deepnog

Release 1.2.1

Lukas Gosch, Roman Feldbauer

Aug 28, 2020

GETTING STARTED

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

CHAPTER

ONE

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 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
```

1.3 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`.

1.4 Supported platforms

`deepnog` currently supports all major operating systems:

- Linux
- MacOS X
- Windows

CHAPTER

TWO

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.inference 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, 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=threshold,
    verbose=3
)
```

CHAPTER

THREE

USER GUIDE

3.1 Concepts

DeepNOG is a command line tool written in Python 3. It uses trained neural networks for extremely fast protein homology predictions. In its current installation, it is based upon a neural network architecture called DeepEncoding trained on the root and bacterial level of the eggNOG 5.0 database (Huerta-Cepas et al. (2019)).

3.1.1 Input Data

As an input DeepNOG expects a protein sequence file which can also be provided gzipped. It is tested for the FASTA file format but in general should support all file formats supported by the Bio.SeqIO module of Biopython. Following the conventions in the bioinformatics field, protein sequences, given no IDs in the input data file, are skipped and not used for the following prediction 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.

3.1.2 Prediction Phase

In the prediction phase, DeepNOG loads a predefined neural network and the corresponding trained weights (defaults to DeepEncoding trained on eggNOG 5.0 (bacterial level)). Then it performs the prediction through 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 multiprocess 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, for details on how to best exploit GPUs for orthologous group predictions using DeepNOG, the reader is referred to the advanced Section 3.3 in this user's guide.

3.1.3 Output Data

As an output DeepNOG generates a CSV file which consists of three columns. First, the unique names or IDs of the proteins extracted from the sequence file, the second column corresponds to the OG-predictions and in the third column the confidence of the neural network in the prediction is stored. Each neural network model has the possibility to define a prediction confidence threshold below which, the neural 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 prediction confidence for any OG for a given input protein sequence is below this threshold, the prediction is left empty. Per default, using DeepEncoding on eggNOG 5.0, the prediction confidence threshold is set to a strict 99%. This threshold can be adjusted by the user.

3.2 Deepnog CLI Documentation

Invocation:

```
deepnog SEQUENCE_FILE [options] > predictions.csv
```

3.2.1 Basic Commands

These options may be commonly tuned for a basic invocation for OG prediction.

```
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 predictions 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.

```
--verbose INT       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)
-ff STR, --fformat STR
                    File format of protein sequences. Must be supported by
                    Biopythons Bio.SeqIO class. (default: fasta)
-of {csv,tsv,legacy}  --outformat {csv,tsv,legacy}
                    The file format of the output file produced by
                    deepnog. (default: csv)
-d {auto,cpu,gpu}, --device {auto,cpu,gpu}
                    Define device for calculating protein sequence
                    classification. Auto chooses GPU if available,
                    otherwise CPU. (default: auto)
-db {eggNOG5}, --database {eggNOG5}
                    Orthologous group/family database to use. (default:
                    eggNOG5)
-t {1,2}, --tax {1,2}
                    Taxonomic level to use in specified database
                    (1 = root, 2 = bacteria) (default: 2)
-nw INT, --num-workers INT
                    Number of subprocesses (workers) to use for data
                    loading. Set to a value <= 0 to use single-process
```

(continues on next page)

(continued from previous page)

```
                    data loading. Note: Only use multi-process data
                    loading if you are calculating on a gpu (otherwise
                    inefficient)! (default: 0)
-a {deepencoding}, --architecture {deepencoding}
                    Network architecture to use for classification.
                    (default: deepencoding)
-w FILE, --weights FILE
                    Custom weights file path (optional) (default: None)
-bs INT, --batch-size INT
                    Batch size used for prediction.Defines how many
                    sequences should be forwarded in the network at once.
                    With a batch size of one, the protein sequences are
                    sequentially classified by the network without
                    leveraging parallelism. Higher batch-sizes than the
                    default can speed up the prediction significantly if
                    on a gpu. On a cpu, however, they can be slower than
                    smaller ones due to the increased average sequence
                    length in the convolution step due to zero-padding
                    every sequence in each batch. (default: 1)
```

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

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

3.3.3 deepnog.dataset

3.3.4 deepnog.inference

3.3.5 deepnog.io

3.3.6 deepnog.models

Description:

Network definitions (PyTorch modules)

3.3.7 deepnog.sync

3.3.8 deepnog.tests

Description:

Helpers for deepnog tests.

Including:

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

Individual tests are located within the respective deepnog subpackages.

3.3.9 deepnog.utils

class deepnog.utils.**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

deepnog.utils.**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.**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.**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.**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.**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.**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.**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. ('deepthought', 'DeepNettigkeit') will load deepnog.models.deepthought.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.**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.**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

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 Register new network architectures

Create a Python module under deepnog/models/<my_network.py>. You can use deepencoding.py as a template. When the new module is in place, also edit deepnog/client.py to expose the new network to the user:

```
parser.add_argument("-a", "--architecture",
                    default='deepencoding',
                    choices=['deepencoding',
                             'my_network',
                             ],
                    help="Network architecture to use for classification.")
```

3.4.2 Register new models

New models for additional taxnomic levels 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
|    |   └── deepencoding.pth
|    └── 2
|        └── deepencoding.pth
| ├── ...
|
|
```

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

```
| deepnog_data
| ├── eggNOG5
|    ├── 1
|    |   └── deepencoding.pth
|    └── 2
|        └── deepencoding.pth
| ├── MyOrthologyDB
| |   └── 1
| |       └── deepencoding.pth
| ├── ...
|
```

Assuming we want to compare `deepencoding` to `my_network`, we add the trained network parameters like this:

```
| deepnog_data
| ├── eggNOG5
|    ├── 1
|    |   ├── deepencoding.pth
|    |   └── my_network.pth
|    └── 2
|        ├── deepencoding.pth
|        └── my_network.pth
| ├── MyOrthologyDB
| |   └── 1
| |       ├── deepencoding.pth
| |       └── my_network.pth
| ├── ...
|
```

Finally, expose the new models to the user by modifying `deepnog/client.py` again. The relevant section is argument parsing for `--database`, and `--tax`, if new taxonomic levels are introduced as well.

```
parser.add_argument("-db", "--database",
                    type=str,
                    choices=['eggNOG5',
                             'MyOrthologyDB',
                             ],
                    default='eggNOG5',
                    help="Orthologous group/family database to use.")
parser.add_argument("-t", "--tax",
                    type=int,
                    choices=[1, 2, ],
                    default=2,
                    help="Taxonomic level to use in specified database "
                         "(1 = root, 2 = bacteria)")
```

3.4.3 Training scripts

Please note, that no training scripts are currently shipped with `deepnog`, as scripts used for the available models rely on in-house software libraries and databases, such as SIMAP2. We are currently working on standalone training scripts, that will be made public asap.

CHAPTER

FOUR

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
```

CHAPTER

FIVE

CHANGELOG

5.1 Next release

...

5.2 1.1.0 - 2020-02-28

5.2.1 Added

- EggNOG5 root (tax 1) prediction

5.2.2 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

5.2.3 Fixes

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

5.2.4 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`.

5.3 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)

CHAPTER

SIX

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.

CHAPTER

SEVEN

USER GUIDE

The User Guide introduces the main concepts of `deepnog`. It also contains complete CLI and API documentations of the package.

CHAPTER

EIGHT

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.

CHAPTER

NINE

WHAT'S NEW

To see what's new in the latest version of `deepnog`, have a look at the changelog.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX