brain outline in autofluorescence

resize resize parameter to match the atlas to your pixel resolution, should be between 0.03 and 0.2 for most applications.

blur blur parameter that sets the smoothness of the tissue outline, if magaded or jagged edges increase. Using a value fo 4 is usually recommended.

downsample downsample, default is set to 0.25 and images with a size of 15000 x 8000 pixels can then usually be run smoothly ...

filter_regions *Filters to chosen base parent regions and all child subregions*

Description

Filters to chosen base parent regions and all child subregions

Usage

```
filter_regions(
  e,
 base_regions = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CBL"),
  ontology = "allen",
  channels = NULL
)
```

Arguments

e	experiment object
base_regions	(default = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU","TH", "HY", "MB", "HB", "CBL")) Region acronyms of base parent regions (and all subregions) to include for analysis
ontology	(str, default = "allen") Options = "allen" or "unified".
channels	(str, default = NULL) NULL option processes all channels. `channels = c("cfos", "eyfp")` specifies exact channels.

Value

e experiment object

Examples

```
## Not run:
e <- filter_regions(e, "Isocortex", channels  = "cfos")

## End(Not run)
```

find_outlier_counts *Detect, log, and remove outlier counts. This function removes any normalized regions counts that are more than `n_sd` standard deviations (default = 2) higher than their cohort mean.*

Description

Detect, log, and remove outlier counts. This function removes any normalized regions counts that are more than `n_sd` standard deviations (default = 2) higher than their cohort mean.

Usage

```
find_outlier_counts(
  e,
  by = c("group", "sex"),
  n_sd = 2,
  remove = FALSE,
  log = TRUE
)
```

Arguments

`e`	experiment object
`by`	(str, default = c("group", "sex")) The mice attributes used to group the datasets into comparison groups.
`n_sd`	(int, default = 2). Number of standards deviations above and below which categorizes outliers.
`remove`	(bool, default = FALSE) Remove all outlier rows from the combined normalized counts dataframe in the experiment object.
`log`	(bool, default = TRUE) Save the logged outlier regions into a csv file in the output folder.

Value

e experiment object. Outlier counts in the experiment object are removed if remove = TRUE.

Examples

```
## Not run:
e <- find_outlier_counts(e, by = c("group","sex"), n_sd = 2, remove = FALSE, log = TRUE)

## End(Not run)
```

`get_cell_table` *Get cell tables*

Description

This function stores a list in a mouse object that is the length of the channels parameter. Each element of the list is a dataframe containing combined cell counts for one channel across all slices processed for that mouse. By default, if one slice has no dataset processed for that particular channel, that slice will be skipped over. The function will run properly but a warning will be thrown indicating you should go back and generate a mapped dataset for that particular slice and channel.

Usage

```
get_cell_table(m, channels = c("cfos", "eyfp", "colabel"))
```

Arguments

`m`	mouse object
`channels`	(vec, default = c("cfos", "eyfp", "colabel")) Channels to process.

Value

m mouse object

Examples

```
## Not run:
m <- get_cell_tables(m, channels = c("cfos", "eyfp", "colabel"))

## End(Not run)
```

`get_correlations` *Get regional cross correlations and their p-values in a correlation list object.*

Description

This analysis will get regional cross correlations based on cell counts normalized by region volume.

Usage

```
get_correlations(
  e,
  by,
  values,
  channels = c("cfos", "eyfp", "colabel"),
  p_adjust_method = "none",
  alpha = 0.05,
  ontology = "allen",
  method = "pearson",
 anatomical.order = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CB"),
  region_order = NULL
)
```

Arguments

`e`	experiment object
`by`	(str) Attribute names to group by, e.g. c("sex", "group")
`values`	(str) The respective values of the attributes entered for the by parameter to generate a specific analysis group, e.g.values = c("female", "AD"). Length must be the same as by.
`channels`	(str, channels = c("cfos", "eyfp", "colabel") The channels to process.
`p_adjust_method`	(bool or str, default = "none") This parameter is fed into the p.adjust function. Options: c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none") Apply the named method to control for the inflated false discovery rate or family wise error rate (FWER). Set to FALSE or "none" to keep "raw" p values. See also `stats::p.adjust()` for the correction options.
`alpha`	(num, default = 0.05) The alpha level for significance applied AFTER p-adjustment.

`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`method`	(str, default = "pearson", options = c("pearson", "spearman")) Specifies the type of correlations to compute. Spearman correlations are the Pearson linear correlations computed on the ranks of non-missing elements,
`anatomical.order`	(str, default = c("Isocortex","OLF","HPF","CTXsp","CNU","TH","HY","MB","HB","CB")) The order of the regions to plot in the heatmaps. using midranks for ties. See also `Hmisc::rcorr()`
`region_order`	(list, default = NULL) optional list with the first element named "acronym" supplying a vector as region acronyms and the second element named "order" supplying an vector of integers determining numerical order, e.g. 1, 1, 2, 2.

Value

e experiment object. The experiment object now has a named `correlation_list` object stored in it. The name of the correlation object is the concatenation of the variable values separated by a "_". This name allows for unambiguous identification of different analysis subgroups in the future.

See Also

`Hmisc::rcorr()`

Examples

```
## Not run:
e <- get_correlations(e, by = c("sex", "group"), values = c("female", "AD"),
channels = c("cfos", "eyfp", "colabel"), alpha = 0.05)

## End(Not run)
```

`get_percent_colabel` *Get the percentage of colabelled cells over either cfos or eyfp channels.*

Description

This analysis will only include common regions that are included in both the colabelled and cfos or eyfp channels. The colabelled percentage of individual animals will be calculated with the option to export the data.

Usage

```
get_percent_colabel(
  e,
  by,
  colabel_channel = "colabel",
  channel = "eyfp",
  save_table = TRUE,
  rois = NULL,
  individual = TRUE
)
```

Arguments

`e`	experiment object
`by`	(str) Attribute names to group by, e.g. by = c("group", "sex"). These will generate analysis subgroups that are averaged together to assess across all rois.
`colabel_channel`	(str, default = "colabel") The channel used as the numerator in fraction counts. The string 'colabel' in the pipeline refers to colocalized 'eyfp' and 'cfos' channels. For other colocalized channels, import the channel using `import_segmentation_custom()` or your own customized import channel.
`channel`	(str, default = "eyfp") The channel used as denominator in fraction counts.
`save_table`	(bool, default = TRUE) Whether to save the output table as a csv in the experiment object output folder.
`rois`	(str, default = NULL) Whether to generate colabelled percentages for only specific regions of interest, e.g. rois = c("HY", "DG"). Child regions of specified rois will also be searched for.
`individual`	(bool, default = FALSE) Whether the data should include individual mouse colabelled percentages rather than the average. If FALSE the colabel percentages are averaged across all analysis subgroups determined by the by parameter

Value

e experiment object with colabelled percentage table stored in it.

Examples

```
## Not run:
e <- get_percent_colabel(e, c("group", "sex"), channel = "eyfp"))

## End(Not run)
```

`get_registered_volumes` *get_registered_volumes (generic function)*

Description

Calculate the registered area (microns^2^) and the regional volumes (microns^3^) of all regions contained in a slice.

Note: Simplification of the analyzed regions by keywords is highly recommended because there are errors in the wholebrain basecode that results in a mismatch between the region acronym mapped to and the actual registration contour based on the region acronym. This mismatch is most notable in the dentate gyrus subregions. If simplification by keywords is used, this will circumvent the errors.

This function also automatically removes parent regions that are redundant, e.g. "CTX" should by volumetrically represented by summing all subregions, but there is a tiny amount of potential space that allows for cells to get mapped to slim spaces between subregions. This potential anatomical space should be ignored.

Calculate the registered area (microns^2^) and the regional volumes (microns^3^) of all regions contained in a slice. Note: Simplification of the analyzed regions by keywords is HIGHLY RECOMMENDED because there are errors in the wholebrain basecode that results in a mismatch between the region acronym mapped to and the actual registration contour based on the region acronym. This mismatch is most notable in the dentate gyrus subregions, where certain regions are represented twice because the DG curve along the rostral caudal axis. If simplification by keywords is used, this will circumvent the errors.

This function also automatically removes parent regions that are redundant, e.g. "CTX" should by volumetrically represented by summing all subregions, but there is a tiny amount of potential space that allows for cells to get mapped to slim spaces between subregions. This potential anatomical space should be ignored.

Usage

```
get_registered_volumes(x, ...)

## S3 method for class 'slice'
get_registered_volumes(
  x,
  simplify_regions = TRUE,
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  ...
)

## S3 method for class 'mouse'
get_registered_volumes(
  x,
  slice_ID,
  hemisphere = NULL,
  simplify_regions = TRUE,
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  replace = FALSE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	further arguments passed to or from other methods.
`simplify_regions`	(bool, default = TRUE) simplify the normalized region counts based on keywords in the internal function, `simplify_keywords`
`simplify_keywords`	(str vec, default = c("layer","part","stratum","division")). Keywords to search through region names and simplify to parent structure
`slice_ID`	(str) ID of slice
`hemisphere`	(str) 'left', 'right' or NULL (both)
`replace`	(bool, default = FALSE). Replace previously calculated volumes for a particular slice.

Value

a mouse or slice object

s slice object with a stored dataframe with columns 'name' (full region name), 'acronym', 'area' (in microns^2^), 'volume' (in microns^3^) 'right.hemisphere'

m mouse object

Examples

```
## Not run:
s <- get_registered_volumes(s)

## End(Not run)
## Not run:
m <- get_registered_volumes(m, slice_ID = "1_10", hemisphere = "left", replace = FALSE)

## End(Not run)
```

---

import_mapped_datasets

Import externally mapped datasets into an experiment

Description

This function takes an experiment object and imports externally mapped datasets.

Usage

```
import_mapped_datasets(e, normalized_count_paths, ...)
```

Arguments

`e`	experiment object
`normalized_count_paths`	(str vec, default = NULL, optional) For importing external datasets ONLY. A character vector, with each element being a path to the respective .csv or .xlsx file in the same order as the `channels` attribute. Must have the same order and length as the `channels` parameter. You can add channels as names to the vector elements to avoid ambiguity and ensure correct importation for the right channel.
`...`	additional parameters to pass to either the `readr::read_csv()` function or `readxl::read_excel()`

Value

e an experiment object with the imported dataset

`import_segmentation_custom`
import_segmentation (generic function)

Description

Custom method for importing segmentation data for a slice object. This is a flexible method for importing channels aside from `cfos` and `eyfp` that use the same ImageJ segmentation macros provided in the SMARTTR pipeline. This method works best when following saving segmentation data with the same naming conventions using the 3D Roi Manager within the 3D ImageJ Suite. Segmentation data is saved in two'.txt' files which output the results of the Measure 3D and Quantif 3D options of the 3D Roi Manager plugin,respectively. A description of what each of those these options measure is provided in the online documentation.

The naming conventions for the ".txt" file storing the Quantif3D results are `Q_*_{channel}_*_{channel}.txt`, where the `*` indicates a wildcard character(s) and `{channel}` is the channel name without brackets. E.g. "Q_G_eYFP_258_1_1_eYFP.txt"

The naming conventions for the ".txt" file storing the Measure 3D are `M_*_{channel}_*.txt` where the `*` indicates a wildcard character(s) and `{channel}` is the channel name without brackets. E.g "M_G_eYFP_258_1_1.txt".

The wildcards characters may be used to store things like date or slice naming information.

The locations of these files must be specified in the `slice_directory` attribute of the slice_object. Otherwise, the root folder containing the registration image is searched. This attribute can be stored when initializing the slice object or can be edited afterwards.

Method for custom importation of segmentation data for a mouse object

Usage

```
import_segmentation_custom(x, ...)

## S3 method for class 'slice'
import_segmentation_custom(
  x,
  channel,
  x_col = NULL,
  y_col = NULL,
  meas_path = NULL,
  quant_path = NULL,
  ...
)

## S3 method for class 'mouse'
import_segmentation_custom(
  x,
  channel,
  slice_ID = NA,
  hemisphere = NULL,
  x_col = NULL,
  y_col = NULL,
  meas_path = NULL,
  quant_path = NULL,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	further arguments passed to or from other methods.
`channel`	(str) channel to import.
`x_col`	(int, optional) The column index of the x pixel location in the txt file result from Measure 3D.
`y_col`	(int, optional) The column index of the y pixel location in the txt file result from Measure 3D.
`meas_path`	(chr, optional, default = NULL). Relative path to file from current directory or absolute path. If NULL, will automatically search for the custom files in the slice directory based on the slice_ID and channel parameters.
`quant_path`	(chr, optional, default = NULL).Relative path to file from current directory or absolute path. If NULL, will automatically search for the custom files in the slice directory based on the slice_ID and channel parameters.
`slice_ID`	(str) ID of slice
`hemisphere`	(str)'left', 'right' or NULL

Value

a mouse or slice object

s slice object

m mouse object

Examples

```
## Not run:
s <-  import_segmentation_custom(s, mouse_ID = "255", channel = "cfos")

## End(Not run)
## Not run:
m <-  import_segmentation(m, slice_ID = "1_10", channels = c("PV"), replace = FALSE)

## End(Not run)
```

`import_segmentation_ij`
import_segmentation (generic function)

Description

Method for importing segmentation data for a slice object

Method for importing segmentation data for a mouse object

Usage

```
import_segmentation_ij(x, ...)

## S3 method for class 'slice'
import_segmentation_ij(x, mouse_ID = NA, channels = NULL, maxdist = 10, ...)

## S3 method for class 'mouse'
import_segmentation_ij(
  x,
  slice_ID = NA,
  hemisphere = NULL,
  channels = NULL,
  maxdist = 10,
  replace = FALSE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	further arguments passed to or from other methods.
`mouse_ID`	(str) ID of mouse
`channels`	(str vector, default = NULL) channels to import. If NULL, defaults to the channels stored in the slice object attributes.
`maxdist`	(int, default = 10) maximum tolerability of character differences to match the string names of the importation files
`slice_ID`	(str) ID of slice
`hemisphere`	(str)'left', 'right' or NULL
`replace`	(bool, default = FALSE) replace existing raw segmentation data

Value

a mouse or slice object

s slice object

m mouse object

Note

The designated `colabel` channel name in this pipeline will auto import the output of the batch_3D_MultiColocalization.ij macro provided in the pre-processing pipeline. If you have a separate method used for detecting colabelled cells, please use a different naming convention for this channel, e.g. "colabel_PV_cfos", and import using a customized import function such as `import_segmentation_custom()`.

Examples

```
## Not run:
s <-  import_segmentation(s, mouse_ID = "255")
s <-  import_segmentation(s, mouse_ID = "255",
channels = c("cfos", "eyfp", "colabel"))
```

```
## End(Not run)
## Not run:
m <-  import_segmentation(m, slice_ID = "1_10",
channels = c("cfos", "eyfp", "colabel"), replace = FALSE)

## End(Not run)
```

---

make_segmentation_object

make_segmentation_object (generic funciton)

Description

Make a wholebrain compatible segmentation object for a slice in a slice object

Make a wholebrain compatible segmentation object for a slice in a mouse object

Usage

```
make_segmentation_object(x, ...)

## S3 method for class 'slice'
make_segmentation_object(
  x,
  mouse_ID = NA,
  channels = NULL,
  use_filter = FALSE,
  ...
)

## S3 method for class 'mouse'
make_segmentation_object(
  x,
  slice_ID = NA,
  hemisphere = NULL,
  channels = NULL,
  replace = FALSE,
  use_filter = FALSE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	(optional) additional volume and overlap parameters for get.colabeled.cells().
`mouse_ID`	(str) ID of mouse
`channels`	(str vector, default = NULL) Channels to process. If NULL, defaults to the channels stored in the slice object attributes.
`use_filter`	(bool, default = FALSE). Use a filter to create more curated segmentation object from the raw segmentation data.
`slice_ID`	(str) ID of slice

`hemisphere`	(str, default = NULL) 'left', 'right' or NULL (both)
`replace`	(bool, default = FALSE) replace existing raw segmentation data

Value

a mouse or slice object

s slice object

Note

If you are processing the colabel channel, the X and Y positions of colabelled cells are the average of the x,y centroid coordinates of the colabelled objects

Examples

```
## Not run:
s <- make_segmentation_object(s, mouse_ID = "255",
channels = c("cfos", "eyfp"), use_filter = FALSE)

## End(Not run)
## Not run:
m <-  make_segmentation_object(m, slice_ID = '1_9', hemisphere = 'left',
channels = c('eyfp', 'cfos', 'colabel'), use_filter = FALSE)

## End(Not run)
```

`map_cells_to_atlas` *map_cells_to_atlas (generic function)*

Description

Method for forward warping segmentation data to atlas space for a slice object. Requires segmentation objects to be made for channels specified and a registration object.

Method for forward warping segmentation data to atlas space for a slice within a mouse object. Requires segmentation objects to be made for the channels specified and a registration.

Usage

```
map_cells_to_atlas(x, ...)

## S3 method for class 'slice'
map_cells_to_atlas(
  x,
  channels = NULL,
  clean = TRUE,
  display = TRUE,
  mouse_ID = NULL,
  ...
)

## S3 method for class 'mouse'
```

```
map_cells_to_atlas(
  x,
  slice_ID = NA,
  hemisphere = NULL,
  channels = NULL,
  clean = TRUE,
  display = TRUE,
  replace = FALSE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	additional parameters besides 'registration', 'segmentation', 'forward.warps', and 'device' to pass to the `wholebrain::inspect.registration()` function
`channels`	(str vector, default = NULL) Channels to process. If NULL, defaults to the channels stored in the slice object attributes.
`clean`	(bool, default = TRUE). Remove cells that don't map to any regions.
`display`	(bool, default = TRUE). Display the results of the forward warp for the slice.display
`mouse_ID`	(str) mouse ID
`slice_ID`	(str) ID of slice
`hemisphere`	(str) 'left', 'right' or NULL (both)
`replace`	(bool, default = FALSE). Replace current forward warped data, both raw and cleaned.

Value

a mouse or slice object

m mouse object

Examples

```
## Not run:
s <- map_cells_to_atlas(s, channels c('cfos' , 'eyfp', 'colabel'),
clean = TRUE, display = TRUE, mouse_ID = "255")

## End(Not run)
## Not run:
 m <- map_cells_to_atlas(m, slice_ID = "1_10", hemisphere = NULL,
 channels = c("cfos", "eyfp", "colabel"), clean = TRUE, replace = FALSE)

## End(Not run)
```

`mouse` *Create a mouse object*

Description

mouse() constructs an S3 object of class 'mouse'. A mouse object consists of a list of slice objects and attributes stored as a list.

The slice objects are added to a mouse object with the function `add_slice()`. Each slice is a named element in the mouse object list, with the naming convention dependent on the slice ID and hemisphere attributes of the slice object.

If you are processing either a left or right hemisphere, the slice is named with the convention: "slice_ID" appended with its "hemisphere" If the hemisphere attribute is NULL, i.e. if the whole slice aligns well with a single atlas plate and there is no need to create separate slice objects per hemisphere, then the slice is named with the convention: "slice_ID"

Usage

```
mouse(
  mouse_ID = "set ID",
  sex = "female",
  age = NULL,
  genotype = NULL,
  reporter = NULL,
  strain = NULL,
  experiment = NULL,
  group = NULL,
  drug = NULL,
  cohort = NULL,
  input_path = "set input path",
  output_path = "set output path",
  ...
)
```

Arguments

`mouse_ID`	(str, default = 'set ID') e.g. '1_1'
`sex`	(str, default = "female")
`age`	(str, default = NULL)
`genotype`	(str, default = NULL)
`reporter`	(str, default = NULL)
`strain`	(str, default = NULL) e.g. 'B6'
`experiment`	(str, default = NULL) e.g. 'sundowning'
`group`	(str, default = NULL) e.g. 'control' or 'AD'
`drug`	(str, default = NULL) e.g. 'vehicle' or 'ketamine'
`cohort`	(str, default = NULL)

`input_path`	(str, default = 'set input path') Root to path containing the mouse data and slice image subfolders. This is useful if you've have changed computers and the drive mapped to the data has slightly changed. TODO: currently not used. Will include search function to parse out files and individual slice information for a mouse.
`output_path`	(str, default = 'set output path') Set the path to the folder you want to save the mouse RDATA file to.
`...`	additional custom keyword pair attributes you'd like to store

Details

The mouse attributes can be assigned as arguments to the mouse constructor function. See the parameters listed for the default values for these attributes Note that you are able to add custom attributes as keyword pairs, if you would like to keep track of an additional piece of information. However, this will only serve a descriptive purpose and will not be used for analysis. You may not need to use all mouse attributes but fill out as many are applicable to your experiment.

Value

A mouse, a colloquial term for an object of class 'mouse'. A 'mouse' object is also a list, with class list.

See Also

See also `slice()` for the description of a slice object and it's attributes. See also `add_slice()` for the description of how to add a slice object to a mouse object.

Examples

```
mouse_example <- mouse() # initializes a mouse object
```

`normalize_cell_counts` *Normalize cell counts per mm^2^ or by mm^3^ (if multiplying by the stack size).*

Description

Run this function after all the slices that you want to process are finished being added and you have combined your cell counts with `get_cell_table()`. This functions process all channels where a cell table was made using the latter function.

Usage

```
normalize_cell_counts(
  m,
  combine_hemispheres = TRUE,
  simplify_regions = TRUE,
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  split_hipp_DV = TRUE,
  DV_split_AP_thresh = -2.7
)
```

Arguments

`m`	mouse object
`combine_hemispheres`	(bool, default = TRUE) Combine normalized cell counts from both hemispheres
`simplify_regions`	(bool, default = TRUE) simplify the normalized region counts based on keywords in the internal function, `simplify_keywords`
`simplify_keywords`	(str vec, default = c("layer","part","stratum","division")). Keywords to search through region names and simplify to parent structure
`split_hipp_DV`	(bool, default = TRUE) Split the subregions of the CA1, CA2, CA3, and DG based on a specified AP coordinate cutoff. This is because the Allen atlas doesn't have a dorsal/ventral region designation for these ROIs.
`DV_split_AP_thresh`	(numeric, default = -2,7) The specified AP coordinate threshold to split hippocampal cell counts into dorsal and ventral.

Value

m mouse object

Examples

```
## Not run:
m <- normalize_cell_counts(m, combine_hemispheres = TRUE, simplify_regions = TRUE)

## End(Not run)
```

`normalize_colabel_counts` *Normalize colabel counts over a designated denominator channel.*

Description

This function can only be run after running `combine_cell_counts()`. It divides the colabelled cell counts by a designated normalization channel to provide a normalized ratio. Please note that the areas and volumes cancel out in this operation. This is not designed to work on multiple hemispheres. Please combine cell counts across multiple hemispheres when you run `normalize_cell_counts()`.

Usage

```
normalize_colabel_counts(e, denominator_channel = "eyfp")
```

Arguments

`e`	experiment object
`denominator_channel`	(str, default = "eyfp") The exact name of the channel used for normalization

Value

e An experiment object with a new dataframe with the normalized ratios of colabelled counts over the designated denominator counts. Because the volumes and region areas cancel out, the values of count, normalized.count.by.area, and normalized.count.by.volume are all the same. This is to provide a consistent input dataframe into the analysis functions.

See Also

`combine_cell_counts()` & `normalize_cell_counts()`

Examples

```
## Not run:
e <- normalize_colabel_counts(e, denominator_channel = "eyfp")

## End(Not run)
```

`ontology` *Ontology*

Description

A custom adjustment of the Allen Common Coordinate Framework where the hippocampus (CA1, CA2, CA3, and DG) and it's subregions are split into dorsal and ventral regions and acronyms. They are given unique IDs and parent ids except for the c("dCA1", "dCA2", "dCA3", "dDG") and c("vCA1", "vCA2", "vCA3", "vDG"), whose parents are c(CA1, CA2, CA3, and DG) respectively.

Usage

```
ontology
```

Format

A dataframe

`ontology.unified` *Unified Kim ontology*

Description

The combined ontology created by the Yongsoo Kim lab. Combines nomenclature of the two most-used mouse brain atlases, the Franklin and Paxinos (FP) and the common coordinate framework (CCF) from the Allen Institute for Brain Science.

Usage

```
ontology.unified
```

Format

A dataframe

`parallel_coordinate_plot` *Create a parallel coordinate plot*

Description

Plot the correlation difference between two comparison groups into a parallel coordinate plot. The function `correlation_diff_permutation()` must be run first in order to generate results to plot.

Usage

```
parallel_coordinate_plot(
  e,
  permutation_comparison = "AD_vs_control",
  channels = c("cfos", "eyfp", "colabel"),
  colors = c("#be0000", "#00782e", "#f09b08"),
  x_label_group_1 = NULL,
  x_label_group_2 = NULL,
  height = 10,
  width = 10,
  print_plot = FALSE,
  save_plot = TRUE,
  reverse_group_order = FALSE,
  force = 1,
  plt_theme = NULL,
  label_size = 30,
  image_ext = ".png",
  nudge_x = 2:5
)
```

Arguments

`e`	experiment object
`permutation_comparison`	The name of the correlation group comparisons to plot.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) channels to plot
`colors`	(str, default = c("#be0000", "#00782e", "#f09b08")) Hexadecimal codes corresponding to the channels (respectively) to plot.
`x_label_group_1`	(str, NULL) The label for the first group in the permutation analysis. Note: this is to customize the graph labels. It does not reverse the group order.
`x_label_group_2`	(str, NULL) The label for the second group in the permutaiton analysis. Note: this is to customize the graph labels. It does not reverse the group order.
`height`	height of the plot in inches.
`width`	width of the plot in inches.
`print_plot`	(bool, default = FALSE) Whether to display the plot (in addition to saving the plot)

`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.
`reverse_group_order`	(bool, default = TRUE) Reverse the order of the groups on the x-axis.
`force`	(default =1) Force of the text repel between text labels.
`plt_theme`	(default = NULL) Add a `ggplot2::theme()` to the plot. If NULL, the default is taken.
`label_size`	(default = 30) Default font size for region labels.
`image_ext`	(default = ".png") image extension to save the plot as.
`nudge_x`	(vec, default = 2:5) a vector determining the jitter between labels.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p_list <- parallel_coordinate_plot(e, permutation_comparison = "Context_vs_Shock",
 channels = "cfos", colors ="#be0000", x_label_group1 = "Context", x_label_group_2 = "Shock")

## End(Not run)
```

`plot_betweenness_regions`
Plot the betweenness distributions across regions

Description

Bar plot of betweenness per region in descending magnitude

Usage

```
plot_betweenness_regions(
  e,
  channels = c("cfos", "eyfp"),
  colors = c("red", "green"),
  network = "AD",
  title = "",
  height = 10,
  width = 20,
  ylim = c(0, 15),
  filter_isolates = TRUE,
  sort_super_region = FALSE,
  region_label_angle = 60,
  label_text_size = 12,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE,
  theme.bar = NULL
)
```

Arguments

`e`	experiment object
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`colors`	(str, default =) String vector of hexadecimal color codes corresponding to to each channel plotted.
`network`	(str, default = "AD") Which network to plot the betweenness distribution across regions
`title`	(str, default = "")
`height`	(int, default = 15) Height of the plot in inches.
`width`	(int, default = 20) Width of the plot in inches.
`ylim`	(vec, default = c(0,15))axes limits of y-axis
`filter_isolates`	(default = TRUE) Avoid plotting isolated nodes (zero value)
`sort_super_region`	(bool, default = FALSE) Whether to divide into subfacets based on which parent region
`region_label_angle`	(int, default = 60) Angle of region labels.
`label_text_size`	(int, default = 12) Font size of region labels.
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.
`theme.bar`	User option to use their own ggplot theme the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_betweenness_regions(e, colors ="#660000", channels = "cfos", region_label_angle = 60,
ylim = c(0, 50), image_ext = ".png")

## End(Not run)
```

plot_correlation_heatmaps *Plot correlation heatmaps*

Description

Plot correlation heatmaps

Usage

```
plot_correlation_heatmaps(
  e,
  correlation_list_name,
  channels = c("cfos", "eyfp", "colabel"),
  colors = c("#be0000", "#00782e", "#f09b08"),
  sig_color = "yellow",
  sig_nudge_y = -0.7,
  sig_size = 7,
  ontology = "allen",
 anatomical.order = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CB"),
  print_plot = FALSE,
  save_plot = TRUE,
  image_ext = ".png",
  plot_title = NULL,
  height = 10,
  width = 10,
 theme.hm = ggplot2::theme(axis.text.x = element_text(hjust = 1, vjust = 0.5, angle =
   90, size = 8), axis.text.y = element_text(vjust = 0.5, size = 8), plot.title =
    element_text(hjust = 0.5, size = 36), axis.title = element_text(size = 18),
   legend.text = element_text(size = 22), legend.key.height = unit(100, "points"),
    legend.title = element_text(size = 22), panel.spacing = unit(0.2, "lines"),
    strip.text.x = element_text(angle = 0, hjust = 0.5, vjust = 0.5, size = 10),
    strip.text.y = element_text(angle = 270,
     hjust = 0.5, vjust = 0.5, size = 10),
   strip.placement = "outside", strip.background = element_rect(color = "black", fill =
    "lightblue"))
)
```

Arguments

`e`	experiment object. Must contain a named correlation_list object generated by `get_correlations()`
`correlation_list_name`	(str) The name of the correlation object generated by `get_correlations()`
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Must exist in the channels attribute of the correlation_list.
`colors`	(str, default = c("#be0000", "#00782e", "#f09b08")) Hexadecimal code for the colors corresponding channels parameter. Color values can also be input compatible with ggplot2 plotting functions.

`sig_color`	(str, default = "yellow") Color of the significance symbol in R
`sig_nudge_y`	(default = -0.7) Relative amount to nudge the significance symbols in the y direction to center over each square.
`sig_size`	(default = 7) Point size for significance symbol.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`anatomical.order`	(default = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU","TH", "HY", "MB", "HB", "CB")) Default way to group subregions into super regions order
`print_plot`	(bool, default = FALSE) Print the plot as graphics windows.
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.
`image_ext`	(default = ".png") image extension to the plot as.
`plot_title`	(str, default = NULL) If NULL, the `correlation_list_name` will used as the title with underscores removed.
`height`	(int) Height of the plot in inches.
`width`	(int) Width of the plot in inches.
`theme.hm`	Option to use custom ggplot2 theme if the user wants. See default values as example.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

See Also

`get_correlations()`

Examples

```
## Not run:
plot_correlation_heatmaps(e, correlation_list_name = "female_AD")

## End(Not run)
```

`plot_degree_distributions` *Plot the degree distributions*

Description

Plot a stacked bar plot of the degree distributions.

Usage

```
plot_degree_distributions(
  e,
  channels = c("cfos", "eyfp"),
  color_palettes = c("reds", "greens"),
  colors_manual = NULL,
  labels = c(female_AD = "female_AD_label", female_control = "female_control_label"),
  title = "my_title",
  height = 15,
  width = 15,
  xlim = c(0, 20),
  ylim = c(0, 15),
  image_ext = ".png",
  print_plot = FALSE,
  theme.gg = NULL,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`channels`	(str, default = c("cfos", "eyfp")) Channels to plot.
`color_palettes`	(str, default = c("reds", "greens")) Color palettes from grDevices::hcl.colors that are used to for plotting networks for each channel, respectively.
`colors_manual`	(str, default = NULL) Manually choose the hexadecimal color codes to create a custom color palette, e.g. colors_manual = c("#660000", "#FF0000", "#FF6666"). Warning: this color will be applied to all channels. It's recommended to set the channels parameter to a single channel if this parameter is used.
`labels`	(e.g. labels = c(network1_name = "network 1 label", network2_name = "network 2 label)) The legend labels to correspond with your respective network names.
`title`	(str, default = "my_title") the experiment object output folder.
`height`	(int, default = 15) Height of the plot in inches.
`width`	(int, default = 15) Width of the plot in inches.
`xlim`	(vec, default = c(0,20)) axes limits x-axis
`ylim`	(vec, default = c(0,15))axes limits of y-axis
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.
`theme.gg`	(default = NULL) Option to use custom ggplot2 theme if the user wants
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_degree_distributions(e, channels = "cfos",
labels = c(female_AD = "female_AD_label", female_control = "female_control_label"),
title = "my title", image_ext = ".png")

## End(Not run)
```

`plot_degree_regions` *Plot the degree distributions across regions*

Description

Bar plot of degree per region in descending magnitude

Usage

```
plot_degree_regions(
  e,
  channels = c("cfos", "eyfp"),
  colors = c("red", "green"),
  network = "AD",
  title = "",
  height = 10,
  width = 20,
  ylim = c(0, 15),
  sort_super_region = FALSE,
  region_label_angle = 60,
  label_text_size = 12,
  filter_isolates = TRUE,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE,
  theme.bar = NULL
)
```

Arguments

`e`	experiment object
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`colors`	(str, default =) String vector of hexadecimal color codes corresponding to to each channel plotted.
`network`	(str, default = "AD") Which network to plot the degree distribution across regions
`title`	(str, default = "") Plot title.
`height`	(int, default = 15) Height of the plot in inches.
`width`	(int, default = 20) Width of the plot in inches.
`ylim`	(vec, default = c(0,15))axes limits of y-axis

`sort_super_region`	(bool, default = FALSE) Whether to divide into subfacets based on which parent region
`region_label_angle`	(int, default = 60) Angle of region labels.
`label_text_size`	(int, default = 12) Font size of region labels.
`filter_isolates`	(default = TRUE) Avoid plotting isolated nodes (zero value)
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.
`theme.bar`	User option to use their own ggplot theme the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_degree_regions(e, colors = c("#660000", "#FF0000"),
channels = "cfos", region_label_angle = 60,
ylim = c(0, 15), image_ext = ".png")

## End(Not run)
```

`plot_joined_networks` *Plot the networks stored in an experiment object*

Description

Plot the networks stored in an experiment object

Usage

```
plot_joined_networks(
  e,
  correlation_list_names = c("male_agg", "female_non"),
  title = NULL,
  channels = "cfos",
  absolute_weight = TRUE,
 edge_colors = c(male_agg_pos = "#06537f", male_agg_neg = "#526c7a", female_non_pos =
    "#C70039", female_non_neg = "#71585f"),
 edge_color_labels = c(male_agg_pos = "Positive male", male_agg_neg = "Negative male",
    female_non_pos = "Positive female", female_non_neg = "Negative female"),
  height = 15,
  width = 15,
  region_legend = TRUE,
  degree_scale_limit = c(1, 10),
  correlation_edge_width_limit = c(0.8, 1),
  image_ext = ".png",
  print_plot = FALSE,
  graph_theme = NULL,
  transparent_edge_group1 = TRUE,
  transparent_edge_group2 = FALSE,
  label_size = 5,
  label_offset = 0.15,
  edge_thickness_range = c(1, 5),
  node_size_range = c(1, 8),
  anatomical.colors = NULL,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`correlation_list_names`	(str vec) character vector of the two correlation lists used to include in a joined network, e.g., `correlation_list_names = c("male_agg", "female_non")`
`title`	(str, default = NULL) Title of network plot
`channels`	(str, default = c("cfos", "eyfp", "colabel"))
`absolute_weight`	(bool, default = TRUE) Whether to plot absolute weights. If TRUE, the edge_colors and edge_colors_label should not contain values for positive and negative correlations.
`edge_colors`	(vec) vector of hexidecimal codes as strings. Assign a group name to the vector element. e.g. c(male_agg_pos = "#06537f", male_agg_neg = "#526c7a", female_non_pos = "#C70039", female_non_neg = "#71585f")
`edge_color_labels`	(vec) vector of edge labels as strings. e.g. c(male_agg_pos = "Positive male", male_agg_neg = "Negative male", female_non_pos = "Positive female", female_non_neg = "Negative female")
`height`	Height of the plot in inches.
`width`	width of the plot in inches.
`region_legend`	(default = TRUE) Boolean determining whether or not to show the region legend categorizing subregions into their largest parent region. Only works well if the Allen ontology is used for the dataset.
`degree_scale_limit`	(vec, default = c(1,10)) Scale limit for degree size
`correlation_edge_width_limit`	(default = c(0.8, 1)) Range for the width size of the edges.
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output. the experiment object output folder.
`graph_theme`	(default = NULL) Add a `ggraph::theme_graph()` to the network graph. If NULL, the default is taken.

`transparent_edge_group1`	(bool) logical to render edges transparent
`transparent_edge_group2`	(bool) logical to render edges transparent
`label_size`	(default = 5) Default font size for network region labels.
`label_offset`	(default = 0.15) Distance of label from nodes.
`edge_thickness_range`	(default = c(1,5))
`node_size_range`	(default = c(1, 8))
`anatomical.colors`	(NuLL or vec, default = NULL) Colors for the parent region as a named vector of hexadecimal regions. Can also be a hexadecimal color code written as a string.
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p_list <- plot_joined_networks(anesthesia, correlation_list_names = c("male_agg", "female_non"),
channels = "cfos", edge_colors = c(male_agg =  "#06537f", female_non = "#C70039"),
edge_color_labels = c(male_agg = "Male aggressor", female_non = "Female non-aggressor"),
degree_scale_limit = c(1,45), correlation_edge_width_limit = c(0.8, 1.0),
height = 30, width = 30, label_size = 13, label_offset = 0.08, image_ext = ".png")

## End(Not run)
```

`plot_mean_between_centrality`

Plot mean betweenness centrality

Description

Plot the mean betweenness centrality of the networks in a barplot. Error bars are plotted as SEM.

Usage

```
plot_mean_between_centrality(
  e,
  color_palettes = c("reds", "greens"),
  colors_manual = NULL,
  channels = c("cfos", "eyfp"),
  labels = c(AD = "AD_label", control = "control_label"),
  title = "my_title",
  height = 10,
  width = 10,
  label_angle = 60,
  rev_x_scale = FALSE,
  ylim = c(0, 50),
  theme.gg = NULL,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`color_palettes`	(str, default = c("reds", "greens")) Color palettes from grDevices::hcl.colors that are used to for plotting networks characteristics for each channel, respectively.
`colors_manual`	(str, default = NULL) Manually choose the hexadecimal color codes to create a custom color palette, e.g. colors_manual = c("#660000", "#FF0000", "#FF6666"). Warning: this will be applied to all channels. It's recommended to set the channels parameter to a single channel if this parameter is used.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`labels`	The labels to correspond with your network names.
`title`	(str, default = "my_title) plot title
`height`	(int, default = 10) Height of the plot in inches.
`width`	(int, default = 10) Width of the plot in inches.
`label_angle`	(int, default = 60)
`rev_x_scale`	(bool, default = FALSE) Reveres the scale of the categorical variables
`ylim`	(vec, default = c(0,10)) Axes limits of y-axis
`theme.gg`	(default = NULL) Option to use custom ggplot2 theme if the user wants
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_mean_between_centrality(e, colors_manual = c("#660000", "#FF0000"),
channels = "cfos", labels = c("AD" = "AD_label", "control" = "control_label"),
title = "my title", ylim = c(0, 50), image_ext = ".png")

## End(Not run)
```

plot_mean_clust_coeff *Plot mean clustering coefficient*

Description

Plot the mean clustering coefficients of the networks in a barplot. Error bars are plotted as SEM.

Usage

```
plot_mean_clust_coeff(
  e,
  color_palettes = c("reds", "greens"),
  colors_manual = NULL,
  channels = c("cfos", "eyfp"),
  labels = c(AD = "AD_label", control = "control_label"),
  title = "my_title",
  height = 10,
  width = 10,
  label_angle = 60,
  rev_x_scale = FALSE,
  ylim = c(0, 0.7),
  theme.gg = NULL,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`color_palettes`	(str, default = c("reds", "greens")) Color palettes from grDevices::hcl.colors that are used to for plotting networks characteristics for each channel, respectively.
`colors_manual`	(str, default = NULL) Manually choose the hexadecimal color codes to create a custom color palette, e.g. colors_manual = c("#660000", "#FF0000", "#FF6666"). Warning: this will be applied to all channels. It's recommended to set the channels parameter to a single channel if this parameter is used.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`labels`	(e.g. labels = c(network1_name = "network 1 label", network2_name = "network 2 label)) The legend labels to correspond with your network names.
`title`	(str, default = "my_title) plot title
`height`	(int, default = 10) Height of the plot in inches.
`width`	(int, default = 10) Width of the plot in inches.
`label_angle`	(int, default = 60)
`rev_x_scale`	(bool, default = FALSE) Reveres the scale of the categorical variables the experiment object output folder.
`ylim`	(vec, default = c(0,10)) Axes limits of y-axis
`theme.gg`	(default = NULL) Option to use custom ggplot2 theme if the user wants

`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_mean_clust_coeff(e, colors_manual = c("#660000", "#FF0000"),
channels = "cfos", labels = c("AD" = "AD_label", "control" = "control_label"),
title = "my title", ylim = c(0, 0.7), image_ext = ".png")

## End(Not run)
```

`plot_mean_degree` *Plot the mean degree of the networks in a barplot. Error bars are plotted as SEM.*

Description

Plot the mean degree of the networks in a barplot. Error bars are plotted as SEM.

Usage

```
plot_mean_degree(
  e,
  color_palettes = c("reds", "greens"),
  colors_manual = NULL,
  channels = c("cfos", "eyfp"),
  labels = c(AD = "AD_label", control = "control_label"),
  title = "my_title",
  height = 10,
  width = 10,
  label_angle = 60,
  rev_x_scale = FALSE,
  ylim = c(0, 70),
  theme.gg = NULL,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`color_palettes`	(str, default = c("reds", "greens")) Color palettes from grDevices::hcl.colors that are used to for plotting networks for each channel, respectively.
`colors_manual`	(str, default = NULL) Manually choose the hexadecimal color codes to create a custom color palette, e.g. colors_manual = c("#660000", "#FF0000", "#FF6666"). Warning: this will be applied to all channels. It's recommended to set the channels parameter to a single channel if this parameter is used.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`labels`	(str) The legend labels to correspond with your network names, e.g. labels = c(network1_name = "network 1 label", network2_name = "network 2 label). These are the same network names used in the function `summarise_networks()`.
`title`	(str, default = "my_title) plot title
`height`	(int, default = 10) Height of the plot in inches.
`width`	(int, default = 10) Width of the plot in inches.
`label_angle`	(int, default = 60)
`rev_x_scale`	(bool, default = FALSE) Reveres the scale of the categorical variables the experiment object output folder.
`ylim`	(vec, default = c(0,10)) Axes limits of y-axis
`theme.gg`	(default = NULL) Option to use custom ggplot2 theme if the user wants
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_mean_degree(e, colors_manual = c("#660000", "#FF0000"),
channels = "cfos", labels = c("AD" = "AD_label", "control" = "control_label"),
title = "my title", ylim = c(0,100), image_ext = ".png")

## End(Not run)
```

`plot_mean_global_effic`

Plot mean global efficiency

Description

Plot the mean global efficiency of the networks in a barplot. Error bars are plotted as SEM.

Usage

```
plot_mean_global_effic(
  e,
  color_palettes = c("reds", "greens"),
  colors_manual = NULL,
  channels = c("cfos", "eyfp"),
  labels = c(AD = "AD_label", control = "control_label"),
  title = "my_title",
  height = 10,
  width = 10,
  label_angle = 60,
  rev_x_scale = FALSE,
  ylim = c(0, 0.7),
  theme.gg = NULL,
  image_ext = ".png",
  print_plot = FALSE,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`color_palettes`	(str, default = c("reds", "greens")) Color palettes from grDevices::hcl.colors that are used to for plotting networks characteristics for each channel, respectively.
`colors_manual`	(str, default = NULL) Manually choose the hexadecimal color codes to create a custom color palette, e.g. colors_manual = c("#660000", "#FF0000", "#FF6666"). Warning: this will be applied to all channels. It's recommended to set the channels parameter to a single channel if this parameter is used.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) Channels to plot
`labels`	The labels to correspond with your network names.
`title`	(str, default = "my_title) plot title
`height`	(int, default = 10) Height of the plot in inches.
`width`	(int, default = 10) Width of the plot in inches.
`label_angle`	(int, default = 60)
`rev_x_scale`	(bool, default = FALSE) Reveres the scale of the categorical variables
`ylim`	(vec, default = c(0,10)) Axes limits of y-axis
`theme.gg`	(default = NULL) Option to use custom ggplot2 theme if the user wants
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output.s
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_mean_global_effic(e, colors_manual = c("#660000", "#FF0000"),
channels = "cfos", labels = c("AD" = "AD_label", "control" = "control_label"),
title = "my title", ylim = c(0, 0.7), image_ext = ".png")

## End(Not run)
```

---

plot_networks *Plot the networks stored in an experiment object*

Description

Plot the networks stored in an experiment object

Usage

```
plot_networks(
  e,
  network_name = "AD",
  title = NULL,
  channels = c("cfos", "eyfp", "colabel"),
  edge_color = "firebrick",
  height = 15,
  width = 15,
  edge_type = "arc",
  region_legend = TRUE,
  degree_scale_limit = c(1, 10),
  anatomical.colors = NULL,
  correlation_edge_width_limit = c(0.8, 1),
  image_ext = ".png",
  print_plot = FALSE,
  graph_theme = NULL,
  label_size = 5,
  edge_thickness_range = c(1, 5),
  node_size_range = c(1, 8),
  label_offset = 0.15,
  save_plot = TRUE
)
```

Arguments

`e`	experiment object
`network_name`	(str, default = "AD")
`title`	(str, default = NULL) Title of network plot
`channels`	(str, default = c("cfos", "eyfp", "colabel"))
`edge_color`	(str, default = "firebrick") Color of the network edges.
`height`	Height of the plot in inches.
`width`	width of the plot in inches.

`edge_type`	(default = "arc") "arc" or "diagonal".
`region_legend`	(default = TRUE) Boolean determining whether or not to show the region legend categorizing subregions into their largest parent region. Only works well if the Allen ontology is used for the dataset.
`degree_scale_limit`	(vec, default = c(1,10)) Scale limit for degree size
`anatomical.colors`	(vector, default = NULL) NULL defaults to viridis. A named vector of hexadecimal codes for the anatomical super regions. e.g. anatomical.colors = c(Isocortex = "#5571a9", OLF = "#64bdc4", HPF = "#d2875b", CTXsp = "#87a3db", CNU = "#466496", TH = "#7e72af", HY = "#8e7960", MB = "#d796c8", HB = "#646464")
`correlation_edge_width_limit`	(default = c(0.8,1))
`image_ext`	(default = ".png") image extension to the plot as.
`print_plot`	(bool, default = FALSE) Whether to print the plot as an output. the experiment object output folder.
`graph_theme`	(default = NULL) Add a `ggraph::theme_graph()` to the network graph. If NULL, the default is taken.
`label_size`	(default = 5) Default font size for network region labels.
`edge_thickness_range`	(default = c(1,5)) Thickness range of the edges.
`node_size_range`	(default = c(1,8)) Node size range. Can also be a hexadecimal color code written as a string.
`label_offset`	(default = 0.15) Distance of label from nodes.
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p <- plot_networks(e, network_name = "Shock", channels = "cfos")

## End(Not run)
```

`plot_normalized_counts` *Plot normalized cell counts*

Description

Plot the cell counts normalized by volume for a given channel

Usage

```
plot_normalized_counts(
  e,
  channels = c("cfos", "eyfp", "colabel"),
  by = c("sex", "group"),
 values = list(c("female", "non"), c("female", "agg"), c("female", "control"), c("male",
    "agg"), c("male", "control")),
  colors = c("white", "lightblue", "black", "red", "green"),
  ontology = "allen",
  title = NULL,
  unit_label = bquote(`Cell counts `("cells/mm"^3)),
 anatomical.order = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU", "TH", "HY", "MB", "HB",
    "CB"),
  height = 7,
  width = 20,
  print_plot = FALSE,
  save_plot = TRUE,
  flip_axis = FALSE,
  reverse_colors = FALSE,
  limits = c(0, 1e+05),
  facet_background_color = NULL,
  strip_background_colors = "lightblue",
 plot_theme = ggplot2::theme(plot.background = element_blank(), panel.grid.major =
   element_blank(), panel.grid.minor = element_blank(), panel.border = element_blank(),
    axis.line = element_line(color = "black"), legend.justification = c(0, 0),
   legend.position = "inside", legend.position.inside = c(0.05, 0.6), legend.direction =
   "vertical", axis.text.y = element_text(angle = 50, hjust = 1, size = 8, color =
    "black"), axis.text.x = element_text(color = "black"), strip.text.y =
    element_text(angle = 0, margin = ggplot2::margin(t = 5,
     r = 5, b = 5, l = 5,
   unit = "pt")), strip.placement = "outside", strip.switch.pad.grid = unit(0.1, "in")),
  image_ext = ".pdf"
)
```

Arguments

`e`	experiment object
`channels`	(str, default = c("cfos", "eyfp", "colabel"))
`by`	(str) Attribute names to group by, e.g. c("sex", "group")
`values`	(list) A list with a length the number of groups desired for plotting. Each element of the list is a vector in the order of the the respective values for the attributes entered for the by parameter to generate a specific analysis group. Each vector should be unique to generate a uniquely colored bar. e.g.values = c("female", "AD").
`colors`	(str, default = c("white", "lightblue")) Hexadecimal codes corresponding to the groups (respectively) to plot. The length of this vector should be the length of the list.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`title`	(str, default = NULL) An optional title for the plot
`unit_label`	(str, default = bquote('Cell counts '('cells/mm'^3))) Default unit label for the graphs

`anatomical.order`	(default = c("Isocortex", "OLF", "HPF", "CTXsp", "CNU","TH", "HY", "MB", "HB", "CB")) Default way to group subregions into super regions order
`height`	height of the plot in inches.
`width`	width of the plot in inches.
`print_plot`	(bool, default = FALSE) Whether to display the plot (in addition to saving the plot)
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the experiment object output folder.
`flip_axis`	plot cell counts on x-axis rather than y-axis.
`reverse_colors`	(bool, default = FALSE) Whether to reverse the color order. This may depend on the order in which you entered the `colors` parameter
`limits`	(c(0,100000)) Range of the normalized cell counts.
`facet_background_color`	(default = NULL) Set to a hexadecimal string, e.g."#FFFFFF", when you want to shade the background of the graph. Defaults to no background when NULL.
`strip_background_colors`	(default = "lightblue) Enter custom codes to control the strip background colors, e.g. c(Isocortex = "#5571a9", OLF = "#64bdc4", HPF = "#d2875b", CTXsp = "#87a3db", CNU = "#466496", TH = "#7e72af", HY = "#8e7960", MB = "#d796c8", HB = "#646464"). If more than one color is used, you must install the package ggh4x.
`plot_theme`	(ggplot2 theme object) This allows for fine tuning the aesthetics of the figure. Default parameters shown: ggplot2::theme(plot.background = element_blank(), panel.grid.major = element_blank(), panel.grid.minor = element_blank(), panel.border = element_blank(), axis.line = element_line(color = 'black'), legend.justification = c(0, 0), legend.position = "inside", legend.position.inside = c(0.05, 0.6), legend.direction = "vertical", axis.text.y = element_text(angle = 50, hjust = 1, size = 8, color = "black"), axis.text.x = element_text(color = "black"), strip.text.y = element_text(angle = 0, margin = ggplot2::margin(t = 5, r = 5, b = 5, l = 5, unit = "pt")), strip.placement = "outside", strip.switch.pad.grid = unit(0.1, "in"))
`image_ext`	(default = ".png") image extension to the plot as.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
p_list <- plot_normalized_counts(e, channels = "cfos", by = c("sex", "group"),
values = list(c("female", "non"), c("female", "agg")), colors = c("white", "lightblue"))

## End(Not run)
```

`plot_percent_colabel` *This function allows for plotting of colabelled cells over either the "cfos" or "eyfp" channels.*

Description

Allows for specification of specific brain regions to plot. Two different mouse attributes can be used as categorical variables to map to either the color or pattern aesthetics of the bar plot, e.g. sex and experimental group. The color aesthetic takes precedence over the pattern aesthetic so if you only want to use one mouse attribute, for plotting set it to the `color_mapping` parameter and set the `pattern_mapping` parameter to NULL.

Usage

```
plot_percent_colabel(
  e,
  colabel_channel = "colabel",
  channel = "eyfp",
  rois = c("AAA", "dDG", "HY"),
  color_mapping = "sex",
  colors = c("#952899", "#358a9c"),
  pattern_mapping = NULL,
  patterns = c("gray100", "hs_fdiagonal",
    "hs_horizontal\n                                             ", "gray90",
    "hs_vertical"),
  error_bar = "sem",
  ylim = c(0, 100),
  plot_individual = TRUE,
  height = 8,
  width = 8,
  print_plot = FALSE,
  save_plot = TRUE,
  image_ext = ".png"
)
```

Arguments

`e`	experiment object
`colabel_channel`	(str, default = "colabel") The channel used as the numerator in fraction counts.
`channel`	(str, default = "eyfp") The channel used as denominator in fraction counts.
`rois`	character vector of region acronyms, e.g. c("AAA", "DG)
`color_mapping`	(str, default = "sex") The name of the categorical variable (e.g., "sex", "age", etc.) to map to the color aesthetic of the bar plot.
`colors`	(str) character vector of the color values desired for the groups.
`pattern_mapping`	(str, default = "sex") The name of the categorical variable (e.g., "sex", "age", etc.) to map to the pattern aesthetic of the bar plot.

`patterns`	(str, default = c("gray100", 'hs_fdiagonal', "hs_horizontal", "gray90", "hs_vertical"), Pattern types to define subgroups.
`error_bar`	(str, c("sd", "sem)) options for which type of error bar to display, standard deviation or standard error of the mean.
`ylim`	(default = c(0,100)) The range of the y-axis
`plot_individual`	(boo) Whether or not to plot multiple
`height`	(default = 8) height of graphics devices in inches
`width`	(default = 8) height of graphics device in inches
`print_plot`	(bool, default = FALSE) whether or not to print the plot for just for display
`save_plot`	(bool, default = TRUE) whether or not to save the plor
`image_ext`	(default = ".png") extension determining the image type to save as

Value

p Plot handle to the figure

Examples

```
## Not run:
 plot_percentage_colabel

## End(Not run)
```

---

print.correlation_list *Print attributes of correlation_list object*

Description

Print attributes of correlation_list object

Usage

```
## S3 method for class 'correlation_list'
print(x, ...)
```

Arguments

`x`	an object used to select a method. A correlation list
`...`	further arguments passed to or from other methods.

`print.experiment` *Print attributes of experiment object*

Description

Print attributes of experiment object

Usage

```
## S3 method for class 'experiment'
print(x, ...)
```

Arguments

`x`	experiment object
`...`	further arguments passed to or from other methods.

Examples

```
e <- experiment()
print(e)
```

`print.mouse` *Print attributes of mouse object*

Description

Print attributes of mouse object

Usage

```
## S3 method for class 'mouse'
print(x, ...)
```

Arguments

`x`	mouse object
`...`	further arguments passed to or from other methods.

Examples

```
m <- mouse()
print(m)
```

`print.slice` *Print attributes of slice object*

Description

Print attributes of slice object

Usage

```
## S3 method for class 'slice'
print(x, ...)
```

Arguments

`x`	slice object
`...`	further arguments passed to or from other methods.

Examples

```
s <- slice()
print(s)
```

`read_check_file` *Read a csv or excel file as a tibble. Checks first that the file exists, and that it is a csv or xlsx format.*

Description

Read a csv or excel file as a tibble. Checks first that the file exists, and that it is a csv or xlsx format.

Usage

```
read_check_file(x, ...)
```

Arguments

`x`	A file path
`...`	additional parameters to pass to either the readr::read_csv function or readxl::read_excel

Value

tibble dataframe

`register` *Register (generic function)*

Description

Register (generic function)

Register a slice in a slice object

Register a slice in a mouse object. If a slice has been previously registered, the default behavior is to continue modifying the previous registration. Use the `replace` parameter to change this behavior.

Usage

```
register(x, ...)

## S3 method for class 'slice'
register(x, filter = NULL, ...)

## S3 method for class 'mouse'
register(
  x,
  slice_ID = NA,
  hemisphere = NULL,
  filter = NULL,
  replace = FALSE,
  ...
)
```

Arguments

`x`	a mouse or slice object
`...`	additional parameters to pass to the SMART::registration2() function, besides 'input', 'coordinate', 'filter' & 'correspondance'
`filter`	(list) Wholebrain filter with parameters.
`slice_ID`	(str) ID of slice
`hemisphere`	(str, default = NULL) 'left', 'right' or NULL if both hemispheres are included
`replace`	(bool, default = FALSE) Replace a registration already contained in a mouse object by resetting to NULL value before registration improvement loop.

Value

a mouse or slice object

s a slice object

m mouse object

Examples

```
## Not run:
s <- register(s)

## End(Not run)
## Not run:
m <- register(m, slice_ID = '1_10', hemisphere = "left", filter = my_filter)

## End(Not run)
```

---

reset_mouse_root   *Reset the root path for the folder containing the registration and segmentation data.*

Description

This function takes a mouse object and also a `input_path` as the root folder for that mouse. It then adjusts all the paths for the registration and segmentation data read to be relative to the root folder. This function is especially useful if you have changed the computers you are analyzing and drive mappings may be different.

Usage

```
reset_mouse_root(m, input_path = NULL, print = TRUE)
```

Arguments

`m`	mouse object
`input_path`	(default = NULL) Reset the root directory of the mouse object.
`print`	(bool, default = TRUE) Print the changes in the console.

Value

m a mouse object

Examples

```
## Not run:
m <- reset_mouse_root(m, input_path = "C:/Users/Documents/Mice/mouse_1/", print = TRUE)

## End(Not run)
```

`rewire_network`	*Implement rewiring algorithms to current empirical networks to randomize certain network properties.*

Description

Not that this keeps other characteristics constant (such as preserved degree sequence). These null networks can them be used to compare against and normalize the empirical networks. Currently this essentially erases edge metrics and treats networks like binary graphs. Edge weights are not used in calculating network topology metrics.

Usage

```
rewire_network(
  e,
  network_name,
  channels = "cfos",
  method = "ms",
  ontology = "allen",
  n_rewires = 10000,
  n_networks = 100,
  return_graphs = FALSE,
  seed = 5
)
```

Arguments

`e`	experiment object
`network_name`	(str) Name of the network
`channels`	(str) Vector of channels to process
`method`	(str, default = "ms") "ms" implements Maslov-Sneppen rewiring approach (annuls all network properties except for network size, connection density, and degree distribution).
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`n_rewires`	(int, default = 10000) The number of rewires for randomization for "ms" rewiring implementation. Recommended to be the larger of either 10,000 or 10*No. edges in a graph.
`n_networks`	(int, default = 100) The number of random networks to create
`return_graphs`	(logical, default = FALSE) if TRUE, returns a list organized by channel containing a sublist, with each element containing a tidygraph object. This must be FALSE if you want to run you want to summarize the null network statistics with `summarize_null_networks()`
`seed`	(int, default = 5) Random seed for future replication.

Value

Summary table of rewired network properties of all nodes showing the average of all randomized network properties generated.

Examples

```
## Not run:
summary_table <- rewire_network(e, network_name = "network1", channels = "cfos",
n_rewire =  igraph::gsize(e$networks$network1$cfos)*100, n_networks = 100)

## End(Not run)
```

`save_experiment` *Save experiment data*

Description

Saves experiment object into it's attribute output path as an RDATA file save_experiment(e)

Usage

```
save_experiment(..., timestamp = FALSE)
```

Arguments

`...`	parameter to pass experiment object
`timestamp`	(bool) save the object with a date tag

Examples

```
## Not run:
e <- save_experiment(e, timestamp = TRUE)

## End(Not run)
```

`save_mouse` *Save mouse data*

Description

Saves mouse object into it's attribute output path as an RDATA file save_mouse(m)

Usage

```
save_mouse(..., timestamp = FALSE)
```

Arguments

`...`	parameter to pass mouse object
`timestamp`	(bool) save the object with a date tag

Examples

```
## Not run:
m <- save_mouse(m, timestamp = TRUE)

## End(Not run)
```

`segmentation.object` *segmentation object compatible with wholebrain package functions*

Description

segmentation object compatible with wholebrain package functions

Usage

```
segmentation.object
```

Format

A

filter list storing parameter use to segment and get brain contours

soma list storing cell count data

`sem` *Standard error function*

Description

Standard error function

Usage

```
sem(x)
```

Arguments

`x`	(vec)

Value

numeric

Examples

```
sem(c(3,4,5))
```

`simplify_by_keywords` *Simplify dataframe by keywords.*

Description

Simplify dataframe by keywords.

Usage

```
simplify_by_keywords(
  df,
  keywords = c("layer", "part", "stratum", "division", "leaflet", "Subgeniculate",
    "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch", "Precommissural"),
  ontology = "allen",
  dont_fold = c("Dorsal part of the lateral geniculate complex",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Substantia nigra")
)
```

Arguments

`df`	(tibble) Must contain columns "acronym" and "name"
`keywords`	(vec, default = c("layer","part","stratum","division", "leaflet", "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch", "Precommissural")) a list of keywords to simplify based on region name.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`dont_fold`	(vec, default = c("Dorsal part of the lateral geniculate complex", "Ventral posterolateral nucleus of the thalamus, parvicellular part", " Ventral posteromedial nucleus of the thalamus, parvicellular part","Ventral posterolateral nucleus of the thalamus, parvicellular part", "Ventral posteromedial nucleus of the thalamus, parvicellular part","Substantia nigra")) Regions that are exceptions to being folded into their parent regions.

Value

df

Examples

```
df <- dplyr::tibble(acronym = c("LGd", "GU4", "dCA1so" ),
name = c("Dorsal part of the lateral geniculate complex",
"Gustatory areas, layer 4", "Field CA2, stratum oriens (dorsal)"))
simplify_by_keywords(df)

df <- dplyr::tibble(acronym = c("LPBD", "MPBE", "CPre"),
name = c("Lateral parabrachial nucleus, dorsal part",
"Medial parabrachial nucleus, external part", "Caudoputamen- rostral extreme"))
simplify_by_keywords(df, keywords = c("dorsal part", "external part",
"Caudoputamen-"), ontology = "unified")
```

`simplify_cell_count` *Simplify the combined cell count table*

Description

This function is designed to offer flexible simplification of mapped cells counts. This can be applied after running `combine_cell_counts()`. However, if mapping is being conducted using the SMARTTR package, we recommend simplifying mapped counts earlier, at the level of mouse objects using `normalize_cell_counts()` because the options offered for simplification are more flexible. The benefit of this function is that it can operate on experiment objects with externally imported combined cell counts tables that are formatted for compatibility. This allows for simplification using other ontologies. See the available atlas options under the `ontology` parameter.

Usage

```
simplify_cell_count(
  e,
  ontology = "allen",
  simplify_keywords = c("layer", "part", "stratum", "division", "leaflet",
   "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch",
    "Precommissural"),
  dont_fold = c("Dorsal part of the lateral geniculate complex",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Substantia nigra")
)
```

Arguments

`e`	experiment object
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`simplify_keywords`	(str vec, default = c("layer","part","stratum","division", "leaflet", "Subgeniculate", "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch", "Precommissural")). Keywords to search through region names and simplify to parent structure. This means the parent structure is also excluded if the list of excluded right and left
`dont_fold`	(vec) vector of regions to not fold in. regions can be further

Value

e experiment object with simplified keywords

Examples

```
## Not run:
e <- simplify_cell_count(e)

## End(Not run)
```

---

simplify_vec_by_keywords

Simplify vector of acronyms by keywords.

Description

Simplify vector of acronyms by keywords.

Usage

```
simplify_vec_by_keywords(
  vec,
  keywords = c("layer", "part", "stratum", "division", "leaflet", "Subgeniculate",
   "island", "Islands", "Fields of Forel", "Cajal", "Darkschewitsch", "Precommissural"),
  ontology = "allen",
  dont_fold = c("Dorsal part of the lateral geniculate complex",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Ventral posterolateral nucleus of the thalamus, parvicellular part",
    "Ventral posteromedial nucleus of the thalamus, parvicellular part",
    "Substantia nigra")
)
```

Arguments

`vec`	(vector) Must contain acronyms
`keywords`	(vec, default = c("layer","part","stratum","division")) a list of keywords to simplify based on region name.
`ontology`	(str, default = "allen") Region ontology to use. options = "allen" or "unified"
`dont_fold`	(vec, default = c("Dorsal part of the lateral geniculate complex", "Ventral posterolateral nucleus of the thalamus, parvicellular part", " Ventral posteromedial nucleus of the thalamus, parvicellular part","Ventral posterolateral nucleus of the thalamus, parvicellular part", "Ventral posteromedial nucleus of the thalamus, parvicellular part","Substantia nigra")) Regions that are exceptions to being folded into their parent regions.

Value

df, dataframe as a tibble with included long name and acronyms that are simplified to parents

Examples

```
acronyms <- c("LGd", "GU4", "dCA1so" )
simplify_vec_by_keywords(acronyms)

acronyms <- c("LPBD", "MPBE", "CPre")
simplify_vec_by_keywords(acronyms,
keywords = c("dorsal part", "external part", "Caudoputamen-"), ontology = "unified")
```

`slice` *Create a slice object*

Description

slice() constructs an S3 object of class 'slice'. A slice object consists of a list of lists storing information about registration, segmentation, raw mapped data and cleaned mapped data. The object attributes are also stored as a list.

Usage

```
slice(
  slice_ID = NA,
  coordinate = -1,
  atlas_plate = NA,
  conversion_factor = 1.0833,
  bin = 1,
  z_width = 24,
  hemisphere = NULL,
  channels = c("eyfp", "cfos", "colabel"),
  registration_path = "set registration image path",
  segmentation_path = "set segmentation data path",
  slice_directory = NULL,
  left_regions_excluded = c("fiber tracts", "VS"),
  right_regions_excluded = c("fiber tracts", "VS"),
  left_regions_included = NULL,
  right_regions_included = NULL,
  ...
)
```

Arguments

`slice_ID`	(str, default = NA) Slice name
`coordinate`	(num, default = -1) Allen mouse brain atlas coordinate aligning with slice.
`atlas_plate`	(int, default = NA) Atlas place number. TODO: Currently unused
`conversion_factor`	(num, 1.0833) pixel-to-micron conversion factor
`bin`	(int, default = 1) Whether the registration image was binned in ImageJ.
`z_width`	(num, default = 24) The z-stack width in um.
`hemisphere`	(str, default = NULL) Hemisphere to process. "left", "right" or NULL is legal.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) The channels to process.

`registration_path`
: (str, default = 'set registration image path') May deprecate this in favor of slice_directory in future versions.

`segmentation_path`
: (str, default = 'set segmentation image path') Path to image used for segmentation function using base `wholebrain::segment()` function.

`slice_directory`
: (str, default = NULL) The root directory where slice information such as the registration or segmentation images or txt files are stored for a given slice. TODO: May change the import and registration functions to only rely on this path.

`left_regions_excluded`
: (str, default = ("layer 1","PIR1","TR1","PAA1","NLOT1","OT1","MOBgl","OV","VLPO","SO", "BA","TU","MEAav","ME","TMv","PVp","SUMl","SCzo","fiber tracts","VS")) The list of acronyms corresponding to regions to exclude for this slice's left hemisphere.

`right_regions_excluded`
: (str, default = ("layer 1","PIR1","TR1","PAA1","NLOT1","OT1","MOBgl","OV","VLPO","SO", "BA","TU","MEAav","ME","TMv","PVp","SUMl","SCzo","fiber tracts","VS")) The list of acronyms corresponding to regions to exclude for this slice's right hemisphere.

`left_regions_included`
: (str, default = NULL) List of acronyms of regions to include from left hemisphere. All other regions will be exsluded. If not NULL, takes precedence over `left_regions_excluded`.

`right_regions_included`
: (str, default = NULL) List of acronyms of regions to include from right hemisphere. All other regions will be exsluded. If not NULL, takes precedence over `right_regions_excluded`.

`...`
: additional custom keyword pair attributes you'd like to store

Details

The slice attributes can be assigned as arguments to the slice constructor function. See the parameters listed for the default values for these attributes Note that you are able to add custom attributes as keyword pairs, if you would like to keep track of an additional piece of information. However, this will only serve a descriptive purpose and will not be used for analysis. You may not need to use all slice attributes but fill out as many are applicable to your experiment.

Value

A slice, a colloquial term for an object of class 'slice'. A 'slice' object is also a list, with class list.

Examples

```
slice_example <- slice() # initializes a slice object
```

`SMARTTR` *SMARTTR: A Mapping, Analysis, and Visualization Package for Wholebrain Dual-Ensemble Coronal Datasets.*

Description

`SMARTTR`. This package allows for the user-friendly pre-processing of segmentation data generated from ImageJ to a be compatible with the `wholebrain` package to generate region-based cell counts that are normalized by volume. It will also provides tools for data analysis based on experimental groupings.

Details

Object descriptions The data for analysis will be stored in objects that allow for more neat bundling of useful information together.

A slice object will contain all the data related to registration, segmentation for each channel, and cell counts for a particular image. It will also contain "metadata" about your experimental images, such as what the experimenter-assigned slice ID is, which brain atlas AP coordinate matches best with the given image, and what the path to the image used for registration is. These metadata are stored as the object's attributes.

A mouse object is an object that will store multiple slice objects (and therefore all the information in it), and will eventually store the combined cell data and the region cell counts normalized by volume. Like a slice, it will also contain "metadata" about your mouse stored as attributes. An experimentAn experiment object consists of a list of processed mouse objects with raw data from slices omitted, and experimental attributes stored as a list. It will also contain "metadata" about your experimental personnel and analysis groups stored as attributes.

The package currently allows for easy implementation of the following steps

1. Setting up the pipeline by specifying experimentparameters, and save directories.
2. The interactive registration process.
3. Importing raw segmentation data from .txt files generated from ImageJ for multiple channels.
4. Optionally creating a filter for the 'cfos' and 'eyfp' channels to clean segmented counts.
5. Creating a segmentation object that is compatible with `wholebrain` functions.
6. Forward warping and mapping the data onto the standardized mouse atlas.
7. Cleaning the mapped data in all the following ways: + Removing cells that map outside the boundaries of the atlas.
    - Omitting regions by a default list of regions to omit.
    - Omitting regions by user specified region acronyms.
    - Removing Layer 1 cells
    - Removing cells from a contralateral hemisphere per slice if the registrations are divided by right and left hemispheres.
8. Obtaining cell counts normalized by region volume (per mm^2^) and region areas (per mm^2^).

`summarise_networks`	*Summarize multiple networks. calculate network statistics for each network. This is not meant to summarize networks created using* `create_joined_networks`*.*

Description

Summarize multiple networks. calculate network statistics for each network. This is not meant to summarize networks created using `create_joined_networks`.

Usage

```
summarise_networks(
  e,
  network_names,
  channels = c("cfos", "eyfp", "colabel"),
  save_stats = TRUE,
  save_degree_distribution = TRUE,
  save_betweenness_distribution = TRUE,
  save_efficiency_distribution = TRUE
)
```

Arguments

`e`	experiment object
`network_names`	(str) The names of the networks to generate summary tables for, e.g. network_names = c("female_AD", "female_control")
`channels`	(str, default = c("cfos", "eyfp", "colabel")) The channels to process.
`save_stats`	(bool, default = TRUE) Save the summary stats as a csv file in the output folder. Note that the clustering calculated is an average of the local vertex clustering.
`save_degree_distribution`	(bool, default = TRUE) Save the network degree distributions (frequencies of each degree) across each comparison group as a csv file.
`save_betweenness_distribution`	(bool, default = TRUE) Save the betweenness distribution and summary as a csv.
`save_efficiency_distribution`	(bool, default = TRUE) Save the efficiency distribution and summary as a csv.

Value

e experiment object

Examples

```
## Not run:
e <- get_network_statistics(e,  network_names = c("female_AD", "female_control"),
channels = c("cfos", "eyfp", "colabel"), save_stats = TRUE, save_degree_distribution = TRUE)

## End(Not run)
```

---

summarize_null_networks

Summarize the parameters of the rewired null networks generated by `rewire_network()`

Description

Summarize the parameters of the rewired null networks generated by `rewire_network()`

Usage

```
summarize_null_networks(
  null_nodes_list,
  network_names = NULL,
  channel = "cfos"
)
```

Arguments

`null_nodes_list`	a list of output summary tables (1 per network) of rewired network properties of all nodes from `rewire_network()`.
`network_names`	(str vec) Name of the networks that were rewired (in the same order as the list). `null_nodes_list` and `network_names` must have the same length.
`channel`	(str) channel to process

Value

a list of length 2. The first element is named `global_summary` and contains a table of global summary statistics. The second element is named `node_summary`, and contained per node statistics averaged from multiple null networks.

Examples

```
## Not run:
rewire_summary <- rewire_network(e, "Context", channels = "eyfp", return_graphs = FALSE)
summarized_null_networks <- summarize_null_networks(rewire_summary,
network_names = "Context", channel = "eyfp")

## End(Not run)
```

---

volcano_plot

Plot the results of the permutation histogram used to determine the p-value of the pairwise region comparison

---

Description

Create a Volcano plot.

Usage

```
volcano_plot(
  e,
  permutation_comparison = "female_AD_vs_male_AD",
  channels = c("cfos", "eyfp", "colabel"),
  colors = c("#be0000", "#00782e", "#f09b08"),
  save_plot = TRUE,
  title = NULL,
  ylim = c(0, 3),
  height = 8,
  width = 10,
  print_plot = FALSE,
  plt_theme = NULL,
  point_size = 1,
  image_ext = ".png"
)
```

Arguments

`e`	experiment object
`permutation_comparison`	The name of the correlation group comparisons to plot.
`channels`	(str, default = c("cfos", "eyfp", "colabel")) channels to plot.
`colors`	(str, default = c("#be0000", "#00782e", "#f09b08")) Hexadecimal code for the colors corresponding to the channels attribute of the correlation_list. Color values can also be input compatible with ggplot2 plotting functions.
`save_plot`	(bool, default = TRUE) Save into the figures subdirectory of the the experiment object output folder.
`title`	Title of the plot.
`ylim`	(vec, default = c(0,y)) Y-axis range (logarithmic).
`height`	height of the plot in inches.
`width`	width of the plot in inches.
`print_plot`	(bool, default = FALSE) Print the plot as graphics windows.
`plt_theme`	(default = NULL) Add a `ggplot2::theme()` to the plot. If NULL, the default is taken..
`point_size`	(default = 1) Size of the plotted points.
`image_ext`	(default = ".png") image extension to save the plot as.

Details

Plot the correlation difference between two comparison groups into a volcano plot. The function `correlation_diff_permutation()` must be run first in order to generate results to plot.

Value

p_list A list the same length as the number of channels, with each element containing a plot handle for that channel.

Examples

```
## Not run:
volcano_plot(e, permutation_comparison = "female_AD_vs_male_AD",
channels = c("cfos", "eyfp", "colabel"), colors =  c("#be0000", "#00782e", "#f09b08"),
save_plot = TRUE, title = NULL, ylim = c(0, 3), height = 8,
width = 10, print_plot = FALSE, image_ext = ".png")

## End(Not run)
```

Index