

---

**deepnog**

***Release 1.2.2***

**Lukas Gosch, Roman Feldbauer**

**Dec 10, 2020**



# GETTING STARTED

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Quick Start Example</b>	<b>5</b>
<b>3</b>	<b>User Guide</b>	<b>9</b>
<b>4</b>	<b>Contributing</b>	<b>29</b>
<b>5</b>	<b>Changelog</b>	<b>33</b>
<b>6</b>	<b>Getting started</b>	<b>37</b>
<b>7</b>	<b>User Guide</b>	<b>39</b>
<b>8</b>	<b>Development</b>	<b>41</b>
<b>9</b>	<b>What's new</b>	<b>43</b>
<b>10</b>	<b>Indices and tables</b>	<b>45</b>
	<b>Python Module Index</b>	<b>47</b>
	<b>Index</b>	<b>49</b>



deepnog is a Python package for assigning proteins to orthologous groups (eggNOG 5) with deep networks.



## INSTALLATION

### 1.1 Installation from PyPI

The current release of deepnog can be installed from PyPI:

```
pip install deepnog
```

For typical use cases, and quick start, this is sufficient. Note that this guide assumes Linux, and may work under macOS. We currently don't provide detailed instructions for Windows.

### 1.2 Alternative: Installation from bioconda

Alternatively, deepnog can be installed from bioconda like this:

```
# With channel setup as described in the `bioconda docs`_
conda config --add channels pytorch
conda install pytorch deepnog
```

### 1.3 Dependencies and model files

All package dependencies of deepnog are automatically installed by pip. We also require model files (= networks parameters/weights), which are too large for GitHub/PyPI. These are hosted on separate servers, and downloaded automatically by deepnog, when required. By default, models are cached in *\$HOME/deepnog\_data/*.

You can change this path by setting the DEEPNOG\_DATA environment variable. Choose among the following options to do so:

```
# Set data path temporarily
DEEPNOG_DATA="/custom/path/models" deepnog infer sequences.fa

# Set data path for the current shell
export DEEPNOG_DATA="/custom/path/models"

# Set data path permanently
printf "\n# Set path to DeepNOG models\nexport DEEPNOG_DATA=\"/custom/path/models\"\n"
↪" >> ~/.bashrc
```

## 1.4 Installation from source

You can always grab the latest version of deepnog directly from GitHub:

```
cd install_dir
git clone git@github.com:univieCUBE/deepnog.git
cd deepnog
pip install -e .
```

This is the recommended approach, if you want to contribute to the development of deepnog.

## 1.5 Supported platforms

deepnog currently supports all major operating systems:

- Linux
- MacOS X
- Windows



## QUICK START EXAMPLE

The following example shows all these steps for predicting protein orthologous groups with the command line interface of deepnog as well as using the Python API. Please make sure you have installed deepnog ([installation instructions](#)).

### 2.1 CLI Usage Example

Using deepnog from the command line is the simple, and preferred way of interacting with the deepnog package.

Here, we assign orthologous groups (OGs) of proteins using a model trained on the eggNOG 5.0 database and using only bacterial OGs (default settings), and redirect the output from stdout to a file:

```
deepnog infer input.fa > assignments.csv
```

Alternatively, the output file and other settings can be specified explicitly like so:

```
deepnog infer input.fa --out prediction.csv -db eggNOG5 --tax 2
```

For a detailed explanation of flags and further settings, please consult the [User Guide](#).

Note that deepnog masks predictions below a certain confidence threshold. The default confidence threshold baked into the model at 0.99 can be overridden from the command line interface:

```
deepnog infer input.fa --confidence-threshold 0.8 > assignments.csv
```

The output comma-separated values (CSV) file *assignments.csv* then looks something like:

```
sequence_id,prediction,confidence
WP_004995615.1,COG5449,0.99999964
WP_004995619.1,COG0340,1.0
WP_004995637.1,COG4285,1.0
WP_004995655.1,COG4118,1.0
WP_004995678.1,COG0184,1.0
WP_004995684.1,COG1137,1.0
WP_004995690.1,COG0208,1.0
WP_004995697.1,,
WP_004995703.1,COG0190,1.0
```

The file contains a single line for each protein in the input sequence file, and the following fields:

- `sequence_id`, the name of the input protein sequence.
- `prediction`, the name of the predicted protein OG. Empty if masked by confidence threshold.

- confidence, the confidence value (0-1 inclusive) that deepnog ascribes to this assignment. Empty if masked by confidence threshold.

## 2.2 API Example Usage

```
import torch
from deepnog.data import ProteinIterableDataset
from deepnog.learning import predict
from deepnog.utils import create_df, get_config, get_weights_path, load_nn, set_device

PROTEIN_FILE = '/path/to/file.faa'
DATABASE = 'eggNOG5'
TAX = 2
ARCH = 'deepnog'
CONF_THRESH = 0.99

# load protein sequence file into a ProteinIterableDataset
dataset = ProteinIterableDataset(PROTEIN_FILE, f_format='fasta')

# Construct path to saved parameters deepnog model.
weights_path = get_weights_path(
    database=DATABASE,
    level=str(TAX),
    architecture=ARCH,
)

# Set up device for prediction
device = set_device('auto')
torch.set_num_threads(1)

# Load neural network parameters
model_dict = torch.load(weights_path, map_location=device)

# Lookup where to find the chosen network
config = get_config()
module = config['architecture'][ARCH]['module']
cls = config['architecture'][ARCH]['class']

# Load neural network model and class names
model = load_nn((module, cls), model_dict, 'infer', device)
class_labels = model_dict['classes']

# perform prediction
preds, confs, ids, indices = predict(
    model=model,
    dataset=dataset,
    device=device,
    batch_size=1,
    num_workers=1,
    verbose=3
)

# Construct results (a pandas DataFrame)
df = create_df(
    class_labels=class_labels,
```

(continues on next page)

(continued from previous page)

```
preds=preds,  
confs=confs,  
ids=ids,  
indices=indices,  
threshold=CONF_THRESH,  
verbose=3  
)
```



## 3.1 Concepts

`deepnog` is a command line tool written in Python 3. It uses deep networks for extremely fast protein orthology assignments. Currently, it is based on a deep convolutional network architecture called DeepNOG trained on the root and bacterial level of the eggNOG 5.0 database (Huerta-Cepas et al. (2019)).

Two subcommand are available:

- `deepnog infer` for assigning sequences to orthologous groups, using precomputed models, and
- `deepnog train` for training such models (e.g. other taxonomic levels or future versions of eggNOG, different orthology databases, etc.)

### 3.1.1 `deepnog infer` for orthology assignments

#### Input Data

DeepNOG expects a protein sequence file as input. It is tested for the FASTA file format, but in general should support all file formats supported by the Bio.SeqIO module of Biopython. Compressed files (.gz or .xz) are supported as well. Protein sequences without IDs in the input data file are skipped and not used for the following assignment phase. Furthermore, if two sequences in the input data file have the same associated ID, only the sequence encountered first in the input data file will be kept and all others discarded before the output file is created. The user will be notified if such cases are encountered.

#### Assignment Phase

In the assignment phase, `deepnog` loads a predefined deep network and the corresponding trained weights (defaults to DeepNOG trained on eggNOG 5.0, bacterial level). Then it performs the assignment by forwarding the input sequences through the network performing the calculations either on a CPU or GPU. `deepnog` offers single-process data loading aimed for calculations on a single CPU core to produce as little overhead as possible. Additionally, it offers parallel multiprocessing data loading aimed for very fast GPU calculations. This is to provide the GPU with data following up the previous forward pass fast enough such that the GPU does not experience idling. In its default parametrization, `deepnog` is optimized for single core CPU calculations, or massively parallel GPU calculations.

## Output Data

As an output deepnog generates a CSV file which consists of three columns:

1. The unique name or ID of the protein extracted from the sequence file,
2. the assigned orthologous group, and
3. the network's confidence in the assignment.

Each deep network model has the possibility to define an assignment confidence threshold below which, the network's output layer is treated as having predicted that the input protein sequence is not associated to any orthologous group in the model. Therefore, if the highest assignment confidence for any OG for a given input protein sequence is below this threshold, the assignment is left empty. Per default, using DeepNOG on eggNOG 5.0, the prediction confidence threshold is set to a strict 99%. This threshold can be adjusted by the user.

### 3.1.2 deepnog train for creating custom models

For details on training new models, see [New models/architectures](#).

## 3.2 Deepnog CLI Documentation

Invocation:

```
deepnog infer SEQUENCE_FILE [options] > assignments.csv
```

### 3.2.1 Basic Commands

These options may be commonly tuned for a basic invocation for orthologous group assignment.

```
positional arguments:
  SEQUENCE_FILE          File containing protein sequences for classification.

optional arguments:
  -h, --help              show this help message and exit
  --version               show program's version number and exit
  -o FILE, --out FILE     Store orthologous group assignments to outputfile.
                          Per default, write predictions to stdout. (default: None)
  -c FLOAT, --confidence-threshold FLOAT
                          The confidence value below which predictions are
                          masked by deepnog. By default, apply the confidence
                          threshold saved in the model if one exists, and else
                          do not apply a confidence threshold. (default: None)
```

### 3.2.2 Advanced Commands

These options are unlikely to require manual tuning for the average user.

<code>--verbose INT</code>	Define verbosity of DeepNOGs output written to stdout or stderr. 0 only writes errors to stderr which cause DeepNOG to abort and exit. 1 also writes warnings to stderr if e.g. a protein without an ID was found and skipped. 2 additionally writes general progress messages to stdout. 3 includes a dynamic progress bar of the prediction stage using tqdm. (default: 3)
<code>--ff STR, --ffformat STR</code>	File format of protein sequences. Must be supported by Biopythons Bio.SeqIO class. (default: fasta)
<code>-of {csv,tsv,legacy} --outformat {csv,tsv,legacy}</code>	The file format of the output file produced by deepnog. (default: csv)
<code>-d {auto,cpu,gpu}, --device {auto,cpu,gpu}</code>	Define device for calculating protein sequence classification. Auto chooses GPU if available, otherwise CPU. (default: auto)
<code>-db {eggNOG5}, --database {eggNOG5}</code>	Orthologous group/family database to use. (default: eggNOG5)
<code>-t {1,2}, --tax {1,2}</code>	Taxonomic level to use in specified database (1 = root, 2 = bacteria) (default: 2)
<code>-nw INT, --num-workers INT</code>	Number of subprocesses (workers) to use for data loading. Set to a value <= 0 to use single-process data loading. Note: Only use multi-process data loading if you are calculating on a gpu (otherwise inefficient)! (default: 0)
<code>-a {deepnog}, --architecture {deepnog}</code>	Network architecture to use for classification. (default: deepnog)
<code>-w FILE, --weights FILE</code>	Custom weights file path (optional) (default: None)
<code>-bs INT, --batch-size INT</code>	The batch size determines how many sequences are processed by the network at once. If 1, process the protein sequences sequentially (recommended on CPUs). Larger batch sizes speed up the inference and training on GPUs. Batch size can influence the learning process.
<code>--test_labels TEST_LABELS_FILE</code>	Measure model performance on a test set. If provided, this file must contain the ground-truth labels for the provided sequences. Otherwise, only perform inference.

## 3.3 API Documentation

This is the API documentation for deepnog.

### 3.3.1 DeepNOG

DeepNOG is a deep learning based command line tool to infer orthologous groups of given protein sequences. It provides a number of models for eggNOG orthologous groups, and allows to train additional models for eggNOG or other databases.

### 3.3.2 deepnog.client package

#### deepnog.client.client module

##### Authors

- Roman Feldbauer
- Lukas Gosch

##### Date

2019-10-18

##### Usage

*python client.py -help*

##### Description

Provides the deepnog command line client and entry point for users.

DeepNOG predicts protein families/orthologous groups of given protein sequences with deep learning.

Since version 1.2, model training is available as well.

File formats supported: Preferred: FASTA DeepNOG supports protein sequences stored in all file formats listed in <https://biopython.org/wiki/SeqIO> but is tested for the FASTA-file format only.

Architectures supported:

##### Databases supported:

- eggNOG 5.0, taxonomic level 1 (root)
- eggNOG 5.0, taxonomic level 2 (bacteria)
- Additional databases will be trained on demand/users can add custom databases using the training facilities.

`deepnog.client.client.main()`  
DeepNOG command line tool.



### 3.3.3 deepnpg.data package

#### deepnpg.data.dataset module

Author: Lukas Gosch

Date: 2019-10-03

Description:

Dataset classes and helper functions for usage with deep network models written in PyTorch.

```
class deepnpg.data.dataset.ProteinIterator (file_, labels: pandas.DataFrame,
                                             aa_vocab, f_format, n_skipped: Union[int,
                                             deepnpg.utils.sync.SynchronizedCounter] =
                                             0, num_workers=1, worker_id=0)
```

Bases: object

Iterator allowing for multiprocessing data loading of a sequence file.

ProteinIterator is a wrapper for the iterator returned by Biopython's Bio.SeqIO class when parsing a sequence file. It specifies custom `__next__()` method to support single- and multi-process data loading.

In the single-process loading case, nothing special happens, the ProteinIterator sequentially iterates over the data file. In the end, it informs the main module about the number of skipped sequences (due to empty ids) through setting a global variable in the main module.

In the multi-process loading case, each ProteinIterator loads a sequence and then skips the next few sequences dedicated to the other workers. This works by each worker skipping `num_worker - 1` data samples for each call to `__next__()`. Furthermore, each worker skips `worker_id` data samples in the initialization.

The ProteinIterator class also makes sure that a unique ID is set for each SeqRecord obtained from the data-iterator. This allows unambiguous handling of large protein datasets which may have duplicate IDs from merging multiple sources or may have no IDs at all. For easy and efficient sorting of batches of sequences as well as for direct access to the original IDs, the index is stored separately.

#### Parameters

- **file** (*str*) – Path to sequence file, from which an iterator over the sequences will be created with Biopython's Bio.SeqIO.parse() function.
- **labels** (*pd.DataFrame*) – Dataframe storing labels associated to the sequences. This is required for training, and ignored during inference. Must contain 'protein\_id' and 'label\_num' columns providing identifiers and numerical labels.
- **aa\_vocab** (*dict*) – Amino-acid vocabulary mapping letters to integers
- **f\_format** (*str*) – File format in which to expect the protein sequences. Must be supported by Biopython's Bio.SeqIO class.
- **num\_workers** (*int*) – Number of workers set in DataLoader or one if no workers set. If bigger or equal to two, the multi-process loading case happens.
- **worker\_id** (*int*) – ID of worker this iterator belongs to

```
deepnpg.data.dataset.collate_sequences (batch: Union[List[deepnpg.data.dataset.sequence],
deepnpg.data.dataset.sequence], zero_padding:
bool = True, min_length: int = 36,
random_padding: bool = False) →
deepnpg.data.dataset.collated_sequences
```

Collate and zero-pad encoded sequence.

#### Parameters

- **batch** (*namedtuple, or list of namedtuples*) – Batch of protein sequences to classify stored as a namedtuple sequence.
- **zero\_padding** (*bool*) – Zero-pad protein sequences, that is, append zeros until every sequence is as long as the longest sequences in batch. NOTE: currently unused. Zero-padding is always performed.
- **min\_length** (*int, optional*) – Zero-pad sequences to at least min\_length. By default, this is set to 36, which is the largest kernel size in the default DeepNOG/DeepEncoding architecture.
- **random\_padding** (*bool, optional*) – Zero pad sequences by prepending and appending zeros. The fraction is determined randomly. This may counter detrimental effects, when short sequences would always have long zero-tails, otherwise.

**Returns** **batch** – Input batch zero-padded and stored in namedtuple collated\_sequences.

**Return type** NamedTuple

`deepnog.data.dataset.gen_amino_acid_vocab(alphabet=None)`

Create vocabulary for protein sequences.

A vocabulary is defined as a mapping from the amino-acid letters in the alphabet to numbers. As this mapping is aware of zero-padding, it maps the first letter in the alphabet to 1 instead of 0.

**Parameters** **alphabet** (*str*) – Alphabet to use for vocabulary. If None, use ‘ACDEFGHIKLM-NPQRSTVWYBXZJUO’ (equivalent to deprecated Biopython’s ExtendedIUPACProtein).

**Returns** **vocab** – Mapping of amino acid characters to numbers.

**Return type** dict

## deepnog.data.split module

```
class deepnog.data.split.DataSplit(X_train: pandas.DataFrame, X_val: pandas.DataFrame,  
X_test: pandas.DataFrame, y_train: pandas.DataFrame,  
y_val: pandas.DataFrame, y_test: pandas.DataFrame,  
uniref_train: Optional[pandas.DataFrame], uniref_val:  
Optional[pandas.DataFrame], uniref_test: Op-  
tional[pandas.DataFrame])
```

Bases: object

Class for returned data, labels, and groups after train/val/test split.

**X\_test:** pandas.DataFrame

**X\_train:** pandas.DataFrame

**X\_val:** pandas.DataFrame

**uniref\_test:** Optional[pandas.DataFrame]

**uniref\_train:** Optional[pandas.DataFrame]

**uniref\_val:** Optional[pandas.DataFrame]

**y\_test:** pandas.DataFrame

**y\_train:** pandas.DataFrame

**y\_val:** pandas.DataFrame

```
deepnog.data.split.group_train_val_test_split(df: pandas.DataFrame, train_ratio:
float = 0.96, validation_ratio: float
= 0.02, test_ratio: float = 0.02, ran-
dom_state: int = 123, with_replacement:
bool = True, verbose: int = 0) →
deepnog.data.split.DataSplit
```

Create training/validation/test split for deepnog experiments.

Takes UniRef cluster IDs into account, that is, makes sure that sequences from the same cluster go into the same set. In other words, training, validation, and test sets are disjunct in terms of UniRef clusters.

#### Parameters

- **df** (*pandas DataFrame*) – Must contain ‘string\_id’, ‘eggnog\_id’, ‘uniref\_id’ columns
- **train\_ratio** (*float*) – Fraction of total sequences for training set
- **validation\_ratio** (*float*) – Fraction of total sequences for validation set
- **test\_ratio** (*float*) – Fraction of total sequences for test set
- **random\_state** (*int*) – Set random state for reproducible results
- **with\_replacement** (*bool*) – By default, scikit-learn GroupShuffleSplit samples objects with replacement. Disabling replacement removes
- **verbose** (*int*) – Level of logging verbosity

**Returns** **data\_split** – Split X, y, groups

**Return type** *NamedTuple*

```
deepnog.data.split.train_val_test_split(df: pandas.DataFrame, train_ratio: float = 0.96,
validation_ratio: float = 0.02, test_ratio: float =
0.02, stratify: bool = True, shuffle: bool = True,
random_state: int = 123, verbose: int = 0) →
deepnog.data.split.DataSplit
```

Create training/validation/test split for deepnog experiments.

Does not take UniRef clusters into account. Do not use for UniRef50/90 experiments.

#### Parameters

- **df** (*pandas DataFrame*) – Must contain ‘string\_id’, ‘eggnog\_id’ columns
- **train\_ratio** (*float*) – Fraction of total sequences for training set
- **validation\_ratio** (*float*) – Fraction of total sequences for validation set
- **test\_ratio** (*float*) – Fraction of total sequences for test set
- **stratify** (*bool*) – Stratify the splits according to the orthology labels
- **shuffle** (*bool*) – Shuffle the sequences
- **random\_state** (*int*) – Set random state for reproducible results
- **verbose** (*int*) – Level of logging verbosity

**Returns** **data\_split** – Split X, y, groups

**Return type** *DataSplit*

### 3.3.4 deepnog.learning package

#### deepnog.learning.inference module

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Predict orthologous groups of protein sequences.

```
deepnog.learning.inference.predict(model, dataset, device='cpu', batch_size=16,  
                                   num_workers=4, verbose=3)
```

Use model to predict zero-indexed labels of dataset.

Also handles communication with ProteinIterators used to load data to log how many sequences have been skipped due to having empty sequence ids.

##### Parameters

- **model** (*nn.Module*) – Trained neural network model.
- **dataset** (*ProteinIterableDataset*) – Data to predict protein families for.
- **device** (*[str, torch.device]*) – Device of model.
- **batch\_size** (*int*) – Forward batch\_size proteins through neural network at once.
- **num\_workers** (*int*) – Number of workers for data loading.
- **verbose** (*int*) – Define verbosity.

##### Returns

- **preds** (*torch.Tensor, shape (n\_samples,)*) – Stores the index of the output-node with the highest activation
- **confs** (*torch.Tensor, shape (n\_samples,)*) – Stores the confidence in the prediction
- **ids** (*list[str]*) – Stores the (possible empty) protein labels extracted from data file.
- **indices** (*list[int]*) – Stores the unique indices of sequences mapping to their position in the file

#### deepnog.learning.training module

Author: Roman Feldbauer

Date: 2020-06-03

Description:

Training deep networks for protein orthologous group prediction.

```
deepnog.learning.training.fit(architecture, module, cls, training_sequences, validation_sequences,
                             training_labels, validation_labels, *,
                             data_loader_params: dict = None, iterable_dataset: bool =
                             False, n_epochs: int = 15, shuffle: bool = False, learning_rate:
                             float = 0.01, learning_rate_params: dict = None, l2_coeff: float
                             = None, optimizer_cls=torch.optim.Adam, device: Union[str,
                             torch.device] = 'auto', tensorboard_dir: Union[None, str] =
                             'auto', log_interval: int = 100, random_seed: int = None,
                             save_each_epoch: bool = True, out_dir: pathlib.Path = None,
                             experiment_name: str = None, config_file: str = None, verbose:
                             int = 2) → deepnog.learning.training.train_val_result
```

Perform training and validation of a given model, data, and hyperparameters.

#### Parameters

- **architecture** (*str*) – Network architecture, must be available in deepnog/models
- **module** (*str*) – Python module containing the network definition (inside deepnog/models/).
- **cls** (*str*) – Python class name of the network (inside deepnog/models/{module}.py).
- **training\_sequences** (*str*, *Path*) – File with training set sequences
- **validation\_sequences** (*str*, *Path*) – File with validation set sequences
- **training\_labels** (*str*, *Path*) – File with class labels (orthologous groups) of training sequences
- **validation\_labels** (*str*, *Path*) – File with class labels (orthologous groups) of validation sequences
- **data\_loader\_params** (*dict*) – Parameters passed to PyTorch DataLoader construction
- **iterable\_dataset** (*bool*, *default False*) – Use an iterable dataset that does not load all sequences in advance. While this saves memory and does not involve the delay at start, random sampling is impaired, and requires a shuffle buffer.
- **n\_epochs** (*int*) – Number of training passes over the complete training set
- **shuffle** (*bool*) – Shuffle the training data. This does NOT shuffle the complete data set, which requires having all sequences in memory, but uses a shuffle buffer (default size: 2\*\*16), from which sequences are drawn.
- **learning\_rate** (*float*) – Learning rate, the central hyperparameter of deep network training. Too high values may lead to diverging solutions, while too low values result in slow learning.
- **learning\_rate\_params** (*dict*) – Parameters passed to the learning rate Scheduler.
- **l2\_coeff** (*float*) – If not None, regularize training by L2 norm of network weights
- **optimizer\_cls** – Class of PyTorch optimizer
- **device** (*torch.device*) – Use either ‘cpu’ or ‘cuda’ (GPU) for training/validation.
- **tensorboard\_dir** (*str*) – Save online learning statistics for tensorboard in this directory.
- **log\_interval** (*int*, *optional*) – Print intermediary results after *log\_interval* minibatches
- **random\_seed** (*int*) – Set a random seed for numpy/pytorch for reproducible results.

- **save\_each\_epoch** (*bool*) – Save the network after each training epoch
- **out\_dir** (*Path*) – Path to the output directory used to save models during training
- **experiment\_name** (*str*) – Prefix of model files saved during training
- **config\_file** (*str*) – Override path to config file, e.g. for custom models in unit tests
- **verbose** (*int*) – Increasing levels of messages

#### Returns

**results** –

**A namedtuple containing:**

- the trained deep network model
- training dataset
- evaluation statistics
- the ground truth labels (*y\_true*)
- the predicted labels (*y\_pred*).

**Return type** namedtuple

### 3.3.5 deepnog.models package

#### deepnog.models.deepencoding module

Author: Lukas Gosch, Roman Feldbauer

Date: 2019-10-09

Description: Convolutional networks for protein orthologous group inference. LEGACY MODULE kept just in case renaming to deepnog.py/DeepNOG causes any unforeseen issues.

#### deepnog.models.deepfam module

Author: Lukas Gosch, Roman Feldbauer

#### deepnog.models.deepnog module

Author: Lukas Gosch, Roman Feldbauer

Date: 2019-10-09

Description: Convolutional networks for protein orthologous group assignment.

### 3.3.6 deepnog.tests package

#### deepnog.tests.utils module

`deepnog.tests.utils.get_deepnog_root()` → `pathlib.Path`

#### Module contents

Description:

Helpers for deepnog tests.

##### Including:

- test data
- test network weights (parameters)
- some helper functions

Individual tests are located within the respective deepnog subpackages.

### 3.3.7 deepnog.utils package

#### deepnog.utils.bio module

`deepnog.utils.bio.parse(p: pathlib.Path, fformat: str = 'fasta', alphabet=None)` → `Iterator`

Parse a possibly compressed sequence file.

##### Parameters

- **p** (*Path* or *str*) – Path to sequence file
- **fformat** (*str*) – File format supported by Biopython.SeqIO.parse, e.g “fasta”
- **alphabet** (*any*) – Pass alphabet to SeqIO.parse

**Returns** **it** – The SeqIO.parse iterator yielding SeqRecords

**Return type** `Iterator`

#### deepnog.utils.config module

`deepnog.utils.config.get_config(config_file: Optional[Union[pathlib.Path, str]] = None)` → `Dict`

Get a config dictionary

If no file is provided, look in the DEEPNOG\_CONFIG env variable for the path. If this fails, load a default config file (lacking any user customization).

This contains the available models (databases, levels). Additional config may be added in future releases.

## deepnog.utils.io\_utils module

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Input/output helper functions

`deepnog.utils.io_utils.create_df` (*class\_labels: list, preds: torch.Tensor, confs: torch.Tensor, ids: List[str], indices: List[int], threshold: float = None*)

Creates one dataframe storing all relevant prediction information.

The rows in the returned dataframe have the same order as the original sequences in the data file. First column of the dataframe represents the position of the sequence in the datafile.

### Parameters

- **class\_labels** (*list*) – Store class name corresponding to an output node of the network.
- **preds** (*torch.Tensor, shape (n\_samples,)*) – Stores the index of the output-node with the highest activation
- **confs** (*torch.Tensor, shape (n\_samples,)*) – Stores the confidence in the prediction
- **ids** (*list[str]*) – Stores the (possible empty) protein labels extracted from data file.
- **indices** (*list[int]*) – Stores the unique indices of sequences mapping to their position in the file
- **threshold** (*float*) – If given, prediction labels and confidences are set to “” if confidence in prediction is not at least threshold.

**Returns** **df** – Stores prediction information about the input protein sequences. Duplicates (defined by their `sequence_id`) have been removed from df.

**Return type** `pandas.DataFrame`

`deepnog.utils.io_utils.get_data_home` (*data\_home: str = None, verbose: int = 0*) → `pathlib.Path`

Return the path of the deepnog data dir.

This folder is used for large files that cannot go into the Python package on PyPI etc. For example, the network parameters (weights) files may be larger than 100MiB. By default the data dir is set to a folder named ‘deepnog\_data’ in the user home folder. Alternatively, it can be set by the ‘DEEPNOG\_DATA’ environment variable or programmatically by giving an explicit folder path. If the folder does not already exist, it is automatically created.

### Parameters

- **data\_home** (*str | None*) – The path to deepnog data dir.
- **verbose** (*int*) – Log or not.



## Notes

Adapted from [SKLEARN\\_DATAHOME](#).

`deepnog.utils.io_utils.get_weights_path(database: str, level: str, architecture: str, data_home: str = None, download_if_missing: bool = True, verbose: int = 0) → pathlib.Path`

Get path to neural network weights.

This is a path on local storage. If the corresponding files are not present, download from remote storage. The default remote URL can be overridden by setting the environment variable DEEPNOG\_REMOTE.

### Parameters

- **database** (*str*) – The orthologous groups database. Example: eggNOG5
- **level** (*str*) – The taxonomic level within the database. Example: 2 (for bacteria)
- **architecture** (*str*) – Network architecture. Example: deepencoding
- **data\_home** (*str*, *optional*) – Specify another download and cache folder for the weights. By default all deepnog data is stored in '\$HOME/deepnog\_data' subfolders.
- **download\_if\_missing** (*boolean*, *default=True*) – If False, raise an IOError if the data is not locally available instead of trying to download the data from the source site.
- **verbose** (*int*) – Log or not

**Returns** `weights_path` – Path to file of network weights

**Return type** Path

## deepnog.utils.logger module

`deepnog.utils.logger.get_logger(initname: str = 'deepnog', verbose: int = 0) → logging.Logger`  
This function provides a nicely formatted logger.

### Parameters

- **initname** (*str*) – The name of the logger to show up in log.
- **verbose** (*int*) – Increasing levels of verbosity

## References

Shamelessly stolen from phenotrex

## deepnog.utils.metrics module

`deepnog.utils.metrics.estimate_performance(df_true: pandas.DataFrame, df_pred: pandas.DataFrame) → Dict`

Calculate various model performance measures.

### Parameters

- **df\_true** (*pandas.DataFrame*) – The ground truth labels. DataFrame must contain 'sequence\_id' and 'label' columns.
- **df\_pred** (*pandas.DataFrame*) – The predicted labels. DataFrame must contain 'sequence\_id' and 'prediction' columns.

**Returns****perf** –**Performance estimates:**

- `macro_precision`
- `micro_precision`
- `macro_recall`
- `micro_recall`
- `macro_f1`
- `micro_f1`
- `accuracy`
- `mcc`

**Return type** dict**deepnog.utils.network module**

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Various utility functions

`deepnog.utils.network.count_parameters` (*model*, *tunable\_only*: *bool* = *True*) → int  
Count the number of parameters in the given model.

**Parameters**

- **model** (*torch.nn.Module*) – PyTorch model (deep network)
- **tunable\_only** (*bool*, *optional*) – Count only tunable network parameters

**References**

<https://stackoverflow.com/questions/49201236/check-the-total-number-of-parameters-in-a-pytorch-model>

`deepnog.utils.network.load_nn` (*architecture*: *Union[str, Sequence[str]]*, *model\_dict*: *dict* = *None*,  
*phase*: *str* = 'eval', *device*: *Union[torch.device, str]* = 'cpu', *verbose*: *int* = 0)

Import NN architecture and set loaded parameters.

**Parameters**

- **architecture** (*str* or *list-like* of two *str*) – If single string: name of neural network module and class to import. E.g. 'deepencoding' will load `deepnog.models.deepencoding.deepencoding`. Otherwise, separate module and class name of deep network to import. E.g. ('deephought', 'DeepNettigkeit') will load `deepnog.models.deephought.DeepNettigkeit`.
- **model\_dict** (*dict*, *optional*) – Dictionary holding all parameters and hyperparameters of the model. Required during inference, optional for training.

- **phase** (`['train', 'infer', 'eval']`) – Set network in training or inference=evaluation mode with effects on storing gradients, dropout, etc.
- **device** (`[str, torch.device]`) – Device to load the model into.
- **verbose** (`int`) – Increasingly verbose logging

**Returns model** – Neural network object of type architecture with parameters loaded from `model_dict` and moved to device.

**Return type** `torch.nn.Module`

`deepnog.utils.network.set_device(device: Union[str, torch.device]) → torch.device`

Set device (CPU/GPU) depending on user choice and availability.

**Parameters device** (`[str, torch.device]`) – Device set by user as an argument to DeepNOG call.

**Returns device** – Object containing the device type to be used for prediction calculations.

**Return type** `torch.device`

### deepnog.utils.sync module

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Parallel processing helpers

**class** `deepnog.utils.sync.SynchronizedCounter` (*init: int = 0*)

Bases: `object`

A multiprocessing-safe counter.

**Parameters init** (*int, optional*) – Counter starts at init (default: 0)

**increment** (*n=1*)

Obtain a lock before incrementing, since += isn't atomic.

**Parameters n** (*int, optional*) – Increment counter by n (default: 1)

**increment\_and\_get\_value** (*n=1*) → `int`

Obtain a lock before incrementing, since += isn't atomic.

**Parameters n** (*int, optional*) – Increment counter by n (default: 1)

**property value**

## 3.4 Deepnog New Models and Architectures

deepnog is developed with extensibility in mind, and allows to plug in additional models (for different taxonomic levels, or different orthology databases). It also supports addition of new network architectures.

In order to register a new network architecture, we recommend an editable installation with pip, as described in *Installation from Source*.

### 3.4.1 Training scripts

Starting with v1.2.0, deepnog ships with functions for training custom models. Consider we are training a DeepNOG model for eggNOG 5, level 1239 (Firmicutes):

```
deepnog train \
  -a "deepnog" \
  -o /path/to/output/ \
  -db "eggNOG5" \
  -t "1239" \
  --shuffle \
  train.faa.gz \
  val.faa.gz \
  train.csv.gz \
  val.csv.gz
```

Run `deepnog train --help` for additional options.

In order to assess the new model's quality, run the following commands:

```
deepnog infer \
  -a "deepnog" \
  -w /path/to/output/MODEL_FILENAME.pth \
  -o /path/to/output/assignments.csv \
  --test_labels test.csv.gz \
  test.faa.gz
cat /path/to/output/assignments.performance.csv
```

This provides a number of performance measures, including accuray, macro averaged precision and recall, among others.

### 3.4.2 Register new models

New models for additional taxonomic levels in eggNOG 5 or even different orthology databases using existing network architectures must be placed in the deepnog data directory as specified by the DEEPNOG\_DATA environment variable (default: `$HOME/deepnog_data`).

The directory looks like this:

```
| deepnog_data
| |--- eggNOG5
| |   |--- 1
| |   |   |--- deepnog.pth
| |   |--- 2
| |   |   |--- deepnog.pth
| |   ...
| |
| |
| |
```

In order to add a root level model for “MyOrthologyDB”, we place the serialized PyTorch parameters like this:

```
| deepnog_data
| |--- eggNOG5
| |   |--- 1
| |   |   |--- deepnog.pth
| |   |--- 2
| |   |   |--- deepnog.pth
| |
| |
```

(continues on next page)

(continued from previous page)

```
| | MyOrthologyDB
| |   | 1
| |   | deepnog.pth
| | ...
|
```

### 3.4.3 Register new network architectures

Create a Python module `deepnog/models/<my_network.py>`. You can use `deepnog.py` as a template. A new architecture `MyNetworkA` would look like so:

```
import torch.nn as nn

class MyNetworkA(nn.Module):
    """ A revolutionary network for orthology prediction. """
    def __init__(self, model_dict):
        super().__init__()
        param1 = model_dict['param1']
        param2 = model_dict['param2']
        param3 = model_dict.get('param3', 0.)
        ...
    def forward(self, x):
        ...
        return x
```

When the new module is in place, also edit `deepnog/config/deepnog_config.py` to expose the new network to the user:

```
architecture:
  netA:
    module: my_network
    class: MyNetworkA
    param1: 'settingXYZ'
    param2:
      - 2
      - 4
      - 8
    param3: 150
    # ... all hyperparameters required for class init

  deepnog:
    module: deepnog
    class: DeepNOG
    encoding_dim: 10
    kernel_size:
      - 8
      - 12
      - 16
      - 20
      - 24
      - 28
      - 32
      - 36
    n_filters: 150
```

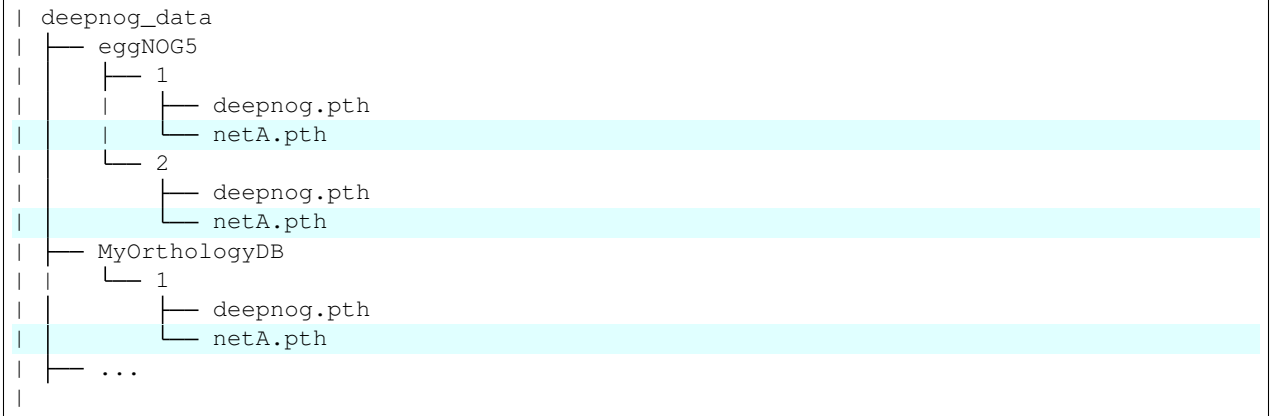
(continues on next page)

(continued from previous page)

```
dropout: 0.3
pooling_layer_type: 'max'
```

The new network can now be used in deepnog by specifying parameter `-a netA`.

Assuming we want to compare deepnog to netA, we add the trained network parameters like this:



Finally, expose the new models to the user by modifying `deepnog/config/deepnog_config.py` again. The relevant section is database.

```
database:
  eggNOG5:
    # taxonomic levels
    - 1
    - 2
    - 1236
    - 1239 # Example 1: Uncomment this line, if you created a Firmicutes model
  MyOrthologyDB: # Example 2: Uncomment this line and the following, if you
    - 1 # created a model for the '1' level of MyOrthologyDB.
```

Notes:

- Currently, a level must be provided, even if the database does not use levels. Simply use a placeholder 1 or similar.
- Indentation matters

## 3.5 File formats

deepnog uses standard file formats, as detailed below for eggNOG 5 (1239, Firmicutes) data.

### 3.5.1 Protein sequences

Protein sequences are expected in FASTA format. Each entry must contain a unique record ID. That is, a `user_data.faa` should look like this:

```
>1000569.HMPREF1040_0002
MMKHDDHVVHQIRTEPIYAILGETFSRGRNTNRQVAKALLGAGVRIIQYREKEKSWQEKYEE
ARDICQWCNEYGATFIMNDSIDLAIACEAPAIHVGQDDAPVAWVRRLAQRDIVVGVSTHT
IAEMKKAVRDGDYVGLGPMYQTTSKMDVHDIADVDKAYALTLPVPVVTIGGIDLIHIR
QLYTEGFRSFAMISALVGATDIVEQIGAFRQVLQEKIDEC
>1000569.HMPREF1040_0003
MATTVGDIVTYLQGIAPLYLKEEWDNPGLLLGNQGDPVSSVLVTLDMEGTVDYAIAEGI
SFIFSHHPLIMKGKAIARTDSYDGRMYQKLLSHHIAVYAAHTNLD SATGGVNDVLAEHLQ
LQHVRFPIPGVSESLYKIAIYVPKGYGDAIREVLGKHDAGHLGAYSYSFSVAGQGRFKP
LAGTHPFIGKRDVLETVEEERETIVEGSRLGEVITAMLAVHPYEEPAYDIYPLYQQRTA
LGLGRLGELATPLSSMAAVQWVKEALHLTHVS YAGPMDRQIQTI AVLGGSGAEFIATAKA
AGATLYVTGDMKYHAAQEAIAIKQGILVVDAGHFGTEFPVIDRMQNIEAENEKQGWHIQCV
VDPTAMDMIQRL
```

Compression is allowed (`user_data.faa.gz`, or `user_data.faa.xz`). For typical usage of deepnog infer for protein orthologous group assignments this is already sufficient.

### 3.5.2 Protein orthologous group labels

Training new models with deepnog train, or assessing model quality with deepnog infer --test\_labels require providing the orthologous group labels.

File format is CSV (comma-separated values) with a preceding header line, and three columns (index, sequence record ID, orthologous group ID).

```
,string_id,eggno_id
1543720,1121929.KB898683_gene1916,1V3NB
351865,536232.CLM_3459,1TPCN
[...]
1570381,1000569.HMPREF1040_0002,1V3ZR
744166,1000569.HMPREF1040_0003,1TQ27
[...]
426023,1423743.JCM14108_56,1TPGE
```

To construct some `user_data.csv`:

- Copy (do not modify) the header line.
- Provide an index in the first column (e.g. 1..N; currently unused, but required).
- Provide the sequence ID (e.g. eggNOG/STRING ID) in column 2.
- Provide its corresponding group label in column 3.
- Sequence IDs in column 2 must match the IDs used in the `user_data.faa`.

### 3.5.3 Assignment output

Orthologous group assignments are output in tabular format (comma-separated).

- Column 1: Sequence ID
- Column 2: Assignment/Orthologous group
- Column 3: Assignment confidence in 0..1 (higher=better).

Example:

```
sequence_id,prediction,confidence
1000565.METUNv1_00038,COG0466,1.0
1000565.METUNv1_00060,COG0500,0.20852506
1000565.METUNv1_00091,COG0810,0.9999591
1000565.METUNv1_00093,COG0659,1.0
1000565.METUNv1_00103,COG5000,0.70716757
1000565.METUNv1_00105,COG0346,0.9999982
1000565.METUNv1_00106,COG3791,1.0
1000565.METUNv1_00114,COG0239,1.0
1000565.METUNv1_00115,COG1643,1.0
```



## CONTRIBUTING

*deepnog* is free open source software. Contributions from the community are highly appreciated. Even small contributions improve the software's quality.

Even if you are not familiar with programming languages and tools, you may contribute by filing bugs or any problems as a [GitHub issue](#).

### 4.1 Git and branching model

We use *git* for version control (CVS), as do most projects nowadays. If you are not familiar with git, there are lots of tutorials on [GitHub Guide](#). All the important basics are covered in the [GitHub Git handbook](#).

Development of *deepnog* (mostly) follows this [git branching model](#). We currently use one main branch: master. For any changes, a new branch should be created. This includes new feature, noncritical or critical bug fixes, etc.

### 4.2 Workflow

In case of large changes to the software, please first get in contact with the authors for coordination, for example by filing an [issue](#). If you want to fix small issues (typos in the docs, obvious errors, etc.) you can - of course - directly submit a pull request (PR).

1. **Create a fork of *deepnog* in your GitHub account.** Simply click “Fork” button on <https://github.com/univieCUBE/deepnog>.
2. **Clone your fork on your computer.** `$ git clone git@github.com:YOUR-ACCOUNT-GOES-HERE/deepnog.git && cd deepnog`
3. **Add remote upstream.** `$ git remote add upstream git@github.com:univieCUBE/deepnog.git`
4. **Create feature/bugfix branch.** `$ git checkout -b bugfix123 master`
5. **Implement feature/fix bug/fix typo/...** Happy coding!
6. **Create a commit with meaningful message** If you only modified existing files, simply `$ git commit -am "descriptive message what this commit does (in present tense) here"`
7. **Push to GitHub** e.g. `$ git push origin featureXYZ`
8. **Create pull request (PR)** Git will likely provide a link to directly create the PR. If not, click “New pull request” on your fork on GitHub.

9. **Wait...** Several devops checks will be performed automatically (e.g. continuous integration (CI) with Travis, AppVeyor).

The authors will get in contact with you, and may ask for changes.

10. **Respond to code review.** If there were issues with continuous integration, or the authors asked for changes, please create a new commit locally, and simply push again to GitHub as you did before. The PR will be updated automatically.
11. **Maintainers merge PR, when all issues are resolved.** Thanks a lot for your contribution!

## 4.3 Code style and further guidelines

- Please make sure all code complies with [PEP 8](#)
- All code should be documented sufficiently (functions, classes, etc. must have docstrings with general description, parameters, ideally return values, raised exceptions, notes, etc.)
- Documentation style is [NumPy format](#).
- New code must be covered by unit tests using [pytest](#).
- If you fix a bug, please provide regression tests (fail on old code, succeed on new code).
- It may be helpful to install *deepnog* in editable mode for development. When you have already cloned the package, switch into the corresponding directory, and

```
pip install -e .
```

(don't omit the trailing period). This way, any changes to the code are reflected immediately. That is, you don't need to install the package each and every time, when you make changes while developing code.

## 4.4 Testing

In *deepnog*, we aim for high code coverage. As of Feb 2020, more than 95% of all code lines are visited at least once when running the complete test suite. This is primarily to ensure:

- correctness of the code (to some extent) and
- maintainability (new changes don't break old code).

Creating a new PR, ideally all code would be covered by tests. Sometimes, this is not feasible or only with large effort. Pull requests will likely be accepted, if the overall code coverage at least does not decrease.

Unit tests are automatically performed for each PR using CI tools online. This may take some time, however. To run the tests locally, you need *pytest* installed. From the *deepnog* directory, call

```
pytest deepnog/
```

to run all the tests. You can also restrict the tests to the subpackage you are working on, down to single tests. For example

```
pytest deepnog/tests/test_dataset.py --showlocals -v
```

only runs tests about datasets.

In order to check code coverage locally, you need the [pytest-cov plugin](#).

```
pytest deepnog --cov=deepnog
```



## CHANGELOG

### 5.1 Next release

#### 5.1.1 Added in 1.2.2

- Install from bioconda
- Support for 109 taxonomic levels in eggNOG 5 (was three before) (e.g. `deepnog infer -db eggnog5 -t 1239` for Firmicutes)
- Support for COG2020 (use `deepnog infer -db cog2020 -t 1`)

#### 5.1.2 Fixes/changes in 1.2.2

- Requirement PyYAML
- Test class imports
- Exit on requesting unavailable device (instead of raising an error)

### 5.2 1.2.1 - 2020-08-28

#### 5.2.1 Added in 1.2.1

- Training custom models: Users can now train additional models for further tax. levels of eggNOG 5 or even different orthology databases
- TensorBoard status reports: Follow training/validation loss online
- Support for configuration file (`deepnog_config.yml`)
- Model quality assessment

### 5.2.2 Changed in 1.2.1

- The command line invocation now uses two subcommands:
  - `deepnog train` for training new models, and
  - `deepnog infer` for general orthologous group assignment (and model quality assessment)

### 5.2.3 Fixed in 1.2.1

- Fixed packaging issue in 1.2.0 (which was subsequently removed altogether)
- Several additional bug fixes and smaller changes

## 5.3 1.1.0 - 2020-02-28

### 5.3.1 Added

- EggNOG5 root (tax 1) prediction

### 5.3.2 Changed

- Package structure changed for higher modularity. This will require changes in downstream usages.
- Remove network weights from the repository, because files are too large for github and/or PyPI. `deepnog` automatically downloads these from [CUBE](#) servers, and caches them locally.
- More robust inter-process communication in data loading

### 5.3.3 Fixes

- Fix error on very short amino acid sequences
- Fix error on unrecognized symbols in sequences (stop codons etc.)
- Fix multiprocessing data loading from gzipped files
- Fix type mismatch in deepencoding embedding layer (Windows only)

### 5.3.4 Maintenance

- Continuous integration on
  - [Travis](#) (Linux, MacOS)
  - [AppVeyor](#) (Windows)
- [Codecov](#) coverage reports
- [LGTM](#) code quality/security reports
- Documentation on [ReadTheDocs](#)
- Upload to [PyPI](#), thus enabling `$ pip install deepnog`.

## 5.4 1.0.0 - 2019-10-18

The first release of deepnog to appear in this changelog. It already contains the following features:

- EggNOG5 bacteria (tax 2) prediction
- DeepEncoding architecture
- CPU and GPU support
- Runs on all major platforms (Linux, MacOS, Windows)





## GETTING STARTED

Get started with `deepnog` in a breeze. Find how to [install the package](#) and see all core functionality applied in a single [quick start example](#).



## USER GUIDE

The [User Guide](#) introduces the main concepts of deepnog. It also contains complete [CLI](#) and [API](#) documentations of the package.



## DEVELOPMENT

There are several possibilities to [contribute](#) to this free open source software. We highly appreciate all input from the community, be it bug reports or code contributions.

Source code, issue tracking, discussion, and continuous integration appear on our [GitHub page](#).



## WHAT'S NEW

To see what's new in the latest version of `deepnpg`, have a look at the [changelog](#).





## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### d

- `deepnog`, [12](#)
- `deepnog.client.client`, [12](#)
- `deepnog.data.dataset`, [13](#)
- `deepnog.data.split`, [14](#)
- `deepnog.learning.inference`, [16](#)
- `deepnog.learning.training`, [16](#)
- `deepnog.models.deepencoding`, [18](#)
- `deepnog.models.deepfam`, [18](#)
- `deepnog.models.deepnog`, [18](#)
- `deepnog.tests`, [19](#)
- `deepnog.tests.utils`, [19](#)
- `deepnog.utils.bio`, [19](#)
- `deepnog.utils.config`, [19](#)
- `deepnog.utils.io_utils`, [20](#)
- `deepnog.utils.logger`, [21](#)
- `deepnog.utils.metrics`, [21](#)
- `deepnog.utils.network`, [22](#)
- `deepnog.utils.sync`, [23](#)



## C

`collate_sequences()` (in module *deepnog.data.dataset*), 13  
`count_parameters()` (in module *deepnog.utils.network*), 22  
`create_df()` (in module *deepnog.utils.io\_utils*), 20

## D

`DataSplit` (class in *deepnog.data.split*), 14  
`deepnog`  
    module, 12  
`deepnog.client.client`  
    module, 12  
`deepnog.data.dataset`  
    module, 13  
`deepnog.data.split`  
    module, 14  
`deepnog.learning.inference`  
    module, 16  
`deepnog.learning.training`  
    module, 16  
`deepnog.models.deepencoding`  
    module, 18  
`deepnog.models.deepfam`  
    module, 18  
`deepnog.models.deepnog`  
    module, 18  
`deepnog.tests`  
    module, 19  
`deepnog.tests.utils`  
    module, 19  
`deepnog.utils.bio`  
    module, 19  
`deepnog.utils.config`  
    module, 19  
`deepnog.utils.io_utils`  
    module, 20  
`deepnog.utils.logger`  
    module, 21  
`deepnog.utils.metrics`  
    module, 21  
`deepnog.utils.network`

module, 22

`deepnog.utils.sync`  
    module, 23

## E

`estimate_performance()` (in module *deepnog.utils.metrics*), 21

## F

`fit()` (in module *deepnog.learning.training*), 16

## G

`gen_amino_acid_vocab()` (in module *deepnog.data.dataset*), 14  
`get_config()` (in module *deepnog.utils.config*), 19  
`get_data_home()` (in module *deepnog.utils.io\_utils*), 20  
`get_deepnog_root()` (in module *deepnog.tests.utils*), 19  
`get_logger()` (in module *deepnog.utils.logger*), 21  
`get_weights_path()` (in module *deepnog.utils.io\_utils*), 21  
`group_train_val_test_split()` (in module *deepnog.data.split*), 14

## I

`increment()` (*deepnog.utils.sync.SynchronizedCounter* method), 23  
`increment_and_get_value()`  
    (*deepnog.utils.sync.SynchronizedCounter* method), 23

## L

`load_nn()` (in module *deepnog.utils.network*), 22

## M

`main()` (in module *deepnog.client.client*), 12  
`module`  
    `deepnog`, 12  
    `deepnog.client.client`, 12  
    `deepnog.data.dataset`, 13  
    `deepnog.data.split`, 14

[deepnog.learning.inference, 16](#)  
[deepnog.learning.training, 16](#)  
[deepnog.models.deepencoding, 18](#)  
[deepnog.models.deepfam, 18](#)  
[deepnog.models.deepnog, 18](#)  
[deepnog.tests, 19](#)  
[deepnog.tests.utils, 19](#)  
[deepnog.utils.bio, 19](#)  
[deepnog.utils.config, 19](#)  
[deepnog.utils.io\\_utils, 20](#)  
[deepnog.utils.logger, 21](#)  
[deepnog.utils.metrics, 21](#)  
[deepnog.utils.network, 22](#)  
[deepnog.utils.sync, 23](#)

## P

[parse\(\) \(in module deepnog.utils.bio\), 19](#)  
[predict\(\) \(in module deepnog.learning.inference\), 16](#)  
[ProteinIterator \(class in deepnog.data.dataset\), 13](#)

## S

[set\\_device\(\) \(in module deepnog.utils.network\), 23](#)  
[SynchronizedCounter \(class in deepnog.utils.sync\), 23](#)

## T

[train\\_val\\_test\\_split\(\) \(in module deepnog.data.split\), 15](#)

## U

[uniref\\_test \(deepnog.data.split.DataSplit attribute\), 14](#)  
[uniref\\_train \(deepnog.data.split.DataSplit attribute\), 14](#)  
[uniref\\_val \(deepnog.data.split.DataSplit attribute\), 14](#)

## V

[value\(\) \(deepnog.utils.sync.SynchronizedCounter property\), 23](#)

## X

[X\\_test \(deepnog.data.split.DataSplit attribute\), 14](#)  
[X\\_train \(deepnog.data.split.DataSplit attribute\), 14](#)  
[X\\_val \(deepnog.data.split.DataSplit attribute\), 14](#)

## Y

[y\\_test \(deepnog.data.split.DataSplit attribute\), 14](#)  
[y\\_train \(deepnog.data.split.DataSplit attribute\), 14](#)  
[y\\_val \(deepnog.data.split.DataSplit attribute\), 14](#)