DEEPNOG
DEEP LEARNING TOOL FOR FAST ORTHOLOGOUS GROUP PREDICTIONS OF PROTEINS

Lukas Gosch
PR Software development project in bioinformatics 2019/20 WS
gosch.lukas@gmail.com

ABSTRACT

DeepNOG is an extremely fast protein homology prediction tool. To classify protein sequences into orthologous groups (OGs) it uses a newly designed convolutional neural network architecture called DeepEncoding trained on the bacterial level of the eggNOG 5.0 database. Compared to the competitor architecture DeepFam, it is independent of the input sequence length and learns a representation of each amino acid in a continuous vector space. DeepEncoding improves upon the state-of-the-art prediction accuracies on the Clusters of Orthologous Groups of proteins (COG) database achieved by DeepFam and the profile hidden Markov model (pHMM) approach used by HMMER. Furthermore, DeepEncoding achieves a considerably higher prediction accuracy than DeepFam on the bacterial level of eggNOG 5.0. On both COG and eggNOG 5.0, DeepEncoding is computationally more efficient than DeepFam and outperforms HMMER by over an order of magnitude. Therefore, DeepNOG using DeepEncoding promises to be a more efficient and accurate alternative to DeepFam and HMMER for fast protein homology prediction. Detecting orthologous proteins with HMMER is currently a major bottleneck in PhenDB, a bacterial phenotypic trait prediction pipeline based on the bacterial level of eggNOG 5.0. Therefore, it is planned to integrate DeepNOG into PhenDB to significantly speed up trait prediction.

Contents

1 Introduction

Understanding the function of proteins is a fundamental problem in molecular biology. Characterizing their function through biological experiments is mostly too expensive and infeasible. Fitch (2000) and Gabaldón & Koonin (2013) have shown that identifying orthologous relationships of proteins is highly informative about their function. For recent suchlike applications of homology, orthologous sequences are often clustered into orthologous groups (OGs). Exemplarily, OG assignments have been used by Feldbauer et al. (2015) to predict microbial phenotypes based on their genome sequence. This is necessary due to the inability of cultivating a majority of microbes in lab cultures (Amann et al. (1995)) and to directly study metagenomic assembled genomes obtained from environmental samples.

Phylogenetic and functional information is already largely stored in the amino acid sequence, i.e. the primary structure of proteins (Passarge (2013)). Therefore, a popular approach to identify orthologous relationships of proteins is to find matching protein sequences in well-annotated orthology resources. A number of such databases exist today such as the hand-curated Clusters of Orthologous Groups (COG) database (Galperin et al. (2015)), KEGG Orthology (Kanehisa & Goto (2000)), OMA (Altenhoff et al. (2018)), Pfam (El-Gebali et al. (2019)) and eggNOG (Huerta-Cepas et al. (2019)). eggNOG 5.0 comprises automatically created orthologous groups in a non-supervised manner. Therefore, OGs in eggNOG are also referred to as NOGs.

The state-of-the-art to detect homologous sequences of proteins are probabilistic profile hidden Markov models (pHMMs). They require multiple sequence alignments for training (Eddy (1998)). Exactly aligning multiple protein-sequences to calculate similarity scores between proteins has been shown by Wang & Jiang (1994) to be an NP-complete problem. This means the computational complexity of aligning k sequences scales exponentially with k leading to exact algorithms being impractical and heuristic methods used in practice (see for example Sievers et. al (2011)). The most used software tool based on pHMMs is HMMER3 (Eddy (2009)). Therefore, HMMER3 is used as a baseline for experiments. For this report, the terms HMMER and HMMER3 are used interchangeably.

The runtime of traditional methods such as the alignment-based pHMMs becomes a limiting factor in experimental design. This is due to the large number of newly sequenced proteins obtained from high-throughput sequencing technologies. From current metagenomic experiments millions and soon to be billions of proteins await analysis (Pasolli et al. (2019), Nayfach et al. (2019), Almeida et al. (2019)). Therefore, new approaches have to be brought forth. One newly investigated alternative are (deep) neural networks. A deep convolutional neural network architecture for orthologous group predictions called DeepFam has been recently published by Seo et al. (2018). While it achieves a significant prediction speed-up compared to HMMER, it still suffers from several shortcomings such as requiring fixed length input sequences and relying on a one-hot encoding of amino acids disregarding biochemical similarities or dissimilarities between amino acids. Therefore, this report introduces a new deep convolutional neural network architecture called DeepEncoding which successfully overcomes the above-mentioned limitations of DeepFam while showing superior prediction accuracy and computational efficiency.

For the easy and practical use of DeepEncoding and other neural networks for homology prediction tasks, this report introduces a Python command line tool called DeepNOG. In its current installation, it supports DeepEncoding trained for predicting NOGs on the bacterial level of the eggNOG 5.0 database. It is readily extensible to arbitrary neural network models implemented in PyTorch and trained on any orthology resource. As input, it expects protein sequences in one of the common protein sequence file formats, such as the FASTA format (Lipman & Pearson (1985)). Then it rapidly classifies the given proteins performing the calculations either on a CPU or GPU, single or multi-threaded and directly stores the predictions with a prediction confidence in an output file.

Orthologous group predictions with HMMER are a major bottleneck in PhenDB, a bacterial phenotypic trait prediction pipeline facilitating the PICA framework (Feldbauer et al. (2015)). Therefore, it is planned to integrate DeepNOG into PhenDB to significantly speed up the bacterial phenotypic trait prediction process.

The report is structured as follows. In Section 2, COG and eggNOG 5.0 are described which are the orthology resources used for the results in this report. DeepFam's architecture including its limitations are summarized in Section 3.1. The DeepEncoding architecture is introduced in-depth in Section 3.2. Furthermore, in Section 4, the Python software tool DeepNOG is discussed in detail. The experiments and results are discussed in Section 5. They empirically show the superiority of the DeepEncoding architecture to DeepFam and pHMMs. The results also highlight certain shortcomings of the DeepEncoding architecture when faced with a high number of OGs with a very imbalanced populations of protein sequences in the training set. This problem is especially relevant for using OG predictions from DeepNOG as alternative to HMMER for phenotypic trait predictions in the PhenDB pipeline. Therefore, Section 5 additionally discusses possible solutions to overcome the limitations of DeepEncoding. In Section 6 a conclusion of the report is given.

2 Databases

Table 1 gives an overview of the most important characteristics and differences between COG and eggNOG 5.0 (bacterial level & single-label[1]).

Comparison: COG and eggNOG 5.0 (bacterial level & single-label)

	COG	eggNOG 5.0
#Proteins	1 674 176	~13 800 000
#Orthologous Groups	4631	206 782
OGs population	1 to 10 632	1 to 96 355
Sequence lengths	21 to 29 202	23 to 24 921

Table 1: Highlighting the main characteristics of the orthology databases COG and eggNOG 5.0 (bacterial level & single-label) used for this report.

2.1 Clusters of Orthologous Groups of Proteins

The Clusters of Orthologous Groups (COG) database is a manually curated orthology resource (Galperin et al. (2015)) that consists of 1 674 176 proteins and 4631 orthologous groups. The orthologous groups are populated with between one[2] (singletons) and 10 631 protein sequences.

Therefore, COG can be seen as a highly imbalanced dataset with respect to class cardinalities, which is a challenge to traditional and modern machine learning methods (Johnson & Khoshgoftaar (2019)), some consequences of which will be discussed in the results section 5.3.3. Furthermore, sequence lengths also show a wide range from 21 to 29 202. COG is used by Seo et al. (2018) to evaluate the DeepFam architecture and to compare the speed and accuracy of this convolutional neural network architecture to HMMER. Therefore, COG is used on the one hand to verify the results reported for DeepFam by Seo et al. (2018), and on the other hand as a baseline to compare DeepEncoding to DeepFam and HMMER.

2.2 eggNOG 5.0

The evolutionary genealogy of genes: Non-supervised orthologous groups database or short eggNOG (Huerta-Cepas et al. (2019)) includes the COG database. It is not manually curated, but the OGs are computed by the non-supervised eggNOG clustering algorithm (Jensen et al. (2008)) and therefore often referred to as NOGs. In its current version 5.0, eggNOG consists of 4.4 million orthologous groups which are distributed across 379 taxonomic levels. For this work, single-label proteins from the bacterial level were considered resulting into 13.8 million proteins and 206 782 OGs. The orthologous groups are populated with between one (singletons) and 96 355 protein sequences. Therefore, eggNOG 5.0 (bacterial level & single-label) can be seen as an even more imbalanced dataset than COG. The sequence length again varies greatly from 23 to 24 921. The bacterial level is of special interest, as it is used by the PICA framework and in the PhenDB pipeline for bacterial phenotypic trait predictions. Therefore, a key goal of this software project was to investigate if the neural network approach shown to be successful by Seo et al. (2018) on COG, can be successfully applied to eggNOG 5.0 (bacterial level & single-label), the results of which can be found in Section

[1]Single-label proteins are associated to only one NOG, which excludes, for example, multi-domain proteins.

[2]Strictly speaking there are no orthologous groups with a population of one, as such a protein sequence is called a singleton. For simplicity, in this report, singletons are synonymously referred to as OGs with a population of one.

5. For a concise presentation, in the rest of this report, eggNOG 5.0 and eggNOG 5.0 (bacterial level) is used synonymously with eggNOG 5.0 (bacterial level & single-label).

3 METHODS

In the following, DeepFam (see Section 3.1) and DeepEncoding (see Section 3.2) are introduced.

3.1 DEEPFAM

DeepFam is a convolutional neural network (LeCun (1989)) introduce by Seo et al. (2018) designed for orthologous group predictions of proteins based on their amino acid sequence. Its architecture is visualized in figure 1.

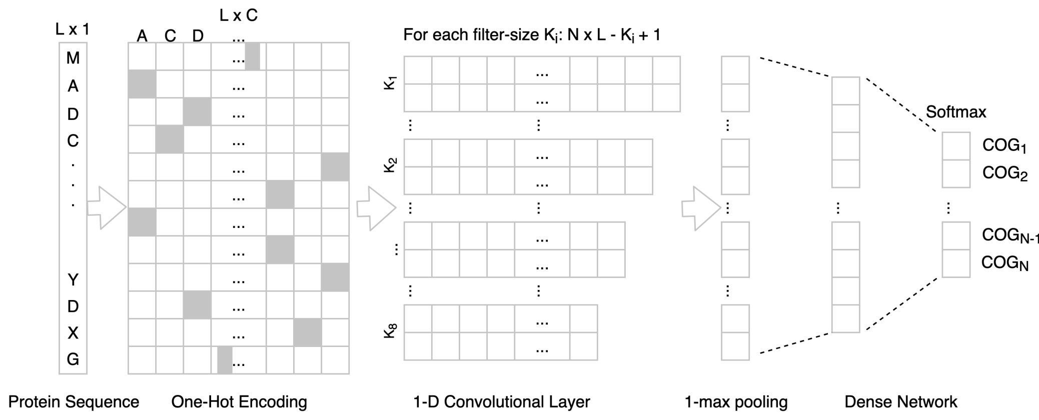

Figure 1: Architecture of DeepFam

3.1.1 ENCODING LAYER

DeepFam processes raw protein sequences as input. Therefore, an embedding is necessary and a one-hot encoding is used. This means if one has an amino acid alphabet of C letters, each amino-acid is a C-dimensional zero-vector with exactly one element set to one in the dimension associated to the amino-acid letter. Therefore, in this embedding, all amino-acids have the same distance from one another independent of their biochemical properties or their importance for the function of the proteins they encode. For example the two neutral aromatic amino acids Phenylalanine (F) and Tryptophan (W) are seen exactly as dissimilar or similar to one another as to the disulphide bonds forming Cysteine (C), which is essential for the three dimensional structure of proteins, or to the positively and negatively charged amino-acids.

3.1.2 CONVOLUTIONAL LAYER

After the encoding, a 1-D convolutional layer is employed. The i-th row in the convolutional layer corresponds to a filter of size K_i which is applied to L-K_i+1 consecutive (encoded) protein subsequences of length K_i resulting in L-K_i+1 convolutional units. 1-D means that the filter treats each column in the encoding as a separate input channel. If the amino-acid alphabet consists of C letters, this results in the filter learning C vectors of weights, each of length K_i. Therefore, the k-th element in a weight vector corresponds to the weight given to a certain amino-acid occurring in the k-th position of the looked upon protein subsequence. The 1-D convolutional operation defining the activation of the convolution unit h_{ij} is formalized through equation (1)

$$h_{ij} = \sigma \left(b_i + \sum_{c=1}^{C} \sum_{l=1}^{K_i} X_{j+l-1,c} W_{i,c,l} \right) \tag{1}$$

where b_i is the bias term and σ the activation function for which DeepFam chooses the ReLU function (Glorot et al. (2011)). Furthermore, it choses 8-different filter sizes 8, 12, 16, 20, 24, 28, 32 and 36. In the most successful parametrization, it uses 250 different filters for each filter-size resulting in 2000 independent filters. Seo et al. (2018) showed that the learned filters are sensitive to sequence motifs typical for certain orthologous groups and their occurrences can distinguish orthologous groups.

One has to note that the convolutional layer is the computational bottleneck in DeepFam and reducing the number of filters significantly increases computation speed (see Section 5).

3.1.3 Pooling Layer

DeepFam employs a standard 1-max pooling layer. This means, the i-th node in the pooling layer corresponds to the maximum value in the i-th row of the convolutional layer and all other values are discarded. The problem in DeepFam is, that a standard 1-max pooling layer exactly requires the length of the input rows to be prespecified. For this reason, Seo et al. (2018) fixed the length L of input sequences in all experiments with DeepFam to 1000 and zero padded smaller input sequences and ignored longer ones. As mentioned in Section 2, COG and eggNOG consist of sequences with very high length variations. Therefore, this is a significant limitation for the applicability of this architecture to arbitrary sequences in COG and eggNOG as well as for classifying arbitrary user sequences using DeepNOG.

3.1.4 Dense Network

After employing a convolutional layer and a 1-max pooling layer, which can be viewed as feature extractors from a given protein sequence, DeepFam uses a standard dense neural network with one hidden layer for classification. It uses a softmax output layer where each node corresponds to an OGs. Therefore, the value of an output node can be interpreted as the confidence the neural network has in associating the input protein sequence to this specific orthologous group (Goodfellow et al. (2016)). The prediction can then be made by outputting the OG for which the neural network has the highest confidence of membership. In the most successful parametrization of DeepFam, it uses 2000 hidden units in its dense network component.

3.1.5 Hyperparameters & Training

As Seo et al. (2018) provided a DeepFam implementation only compatible with Python 2.7, DeepFam was reimplemented in Python 3 with PyTorch and trained as described in Seo et al. (2018). Two DeepFam architecture were trained. One corresponding to the best parametrization as reported in Seo et al. (2018) which is called DeepFam for the rest of this report. As DeepFam was not trainable on eggNOG 5.0 with the computational constraints of the available graphic card for this project[3], DeepFam Light refers to a slightly more lightweight architecture parametrization using 150 instead of 250 different filters for each filter size and 1500 instead of 2000 hidden units. This design decisions have been made in accordance to the error rates reported for different hyperparameter choices by Seo et al. (2018). Furthermore, DeepFam Light was trained with a batch size of 50 instead of 100) to significantly reduce memory consumption.

3.2 DeepEncoding

DeepEncoding is a convolutional neural network architecture based on DeepFam designed to overcome some of its shortcomings, especially the biochemical dissimilarities disregarding one hot encoding (see Section 3.1.1) and the limitation to fixed length protein sequences (see Section 3.1.3). Figure 2 shows the architecture of DeepEncoding. For the results comparing DeepEncoding with DeepFam, see Section 5.

3.2.1 Encoding Layer

DeepEncoding exchanges the one-hot encoding with a learnable embedding in $\mathbb{R}^D$. This means, the embedding layer learns a D-dimensional vector representation of each amino acid in the alphabet.

[3]Quadro P6000, 24GB memory

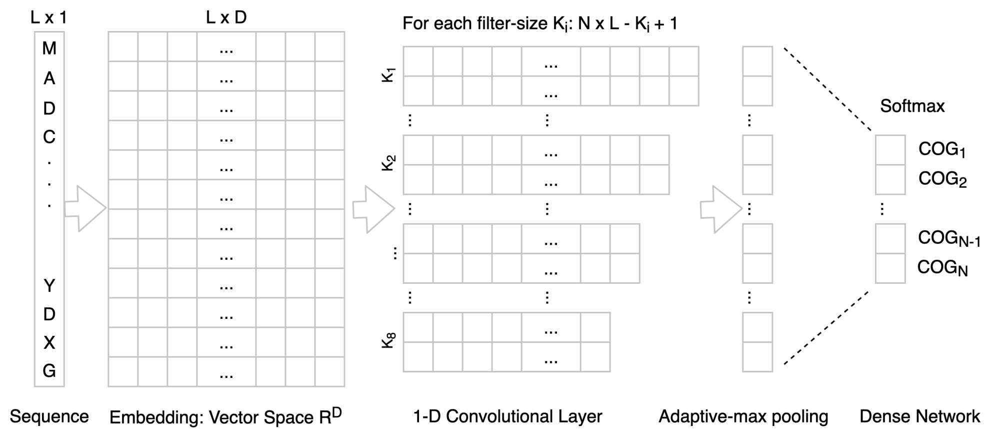

Figure 2: Architecture of DeepEncoding

Therefore, it introduces C x D new learnable parameters which is negligible compared to the other layers.

Now, distances between amino acids are no longer exactly the same, a low dimensional visualization of a learned embedding can be found in section 5.3.1. The embedding dimension can be chosen a lot smaller than the size of the alphabet. For our application, we have chosen an embedding dimension of 10.

3.2.2 Convolutional Layer

DeepEncoding's convolutional layer stays unchanged to DeepFam's one. Employing a 1-D convolution means that each dimension in the embedding vector space $\mathbb{R}^D$ is treated by the filter as an independent input feature vector. The filter then weights the importance of the positive or negative strength of a feature at a certain position in the input protein subsequence for each input feature vector separately. As the positive or negative strength of a feature at a certain position is determined by the concrete amino acid in this specific position in the subsequence, the interpretation of filters getting sensitive to discriminatory motifs for OG predictions, as put forth by Seo et al. (2018), holds.

Furthermore, the continuous vector space embedding has a significantly lower dimension then the one hot-encoding. Therefore, the filters in the convolution layer have significantly fewer parameters to learn.

3.2.3 Pooling Layer

DeepEncoding exchanges the 1-max pooling layer of DeepFam (see Section 3.1.3) with an adaptive max-pooling layer. It has the same function of looking for the maximum value in an input row of the convolutional layer, storing it and discarding the rest, but it does not need to be told the input row length in advance. The length L of an input protein sequence determines exactly two things, the length of the embedding and the length of the rows in the convolutional layer (given the filter sizes are fixed). As the convolutional layer by definition can work with arbitrary L, changing L only effects the length of the rows. Due to the ability of the adaptive-max pooling layer to deal with arbitrary row-input lengths, DeepEncoding has no upper bound on the length of the input sequences it can process. Extremely short input sequences of lengths smaller than the biggest filter size K have to be zero padded to length K to be correctly processed. As K is chosen to be 36, sequences have to be zero padded to a length of 36 and in difference to DeepFam which requires zero-padding to a length of 1000. This makes DeepEncoding practically independent of the input protein sequence length.

3.2.4 Dense Network

As in DeepFam (see Section 3.1.4), a dense neural network is employed for classification based upon the extracted features from the convolutional layers. Instead of using an additional hidden layer, DeepEncoding directly places the softmax output layer after the max pooling layer. As a softmax layer is equivalent to a logistic regression, DeepEncoding can be understood as a two-stage machine learning model. In the first stage, it extracts the (strength of) occurrence of sequence motifs with discriminatory information regarding OGs. In the second stage, it applies a logistic regression model, using the (strength of) occurrence of sequence motifs as input, to predict the most likely orthologous group for the input sequence.

Omitting the hidden layer also saves parameters in the model. [4]

3.2.5 Hyperparameters & Training

DeepEncoding uses scaled exponential linear units (SELUs) introduced by Klambauer et al. (2017) as non-linearities compared to ReLUs paired with batch normalization (Ioffe & Szegedy (2015)) as done by DeepFam. Even though Santurkar et al. (2018) showed that batch normalizations' success is linked to the geometry of the loss surface rather than the internal covariate shift[5], for our case using SELUs achieves very good empirical results while being computationally more efficient than introducing a batch normalization layer. This allows DeepEncoding to be trained without batch normalization (Ioffe & Szegedy (2015)) as the internal covariate shift is prevented by the self-normalizing property induced by the SELUs which increases training speed[6]. The convolutional and dense layers of DeepEncoding are initialized as described by Klambauer et al. (2017) for self-normalizing networks. As a regularization technique, dropout with $p = 0.3$ is employed (Hinton et al. (2012)). Additionally alpha-dropout with $p = \{0.05, 0.10, 0.20\}$ (as suggested by Klambauer et al. (2017)) as well as dropout with $p = 0.5$ (as suggested by Hinton et al. (2012)) was investigated, but cross-validation performance on COG was lower than using dropout with $p = 0.3$. Due to computational limitations and the similarity of COG to eggNOG 5.0, the performance of hyperparameter choices on COG was used as an estimator for the performance on eggNOG 5.0. Therefore, DeepEncoding trained on eggNOG 5.0 used dropout with $p = 0.3$. Furthermore, DeepEncoding uses the same filter sizes 8, 12, 16, 20, 24, 28, 32 and 36 as DeepFam and 150 filters for each size. For stochastic optimization, Adam (Kingma & Ba (2014)) is used with a learning rate set to 0.01 in conjunction with a learning rate scheduler decreasing the learning rate by 0.75 after each epoch[7]. The other parameters in Adam are set as recommended by Kingma & Ba (2014). Furthermore, the batch size is set to 16.

4 Software: DeepNOG

DeepNOG is a Python command line tool. It uses deep networks for fast protein homology predictions. In its current installation, it supports the DeepEncoding architecture trained on the bacterial level of the eggNOG 5.0 database to be directly applicable, instead of HMMER, in the PhenDB pipeline. The reasons for choosing DeepEncoding instead of DeepFam are highlighted in Section 5. In general, DeepNOG is independent of a specific neural network architecture. It is written with extensibility in mind. Therefore, one can easily register a new neural network to be used by DeepNOG trained on arbitrary orthology databases. The only requirement is that a model definition in PyTorch is available as well as trained weights can be provided. For details on the usage of DeepNOG as

[4]As highlighted by Belkin et al. (2019), parameters in a model is not necessarily a good measure of its complexity and especially neural network have been shown to have the ability to generalize better in an over-parametrized regime (i.e. where they have more parameters then training data points). For the purpose of this project, reducing parameters should be seen as increasing performance rather than as a regularization method for limiting the capacity to learn better generalizing models.

[5]Preventing the internal covariate shift in neural networks by an activation function which induces a self-normalization was the original motivation behind SELUs.

[6]Though there is a debate on

[7]Actually, Adam calculates an individual learning rate for each parameter depending on the respective gradient and includes a learning rate decay in its calculation. The learning rate given to Adam is the upper bound for the calculated individual learning rates. Therefore, a learning rate scheduler decreases this upper bound each epoch.

well as on how to extend DeepNOG to support custom neural networks, the reader is referred to the user & developer guide accompanying this report.

DeepNOG is packaged, meaning it can be installed from source with the Python package manager pip. Software tests have been developed covering the main functionality of the code. Furthermore, the code is documented throughout and in the development of the code base state-of-the-art code style guidelines[8] were kept in mind. The project is versioned with git following the branching model outlined in this blog post[9].

4.1 Flowchart

DeepNOG's workflow is visualized in the process diagram in figure 3.

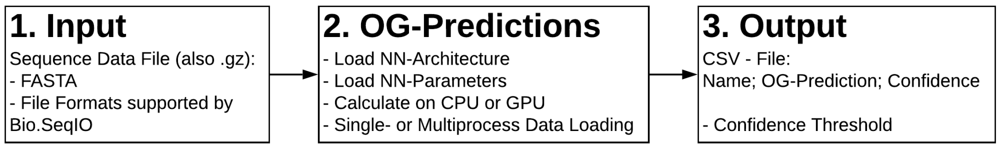
1. Input
Sequence Data File (also .gz):
- FASTA
- File Formats supported by Bio.SeqIO
2. OG-Predictions
- Load NN-Architecture
- Load NN-Parameters
- Calculate on CPU or GPU
- Single- or Multiprocess Data Loading
3. Output
CSV - File:
Name; OG-Prediction; Confidence
- Confidence Threshold

Figure 3: Workflow of DeepNOG to create orthologous group predictions based on input protein sequences. For a detailed explanation of the individual steps, the reader is referred to the description in the text.

4.1.1 Input Data

As an input DeepNOG expects a protein sequence file which can also be provided gziped. It is tested for the FASTA file format, but 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.

4.1.2 Prediction Phase

In the prediction phase, DeepNOG loads a predefined neural network and the corresponding pre-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. The user can either choose the hardware explicitely, or the program determines the hardware and chooses automatically. It offers singleprocess 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 in the user guide.

4.1.3 Output Data

DeepNOG generates CSV output files, which consists of three columns. The first column contains 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. Predictions with confidences below the threshold are discarded, so that the

[8]Especially referring to the PEP8 style guide.

[9]For printed versions: https://nvie.com/posts/a-successful-git-branching-model/

model does not associate the input protein sequence with any orthologous group in the database. Therefore, if the highest prediction confidence for any OG for a given input protein sequence is below this threshold, the prediction is left empty. Based on the prediction confidence analyses outlined in Section 5.3.2, for DeepEncoding on eggNOG 5.0, the prediction confidence threshold is set to 99%.

5 Experiments and Results

DeepFam (see Section 3.1) and DeepEncoding (see Section 3.2) were evaluated on both COG (see Section 2.1) and eggNOG 5.0 (bacterial level) (see Section 2.2). COG was used to successfully reproduce the findings reported by Seo et al. (2018) with our own PyTorch implementation of DeepFam as well as to compare DeepEncoding with DeepFam and HMMER. The findings for COG are summarized in Section 5.2. As it is of special interest to use DeepNOG instead of HMMER in the PhenDB pipeline, DeepEncoding and DeepFam are applied to eggNOG 5.0 and their efficiency compared to HMMER. The findings for eggNOG 5.0 are summarized in Section 5.3.

5.1 Addressing Imbalanced Orthologous Group Populations

As stated in Section 2, COG as well as eggNOG 5.0 constitute very imbalanced datasets as their classes range from singletons to orthologous groups populated by tens of thousands of different protein sequences. Therefore, it is very questionable if DeepFam or DeepEncoding (or any machine learning algorithm) can sensibly learn assignments to orthologous groups which have a population below a certain threshold without introducing measures to combat the imbalance in the data (Johnson & Khoshgoftaar (2019)). For example, in the case of singletons, not even train-test or train-validation splits can be made. Therefore, Seo et al. (2018) introduced three minimum population thresholds for OGs (100, 250 and 500) with the additional goal to investigate, how the methods scale to higher numbers of OGs with more and more imbalance. For this report, the edge cases of minimum population thresholds of 100 and 500 for both COG and eggNOG 5.0 were investigated as they are the most informative about the scaling behaviour of the methods.

An additional consequence of the imbalanced datasets is that measuring a model with its total accuracy can be misleading. For example, if one only has two classes, one with 100 000 examples and the other one with 10, a model achieves nearly perfect accuracy by always predicting the majority class. This means, it could well be that high accuracy scores of models, in an imbalanced data setting, actually result from predicting high population orthologous groups very well but perform very weak on a much higher number of low population classes. Therefore, only looking at total accuracy can mask fundamental flaws in the model. As a consequence, this report addresses the problem of imbalanced training datasets by introducing an evaluation metric not used by Seo et al. (2018) but common for imbalanced, multi-class machine learning problems, called the *mean accuracy*. It is defined by calculating the prediction accuracy for each orthologous group separately and then averaging over all OGs. This results in an additional accuracy score which weights each orthologous group the same independent of its size. The author believes that reporting mean accuracy scores should be a standard when evaluating multi-class machine learning models on large imbalanced datasets such as the common orthology databases COG and eggNOG 5.0.

5.2 COG

To apply the DeepFam architecture to the COG database, besides setting a minimum population threshold, Seo et al. (2018) set the maximum sequence length to 1000 and longer sequences were dropped from the training set. This is due to the fact that DeepFam can only process fixed length sequences (see Section 3.1). Applying the above mentioned restrictions to the COG database results in the datasets outline in table 2.

From table 2 it follows that applying the length restriction removes 1.3% of the data and 4 classes. Additionally, requiring a minimum population of 100, removes nearly 1.8 thousand OGs and roughly 6.5% of the protein sequences. Setting the minimum population threshold to 500, removes nearly 4/5 of the orthologous groups and over 30% of the sequences. Therefore, a population threshold of 500 is quite restrictive compared to 100.

Resulting COG datasets

Name	Min. Population	OGs	% of COG sequences	Length restriction
COG	-	4659	100	-
-	-	4655	98.7	1000
COG-100-2892	100	2892	93.5	1000
COG-500-1074	500	1074	67.5	1000

Table 2: Datasets derived from the COG database based on different restrictions.

The *total accuracy* and *mean accuracy* results of the different inference methods on the COG datasets outlined in table 2 are summarized in table 3. The results are obtained by averaging the three-fold cross-validation results with the cross-validation splits obtained from Seo et al. (2018).

	(Total) Accuracy in %		**Mean Accuracy in %**	
Method	COG-500-1074	COG-100-2892	COG-500-1074	COG-100-2892
pHMMs*	91.75	91.67	-	-
DeepFam	95.40*	91.40*	95.68	85.87
DeepFam Light	94.4	87.27	94.35	77.1
DeepEncoding	**95.66**	**94.24**	**96.07**	**93.03**

Table 3: Total and mean accuracy results of state-of-the-art pHMMs as well as the different deep learning architectures on different COG datasets. Results marked with * taken from Seo et al. (2018). Best scores achieved written in bold.

As a baseline for the total accuracy results outline in table 3, profile hidden Markov models as produced with HMMER have been used. They achieve 91.75 % accuracy on COG-500 and 91.67 % accuracy on COG-100. DeepFam improves upon the HMMER results significantly to 95% for COG-500 but decreases to 91.40% in COG-100. This indicates that DeepFam seems to scale sub-optimal to higher number of OGs. These results are taken from Seo et al. (2018). DeepFams results have been reproduced with an independent PyTorch reimplementation. As stated in detail in Section 3.1.5, DeepFam was not trainable on eggNOG 5.0 with the available resources. Therefore, a more lightweight DeepFam architecture called DeepFam Light has been devised. DeepFam Light achieves very good but not optimal results on COG-500 but decreased by over 7%-pts accuracy on COG-100 exhibiting an even worse scaling behaviour than DeepFam. DeepEncoding marginally beats DeepFam on the COG-500 dataset but significantly outperforms all other methods, especially HMMER, on COG-100 achieving over 94% accuracy. Therefore, the accuracy score only decreased by roughly 1%-pts when nearly tripling the number of classes. This implies a superior scaling behaviour to higher numbers of OGs than DeepFam. As eggNOG 5.0 comprises even more (highly imbalanced) OGs, this is a very important property when applying these methods in the DeepNOG tool and in the PhenDB pipeline.

Columns three and four in table 3 summarize the *mean accuracy* results achieved by the different deep learning architectures on the COG datasets to investigate the behaviour of the different models in the face of imbalanced training data. From it, one can see that on COG-500, all three deep learning architectures achieve a comparable results to their total accuracy scores. Therefore, no problem with the imbalanced OGs can be detected and all methods seem to behaviour equally well for highly and lowly populated classes. This picture changes drastically by looking at COG-100. There, DeepFam achieves a mean accuracy of slightly below 86%.

This is 6%-pts below its total accuracy. This discrepancy even worsens for DeepFam Light where its mean accuracy is more then 10%-pts below its total accuracy. Therefore, as argued in Section 5.1, this implies that both DeepFam architectures perform well on highly populated OGs but worse on lowly populated ones and that this effect has been masked by a high total accuracy. On the other hand, DeepEncoding still achieves 93% mean accuracy on COG-100 which is only 1%-pts lower than its total accuracy. This means that even though the number of lowly populated classes has increased significantly, DeepEncoding performs very well in predicting lowly populated orthologous groups. Therefore, one can conclude that on COG DeepEncoding does not only scale better than DeepFam in a high number of classes setting, but also scales significantly better to a more and more imbalanced training dataset.

From prediction accuracy alone, it is not possible to judge if a certain machine learning method is superior for orthologous group predictions, as the most important constraint for current applications is prediction speed. To investigate the speed of the methods, again HMMER is used as a baseline. In 5 trials, 1000 proteins were classified with HMMER and DeepNOG using either DeepFam or DeepEncoding on one Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz [10].

The process time has been recorded and averaged over all trials and results are summarized in table 4.

Average Prediction Speed for classifying 1000 Proteins in Seconds ($n = 5$, 1 CPU)

Method	COG-500-1074	COG-100-2892
pHMMs	96.3	207.0
DeepFam	34.1	35.6
DeepFam Light	23.1	24.7
DeepEncoding	**21.8**	**22.9**

Table 4: Average prediction speed of the different machine learning models trained on different COG databases to classify 1000 proteins. Results obtained using one CPU and five trials. Fastest results written in bold.

HMMER needed nearly 100 seconds in the case of COG-500 and over double the amount of time for COG-100 for which the number of classes tripled. This implies that HMMER does not scale very efficiently to higher numbers of classes as expected. This is very different for neural networks. DeepFam needs only 34 seconds to annotate the proteins if trained on COG-500 and is only one second slower for COG-100. DeepFam Light even manages to annotate the proteins in 23 seconds if trained on COG-500 and 24 seconds if trained on COG-100. The fastest method of all is DeepEncoding. It takes only 21.8s or 22.9s respectively. Therefore, deep neural networks in general scale, with respect to their prediction speed, significantly better to higher numbers of included OGs in the model than the pHMM approach. DeepEncoding trained on COG-100 achieves a performance increase of over one order of magnitude compared to HMMER. DeepEncoding beating DeepFam with respect to prediction speed can be explained by the fact, that it only needs 1/3 of the total number of parameters compared to DeepFam. Therefore, one can see that DeepEncoding on the COG dataset achieves not only the best accuracies of all methods but is also the fastest method of all.

5.3 EGGNOG

For the results reported on eggNOG 5.0 (bacterial level) and with its application in the DeepNOG tool in mind, DeepFam's pooling layer was changed to an adaptive max pooling layer as described in Section 3.2 such that it is applicable to arbitrary protein sequences lengths. Therefore, no length restriction has been applied on eggNOG 5.0. Applying the minimum orthologous group population thresholds of Section 5.1, results in the datasets outlined in table 5.

Resulting eggNOG 5.0 datasets (bacterial level & single-label)

Name	Min. Population	OGs	% of eggNOG 5.0 sequences	Length restriction
eggNOG 5.0	-	206 782	100	-
eggNOG-100-6217	100	6217	90.2	-
eggNOG-500-3373	500	3373	85.87	-

Table 5: Datasets derived from the eggNOG 5.0 database based on different orthologous group population thresholds.

From table 5 it follows that if requiring a minimum population of 100 protein sequences per orthologous group, 6217 classes survive the cut which corresponds to more than double the number of classes as found in the COG-500 dataset (see table 2) and even more classes than found in total in COG. For a population threshold of 500, over 3373 orthologous groups are left, which is still over 500 classes more than found in COG-500. These statistics again highlight that eggNOG 5.0 constitutes an even more imbalanced training dataset than COG.

[10]Experiments were run on accelerator.

The *total* and *mean accuracy* results of applying DeepFam Light (see Section 3.1.5) and DeepEncoding to the databases from table 5 can be found in table 6. The reported results correspond to the accuracies on the validation set created from a 81% training, 9% validation and 10% test set split.

	(Total) Accuracy in %		**Mean Accuracy in %**	
Method	eggNOG-500-3373	eggNOG-100-6217	eggNOG-500-3373	eggNOG-100-6217
DeepFam Light	83.23	78.95	68.59	37.81
DeepEncoding	**93.18**	**93.17**	**93.14**	**92.73**

Table 6: Total and mean accuracy results of the two competing deep learning architectures DeepFam Light and DeepEncoding on different eggNOG datasets. Best results written in bold.

Table 6 shows that DeepFam Light achieves only 83.23% accuracy on eggNOG-500 compared to 93.18% achieved by DeepEncoding. On eggNOG-500, DeepFam Light achieves only 68.59% of mean accuracy which constitutes ~15%-pts decrease to its total accuracy. This means, DeepFam Light performs good classifying highly populated OGs but performs much worse for smaller ones. On the other hand, DeepEncoding stays very stable and achieves 93.14% mean accuracy on eggNOG-500 nearly matching its total accuracy score. This picture is amplified looking at eggNOG-100. DeepFam Light achieves 78.95% total accuracy and only 37.81% mean accuracy on eggNOG-100. This is compared to 93.17% total accuracy and 92.73% mean accuracy achieved by DeepEncoding on eggNOG-100. Therefore, DeepFam Light's problem of bad performance on lowly populated OG's has significantly worsened when lowering the population threshold. On the other hand, DeepEncoding does achieve roughly the same total accuracy score on eggNOG-100 and eggNOG-500, already indicating very good scaleability to higher class numbers. DeepEncoding's mean accuracy score has also nearly stayed the same dropping by only 0.41%-pts. Therefore, DeepEncoding performs very well even in the face of the highly imbalanced eggNOG 5.0 database clearly outperforming DeepFam Light.

To compare the efficiency of DeepFam Light and DeepEncoding, they again have been matched against HMMER in 5 trials, where 1000 proteins were classified using one Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz on accelerator and the process time averaged (as already done on COG in table 4). The speed results can be seen in table 7.

Average Prediction Speed for classifying 1000 Proteins in Seconds ($n = 5$, 1 CPU)

Method	eggNOG-500-3373	eggNOG-100-6217
pHMM	218.9	253.7
DeepFam Light	24.9	26.3
DeepEncoding	**23.5**	**24.7**

Table 7: Average prediction speed of HMMER, DeepFam Light and DeepEncoding on different eggNOG databases to classify 1000 proteins. Results obtained using one CPU and five trials. Fastest results written in bold.

Table 7 shows that HMMER needs 218.9s on average to classify 1000 proteins if 3373 pHMMs are used and 253.7s if 6217 pHMMs are used, this corresponds to a performance decrease of 15.9%-pts. These results highlight that the pHMMs approach suffers from poor performance scalability if on increases the number of orthologous groups. On the other hand, both neural networks perform faster and scale better. DeepEncoding is again faster than DeepFam for the same reasons as already outline for COG in Section 5.2. DeepEncoding does only need 23.5s on average if trained on eggNOG-500 and only 24.7s on average if trained on eggNOG-100. Therefore, DeepEncoding's performance decreased by only ~5.1%-pts. when the number of OGs nearly doubled. This means that DeepEncoding scales more than three times better than HMMER and on eggNOG-100, achieves a speed-up of over one order of magnitude. If the number of OGs would increase further, one could expect DeepEncoding to achieve bigger and bigger speed-ups compared to HMMER. Additionally, DeepEncoding is more memory efficient than HMMER. DeepEncoding's weights needed to be loaded into memory have 51.9 MB for eggNOG-500 and 92.9 MB for eggNOG-100. However, the pHMMs for eggNOG-500 and eggNOG-100 need 1.2 GB and 1.7 GB, respectively.

To summarize, DeepEncoding clearly outperforms DeepFam Light on every metric on eggNOG 5.0 as it already did on COG. Furthermore, due to the large speed-up gained compared to HMMER, it could be a viable alternative for usage in the PhenDB pipeline. Therefore, DeepEncoding is used instead of DeepFam in DeepNOG. The next sections discuss DeepEncoding and its application in DeepNOG in more detail.

5.3.1 Amino Acid Embedding

The most important design difference between DeepEncoding and DeepFam is in the encoding layer where DeepEncoding learns an amino acid embedding in $\mathbb{R}^{10}$ instead of using a one-hot encoding. Figure 4 uses t-Distributed Stochastic Neighbor Embedding (t-SNE) by van der Maaten & Hinton (2008) to visualize the ten dimensional embedding learned by DeepEncoding from eggNOG-100 using two dimensions. Therefore, if the learned embedding indeed clusters amino acids with similar biochemical properties together, some of this structure should be visible with high probability in $\mathbb{R}^2$.

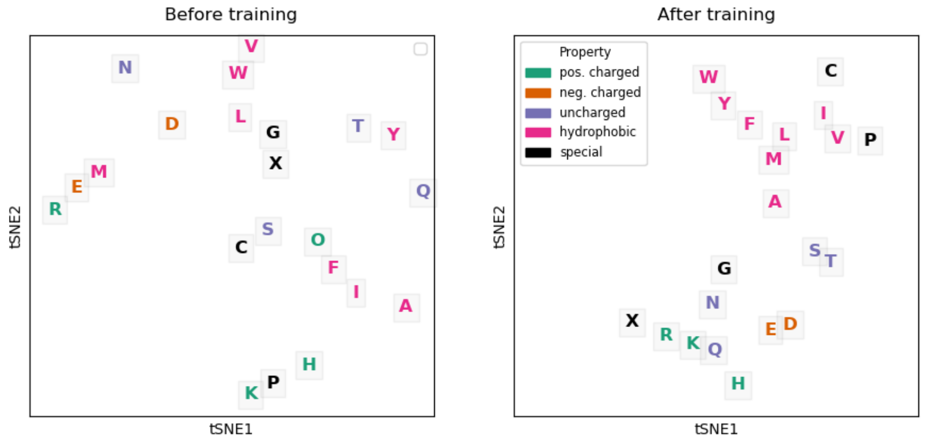

Figure 4: Amino acid embeddings ($\mathbb{R}^{10}$reduced to $\mathbb{R}^2$ using t-SNE). Visualized from DeepEncoding, before training (random initialization) and after being trained on eggNOG-100. Amino acids are coloured based on biochemical properties.

The codes for Selenocystein (U), Asparagine/Aspartic acid (B), Glutamine/Glutamic acid (Z) and Leucine/Isoleucine (J) are excluded from figure 4, as they appear in almost no protein in eggNOG 5.0. The left picture in figure 4 shows the distribution of the amino acids before the network has been trained. Therefore, it is based on a random initialization in $\mathbb{R}^{10}$. The amino acids are coloured based on a rough categorization of their biochemical properties and indeed no obvious structures can be found. The right picture visualizes the amino acid embedding after the training. The amino acids clearly show a grouping based on their biochemical categorization. Furthermore, a finer topology based on biochemical properties can be identified. For example the aromatic amino acids Phenylalanine (F), Tryptophan (W) and Tyrosine (Y) are distinctly grouped together. To note is that Pyrrolysine (O) it not displayed in the right picture, as it got mapped to a very distant point in two-dimensional space compared to all the other amino acids. This could be explained as an artefact of the dimensionality reduction. Indeed, inter-cluster distances typically hold no meaning in t-SNE visualizations, and Pyrrolysine is rare.

As the other layers in DeepEncoding are similar to DeepFam and DeepFam uses an additional hidden layer which can be believed to increase generalization rather than to hinder it, the superior performance of DeepEncoding compared to DeepFam can be attributed to the usage of a learnable embedding in $\mathbb{R}^{10}$ as an encoding layer.

5.3.2 Prediction Confidence Threshold

As mentioned in Section 4, DeepNOG supports setting a prediction confidence threshold to decide, based on the neural network's confidence in the orthologous group prediction, if an input protein can be associated to any OG included in the model. In this Section, it is empirically verified that DeepEncoding's prediction confidence can be used for such a decision and a concrete confidence threshold for DeepEncoding trained on eggNOG-100 is determined. For the empirical verification, 100 000 proteins are randomly sampled from the eggNOG-100 test set. Therefore, they are members of orthologous groups included in the model but have not been seen during training. Furthermore, 100 000 proteins are randomly sampled from orthologous groups in eggNOG 5.0 which are not included in eggNOG-100. Therefore, they correspond to members of OGs not included in the model, i.e. never seen by DeepEncoding. The above sampled proteins are then classified with DeepEncoding using DeepNOG and figure 5 shows a histogram of the obtained prediction confidences with a resolution of 1%.

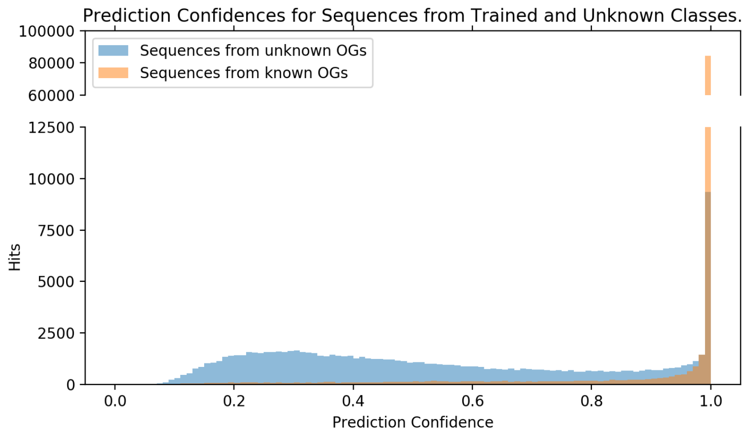

Figure 5: Histogram of DeepEncoding's prediction confidences. Created by classifying 100 000 proteins from the eggNOG-100 test set and by classifying 100 000 proteins from orthologous groups not included in eggNOG-100. The histogram has been cut between 12 500 and 60 000 hits for the purpose of readability. In the 98% to 99% bin, there are 1438 sequences from unknown OGs and 1434 sequences from known OGs counted. Therefore the orange sequences only dominate the 99% to 100% confidence bin with 84 302 orange sequences compared to 9356 blue sequences.

Figure 5 clearly shows that both groups of protein sequences have a very different prediction confidence distribution. The neural network is very confident for proteins which are members of known OGs (orange), i.e. OGs included in the model but which have not been seen during training, but is much less confident for proteins from unknown OGs (blue), i.e. which are members of OGs not included in the model. Therefore, a sensible confidence threshold can be set to a value, where for each prediction confidence level at and above this threshold more protein sequences from known OGs compared to from unknown OGs are counted. In the histogram in figure 5, below 99% prediction confidence, blue dominates every bin. Orange clearly dominates the 99% to 100% bin with 84 302 hits compared to 9356 blue sequences. In the 98% to 99% bin, there are 1438 sequences from unknown OGs and 1434 sequences from known OGs counted. Therefore, the known orange sequences only dominate the last bin and 99% is set as the confidence threshold for DeepEncoding in DeepNOG, below which a protein sequence is treated as being not part of any OG included in the model.

5.3.3 Investigating the Population Threshold

The total and mean accuracy results of DeepEncoding listed in table 6 indicate that the minimum population threshold to include an orthologous group in the model could be lowered below 100. The main application for DeepNOG using DeepEncoding is as an alternative to HMMER in the PhenDB pipeline. There it should provide the input for the PICA framework (Feldbauer et al. (2015)) which bases bacterial phenotypic trait predictions upon a binary list which indicates, if certain NOGs are present in the bacterial genome or not. For each phenotypic trait, PICA uses a separate support vector machine (SVM) model with a linear kernel (Cortes & Vapnik (1995)). Therefore, the importance of an orthologous group in a phenotypic trait model can be assessed by looking at its associated weight value. The bigger the absolute weight value, the more important an OG is in deciding for or against a certain phenotypic trait. Therefore, averaging the weight values associated to a specific OG over all phenotypic trait models including it, gives an estimate on the importance of this OG for PICA.

In the following, this analysis is performed for all orthologous groups up to a population of 100 on 70 phenotypic trait models using in total 124 704 unique NOGs and on average 5833 NOGs. For the results visualized in figure 6, the weights have additionally been normalized to [0, 1] through dividing by the biggest weight in each model.

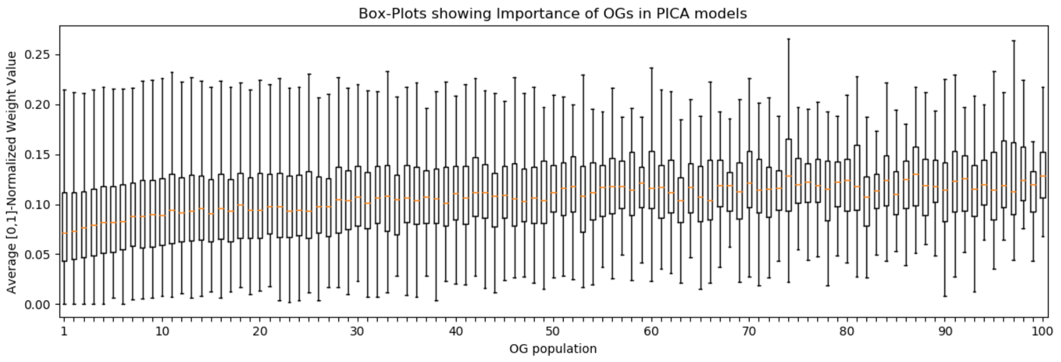

Figure 6: Box-Plots visualizing the normalized and averaged weights from 70 PICA models corresponding to OGs with a population of up to 100. The x-axis is cut at 100 as OGs with a population above 100 are already included in DeepEncoding and the goal of this analysis is to investigate the importance of OGs excluded from the model. Furthermore, restricting the x-axis increases the readability of the visualization.

In figure 6 a trend can be seen that in general, higher populated OGs seem to be slightly more relevant in the PICA models than lowly populated OGs. But no clear cut-off value can be seen, below which orthologous groups seem mostly irrelevant. Furthermore, figure 7 highlights that the majority of OGs used in the PICA models have very low population. Due to the mass of lowly populated OGs in the PICA models compared to the cumulative number of OGs included in DeepEncoding, lowly populated OGs could outweigh the more populous OGs in a phenotypic trait decision, even though they individually have slightly less importance.

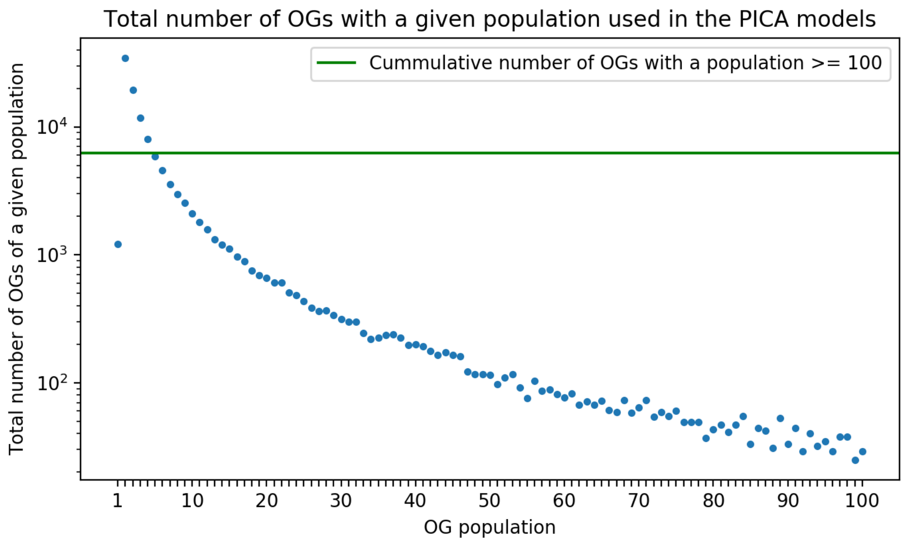

Figure 7: Number of orthologous groups with a population of up to 100, used in the PICA models. The green line shows the cumulative number of OGs with a population bigger or equal to 100, corresponding to the number of orthologous groups included in DeepEncoding. The x-axis is cut at 100 for the same reasons as outlined in figure 6.

From figures 6 and 7 one can conclude that no minimum population threshold can be defined without excluding a large number of orthologous groups with high (cumulative) importance in the PICA models. Future work will have to show, if PICA based on DeepEncoding, which only includes OGs with a population of at least 100, can still make sensible phenotypic trait predictions. Otherwise, these results point in the direction that all orthologous groups from eggNOG 5.0, which are also used in PICA, should be included in the used deep learning model independent of their population size. As this corresponds to a very large number of classes paired with a very imbalanced dataset problem, DeepEncoding's architecture may come to its limits. Therefore, new neural network architectures might have to be developed to approach this problem. One inspiration could come from the field of natural language processing (NLP). There, language models have to be trained with very large vocabularies and a very imbalanced occurrence of the individual words. One interesting approach from NLP is by Grave et al. (2017) and introduces an adaptive softmax layer. It bypasses the linear computational time dependence on the number of classes by explicitly exploiting the imbalanced nature of the underlying dataset. It could be directly applicable DeepEncoding replacing its dense neural network part with the adaptive softmax incorporating all NOGs from eggNOG 5.0.

6 Conclusion

This report introduced DeepEncoding, a new deep neural network architecture for extremely fast classification of protein sequences into orthologous groups. Additionally, to correctly evaluate the model on imbalanced datasets such as COG and eggNOG 5.0, mean accuracy got introduced as the appropriate evaluation metric. Compared to DeepFam, DeepEncoding achieved superior classification accuracies on COG and eggNOG 5.0 (bacterial level). Using mean accuracy, it was uncovered that DeepFam's good total accuracy results masked a problem that it was prediction high populous orthologous groups very well but performed weak on low populous orthologous groups. On the tested datasets, DeepEncoding did not exhibit this problem.

To facilitate DeepEncoding as an easy to use alternative to HMMER, the software tool DeepNOG was created. It allows to register and use any neural network architecture implemented in PyTorch for homology prediction of custom proteins. With DeepNOG leveraging DeepEncoding, a speed-up of over one order of magnitude compared to HMMER could be achieved on eggNOG 5.0. In addition to the speed-ups reported by Seo et al. (2018) on other orthology resources, it can be established that deep learning approaches to protein homology prediction are significantly faster than state-of-the-art pHMMs. Additionally, results on COG indicate that deep learning approaches can have superior prediction accuracies. Future work in this direction will have to perform comprehensive empirical studies comparing total and mean prediction accuracies of deep learning approaches with pHMMs in practical settings. This would establish whether superior prediction accuracies can be achieved consistently by neural networks.

This report focused on the application of protein homology predictions for bacterial phenotypic trait predictions in the PhenDB pipeline. One major shortcoming of current neural network architectures such as DeepFam and DeepEncoding is that only a fraction of all orthologous groups used by PICA or found in eggNOG 5.0 can be included in the models. This is due to the highly imbalanced nature of the underlying dataset. Future work will have to focus on novel neural network architectures to scale deep learning approaches to hundreds of thousands of orthologous groups with highly imbalanced protein sequence populations.

References

Alexandre Almeida et al. A new genomic blueprint of the human gut microbiota. *Nature*, 568: 499–504, 2019.

Adrian M. Altenhoff et al. The oma orthology database in 2018: retrieving evolutionary relationships among all domains of life through richer web and programmatic interfaces. *Nucleic Acids Res.*, 46(D1):D477–D485, 2018.

R.I. Amann et al. Phylogenetic identification and in situ detection of individual microbial cells without cultivation. *Microbiol. Rev.*, 59(1):143–69, 1995.

Mikhail Belkin et al. Reconciling modern machine learning practice and the bias-variance trade-off. *arXiv*, pp. 1812.11118, 2019.

Corina Cortes and Vladimir N. Vapnik. Support-vector networks. *Machine Learning*, 20:273 – 297, 1995.

Sean R. Eddy. Profile hidden markov models. *Bioinformatics Review*, 14(9):755–763, 1998.

Sean R. Eddy. A new generation of homology search tools based on probabilistic inference. *Genome Inform.*, 23:205–2011, 2009.

Sara El-Gebali et al. The pfam protein families database in 2019. *Nucleic Acids Res.*, 47(Database issue):D427–D432, 2019.

Roman Feldbauer et al. Prediction of microbial phenotypes based on comparative genomics. *BMC Bioinformatics*, 16(14):S1, 2015.

W.M. Fitch. Homology a personal view on some of the problems. *Trends Genet.*, 16:227–231, 2000.

T. Gabaldón and E.V. Koonin. Functional and evolutionary implications of gene orthology. *Nat. Rev. Genet.*, 14:360–366, 2013.

M.Y. Galperin et al. Expanded microbial genome coverage and improved protein family annotation in the cog database. *Nucleic Acids Res.*, 43(Database issue):D261–9, 2015.

Xavier Glorot et al. Deep sparse rectifier neural networks. *PMLR*, 15:315–323, 2011.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Édouard Grave et al. Efficient softmax approximation for gpus. *PMLR*, 70, 2017.

Geoffrey E. Hinton et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*, pp. 1207.0580, 2012.

Jaime Huerta-Cepas et al. eggnog 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic Acids Res.*, 47(Database issue):D309–D314, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. *ICML*, pp. 448–456, 2015.

L.J. Jensen et al. eggnog: automated construction and annotation of orthologous groups of genes. *Nucleic Acids Res.*, 36:D250–D254, 2008.

Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *J. Big Data*, 6:27, 2019.

M. Kanehisa and S. Goto. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, 28:27–30, 2000.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Günter Klambauer et al. Self-normalizing neural networks. *NIPS*, pp. 972–981, 2017.

Y. LeCun. Generalization and network design strategies. *Technical Report CRG-TR-89-4, University of Toronto*, 1989.

D. J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693): 1435–41, 1985.

Stephen Nayfach et al. New insights from uncultivated genomes of the global human gut microbiome. *Nature*, 568:505–510, 2019.

Edoardo Pasolli et al. Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle. *Cell*, 176:649–662, 2019.

Eberhard Passarge. *Color Atlas of Genetics*. 4th Edition, Thieme Medical Publishers, 2013.

Shibani Santurkar et al. How does batch normalization help optimization? *NIPS*, 31, 2018.

Seokjun Seo et al. Deepfam: deep learning based alignment-free method for protein family modeling and prediction. *Bioinformatics*, 34:i254–i262, 2018.

F. Sievers et. al. Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Mol. Syst. Biol.*, 7:539, 2011.

L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1: 337–348, 1994.