universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Virus Orthologous Groups Assignment with DeepNOG and VOGDB“

verfasst von / submitted by

Alexander Pfundner BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Computational Science UG2002
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Thomas Rattei

Declaration of Authorship

I, Alexander Pfundner, declare that this thesis and the work presented is my own personal work. I confirm that it is always clearly stated when I have consulted the work of others. There are no co-authors in this work, any use of *we* refers to *the reader and I*, and is solely a design on phrasing.

Acknowledgments

First and foremost, I want to thank my supervisors Thomas Rattei and Roman Feldbauer for giving me the opportunity to work on this very interesting topic, for their ongoing support and constructive criticism, working on this project was a great pleasure for me. Many thanks go to all my colleagues at CUBE, Lovro, Roko, Ade, Michael, Lukas and Li for providing such a kindly working atmosphere and also valuable advice.
My wholehearted gratitude goes to my family, my inspiring wife Alex, my wonderful son Moritz, my caring parents Rudi and Gerlinde, as well as the best dogs in the world Lore, Berta and Charlie. This accomplishment would not have been possible without you, thank you for sticking with me.

Abstract

Assigning functions to proteins is a central task in molecular biology. The amount of sequence data that is produced nowadays can no longer be processed efficiently through biological experiments, therefore computational methods are needed. The function of a protein can be explained by its structure, which is largely encoded by the amino acid sequence. Orthologs are genes that descended from a last common ancestor and therefore have a high sequence similarity. Orthology resources are databases that cluster orthologous sequences in groups, thus making it possible to transfer annotations from one group to an unknown sequence that maps to this group.

This thesis investigates how recent developments in orthologous groups assignment (DeepNOG) and the availability of a comprehensive virus orthology resource (VOGDB) can be utilized to create a fast and accurate method for virus orthologous group assignment. DeepNOG is a deep learning neural network approach that assigns sequences to orthologous groups. VOGDB is a viral orthology database that includes all available viral proteins from RefSeq and uses Hidden Markov models to model orthologous groups. Data preparation for training a DeepNOG model from VOGDB data has to consider the biased nature of the dataset in terms of the class cardinalities. The performance of the DeepNOG model is compared to the Hidden Markov models of VOGDB in terms of assignment confidence, runtime and memory consumption. Model applicability to real-world scenarios is evaluated on external datasets.

The trained DeepNOG models achieve similar accuracy as the Hidden Markov models and reduce the assignment runtime by at least a 100-fold and the memory consumption by a 30-fold. Hyperparameter tuning revealed only marginal improvements when deviating from the default configuration. Both methods agree and produce high confidence assignments on other virus orthology resources. Performance for data from uncultivated viral genomes was equally low for both methods with significant disagreement.

Kurzfassung

Proteinen Funktionen zuzuordnen ist eine zentrale Aufgabe der Molekularbiologie. Die Menge an Sequenzdaten, die heute anfällt, kann durch biologische Experimente nicht mehr effizient verarbeitet werden, daher werden computergestützte Methoden benötigt. Die Funktion eines Proteins lässt sich durch seine Struktur erklären, die weitgehend durch die Aminosäuresequenz kodiert wird. Orthologe sind Gene, die von einem letzten gemeinsamen Vorfahren abstammen und daher eine hohe Sequenzähnlichkeit aufweisen. Orthologieressourcen sind Datenbanken, die orthologe Sequenzen gruppieren, wodurch es möglich ist, Annotationen von einer Gruppe auf eine unbekannte Sequenz zu übertragen, die sich zu dieser Gruppe zuordnet.

Diese Arbeit untersucht, wie die jüngsten Entwicklungen in der orthologen Gruppenzuordnung (DeepNOG) und die Verfügbarkeit einer umfassenden Orthologieressource für Viren (VOGDB) genutzt werden können, um eine schnelle und akkurate Methode für die Zuordnung von Virusproteinen zu Orthologiegruppen zu erstellen. DeepNOG ist ein Deep Learning Ansatz, der Sequenzen orthologen Gruppen zuweist. VOGDB ist eine virale Orthologiedatenbank, die alle verfügbaren viralen Proteine von RefSeq enthält und Hidden Markov Modelle verwendet, um orthologe Gruppen zu modellieren. Die Datenvorbereitung zum Trainieren eines DeepNOG-Modells aus VOGDB-Daten muss die Eigenschaften des Datensatzes in Bezug auf die Klassenkardinalitäten berücksichtigen. Die Performanz des DeepNOG-Modells wird in Bezug auf Zuversichtlichkeit der Zuordnungen, Laufzeit und Speicherverbrauch mit den Hidden Markov Modellen von VOGDB verglichen. Die Anwendbarkeit des Modells auf reale Szenarien wird anhand externer Datensätze bewertet.

Die trainierten DeepNOG-Modelle erreichen eine ähnliche Genauigkeit wie die Hidden Markov Modelle und reduzieren die Zuweisungslaufzeit um mindestens das 100-fache und den Speicherverbrauch um das 30-fache. Das Hyperparametertuning zeigte beim Abweichen von der Standardkonfiguration nur marginale Verbesserungen. Beide Verfahren stimmen überein und erzeugen Zuweisungen mit hoher Konfidenz für andere virale Orthologieressourcen. Die Leistung für Daten aus nicht kultivierten viralen Genomen war für beide Methoden gleich niedrig und Zuweisungen stimmten meistens nicht überein.

Contents

1 Introduction

Linking proteins with function is a key issue in molecular biology. Today's large-scale metagenomic projects produce immense amounts of sequencing data that can no longer be processed efficiently with biological experiments [36]. Therefore, computational methods that are able to extract functional information from protein sequences are needed.
There is broad agreement that the function of a protein can be explained by its structure. At the same time, the amino acid sequence contains enough information to encode for the structure [39]. Thus, logically the function of the protein is mainly determined by its sequence.
Protein families are groups of proteins that descended from a common ancestor, having similar functions and high sequence similarity. Assigning protein sequences of interest to families enables us to gain annotation in a transitive manner. From this, two problems emerge, firstly we have to group together protein sequences according to their orthologous relationship, secondly new sequences need to be assigned correctly to the best fitting group.
Orthologs are genes derived from a single ancestral gene in the last common ancestor of the compared species [28]. It is assumed, that sequences of orthologous genes have higher sequence similarity to each other than to any other genes from the compared genomes, therefore symmetrical best hits can be used to identify orthologs.
The "Clusters of Orthologous Genes" (COG) [18] was the first genome-wide evolutionary classification system for protein families. The term *orthologous group* (OG) extended the notion of a genome-specific best hit to multiple genomes, thereby constructing clusters of consistent best hits. Nowadays there are further orthology resources around, some of them covering thousands of organisms from all domains of life and viruses, the most popular ones include eggNOG [24], OMA [4] and OrthoDB [56].
In this thesis, we solely focus on the problem of assigning new sequences to precomputed OGs. The two main approaches in assigning sequences of interest to OGs are alignment-free and alignment-based methods, whereas the latter, although computationally expensive [55], is the defacto standard, relying on Hidden Markov Models (HMM) built from multiple sequence alignments (MSA) [11]. Alignment-free methods utilize concepts like k-mers to

create feature vectors, which results in loosing the order of the protein sequence and thus structural information, which leads to lower sensitivity.
Recent benchmark studies for protein orthology assignments reported AUC values in the area of 0.70-0.80 for current state-of-the-art alignment-free and alignment-based methods [57], [7]. The performance of alignment-free methods tends to vary stronger for different datasets and scenarios. In contrast to this, alignment-based methods generalize better, but with the cost of higher time complexity.
In large studies alignment-based methods manifest a common computational bottleneck [15], issuing a demand for fast and accurate alignment-free methods. Recently, approaches based on deep learning have shown to deliver a speed-up of multiple folds while still obtaining the same high accuracy as alignment-based methods [14], [41]. These methods reveal a new way of alignment-free methods, that are able to directly process raw sequences without loosing information about sequence order.
In 2018 DeepFam [41] was published as the first deep learning model for OG assignment, in 2020 DeepNOG [14] followed. Both models use a feedforward architecture with convolutional units to extract knowledge from the input data. DeepFam uses a pseudo one-hot-encoding to encode the amino acid sequence, DeepNOG utilizes a word embedding. There were two models trained with DeepFam, one was trained on the COG database [18] (which is manually curated and thus relatively small) for assigning COGs. The second model was trained with the even smaller G protein-coupled receptor (GPCR) [6] dataset on the sub-subfamily level, with a simple bottom-up approach for assignments of the sub-family and family level.
DeepNOG was trained on the eggNOG [24] database, root and bacteria level. While including COG as a supervised core, the main parts of eggNOG are built using an unsupervised clustering approach. Performance of DeepNOG is superior to DeepFam for both, COG and eggNOG databases [14] (the DeepNOG team rebuilt the DeepFam model and trained it also on eggNOG).
When we want to assign viral proteins to viral orthologous groups (VOGs) we need a resource that includes viral proteomes. Unfortunately, the COG database does not include any viral proteomes and eggNOG solely includes viral proteomes from the UniProt Knowledgebase [49], thereby not covering all available viral proteomes. Thus, in terms of VOG assignment the usage of DeepNOG is strongly limited and it is therefore interesting to find out if we can train a DeepNOG model from another orthology database that specializes in viral proteins. VOGDB is an extensive resource for viral orthology [2], which includes all publicly available genomes from RefSeq, therefore depicting a broad range of taxonomic diversity.

In this thesis, we investigate the possibility of creating a deep learning model, similar to DeepNOG and DeepFam, for VOG assignment, but trained with data from VOGDB. Since eggNOG only includes a fraction of the viral proteomes contained in VOGDB, it is expected that a similar deep learning model trained on VOGDB will have superior capabilities to DeepNOG.
In contrast to the orthology of cellular organisms, we have to address multiple challenges when working with viral genomes, e.g. very rapid and often massively varying rates of evolution or the large-scale propensity of viruses for reassortment or recombination [44]. Also, we have to keep in mind, that for a large number of viruses the sequence data is incomplete or not yet available. These problems have the consequences of no existing universal approach for inferring virus orthology and therefore missing gold standard data. With this, we expect various challenges in the creation of the model. For example, when training a model, splitting the dataset into training, validation and test sets, requires high class cardinalities. A more general problem arises when we want to evaluate the models performance, as there is no gold standard dataset for VOGs that can aid us in benchmarking. Thus, leaving us with the task of developing new metrics for evaluation. In addition, VOGDB is constantly updated, so we need to design an automatic pipeline that seamlessly integrates the deep learning model and tells us when retraining or reevaluation is needed.

1.1 Biological Background

To investigate the evolutionary relationships between sequences we analyze the similarities between sequences. We group sequences that are related to each other via an ancestral sequence together and form clusters of sequences that share phylogenetic and functional information. This enables us to transfer information to any new sequences that map to these clusters. Considering the orthology of viruses, we are confronted with strongly limited resources, which is unfortunate, since viral dark matter is flooding the sequence repositories and the amount is expected to increase in the future [31]. In the following sections, we will give an overview of the concepts needed for creating and applying orthology resources, to gain functional annotation, focusing on viral genomes.

1.1.1 Sequence Homology

In comparative genomics, we look at conserved regions in genomes of organisms that share the same traits. Homologous genes derived from a common origin and have the

possibility to fulfill similar biological functions. Homology of sequences is indicated by sequence similarity, which is the percentage of identical residues, also called sequence identity. If two sequences are very similar, i.e. they share more similarity than expected by chance, we get strong evidence that these sequences are related evolutionary via a common ancestral sequence. However, there is no need for sequences to share significant sequence identity to be homologous, there are multiple examples of obviously homologous sequences based on e.g. statistically significant structural similarity [37]. Shared ancestry between sequences can manifest from a speciation event, a duplication event or a horizontal gene transfer (HGT) event, see figure 1.1 for a depiction of orthologous and paralogous relationships of genes. Homologs created only by a speciation event are called orthologs and homologs created by a duplication event are called paralogs. HGT is the direct movement of genetic information between different organisms without a parent to offspring transmission. Combining speciation, duplication and HGT with the concepts of gene loss and rearrangements creates a complex structure of relationships around orthologs and paralogs, which needs to be analyzed carefully.

Orthology

In order to give an accurate definition of orthology, it is important to clarify ones perspective on the problem. Biologists define orthology in terms of the functions of the gene products and bioinformaticians define orthology in terms of the similarity of sequences. In this work, we adopt the bioinformatics perspective and use the definitions of orthology as given by Eugene V. Koonin. Orthologous genes originated from a single gene of the last common ancestor of the compared species [28]. This statement gives rise to two requirements that have to be met. At first, we require a single ancestral gene. If it can be shown that the ancestral genome contained two paralogous genes that resulted in the genes in question, we cannot consider an orthologous relationship. Secondly, the ancestral gene must specifically come from the last common ancestor and not from a more ancient ancestor. It is important to mention that there is no connection between orthology and function, in this basic definition of orthology, but it is generally assumed that orthologs share equivalent biological functions over different species [47].
When identifying orthologs, several accomplishments can be made, including the investigation of the evolutionary process or creating clusters of genes to analyze their functional potential [13]. The creation of OGs is important to many tasks in biology, such as the before mentioned transfer of annotation to new sequences. In the past, several projects focused on creating OGs using different construction approaches. The methods used

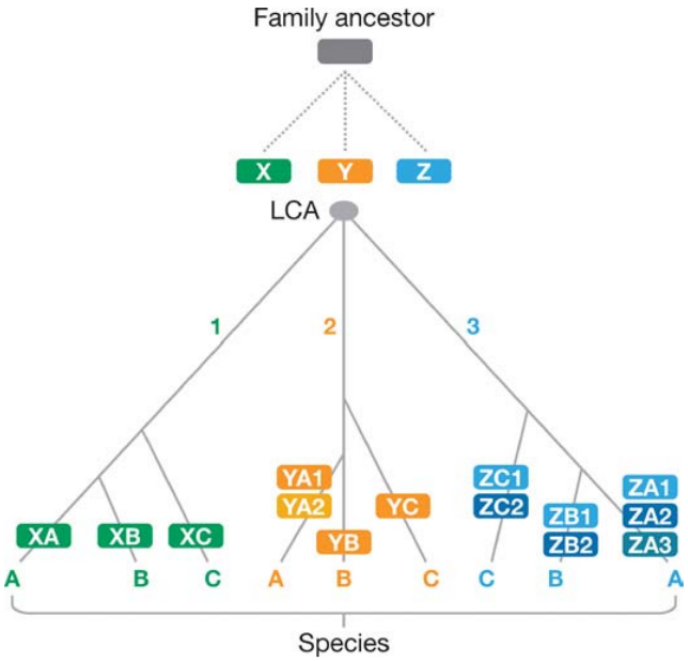

Figure 1.1: Example phylogenetic tree of a gene family, depicting orthologous and paralogous relationships. Branch 1 shows an orthologous relationship of gene X in the species A, B and C. Branch 2 features a duplication of gene Y in species A creating a paralog in this species and co-orthologs to the genes in species B and C. Branch 3 depicts a complex scenario with multiple paralogous and orthologous relationships. Figure taken from [28].

can be split into two categories, namely clustering pairs of genes for similar functions or utilizing phylogenetic trees to identify events producing different functions.
Typical errors in the creation of OGs often result from the presence of xenologs, as these special types of homologs, violate the definition of orthology through the horizontal transfer of genes [28].

1.1.2 Functional Annotation

With the computational background of this work and when speaking of functional annotation, where we associate biological information with gene sequences, we always consider an automatic annotation process. Manual annotation is still considered to be the gold standard, but often not feasible in today's NGS projects [10] and mostly limited to model organisms. When performing an automatic prediction of function we utilize local alignments of a query sequence against a sequence database. We then assign the

annotation of the highest scoring result to the query sequence, following our assumption that sequences that have high identity to each other evolved from a single ancestral sequence. There are manually curated and automatically annotated protein databases, where the latter have a much larger content, but also introduce inaccuracies, making the transfer of annotation a difficult task.
Today, annotation of prokaryotes has matured by having a plethora of tools and extensive annotation pipelines available [30]. In contrast, the task of annotating viruses is still associated with limitations and there is for example no general approach that can deal with several virus families. Viral genomes can be considered as very dynamic with much higher mutations rates than cellular organisms [8]. This leads to a rapid decline of sequence similarity, which is problematic for the creation and use of OGs based on bidirectional best hits. For example, the evolutionary relationship of the enterobacterial phage PRD1 has been traced with the help of the virus capsids and not with the sequence data [35].
To classify viral genes, we distinguish the three categories: genes with homologs to cellular organisms, virus-specific genes and viral hallmark genes. Since there is no single gene shared by all known viruses [9], we have to look at proteins that are conserved over larger taxonomic groups, i.e. proteins involved with housekeeping functions, encoding for example virus structure or replication.

1.1.3 Viral Genomes

In this section, we want to discuss some of the properties that are special to viruses and viral genomes when compared to classical cellular organisms. Further, we will highlight the impact of these properties on our use case of transferring annotation between viral genes with the concepts of orthology.
Viruses descended from multiple discrete viral ancestors [22], making them of polyphyletic origin with no last universal common ancestor (LUCA). We classify viruses by the type of molecule used for the organization of the genomic nucleic acids, known as the Baltimore scheme [29]. In this classification, we differentiate between RNA and DNA viruses and their strandedness (single or double) with positive or negative sense (in case of single-stranded), which results in seven groups (see table 1.1 for Baltimore groups and designated group names).
Viruses come with very different genome sizes, but compared with cellular organisms, their genomes are mostly smaller. Genomes of RNA viruses are small and also have little variation in genome size. DNA viruses show a much larger variation in their genome sizes, from very small single-stranded DNA viruses having about 2 kb to giant double-stranded

Group name	Description	Genome type
Group I	double-stranded DNA viruses	DNA(+/-)
Group II	single-stranded DNA viruses	DNA(+)
Group III	double-stranded RNA viruses	RNA(+/-)
Group IV	positive sense single-stranded RNA viruses	RNA(+)
Group V	negative sense single-stranded RNA viruses	RNA(-)
Group VI	single-stranded RNA viruses with intermediate DNA in their life cycle	RNA(+)
Group VII	double-stranded DNA viruses with intermediate RNA in their life cycle	DNA(+/-)

Table 1.1: The seven groups of the Baltimore scheme. Classifying viruses depending on their genome type and replication method.

DNA viruses with about 2 Mb, figure 1.2 shows the genome sizes and variation for every Baltimore group. In accordance with the genome size, different viruses encode a different number of proteins in the genome, whereby the length of the encoded proteins can also involve large variation.

1.2 Computational Background

In the following sections, we will draw attention to the essential computational concepts when learning how to assign VOGs to new sequences. We will give an overview of the concepts behind machine learning and highlight the methods used in this thesis. Further, we will take a look at the data, that is the OGs and how databases for OGs and more specifically VOGs are constructed and applied.

1.2.1 Machine Learning Basics

Machine learning is a sub-area of artificial intelligence that enables systems to automatically learn from experience (data) and to improve without being explicitly programmed. Machine learning can automatically generate knowledge, identify relationships and recognize unknown patterns. These identified patterns and relationships can be applied to a new, unknown data set in order to make predictions.
The field is divided into three kinds of methods, namely supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the relationship to a target variable is always learned, the target variable can be a class or a numerical value. In

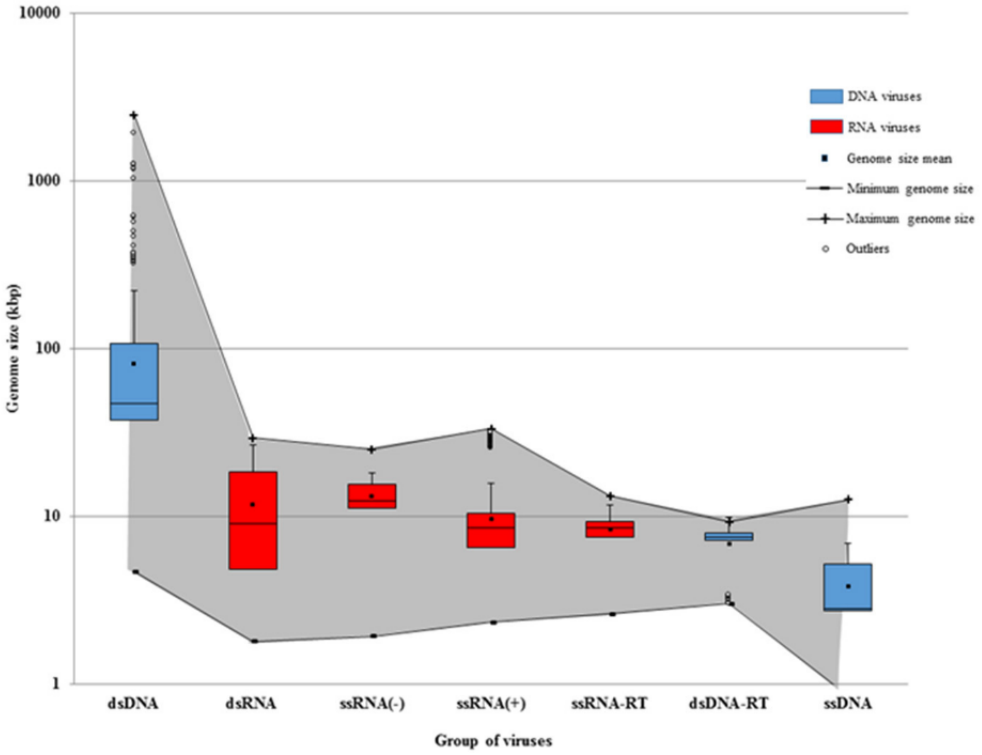

Figure 1.2: Genome sizes of viruses for each Baltimore group. The gray area indicates the maximum and minimum genome size. Figure taken from [5].

unsupervised learning, the algorithm does not receive any sample data, but rather data from which the algorithm should independently identify hidden patterns. The fundamental difference to supervised learning is that unsupervised learning is not designed to calculate a prediction for a known target variable. With reinforcement learning, the algorithm is not shown which action is the right one for each situation, but rather it receives positive or negative feedback from a cost function. The cost function is then used to estimate which action is the right one at which point in time. The key difference to supervised and unsupervised learning is that reinforcement learning does not require any sample data beforehand, the algorithm develops its own strategy in an iterative simulation.
In order for machine learning to work and the software to be able to make decisions, a human has to train the algorithm. By providing training and sample data, the algorithm can recognize patterns and relationships and thus learn from the data. This process is also called model training. After a successfully completed learning process, the trained model is used to evaluate unknown data. This means that better decisions can be made based on these predictions. The main goal is to learn it automatically without human

intervention and adjust actions accordingly.
As the extraction of knowledge from raw data is often very difficult and the number of features has a direct impact on the computational effort, feature engineering is required for most algorithms. Feature engineering is the process of selecting, manipulating, and converting raw data into features that can be used in machine learning, with the aim of simplifying and accelerating data transformations while improving model accuracy. Most of the time, the task of feature engineering is difficult, since it requires a lot of domain knowledge. The ever-increasing amounts of input data in machine learning projects lead to increased popularity of algorithms that can carry out feature engineering by themselves.

1.2.2 Neural Networks

Neural networks are algorithms that are modeled on the human brain, which results in an abstract model of artificial neurons that are able to solve complex tasks. Currently, neural networks are a very active research area, with multiple improvements and a lot of applications every year. Neural networks can have different levels of complexity, but essentially have the structures of directed graphs. If a neural network has particularly deep structures, it is called deep neural network and the associated learning process is then called deep learning.
Deep neural networks are able to process raw or unstructured data and require little domain knowledge, as the algorithm performs the feature engineering itself. This advantage comes with the cost of the need for strong hardware (high memory consumption and need of a GPU), long-running training times and a lack of interpretability (black-box model). In the following, we will introduce the concepts behind neural network models by looking at the simplest form, which is the feedforward network. In addition to that, we will also introduce the concepts behind the network architecture used in DeepNOG and DeepFAM which is the convolutional neural network.

Feedforward Neural Networks

The feedforward neural network (FNN) is the basic model for all modern deep learning approaches [20]. Intuitively, FNNs approximate functions, by learning the mapping y = f(x,W), with x as the input data, W as the parameters to learn and y as the target output (label, in case of a classification task). The network aspect arises from the fact that FNNs are represented by chaining several functions together. In the network architecture, we address these functions as layers, where we have input layers, output layers and hidden layers. The number of the hidden layers is the depth of the network and defines the

ability to approximate complex functions, figure 1.3 depicts a network diagram for a fully connected (dense) FNN. More hidden layers raise the models ability, but also increase the likelihood of overfitting and introduce a higher computational effort when training the model.

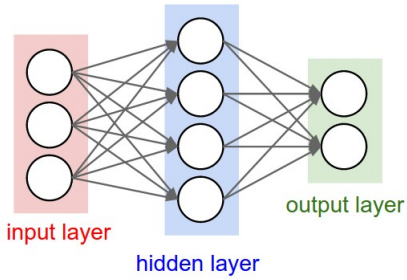

Figure 1.3: Directed acyclic graph for a fully connected (dense) FNN. Edges represent weights and nodes represent input, hidden and output values. Figure taken from [45].

Linear models for classification use M linear combinations of non-linear basis functions ϕ to transform the inputs x such that

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right)$$

where f is a non-linear activation function. In FNNs we let the basis functions depend on tuneable parameters, such that they can be adjusted along with the coefficients when training the model. Every layer consists of activations a_j, e.g. the first layer is calculated as

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} + w_{j0}^{(1)}$$

which is the sum of all combinations of weights w_{ji} and biases w_{j0} applied to the data x. Consequently an activation function, like sigmoid, tanh or ReLU, is applied to every computed activation. The outputs of the first layer are the units of the hidden layer, which then again can be combined into another hidden layer. In the last layer, that is the output units, depending on the use case, a final activation function, e.g. softmax, is applied.

The flow of data through the network, we have described so far, is called forward propagation. During training, we notice the error of the model, which is manifested by the error function. Our goal is to approximate the minimum of this error function by

adjusting the weights accordingly, this is done in three steps. Firstly, we acquire the gradient of the loss function by the backpropagation algorithm. Secondly, we evaluate the derivatives with respect to the weights. Thirdly, we use the derivatives to adjust the weights accordingly, usually making use of stochastic gradient descent algorithms.

Convolutional Neural Networks

A convolutional neural network (CNN) is a specific type of neural network optimized for data that has a grid-structured topology (e.g. images). In a CNN we replace the general matrix multiplication by a convolution, in at least one layer [20]. In the field of neural networks, convolution is a mathematical operation, where we apply a small matrix, named kernel, to the input, to produce a feature map. This technique allows for the extraction of high-level features, while requiring fewer operations than a full matrix multiplication. The kernels (often called filters) are applied to the input in a sliding-window approach, see figure 1.4 for a depiction of this process.

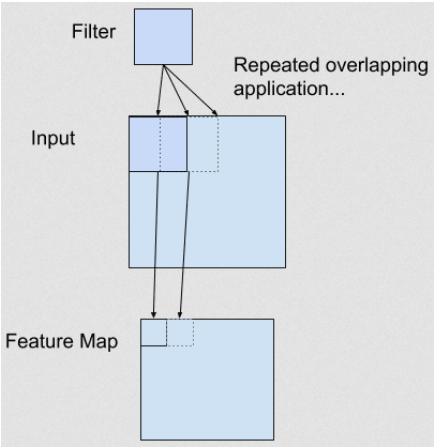

Figure 1.4: Example of a convolution. A filter is applied in a sliding-window approach on a two-dimensional input to produce a feature map. Figure taken from [45].

Each layer in a CNN is made up of three consecutive parts. In the first part, multiple convolutions are performed in parallel and produce a set of linear activations. In the second part, called detector, each activation is run through a nonlinear activation function (e.g. ReLU). In the third part, we use a pooling function to summarize nearby neurons (e.g. max-pooling for taking the maximum value of a rectangular neighborhood). In figure 1.5 we see a CNN which is able to detect handwritten digits, two convolutional layers with max-pooling are used to extract features, followed by two dense layers performing the classification.

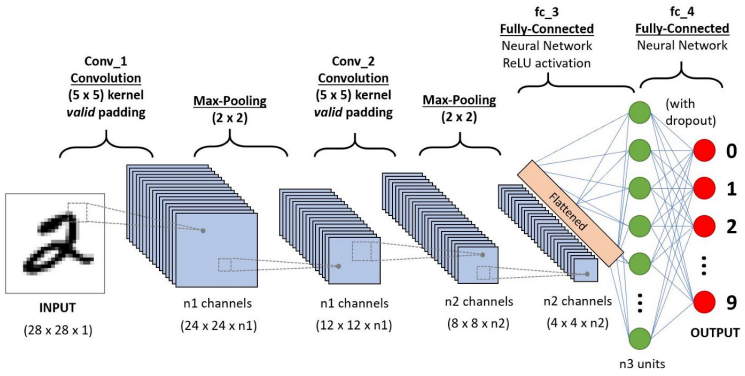

Figure 1.5: A CNN to classify handwritten digits. The network consists of two convolutional layers with max-pooling for detecting features and two dense layers for making the classification. Figure taken from [51].

Database	**Taxonomic Range**	**# Proteomes**
COG	bacteria and archaea	1,309
eggNOG	all domains of life, viruses	7,592
OMA	all domains of life	2,326
OrthoDB	all domains of life, viruses	7,588
OrthologeDB	bacteria and archaea	2,069

Table 1.2: Overview of a selection of orthology resources, depicting the included taxonomic range and the number of proteomes. Full table available at *https://questfororthologs.org/orthology_databases.*

1.2.3 Databases of Orthologs

The content of today's orthology databases covers a very broad spectrum of applied methodologies and specific use cases. Resources strongly differ for example in the number of included species or the available taxonomic range. At the moment, the Quest for Ortholog consortium lists about 50 different orthology resources on their website [19]. This group of scientists working in the field of orthology has set itself the task of collecting and discussing the available resources in orthology and providing benchmark data sets for the evaluation of these resources. In table 1.2 we can see an excerpt of this information concerning the most important orthology databases, in general or specific for the use case of this thesis.

In the following subsections, we will highlight the orthology resources that are most important for our use case, namely COG, eggNOG and VOGDB. COG was one of the first orthology resources and served as a seed resource for the construction of the eggNOG

database. COG was the dataset used for the development of DeepFam and eggNOG was the dataset used for the development of DeepNOG. Further, the construction approach of eggNOG guided the development of VOGDB.

COG

Since first published in 1997, the "Clusters of Orthologous Genes" database (COG) has been a widely used tool for genome annotation and comparative genomics of microbes [17]. The latest update of COG was released in 2020 and contains 4,887 OGs, covering 1187 bacteria and 122 archaea [18]. The approach behind COG only uses complete genomes and builds upon the assumption that any set of at least three proteins are most likely true orthologs, if they have high similarity to each other and low similarity to any other proteins from the same genomes. The success of previous COG versions is based on extensive manual curation as well as extensive computational analysis for annotating poorly characterized COGs. Drawbacks of the COG database include the lack of hierarchy between OGs, the high sensitivity to domain rearrangements and the lack of scalability due to the needed labour-consuming manual curation. [17].

eggNOG

The "Evolutionary genealogy of genes: Non-supervised Orthologous Groups" (eggNOG) covers orthology relationships for thousands of organisms from all domains of life and viruses. The database provides extensive functional annotations for its OGs, which cover a broad taxonomic range. Further, eggNOG offers a hierarchy between OGs for different resolutions, which in its actual release covers 379 taxonomic levels. At its core, eggNOG uses a similar construction approach like COG, where three-way similarities between genes are utilized to cluster sequences to OGs. The most recent version of eggNOG contains 4.4M OGs, covering 5090 organisms (4445 bacteria, 168 archaea, 477 eukaryotes) and 2502 viruses. New releases of eggNOG are published every two years, incorporating new source databases and algorithmic improvements [24].

VOGDB

At the moment VOGDB is the largest resource for viral orthology. The current release (vog207 - Oct 22, 2021) contains 28,386 VOGs with 438,852 proteins assigned to them, coming from 10,046 genomes. The release cycle of RefSeq outputs a new release every two months. A new release of VOGDB is computed and released also every two months, referencing the latest RefSeq release by using the same release numbers. The VOGDB

pipeline features three consecutive stages, see figure 1.6 for an overview. In the first stage, viral genomes are extracted from RefSeq and filtered for quality and completeness. Also, the algorithm tries to reannotate any polyproteins. In the second stage, the so-called premature VOGs (preVOGs) are built via pairwise alignment and bidirectional best hits, which then are clustered with a procedure similar to eggNOG and COG. In the last stage, the preVOGs are enriched with remote homology information to become (mature) VOGs. For this, HMMs are computed for each preVOG and aligned against each other to produce an all-against-all matrix that can again be clustered to produce the matured VOGs.

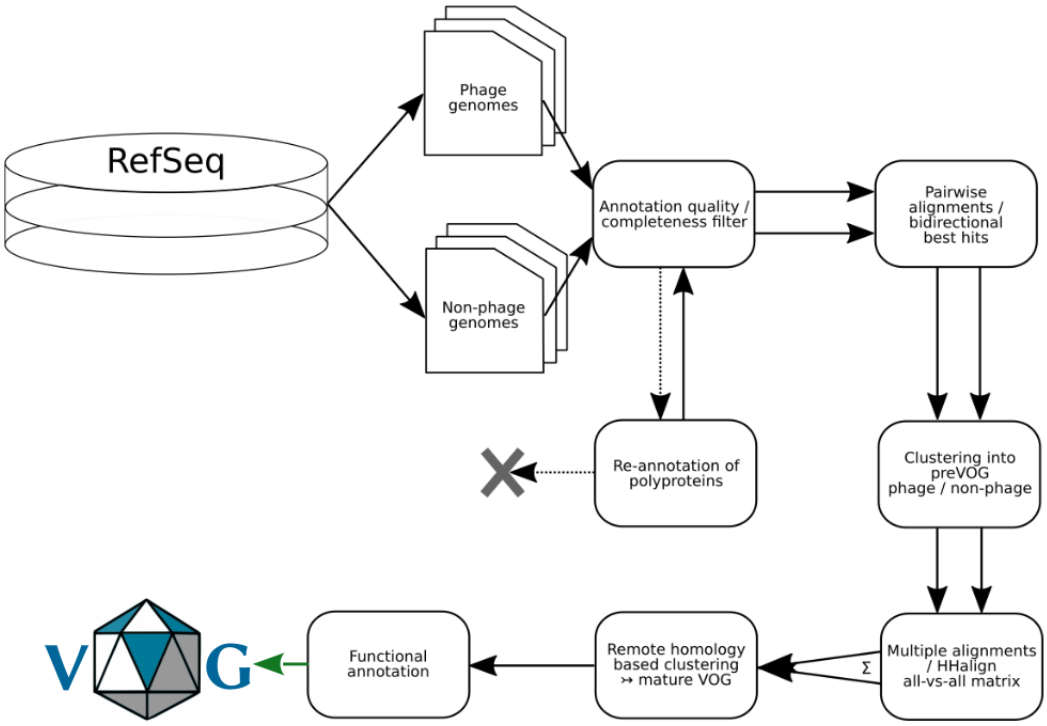

Figure 1.6: Overview of the VOGDB pipeline. Figure taken from [2].

In the last years, VOGDB has been used in several projects, including a metagenomic analysis investigating the viral distribution patterns in reef invertebrates [32] and another study analyzing the mobility of genetic elements of viruses in eukaryotic cells [48].

1.2.4 Orthologous Groups Assignment

Computational methods for assigning new sequences to precomputed OGs can be grouped into alignment-based and alignment-free methods. Alignment-based methods rely on comparisons of multiple sequences and HMMs for database searching of proteins by

sequence homology [14]. To assign a sequence to an OG, an inference with all HMMs of the database is required, which makes the approach computationally expensive. Alignment-free methods offer an alternative to this problem, which is why they are actively researched.

Alignment-based Methods

HMMs have become very important in the field of protein family analysis, where these methods have the advantage of representing a family of hundreds of homologous sequences with a single model and thereby compensating the difference in speed when compared to BLAST [43]. In the last years, HMMER's performance has improved and in its actual release (HMMER3) the speed difference to BLAST is negligible [16]. When looking at the model performance, HMMs are still considered to be state-of-the-art in terms of accuracy and sensitivity [14]. However, when assigning a protein sequence to a VOG, one must search the sequence against all HMMs in the database, since every VOG is modeled by a single HMM. Therefore, in this work special attention is paid to the extent to which an alignment-free method based on deep learning can overcome these computational problems.

Alignment-free Methods

The central question when developing an alignment-free method is how to derive feature vectors from raw sequences. Before DeepFam was published, the best approach utilized k-mers to create feature vectors [53], which in some cases showed better performance than alignment-based methods [34]. Using k-mers introduces two obvious problems, namely the loss of order information in sequences and the requirement of exact matches to the k-mers which is not backed biologically since amino acids share biochemical properties.
DeepFam was the first alignment-free method that utilized a CNN to model OGs. A DeepFam model is a multiclass classifier that assigns OGs from the COG database. DeepNOG, inspired by DeepFam, is also a CNN which extracts patterns from protein sequences and was developed to overcome several limitations of DeepFam [14]. DeepNOG was initially developed to assign sequences to OGs from the eggNOG database, but it also features a training module that allows the training for custom databases. Raw protein sequences are taken as input and the model returns class labels for each of them. This catalogs DeepNOG as an end-to-end learning method, where no feature engineering and extraction is needed. A DeepNOG model is able to model all VOGs within a single multiclass classifier, but the training process is still expensive and might require multiple CPUs or even multiple GPUs. Later in the text, when we evaluate the DeepNOG model,

we will compare DeepNOG extensively with the VOGDB HMMs, in terms of runtime, memory consumption and of course assignment performance.

1.3 Problem Description and Research Goals

DeepNOG offers fast and accurate functional and taxonomic annotation for genomes from cellular organisms. To this day, a similar method for annotating the genomes of viruses does not exist. HMMs form a good baseline for the annotation of viral genomes, but they are too computationally expensive to be applied in large-scale projects.
In this thesis, we want to investigate, if we can train a DeepNOG model with data from VOGDB, to get a model for fast and accurate VOG assignment that has similar confidence in its predictions like the state of the art HMMs.
Since the DeepNOG software offers us several parameters for tuning the training procedure, we want to explore different parameterizations and the effects they have on the learning process and the model performance. Further, we will also compare the DeepNOG model to the HMMs from VOGDB in terms of execution runtime and memory consumption as well as the applicability of both methods to real-world scenarios and uncultivated genomes.
From this proposed workflow, the following key problems emerged. Firstly, we must prepare the data to be used with DeepNOG by taking the number of sequences and VOGs needed for training and evaluation into account, while keeping an eye on the class cardinalities. Secondly, we must train a model that generalizes well and tune the hyperparameters offered by DeepNOG to maximize model performance for this specific dataset. Thirdly, an extensive performance evaluation has to be done, where we find out how much loss in accuracy we get by using DeepNOG instead of the VOGDB HMMs, we achieve this by comparing the two methods on internal (i.e. VOGDB data) and external datasets. For the fourth and last step, we must find a way on how to integrate the model retraining on new VOGDB releases into the VOGDB landscape.
Specific use cases for the resulting models include viral genome annotation, comparative analysis of viruses, and metagenomic projects that focus on viruses. In particular, the reannotation of viral genomes is strongly supported by the models, as we are subject to time restrictions here and the method developed will be much faster than the methods currently used.

2 Materials and Methods

2.1 VOGDB

The VOGDB dataset consists of several files (e.g. tab-separated, FASTA) and can be downloaded from http://vogdb.org/download. Table 2.1 shows basic statistics of the actual VOGDB release vog208 and table 2.2 gives an overview of all dataset files. The files *vog.faa.tar.gz*, *vog.members.tsv.gz* and *vog.hmm.tar.gz* are particularly important in this work. To assign VOGs for a given set of protein sequences the HMMER software package is used. We use the tool hmmscan to search sequences against VOGs, see listing 2.1 for more details. After this, we parse the results file for the columns target name (=VOG), query name and E-value and take the entry with the lowest E-value as the assignment.

```
# Concat all HMMs into one single file
tar -xzOf vog.hmm.tar.gz > hmmdb
# Create HMM database for hmmscan
hmmpress hmmdb
# Search protein sequences against HMM database
#    --tblout    generates tab-separated output file
hmmscan --tblout result.txt hmmdb seqs.faa
```

Listing 2.1: Steps needed to perform a search of sequences against an HMM database.

Release date	Nov 08, 2021
Release number	vog208
Data source	NCBI RefSeq release 208
Number of genomes	10,046
Number of proteins in VOGs	438,852
Number of VOGs	28,386
Virus specific VOGs (high/medium/low stringency)	22,597/24,359/25,131

Table 2.1: Statistics of the latest VOGDB release vog208.

File	Filetype	Description
vog_functional_categories.txt	text	Letter codes for indicating functional category.
vog.species.list	TSV	All genomes used for constructing VOGs.
vog.proteins.all.fa.gz	FASTA	All proteins from the genomes in vog.species.list.
vog.genes.all.fa.gz	FASTA	All gene sequences from the genomes in vog.species.list.
vog.faa.tar.gz	FASTA	A file for every VOG and the protein sequences assigned.
vog.raw_algs.tar.gz	MSA	Multiple sequence alignments for each VOG.
vog.hmm.tar.gz	HMMER	HMMs from MSAs for each VOG.
vog.members.tsv.gz	TSV	Proteins to VOGs mapping.
vog.annotations.tsv.gz	TSV	Annotations from SwissProt and RefSeq.
vog.lca.tsv.gz	TSV	Taxonomic lineage of the LCA for each VOG.
vog.virusonly.tsv.gz	TSV	Occurrence of VOGs in virus genomes for different stringencies.

Table 2.2: The complete VOGDB dataset.

2.2 DeepNOG

Initially trained and optimized to assign OGs from eggNOG, the DeepNOG software allows also training new models for custom orthology databases. For this, DeepNOG requires a specific set of input mapping and sequence files, table 2.3 gives details on the file formats. DeepNOG offers multiple parameters to steer the training process, in listing 2.2 the most straightforward version for training a custom model is shown. A successful training outputs 3 files, namely a model file *model.pth*, a labels file *labels.npz* and an evaluation file *eval.csv*. The model file consists of all the network weights and is therefore the file we later use for assignments. The labels file holds the ground truth and the assigned labels over each epoch. The evaluation file depicts the learning process by listing accuracies and losses for training and validation for each epoch.

```
# Train DeepNOG model for custom databases
#     -a     network architecture used
#     -o     output directory for the model files
#     -db    database name
#     -t     taxonomic level in database
deepnog train
    -a "deepnog" \
    -o . \
    -db "VOGDB" \
    -t "1"
    train.faa \
    val.faa \
    train.csv \
    val.csv
```

Listing 2.2: Call of DeepNOG for training a model from a custom database.

2.2.1 Architecture

DeepNOG uses a feedfoward architecture with convolutions to extract patterns, figure 2.1 depicts the setup of the different layers in the network's architecture. To get a numerical representation of an input sequence, the sequence is mapped to a vector space, in machine learning terms this is called a *word embedding* [12]. Applying this technique to protein sequences is, to our knowledge, unique to DeepNOG and will be explained in more detail in the following section. Then, a 1-D convolutional layer is applied to the embedding, meaning that filters are moved over each column of the embedding. Filters come in eight different sizes (8-36) and are used multiple times (up to 250). This creates a vector of

File	Description
train.csv	Sequence-ID to VOG mapping, training split. Columns: Row-Nr, Sequence-ID, VOG-ID Mandatory header line: ",string_id,eggnog_id"
val.csv	Sequence-ID to VOG mapping, validation split. Columns: Row-Nr, Sequence-ID, VOG-ID Mandatory header line: ",string_id,eggnog_id"
train.faa	FASTA file for model training, containing all sequences for the sequence-IDs from train.csv.
val.faa	FASTA file for model validation, containing all sequences for the sequence-IDs from val.csv.

Table 2.3: Files needed for training a DeepNOG model on a custom database. CSV-files represent sequence-ID to VOG mappings, FASTA-files contain the sequences for the corresponding split.

weights modeling the occurrences of amino acids at specific positions in the subsequence. The weights are then handed over to a SELU activation function, which was chosen over a ReLU activation function for stability and efficiency reasons [27]. This results in filters that are sensitive to discriminatory motifs for OG assignments. DeepNOG downsamples the outputs of the convolutional layer utilizing adaptive-max pooling and then directly places the classification layer (softmax layer) after it, which results in only one hidden layer (dense layer) instead of two, thereby reducing the number of parameters greatly.

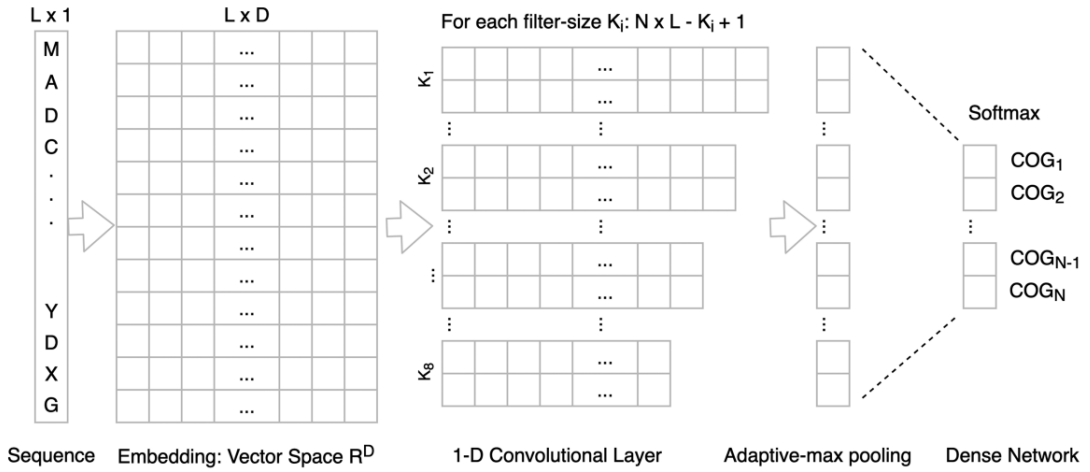

Figure 2.1: DeepNOG end-to-end model architecture consisting of three main parts, that is the word embedding, the convolutional layer and the classification layer.

2.2.2 Word Embedding

To be used with deep networks, sequences must be transformed into numerical vectors. The most straightforward way of doing this is to use a procedure called one-hot encoding. In one-hot encoding vectors have a fixed length, in our case this is the length of the amino acid alphabet, and are set to 1 at a specific position (representing a specific amino acid) and 0 elsewhere. This sort of encoding enforces equidistance between all amino acids and thereby loosing chemical and biological properties, which represent important features in our use case. To account for the similarities of amino acids, DeepNOG embeds them into a word embedding, which is a vector representation that basically starts with a randomized one-hot-encoding which is then jointly optimized during the training of the network. This learned numerical representation of amino acids can have a lower dimension than the length of the amino acid alphabet and we are also able to inspect it visually for biological plausibility, see figure 2.2.

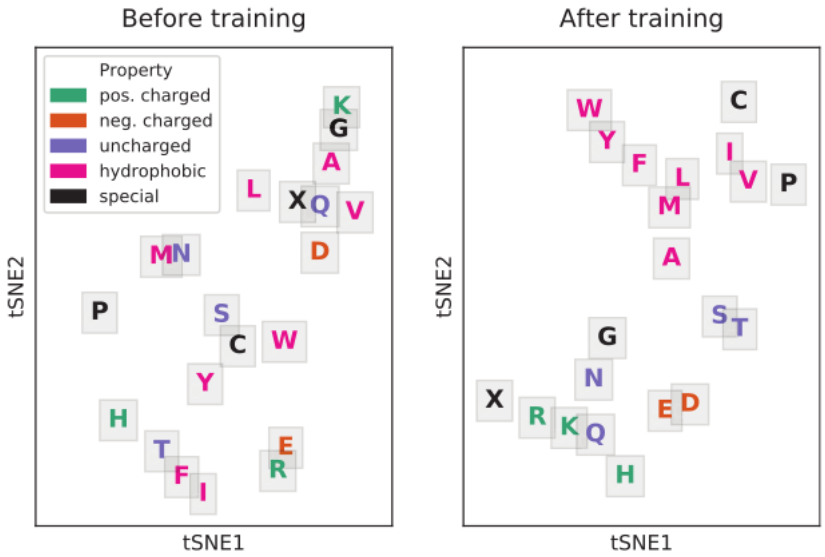

Figure 2.2: Representation of amino acids in the word embedding layer of DeepNOG. On the left the random initializations before the training, on the right the learned representations after training.

2.2.3 eggNOG Dataset

In the development of DeepNOG, the focus was set to the root and bacterial level of eggNOG, as these taxonomic levels are highly relevant to metagenomic studies [14]. Most parts of the eggNOG database are constructed in an unsupervised manner, where

	eggNOG (bacteria)	**VOGDB**
#OGs	206,782	28,386
mean	74	16
std	904	85
min	2	2
25%	2	2
50%	3	4
75%	7	8
max	97,670	7,764

Table 2.4: Comparison of the eggNOG (bacteria level) and VOGDB datasets. Descriptive statistics for the number of OGs in the datasets, as well as OGs size distribution.

assignment of one protein to multiple OGs is allowed. DeepNOG assigns a single class label (VOG) to every protein, that is the label with the highest prediction probability. In the case of low confidence assignments, no label is assigned. Therefore, when training the model, DeepNOG only considers proteins from eggNOG that are assigned to a single OG, any multi-domain proteins are excluded. For the eggNOG root and bacteria levels, this means that about a fraction of 3-4% of proteins is excluded [24]. To get an idea of how the eggNOG dataset differs from the VOGDB dataset, we calculated descriptive statistics on the number of OGs and OGs size for eggNOG bacteria level and VOGDB, see table 2.4. In the case of multiclass classification, class cardinalities are of particular interest, figure 2.3 shows the similarities between OG size distributions for eggNOG and VOGDB. Both datasets mainly consist of OGs with less than 10 members, as well as a few OGs with a high count of members.

2.2.4 Comparison to DeepFam

DeepFam was the first alignment-free method that was based on deep learning [41]. It was also the first alignment-free method that was fast, while at the same time being highly accurate, when compared to the commonly used HMMER software. Besides that, the DeepNOG team identified several limitations in the DeepFam approach. DeepFam utilizes the COG database, which is manually curated but orders of magnitudes smaller than eggNOG. This not only limits DeepFam's applicability but also introduces scaling issues when training on larger datasets. On top of that, DeepFam restricts the length of the input sequences. Instead of adaptive-max-pooling, DeepFam uses a standard 1-max-pooling layer which enforces a specific length of all input sequences (L=1000). Shorter sequences

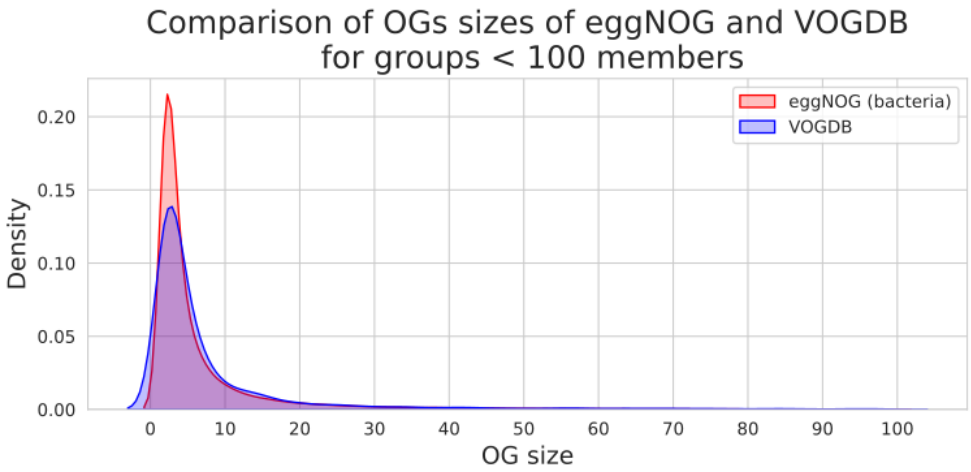

Figure 2.3: Similarities between OG size distributions for eggNOG and VOGDB.

get zero-padded and longer sequences are removed from the dataset. Thus, DeepFam is not applicable to arbitrary sequences which is a hard limitation, especially when looking at real-world applications.

2.3 Pipeline Overview

To find out whether VOGDB data is applicable to DeepNOG and whether the resulting model can be used for VOG assignment, we create the following pipeline:

1. Data preparation
    a) Extract Sequence-ID to VOG mapping from VOGDB
    b) Align VOGDB sequences to UniRef to get UniRef cluster labels
    c) Create member threshold datasets
    d) Split the datasets into training, validation and test sets
2. Model training
    a) Train a model for every UniRef-threshold combination
    b) Evaluate the training processes and the model performances
    c) Retrain and re-evaluate for different model parameterizations
3. Model inference

a) Perform assignments for every model on every test set

b) Evaluate model performances

c) Perform assignments with HMMs on every test set

d) Compare model performance of DeepNOG and HMMs

e) Compare runtimes and memory consumption

4. Performance on external datasets

a) Get subsets of pVOG and UViG datasets

b) Redundancy analysis

c) Perform assignments with DeepNOG and HMMs

d) Evaluate performances and compare results

2.3.1 Software Stack

In this work, depending on the task type, different tools and environments were used. The task types included are: pure computational, investigative computational, data analysis and orchestration. Pure computational tasks were done with a mixture of Shell and Python scripts. Investigative computational tasks and data analysis were done using Jupyter notebooks with Python. For putting it all together, i.e. orchestration, the Nextflow workflow manager was used. See table 2.5 for more details on tools, software packages and versions. For running data analysis tasks in Jupyter notebooks, a separate python environment was created with Conda and loaded when needed. Computational expensive tasks, as well as tasks that needed a GPU, were run on the Life Science Compute Cluster [1] with an appropriate configuration.

2.4 Data Preparation

In a first step, we need to get a simple 1:1 protein to VOG mapping for all proteins in VOGDB that are actually assigned to a VOG. The VOGDB file *vog.members.tsv.gz* lists the VOG to Sequence-ID mapping in a 1:n fashion and therefore needs flattening first to be further processed. The list of proteins needs then to be cleaned since it can contain duplicate or empty entries. The resulting set of proteins was then used to create a FASTA file containing solely assigned sequences. At this point, we would already be able to split the dataset into a training, validation and test set and start training the DeepNOG model right away. Unfortunately, randomly splitting (stratified split) would

Software	Version	Library / Tool
Python	3.9.0	pandas 1.3.2 matplotlib 3.4.2 seaborn 0.11.2 scikit-learn 0.24.2 numpy 1.20.3 scipy 1.7.1 biopython 1.78 pytorch 1.8.0
Jupyter	1.0.0	jupyter lab 3.1.7
MMSeqs2	11-e1a1c	createdb search convertalis
DeepNOG	1.2.3	
HMMER	3.3	hmmpress hmmscan
MEME	5.1.1	fasta-subsample
Nextflow	21.04.0	
Conda	4.10.3	

Table 2.5: Software packages, tools and libraries used in this work.

introduce unwanted effects, like strong over-fitting, since we are facing here a dataset with low class cardinalities and at the same time high class imbalance. These problems limit the model's applicability and performance. Also important to mention is that we don't have control over the used labels and have to take what the randomization gives us.

2.4.1 Group Splits

To overcome the unwanted effects of a stratified split, we make use of the so-called group split. In this technique, the splitting is performed on a third-party label which ideally encodes domain-specific stratifications [38]. In the case of sequences and OGs this is, for example, any information that represents similarity between sequences, since OGs are for the most part built on sequence similarity measures. UniRef clusters provide clustered sets of sequences from the UniProt Knowledgebase [50] at several resolutions of identity (100, 90 and 50), thereby uniting similar sequences under a single UniRef cluster label. To obtain a protein to UniRef mapping, a sequence alignment of VOGDB to UniRef is necessary. This resulted in 3 alignments, one for each UniRef resolution, which was done by using the MMseqs2 software package [46]. The group split itself then was performed with the DeepNOG utility function *data.group_train_val_test_split()* which makes use

of scikit-learn's GroupShuffleSplit.

2.4.2 Member Thresholds

Since most VOGs have a very low class cardinality, it is important to investigate the effects where we remove small VOGs from the dataset, thereby boosting the learning process. Since fewer labels have to be classified and on average more data is available per label, the model's performance is expected to increase. To achieve this we introduced two VOG member thresholds, namely one where we kick out VOGs with very few members and one where we only include bigger VOGs, thereby reducing the scope of the dataset strongly.

2.5 Model Training

When we combine the multiplicities of UniRef resolutions and member thresholds we get the following 9 datasets:

- UniRef100 with no threshold
- UniRef100 with 10+ threshold
- UniRef100 with 100+ threshold
- UniRef90 with no threshold
- UniRef90 with 10+ threshold
- UniRef90 with 100+ threshold
- UniRef50 with no threshold
- UniRef50 with 10+ threshold
- UniRef50 with 100+ threshold

This means that for every DeepNOG parameterization, 9 model trainings have to be done and evaluated. After a first training run with DeepNOG's default parameterization, we changed some of the default hyperparameters and retrained new models. We repeated this process until we got a clear picture of the reachable model performance and the influence of the different hyperparameters. To evaluate the training process we looked at the learning curves, i.e. change in accuracy and loss for training and validation set for each epoch, as well as the training runtime for every model.

2.5.1 Hyperparameters

DeepNOG offers multiple parameters to tune the training process and its learning behavior. Most of these parameters are passed as command line arguments, some of them have to be specified in DeepNOG's own configuration file. In the following text we will highlight and explain the most important tunable hyperparameters of DeepNOG in our setup.

Batch Size

This parameter defines how many sequences are processed between the updates of the internal network parameters. The default batch size of DeepNOG is set to 64 sequences, for CPUs a batch size of 1 is recommended. Since larger batch sizes lead to fewer updates of the network and vice-versa, this parameter can have an influence on the learning process.

Dropout Rate

This parameter is set within the DeepNOG configuration file *deepnog_ config.yml* in the section *architecture*, with 0.3 being the default value. Dropout is a form of regularization in which a fraction of the neurons gets randomly chosen to be dropped out (ignored) and don't receive updates from the backward pass. The resulting effect is that the network is prevented from focusing on specific neurons and with that also from overfitting.

Number of Epochs

An epoch represents one full pass of the training data through the network, per default DeepNOG does 15 epochs per training. Deep neural networks are usually optimized by a set of iterative algorithms (e.g. gradient descent). The iterative nature of these algorithms requires multiple training runs to ensure that it converges to the global minimum. The number of epochs is the hyperparameter that influences the training runtime the most, so it is recommended to start with a moderate amount of epochs and raise it if needed or the performance is still expected to improve.

Learning Rate and Decay

The learning rate starts out with a fixed value and is then reduced after every epoch by the learning decay. The default value for the learning rate is 0.02 and 0.75 for the learning decay. With these parameters, we control the weights updates in feedback to

File	Description
test.faa	FASTA file with sequences from which VOG assignments should be made.
test.csv	Ground truth mapping of Sequence-ID to VOG. Columns: Row-Nr, Sequence-ID, VOG-ID Mandatory header line: ",string_id,eggnog_id"

Table 2.6: Input files needed by DeepNOG for assigning VOGs to sequences, including performance evaluation via ground truth labels.

the estimated error. Too small learning rates may lead to long training processes, on the other hand, too large learning rates lead to instability.

2.6 Model Inference

To evaluate the performance of the trained models for unseen sequences, we use DeepNOG's inference module to assign VOGs for the test split of every dataset, see listing 2.3 for an example call and table 2.6 for information on the supplied input files. For every sequence, DeepNOG outputs an assignment accompanied by a confidence value. If we also supply the ground truth labels, DeepNOG will report several performance measures that are useful for evaluating multiclass problems.

```
# Assign VOGs for a set of sequences
#     -w    weights file of the trained model
#     -t    taxonomic level in database
#     -a    network architecture used
#     --test_labels    ground truth labels
deepnog infer \
    test.faa \
    -w model.pth \
    -t "1" \
    -a "deepnog" \
    --test_labels test.csv
```

Listing 2.3: Call of DeepNOG for assigning VOGs to sequences with performance evaluation via ground truth labels.

2.6.1 Metrics

To quantify the quality of VOG assignments, DeepNOG offers us the following performance measures. Since DeepNOG models are multiclass classifiers, it is important to mention,

that the multiclass versions of these performance measures are involved. In contrast to the more common versions coming from binary classification problems.

Accuracy

The fraction of correct predictions made by the model, i.e.

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

with $\hat{y}_i$ being the predicted label of the i-th row and y_i being the true label. Accuracy is usually the first metric to look at when evaluating the performance of a multiclass classifier, but since this dataset features class imbalances, we need to look at further metrics that aren't that vulnerable.

Precision, Recall and F-measures

Considering a binary classification task, precision is the ability of the model to not classify samples as positive when they are in truth negative. Furthermore, recall is the ability of the model to retrieve all positive samples. F-measures are different weighted harmonic means of precision and recall. In our case we only consider the F1 score, where recall and precision contribute equally. For a multiclass classifier, the concepts of precision, recall and F1 are transferred by looking at each label independently. How to consider the labels independently is set by type of averaging. A macro-average means that the metric is computed independently for each class and then averaged, so each class is treated equally. On the other side, a micro-average will collect all the contributions of all classes to compute the metric, thereby being more robust in case of a class imbalance. The metrics are defined as followed. With

- y as the set of predictions and $\hat{y}$ as the truth
- L as the set of class labels
- y_l as the subset of predictions that have label l
- $\hat{y}_l$ as the subset of the truth that have the label l

we get the metrics

- $P(y, \hat{y})$ as micro-average precision and $\frac{1}{|L|} \sum_{l \in L} P(y_l, \hat{y}_l)$ as macro average precision.

- $R(y, \hat{y})$ as micro-average recall and $\frac{1}{|L|} \sum_{l \in L} R(y_l, \hat{y}_l)$ as macro-average recall.
- $F1(y, \hat{y})$ as micro-average F1 score and $\frac{1}{|L|} \sum_{l \in L} F1(y_l, \hat{y}_l)$ as macro-average F1 score.

where

- $P(A, B) := \frac{|A \cap B|}{|A|}$
- $R(A, B) := \frac{|A \cap B|}{|B|}$
- $F1(A, B) := 2\frac{P(A,B) \times R(A,B)}{P(A,B)+R(A,B)}$

Matthews Correlation Coefficient

The Matthews correlation coefficient (MCC) is a measure that focuses its applicability to problems that have a huge class imbalance [26]. Considering a multiclass classifier, the MCC is defined with the help of a confusion matrix C and the set of all classes K.

$$MCC = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{(s^2 - \sum_k^K p_k^2) \times (s^2 - \sum_k^K t_k^2)}}$$

with

- t_k as the number of true occurrences of class k
- p_k as the number of predictions of class k
- c as the number of correct predictions
- s as the number of samples

2.6.2 Assignment Confidence

For every sequence, DeepNOG outputs a probability distribution over the available classes. The output of a single neuron represents DeepNOG's confidence, that the sequence belongs to the corresponding class from the set of available classes, keeping in mind that in this problem and its setup we don't know all the classes. Firstly, as mentioned above, the viral protein sequence space is only marginally explored, so there always will be VOGs that are missing in the dataset. Secondly, when splitting the data into training, validation and test set, it is possible that whole VOGs, i.e. all sequences from a VOG are put into the

validation or test set, thereby missing in the training procedure and becoming unknown to the model. When thinking about applicability in a real-world scenario, at first glance, it might seem odd to omit VOGs completely, since the predictive power in terms of assignable labels is reduced. At a second glance, our model becomes the opportunity to learn the possibility that a sequence doesn't map to any VOG, thereby strengthening the models confidence. Furthermore, with VOGs being able to be unique to the validation or test set we again prevent over-fitting and therefore get a more accurate validation process and a much clearer performance evaluation. When evaluating the performance, the fraction of assignments that were made with high confidence is of great interest. We therefore count the number of assignments that pass a certain confidence threshold and then show this together with e.g. the accuracy, so we immediately see if a model is able to make good and meaningful assignments. The default confidence threshold by DeepNOG is set to 99% which was determined by looking at the Pareto boundary of minimizing false positives and false negatives for the eggNOG bacterial level.

2.6.3 Comparison with HMMs

To compare DeepNOG to the VOGDB HMMs, we use the tool hmmscan of the HMMER suite to assign VOGs to the sequences of the test sets. With this, we are able to make comparisons in terms of model performance, inference runtime and memory consumption.

HMMs Performance

The tabular output format of hmmscan is a HMMER specific file format, which can be parsed by using the HmmerIO module of Biopython with option *hmmer3-tab*. Sequences that aligned to HMMs are called hits, for every hit the E-value is reported. We determine the hit with the lowest E-value for every sequence, according to a maximum threshold to separate high from low confidence assignments. Since hmmscan doesn't necessarily find hits for all sequences, we have to collect the missing sequences and account them as low confidence assignments, in order to have a comparable setup between DeepNOG and VOGDB HMMs. In the last step, for the actual performance comparison, we compute all the performance measures from DeepNOG for the HMMs assignments.

Memory and Runtime

To compare the memory consumption of both methods we look at the weights file of DeepNOG and the HMM database of VOGDB. For a runtime comparison, we must differ between DeepNOG running on a CPU or a GPU, as well as hmmscan running on a single

CPU or in a parallel setup. This results in 2 runs of DeepNOG and 2 runs of hmmscan. We repeat this process multiple times and note the average runtime.

2.7 Performance on External Datasets

In this section, we further evaluate the model performance considering real-world scenarios. For this we picked subsets of two external datasets, pVOG [21] and IMG/VR [40], and run assignments with both DeepNOG and the VOGDB HMMs.

2.7.1 pVOG

The pVOG dataset was released in 2014 and is a small VOG resource. We expect the sequences from the pVOG dataset to be highly redundant with VOGDB. To compare assignment results of DeepNOG and VOGDB HMMs we simply calculate and visualize the overlap per pVOG, for a subset of pVOGs.

2.7.2 IMG/VR

The IMG/VR dataset is a huge dataset of sequences of uncultivated viral genomes (UViGs). UViG sequences are not contained in RefSeq, therefore we expect a very low number of redundant sequences with VOGDB. Since IMG/VR is not a VOG resource we cannot compare the overlap of assigned VOGs and have to deviate to other ways of comparison, like clustering metrics.

2.7.3 Comparing Clusterings

To compare assignments by DeepNOG and VOGDB HMMs on a dataset that is not a VOG resource, we utilize the assignments of the VOGDB HMMs as the ground truth and the DeepNOG assignments as the predictions of a clustering problem. We then compute two clustering performance measures, namely the Rand Index [23] and the Mutual Information [54] metric. These clustering performance measures evaluate how similar the separations of the data are between the predictions and the ground truth. In other words, they measure the agreement between two clusterings, ignoring permutations.

Rand Index

The adjusted Rand Index obtains a value close to 0 for a random assignment and a value of 1 in the case of a perfect agreement. With C as the ground truth and K as the

clustering we define

- a as the count of pairs of elements that are in the same set in C and K
- b as the count of pairs of elements that are in different sets in C and K
- $C_2^{n_{samples}}$ as the count of all possible pairs

Then the (unadjusted) Rand Index (RI) is defined by

$$\text{RI} = \frac{a+b}{C_2^{n_{samples}}}$$

The adjusted Rand Index (ARI) corrects the RI for chance by discounting the expected RI $E[RI]$ of random assignments and is defined by

$$\text{ARI} = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]}$$

Mutual Information

For a random assignment, adjusted Mutual Information scores close to 0. In case of a perfect agreement, a score of 1 is obtained. In this measure, the concept of entropy is used as the amount of uncertainty of a set of partitions. The entropy of an assignment U is defined by

$$H(U) = -\sum_{i=1}^{|U|} P(i)\log(P(i))$$

with $P(i)$ being the probability that a randomly picked item from U falls into the class U_i. The Mutual Information (MI) between two assignments U and V is then defined by

$$\text{MI}(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|}\frac{|U_i \cap V_j|}{N}\log\left(\frac{N|U_i \cap V_j|}{|U_i||V_j|}\right)$$

witch $P(i,j)$ being the probability that a randomly picked item falls into class U_i and V_j. Similar to the ARI, the adjusted Mutual Information (AMI) score corrects for chance by discounting the expected MI $E[MI]$ and is defined by

$$\text{AMI} = \frac{\text{MI} - E[\text{MI}]}{\text{mean}(H(U), H(V)) - E[\text{MI}]}$$

3 Results

3.1 Data Preparation

In this section, we will highlight the characteristics of the intermediate and final datasets, including any supplementary data that was used in the data preparation step. In the first step, we extract all sequence-IDs and sequences from VOGDB that are assigned to a VOG. The release *vog208* from November 8, 2021, was used. The resulting sequence-ID to VOG mapping yields 434,836 unique entries. In the resulting FASTA file the number of sequences matches this number exactly.

3.1.1 Group Splits

Training a model from a completely randomized stratified split of this dataset would introduce unwanted over-fitting effects. By using a group split on a third-party label we achieve a higher balance between in-model and out-of-model VOGs, which prevents over-fitting, as it addresses the weaknesses of the dataset. To perform the group splits on a third-party label, the cluster labels of UniRef100, UniRef90 and UniRef50 were used. See table 3.1 for statistics of the latest release *2021_03* for the different sizes of the datasets and the distributions of clusters according to size.

DeepNOG used a 96%/2%/2% training, validation and test split for the eggNOG data. For VOGDB data we decided to stay with these percentages. These values already focus on the maximum usage of the data for training which is, in the VOGDB case, even more important, since the dataset is smaller than e.g. the eggNOG bacteria level. To

	# Clusters	**Singleton clusters**	**<5 members clusters**
UniRef100	277,482,692	91%	99%
UniRef90	135,301,051	71%	93%
UniRef50	48,531,432	63%	87%

Table 3.1: Properties of the latest UniRef release *2021_03* for all three levels of resolution. Number of clusters per dataset and fractions of singleton clusters (i.e. clusters of size 1) and clusters with less than 5 members.

ensure the occurrence of the wanted effects of using a group split over a stratified split, we investigated the overlap of VOGs between the training, validation and test split for both splitting techniques. In the stratified split, the VOGs of the training set strongly overlap with the VOGs from the validation an test set. As we can see in figure 3.1 in the stratified split, the training set includes nearly all VOGs and there is also substantial overlap between the test and validation set. Both of these effects would clearly increase over-fitting. With the group splits we manage to set aside 10 to 50 times the number of VOGs solely for the validation and test set, which we consider an acceptable result, that will clearly aid the training process later.

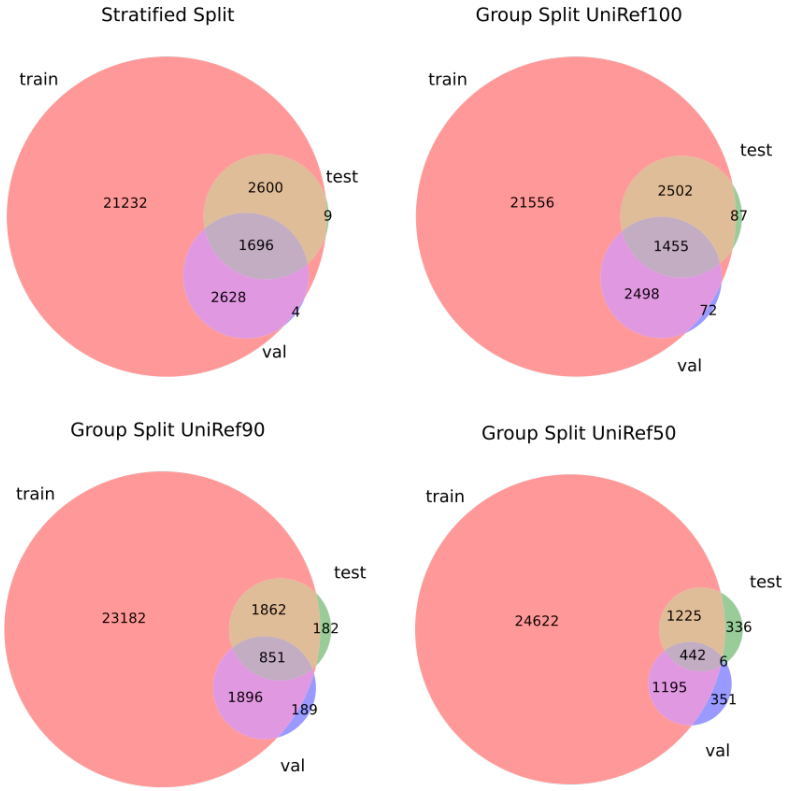

Figure 3.1: Comparison of stratified split and group splits for different UniRef resolutions. In the case of the stratified split, the training set includes nearly all VOGs and there is also a substantial overlap between the test and validation set. In the case of the group splits, a significant portion of VOGs is not included in the training set.

Threshold	# VOGs	% VOGs	# Sequences	% Sequences
all (no threshold)	28,210	100%	434,836	100%
10+	6,408	23%	357,341	82%
100+	704	3%	204,450	47%

Table 3.2: Properties of the datasets resulting from applying membership thresholds. Depicting a nonlinear reduction of the number of VOGs and sequences in the threshold datasets.

3.1.2 Member Thresholds

To investigate the effects of low vs. higher class cardinalities we extracted two subsets from the dataset where we limit us to VOGs that surpass certain membership thresholds. For a subset where we only select VOGs that have at least 10 members, the number of VOGs and sequences in the dataset reduces greatly. See table 3.2 for details on the resulting dataset and the implications of thresholds on the number of VOGs and sequences, as well as figure 3.2 for a comparison of the VOG size distributions. For a 100+ members threshold, the dataset reduces further, with only 3% of VOGs remaining while still having about half of the sequences. In this case, the machine learning problem is decisively simplified, so we expect an improved performance over models trained from the no threshold or 10+ members threshold datasets.

3.1.3 Final Datasets

If we combine all member thresholds with all UniRef resolutions, we get a total of 9 different datasets. A separate model will be trained on all of these datasets, see table 3.3 for the final number of VOGs included in the training, validation and test.

3.2 Training Process

In this section, we will cover the outcomes of monitoring the training process, as well as evaluating different hyperparameter configurations. For the most part, we will investigate the accuracies of the models, i.e. the progression of accuracy in the training itself and the variation of accuracies on all datasets for every hyperparameter configuration. It is important to note, that the performance of the model during and at the end of the training process must not have direct implications on the assignment performance, at this point of the machine learning process, it can just be seen as some kind of orientation.

UniRef	**Threshold**	**# Train VOGs**	**# Validation VOGs**	**# Test VOGs**	**# Retained VOGs**	**% Retained VOGs**
100	all	28,011	4,025	4,044	159	0.6%
100	10	6,406	2,575	2,599	1	0.02%
100	100	704	643	647	0	0%
90	all	27,791	2,936	2,895	371	1.3%
90	10	6,396	1,722	1,760	12	0.2%
90	100	704	493	492	0	0%
50	all	27,484	1,194	2,009	693	2.5%
50	10	6,364	1,120	1,137	42	0.7%
50	100	703	348	341	1	0.1%

Table 3.3: Counts of the included VOGs for all parts of the split for every dataset. In addition, the number of retained VOGs, i.e. VOG labels that are not included in the training split, is shown.

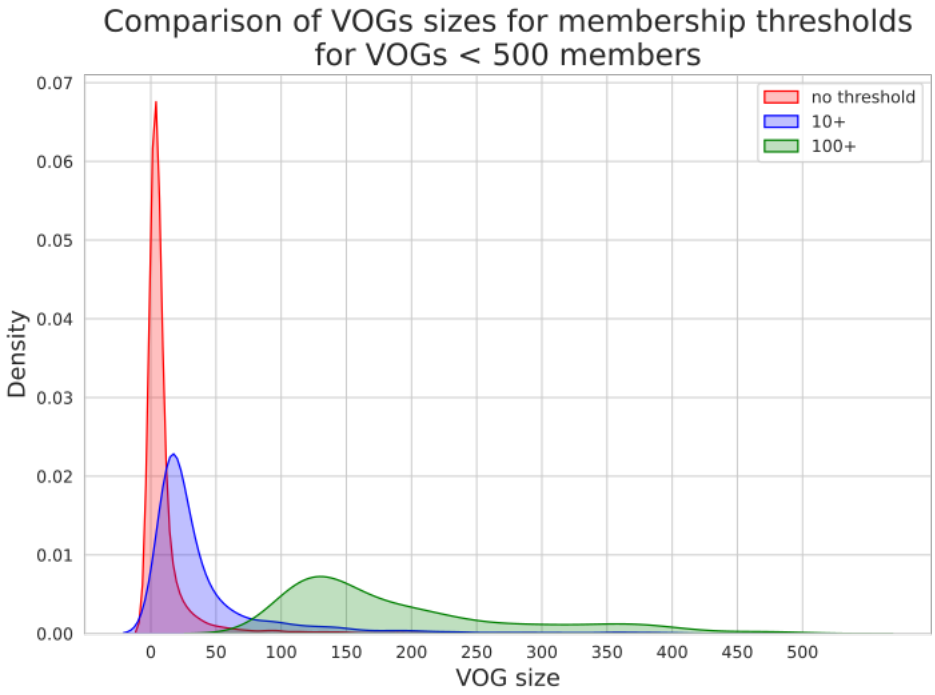

Figure 3.2: Density estimations of the VOG sizes for the membership threshold datasets.

3.2.1 Learning Curves

DeepNOG reports the accuracy and loss (i.e. the error of the model) on the training and validation set for every epoch. Comparing the progression of these measures for training and validation offers us a way of evaluating the training process quickly. Having all the training progressions side-by-side gives us the opportunity to spot-check for possible changes in the hyperparameter configuration, figures 3.3 and 3.4 depict such comparisons.

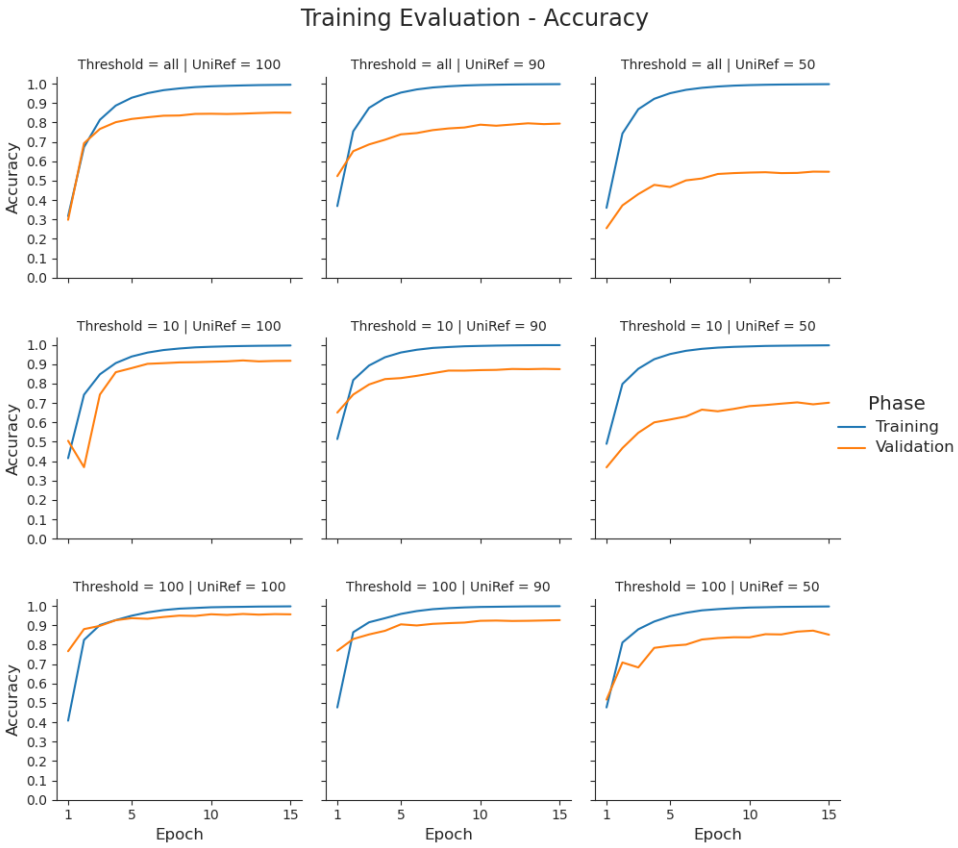

Figure 3.3: Progression of model accuracy on training and validation set over epochs. In an ideal setting, the curves for training and validation will converge to each other and then stay close to each other while having a high accuracy outcome. This would then mean that the model is not or only slightly overfitted. As we can see here this is only the case for some of the datasets, which was expected. The strongly reduced sets for the 100+ thresholds (bottom row) are easier to learn. The same goes for sets that were split on the UniRef100 labels (left column), where fewer VOGs are retained to the validation and test set.

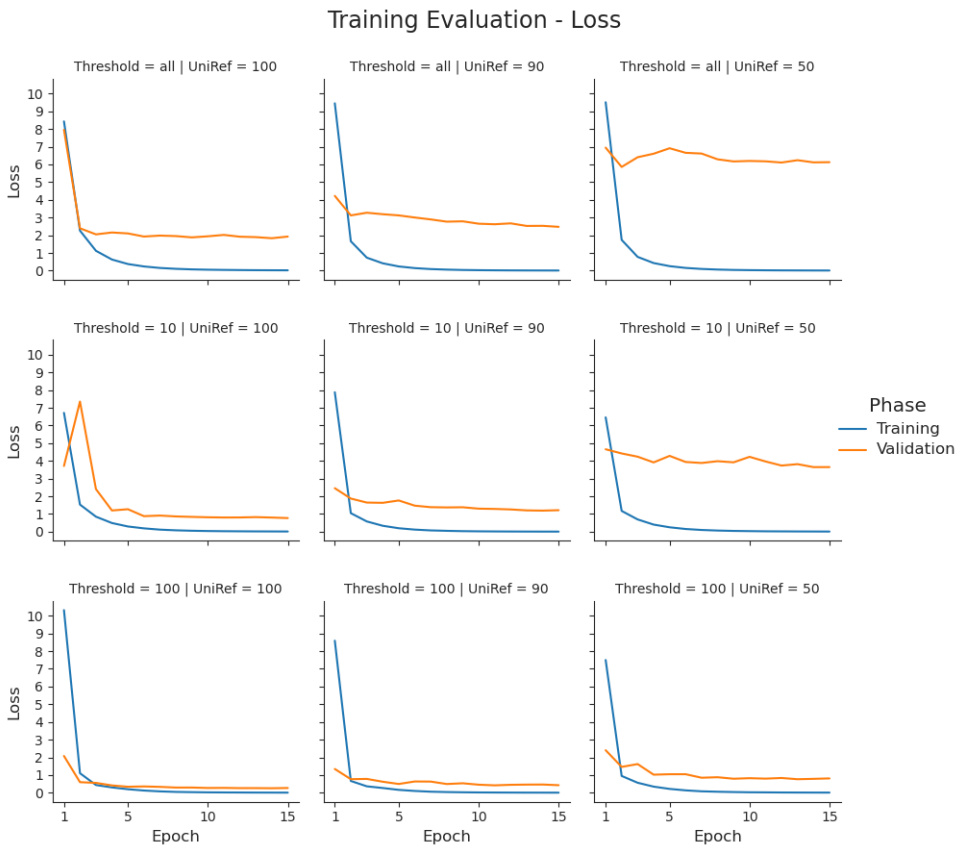

Figure 3.4: Progression of the error of the model on the training and validation set over epochs. Similar to the accuracy curves, we want the two curves to converge to each other and in this case of an error measure to go down and stay down.

3.2.2 Hyperparameter Optimization

To investigate the possibility of improving the training process we started out by tuning single hyperparameters to get an initial picture of where there might be improvements possible. In a second round, we then evaluated combinations of tuned parameters. This resulted in the following deviations from the default hyperparameter configurations:

1. First round of tuning
    - Dropout rate 0.4
    - Dropout rate 0.5
    - Dropout rate 0.6
    - Dropout rate 0.7

- Dropout rate 0.8
- Batch size 8
- Batch size 16
- Batch size 32
- Batch size 128
- Learning rate 0.005

2. Second round of tuning
    - Epochs 30, Learning rate decay 0.375
    - Dropout rate 0.4, batch size 16
    - Dropout rate 0.4, batch size 128
    - Dropout rate 0.6, batch size 16
    - Dropout rate 0.6, batch size 32

The hyperparameter mostly investigated was the dropout rate, which was obvious since the learning curves indicated over-fitting for most datasets and this is the hyperparameter that directly influences the generalization. Smaller batch sizes were investigated because of the concept of mini-batches [33], the higher batch size of 128 was investigated also for influences on generalization. For tuning the hyperparameters that influence the optimizer, learning rate and decay, little room for lowering were available, since the default values are already very low. For example, tuning the learning rate decay was only possible when doubling the number of epochs, since the learning process became unstable for the default of 15 epochs. As tuning dropout rate and batch size looked promising, multiple combinations were tested. In figure 3.5 we can see a comparison of all hyperparameter configurations that were tested. Only a few configurations have an improved or slightly similar accuracy as the default configuration. Unfortunately, changes of the hyperparameters often led to a higher variation in the accuracies, which questions the applicability of these configurations.

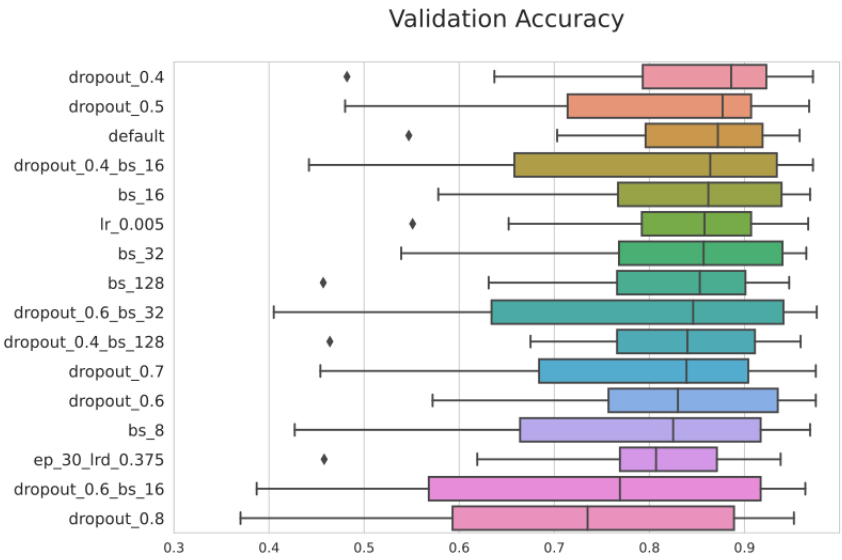

Figure 3.5: Boxplot of achieved validation accuracies on the 9 datasets for every hyperparameter configuration, ordered by mean accuracy. As we can see, only a few configurations come close to the performance of the default parameterization. Further, we see that changes of the hyperparameters often also introduce more variation in the accuracies.

3.3 Assignment Performance

To get a clear picture of the true capabilities of the trained models, we analyze the different models and hyperparameter configurations. We start with evaluating the performance of the test sets, followed by a comparison with the performance that the VOGDB HMMs have on these test sets. In the last step, we also compare DeepNOG and HMMs on two subsets of external datasets, to get an assessment towards real-world applications.

3.3.1 Assignments of the Test Set

To evaluate the performance of the trained models for the test split, we supply the truth labels when running the DeepNOG assignment module. With this DeepNOG is able to calculate and report the previously mentioned performance metrics, see table 3.4 for the performance of the model with the default parameterization.

As the evaluation of the training process previously predicted, only a few hyperparameter configurations should be able to raise the model's performance. In figure 3.6 we see the reported accuracies for all 9 datasets for every hyperparameter configuration on the

UniRef	Threshold	Accuracy	Macro Precision	Macro Recall	Macro F1	Micro Precision	Micro Recall	Micro F1	MCC
100	all	0.83	0.71	0.73	0.71	0.83	0.83	0.83	0.83
100	100	0.96	0.94	0.93	0.93	0.96	0.96	0.96	0.96
100	10	0.91	0.84	0.84	0.84	0.91	0.91	0.91	0.91
90	all	0.77	0.56	0.59	0.56	0.77	0.77	0.77	0.76
90	100	0.94	0.90	0.89	0.88	0.94	0.94	0.94	0.94
90	10	0.86	0.76	0.76	0.75	0.86	0.86	0.86	0.86
50	all	0.50	0.23	0.25	0.22	0.50	0.50	0.50	0.50
50	100	0.85	0.73	0.71	0.70	0.85	0.85	0.85	0.85
50	10	0.63	0.40	0.41	0.39	0.63	0.63	0.63	0.63

Table 3.4: Performance measures for assignments for all 9 test sets with the default model parameterization.

test set. As expected, most parameterizations have lower performance than the default settings. Also, like in the training process, the tuned hyperparameters introduce more variation in the reported accuracies.

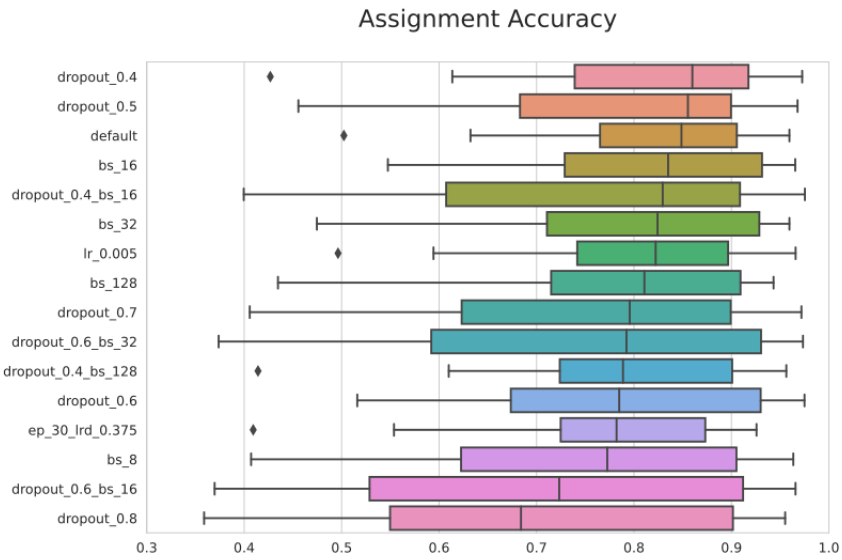

Figure 3.6: Boxplot of achieved assignment accuracies on the 9 datasets for every hyperparameter configuration, ordered by mean accuracy. Again, only a few configurations come close to the default parameterization and often more variation in accuracy is introduced.

Besides the ability of making accurate VOG assignments, it is crucial to analyze the fraction of high confidence assignments the models are able to make, see table 3.5 for details on high confidence assignments for the default configuration. In figure 3.7 we can see the tuned models, that were able to raise the models accuracy as well as raising the number of high confidence assignments. The cutoff value for an assignment being of high confidence was set to c $>=$ 0.95.

UniRef	Threshold	# Assignments	# High conf. assignments	% High conf. assignments
100	all	6,609	5,480	83%
100	100	2,518	2,369	94%
100	10	5,028	4,525	90%
90	all	6,536	5,082	78%
90	100	2,452	2,238	91%
90	10	5,377	4,549	85%
50	all	6,490	3,722	57%
50	100	2,214	1,805	82%
50	10	4,663	3,095	66%

Table 3.5: Statistics on assignment confidence for all 9 datasets for the default model parameterization.

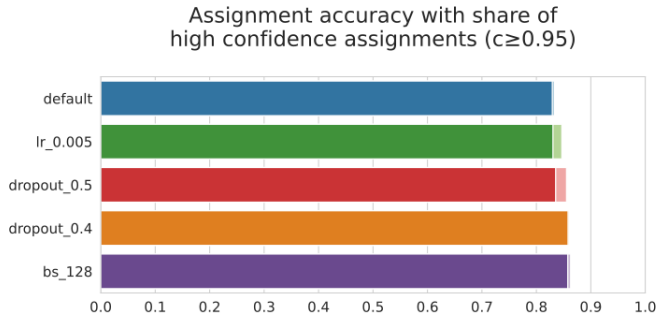

Figure 3.7: Top-performing hyperparameter configurations in comparison with the default configuration (UniRef100, no member threshold). Raising the models accuracy doesn't necessarily improve the number of high confidence assignments. Also, these configurations have different training runtimes underlying, e.g. reducing the learning rate increases execution time, while raising batch size decreases it.

3.3.2 Comparison with VOGDB HMMs

We used hmmscan from the HMMER suite to search every sequence from the test set against the collection of HMMs from VOGDB. The resulting output file lists all the sequences and their hits in the HMM database using an E-value to assess the quality of that hit. To get the actual assignment we retrieve the VOG with the lowest E-value for every sequence. Similar to the confidence threshold used for the DeepNOG assignments, we only count an assignment as high confident if it has an E-value < 0.00001. In figure 3.8 we can see a performance comparison of VOGDB HMMs and DeepNOG on the test

set in terms of accuracy and number of high confidence assignments. On the complete dataset, the performance of DeepNOG is lower than the performance of the HMMs, also in terms of high confidence assignments. For the reduced datasets DeepNOG is able to raise the performance and the number of high confidence assignments, surpassing HMMs in the case of the 100+ members threshold dataset.

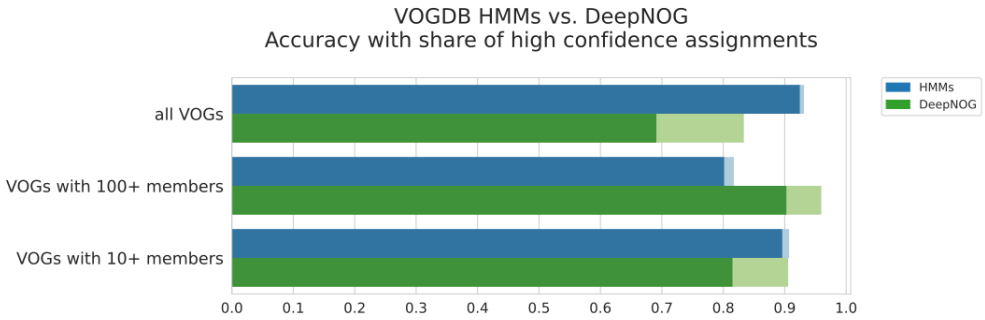

Figure 3.8: Comparison of VOGDB HMMs and DeepNOG assignment performance on the UniRef100 test set. For the complete dataset, the HMMs surpass the performance of DeepNOG greatly, both in accuracy and number of high confidence assignments. For the 10+ members threshold dataset, DeepNOG is able to gain performance and in the case of the 100+ members threshold dataset, it achieves higher performance than the HMMs.

To investigate the reduction in runtime we ran an experiment where we assigned the sequences of the UniRef100 no member thresholds test set for different setups of DeepNOG and HMMER. DeepNOG was once run on a CPU (single-core) and once on a GPU. HMMER was run in a single-core setup, as well as a parallel setup with 32 cores. Every run was repeated 3 times and the average number of seconds for processing 1000 sequences was reported. To investigate a connection between sizes of datasets, the whole experiment was repeated with the 100+ members threshold dataset, see figure 3.9 for results of these experiments.

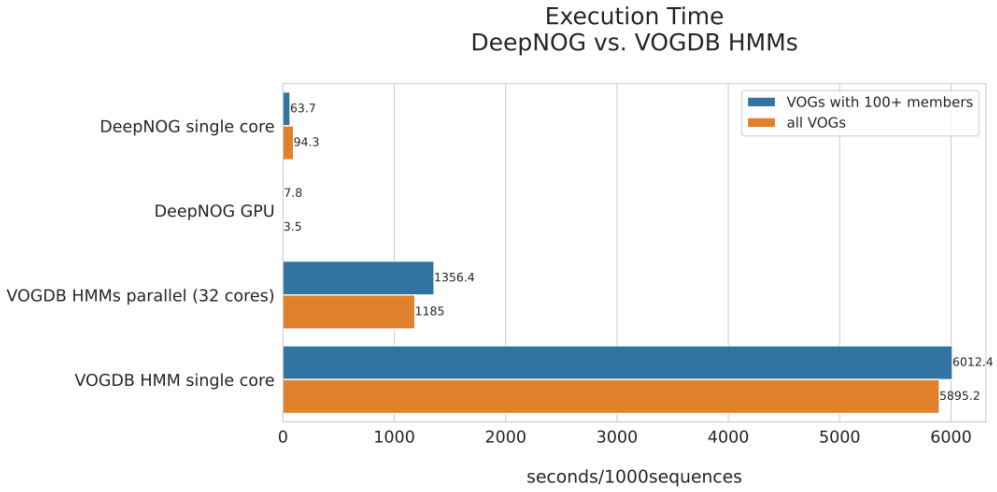

Figure 3.9: Comparison of execution time for DeepNOG and VOGDB HMMs in seconds per 1000 sequences. DeepNOG manages to be faster than the HMMs by a 100-fold when run on a CPU and a striking 1000-fold when run on a GPU. Interesting to see is that the size of the dataset has no noteworthy influence on the execution time.

In the last comparison, we looked at the needed memory in the assignment processes of DeepNOG and the VOGDB HMMs. The HMM database created by HMMER needs to be loaded in the memory as a whole for the entire assignment process, this database has an approximate size of 4.2 GB. A DeepNOG model needs about 120 MB to load the weights for making assignments, which is substantially smaller.

3.3.3 Comparison on External Datasets

We used subsets of the pVOG and IMG/VR datasets to compare DeepNOG and the VOGDB HMMs further. Both datasets are very different in terms of size, actuality and in their contemplated use cases. pVOG constitutes a VOG resource similar, but not as extensive as VOGDB. IMG/VR is a massive resource for uncultivated viral genomes (UViGs).

pVOG Subset

We randomly picked 16 VOGs from the pVOG dataset with 300-500 members each. This resulted in a set of 5885 sequences for VOG assignment. The redundancy of these sequences in VOGDB was analyzed for 0.1, 0.2 and 0.3 levels of difference in sequence identity and resulted in 98% of pVOG sequences being already included in VOGDB. Both

DeepNOG and the HMMs were able to get a high number of high confidence assignments for these sequences (DeepNOG 97%, HMMs 99%). In addition, we analyzed the similarity of the DeepNOG and HMMs assignments for agreement, figure 3.10 shows the overlap of these assignments.

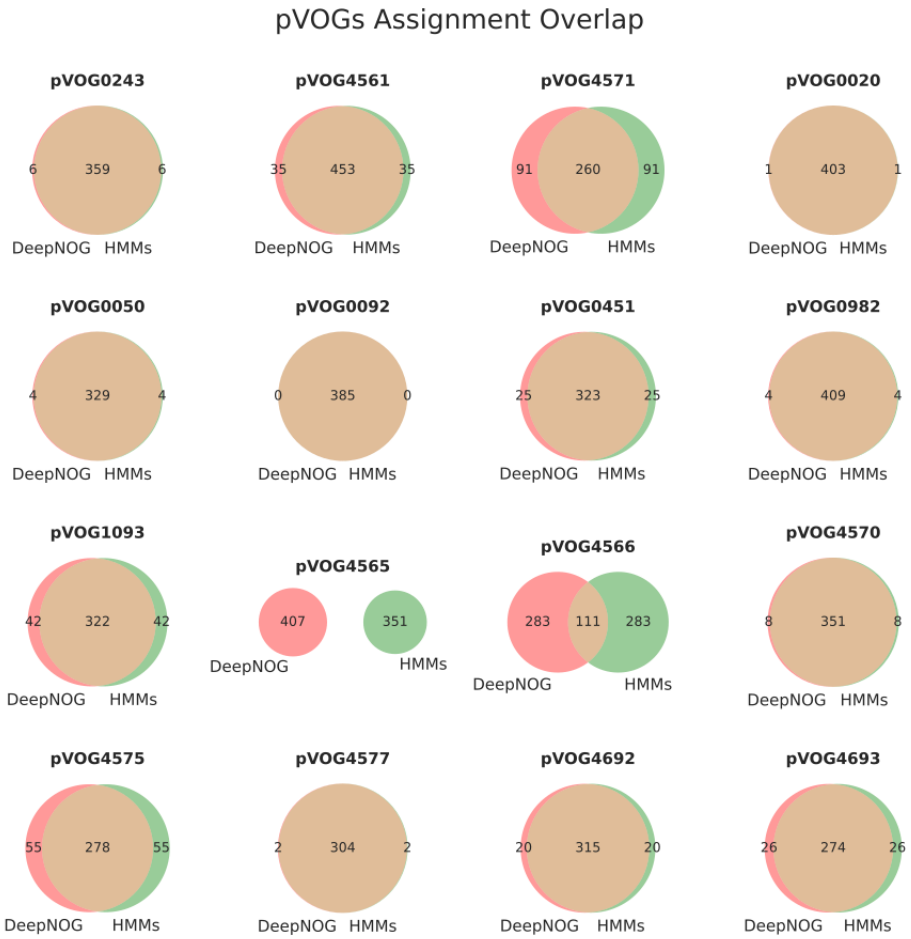

Figure 3.10: Overlap of DeepNOG and HMMs high confidence assignments. Both methods agree on 14 out of the 16 pVOGs.

UViG Subset

We randomly picked 10,000 sequences from the IMG/VR dataset, with 2-4% redundancy in VOGDB, which was expected since UViGs are not included in RefSeq. DeepNOG managed to assign 27% of sequences with high confidence, the HMMs managed to get 39%. Since IMG/VR is not a VOG resource we analyze the similarities between the

assignments by taking them as the result of a clustering procedure. The computed ARI of 0.2 indicates poor agreement between the two assignments. The computed AMI of 0.5 indicates some low-level form of agreement. To check for the possible case that the low confidence assignments all belong to similar families, we ran a quick check, where we used BLAST to retrieve annotations for 20 of these sequences from the UniProt Knowledgebase. This check returned a diverse set of annotations, which indicates that the sequences with the low confidence assignments belong to VOGs that have not yet been incorporated into VOGDB and therefore the HMMs and DeepNOG are not able to classify them correctly.

3.4 Deployment

To integrate a DeepNOG model trained from VOGDB data into the VOGDB landscape, we need a pipeline that trains a new model for every VOGDB release. This pipeline was implemented with the help of the Nextflow workflow manager and resulted in a workflow that consists of 3 processes that run locally and 5 processes that make use of the LiSC (2 of them need a GPU). The output of the pipeline is one DeepNOG model which is trained on 99% of the data with 1% for validation, a test split is not needed and therefore omitted. The dataset for training has no threshold on VOG sizes and UniRef100 cluster labels are used for performing the group split. Further, the pipeline outputs a small report on the training process, including training and validation accuracies, the set of class labels and plots for the learning curves. Three of the processes have a runtime of more than an hour. The VOGDB to UniRef100 alignment takes approximately 10h, the group split about 1h and the training about 3h (all processes ran on multiple CPUs or GPUs). Figure 3.11 depicts the execution plan of the Nextflow workflow with a directed acyclic graph.

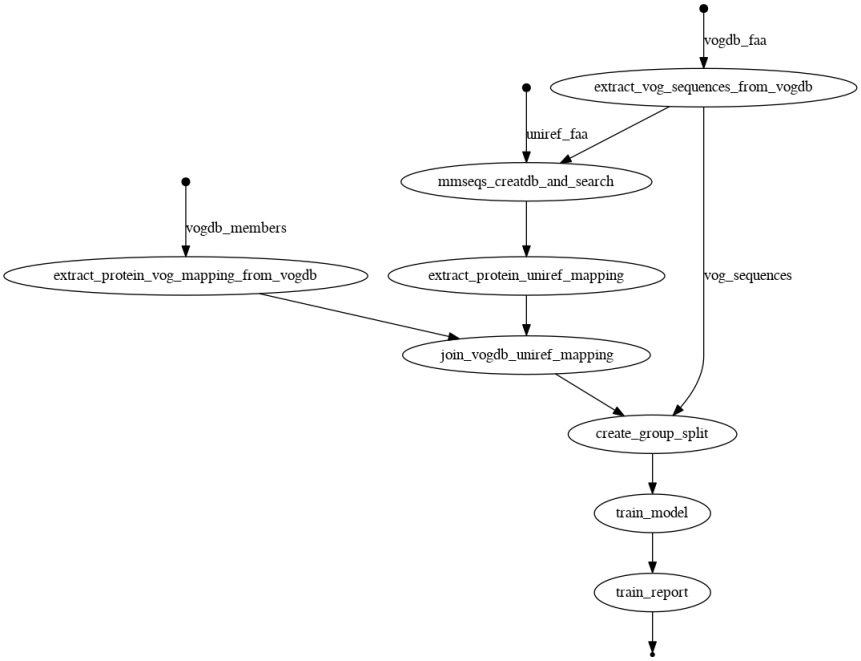

Figure 3.11: Execution plan of the production workflow for retraining DeepNOG models for every new VOGDB release. Ellipses depict processes, dots with arrows downwards depict input files.

4 Discussion

In this thesis, we investigated if a DeepNOG model can be trained from VOGDB data to make accurate and high confidence VOG assignments, similar to the HMMs from VOGDB. Further, we evaluated the possibilities of tuning the DeepNOG model by deviating from the default neural network parameterization. In the following sections, we will further mention possible investigations considering this work, as well as some problems and shortcomings of the VOGDB dataset, the DeepNOG implementation and the resulting pipeline itself.

4.1 Challenges of the VOGDB Data

Multiple steps have been taken to prepare the data in accordance with the input requirements of DeepNOG and to challenge the training process in a way that overfitting is reduced as much as possible. Here we want to address two problems that accompanied the development. The first problem considers the huge amount of classes and the distribution of VOG sizes, where we have mostly very small VOGs, that barely fulfill the requirements to be used in such a massively multiclass classifier. To investigate this problem, we created subsets where we discarded small VOGs from the dataset for two levels of membership. This approach could be considered rough, since it for example ignores the fact that there surely are VOGs in the dataset from which training almost never makes sense, due to the lack of enough sequences or the properties of these sequences (e.g. length, similarity to each other) and the VOG itself (e.g. closeness to other VOGs). Another thing worth investigating would be the effects of removing the few really large VOGs from the model and maybe training them in a separate model. The second problem considers the characteristics of protein sequences, because sequences are to some extent almost always similar to each other. Any machine learning approach that wants to process sequences must find a way to incorporate, let's say, just the right amount of similarities between sequences, to succeed. In this work, these problems also transfer to the VOGs and we think that incorporating the closeness of VOGs to each other in the class space would lead to better results and more applicability in a real-world scenario.

4.2 Strengths and Weaknesses of DeepNOG

In this work, we showed that DeepNOG is already highly optimized towards the use case of orthologous group assignment. Due to the possible output of several performance measures, the evaluation of a DeepNOG model can be done in a comprehensive way. DeepNOG modules for training and assignment are comfortable to use and utility functions inside the DeepNOG software package are well documented and accessible. To train a neural network we process batches of sequences, after each batch, the model weights are updated. If we have a structure in the sequence file for training, then it becomes possible that all the sequences of a batch belong together (in this scenario the same VOG) which could lead to a biased update which then enables overfitting. DeepNOG has a feature to shuffle the input sequences, but this shuffling procedure is limited to a rather small shuffle buffer of 2^{16} sequences which doesn't ensure that all sequences are picked randomly. To overcome this shortcoming, any user of DeepNOG needs to shuffle the data before processing it with DeepNOG. This creates an odd situation since the user is then required to be familiar with randomization and shuffling procedures. Therefore, we propose that the shuffling feature has to be improved in the DeepNOG software. Another problem of DeepNOG comes from the strong optimization of DeepNOG towards GPUs, e.g. the default batch size of 64. When training on a CPU, the sequences are processed one by one, which constitutes a batch size of 1, so the training process is essentially different. We think that this could reduce applicability and therefore needs to be investigated further.

4.3 Potential for a Real-World Application

Here we consider the use of DeepNOG for VOG assignment in real-world scenarios where e.g. resources are limited or knowledge about deep learning approaches is little. In the following sections, we will mention the requirements needed for a successful application of our approach. In addition to that, we will discuss potential improvements.

4.3.1 Overfitting

VOGDB is already a resource of predicted information, which makes DeepNOG trained on VOGDB data, a machine learning model that makes predictions of predictions. The set of VOGs in VOGDB is not the complete set of VOGs and there is the possibility of far more VOGs out there. In this work, we showed this fact by making assignments for sequences from UViGs, which resulted in a high number of low confidence assignments for both DeepNOG and the VOGDB HMMs. We can also see this in terms of overfitting

when we look at the learning curves, where we see that every of the 9 trained models is overfitted. Sure, in some cases, the overfitting is only small, but we have to remember that we already work with a model optimized for OG prediction and that the dataset has undergone several adjustments to reduce overfitting.

4.3.2 Production Pipeline

The number of viral protein sequences added to each RefSeq release is not always the same. Also, the increase of VOGs and the creation of new VOGs in VOGDB doesn't follow a linear pattern. If we train a DeepNOG model in accordance with every VOGDB release, we might face different scenarios, which could influence the training process. To keep track of things, we need to find a way of comparing the latest model to previous ones. Due to the needed alignment of VOGDB against UniRef100, the data preparation step becomes the most computationally expensive and long-running part of this pipeline. To address this imparity, we could investigate an approach where we only consider the new sequences of the VOGDB release for the alignment and use the UniRef labels from mappings of previous releases.

5 Conclusion and Outlook

In previous sections, we showed that DeepNOG can be trained with VOGDB data to get a very fast method for VOGs assignment, that has similar accuracy when compared to HMMs, which are the state of the art. Deviations from the default hyperparameters were investigated and allowed only for marginal improvements of accuracy and number of high confidence assignments. Further, we investigated the agreement of DeepNOG and HMMs on seen and unseen data. For data that already is included in VOGDB, both methods strongly agree in the made assignments, which is only barely the case for unknown data. To conclude this work we give an overview of the final model statistics trained from the latest VOGDB release *vog208*, see table 5.1 for details on the latest model created with the production pipeline. In addition, we will present some ideas on tackling the problems of the mentioned weaknesses of the dataset and DeepNOG, as well as an outlook of possible impacts on our use case from recent developments in the field of protein structure prediction.

5.1 Data Augmentation

A possible way of reducing the problem of low class cardinalities in the VOGDB dataset would be the addition of synthetic sequences to small VOGs. The idea is to take the sequences of a small VOG and simulate slightly different sequences from them, while preserving the characteristics that define the VOG membership of course. This would allow to augment small VOGs and thereby raise the class cardinalities of the dataset. A first

Number of classes	28,148
Number of classes retained	62
Accuracy (validation)	91.6 %
Execution time	CPU: 70 s/1000seqs GPU: 6 s/1000seqs
Memory consumption	131 MB

Table 5.1: Properties of the latest model built from VOGDB release vog208.

and very simple approach would be to randomly make substitutions in the sequences and then checking if these substitutions still manifest conservative substitutions. This could be achieved by checking for low differences in the BLAST positive scores (e.g. $p = 0.1$), which would ensure that the substitutions are conservative and the VOG membership is thereby not violated. A second, more sophisticated approach comes from recent developments in the field of data augmentation of sequence models [42], where the models are trained on the unaugmented dataset first and then synthetic sequences are added in a semi-supervised manner. For this approach, the authors defined a replacement dictionary that models the chemical properties of amino acids and makes substitutions in accordance with a probability parameter. This parameter constitutes a new hyperparameter of the model and is then assessed on how it affects the downstream assignments.

5.2 Learning Rate Schedulers

DeepNOG uses a very simple approach to change the learning rate throughout the training process. In this so-called time-based schedule the learning rate for the next epoch is lowered by a predefined fraction (learning rate decay) applied to the learning rate of the current epoch. This means that the rate is gradually lowered and the optimizer uses smaller step sizes as the training progresses. Other rather simple approaches that can be used in training neural networks include reducing the learning rate by fixed steps or with exponential decay, thereby gaining control over the learning rate in later epochs. More sophisticated approaches, called adaptive learning rate methods, make use of heuristics or utilize callbacks to performance measures to decide on how to adjust the learning rate. Heuristic approaches introduce multiple new parameters which of course makes subsequent hyperparameter optimization more challenging. Pytorch, the deep learning framework used by DeepNOG, offers a learning rate scheduler called ReduceLROnPlateau which allows for callbacks. The scheduler accesses a metric (e.g. accuracy) and reduces the learning rate strongly if little or no improvements are seen over a predefined number of epochs. Adding this feature to DeepNOG would surely improve the training experience.

5.3 Implications of AlphaFold

Published in mid 2021, AlphaFold is the first computational approach that is able to predict protein structures close to experimental accuracy [25]. Behind these improvements is a new type of neural network architecture and also a training procedure that incorporates the evolutionary and physical constraints of protein structures. Recently, AlphaFoldDB

was released [52], covering structure predictions for the proteomes of human and 20 other model organisms. For the protein folding problem, this is a game-changer and AlphaFold is predicted to have a major impact on research in life sciences and medicine in the future. Similar to the biomedical consequences, the implications for studies of molecular evolution could just be as profound [3], making structural biology become part of the tools used in evolutionary research. In molecular evolutionary biology, the attention is usually set to proteins where a small change in the sequence will lead to a different function. This may limit the applicability of AlphaFold, since it is based, among other things, on the assumption that similar sequences assemble similar structures. The expected range of possibilities for evolutionary research offered by AlphaFold is currently set very broad, ranging from making good predictions at the family level to predicting functional effects of mutations or even the ability to reconstruct ancestral structures.

Abbreviations

- AMI - Adjusted Mutual Information
- ARI - Adjusted Rand Index
- AUC - Area Under the Receiver Operating Characteristic Curve
- CNN - Convolutional Neural Network
- CSV - Comma-separated Values
- CPU - Central Processing Unit
- DNA - Deoxyribonucleic Acid
- FNN - Feedforward Neural Network
- GPU - Graphics Processing Unit
- HGT - Horizontal Gene Transfer
- HMM - Hidden Markov Model
- LCA - Last Common Ancestor
- LUCA - Last Universal Common Ancestor
- MCC - Matthews Correlation Coefficient
- MI - Mutual Information
- MSA - Multiple Sequence Alignment
- NGS - Next-generation Sequencing
- OG - Orthologous Group
- RI - Rand Index
- RNA - Ribonucleic Acid
- TSV - Tab-separated Values
- UViG - Uncultivated Viral Genome
- VOG - Virus Orthologous Group

List of Tables

List of Figures

Listings

References

[1] LiSC - Life Science Compute Cluster, University of Vienna, https://cube.univie.ac.at/lisc, Accessed Dec 2021.

[2] VOGDB: Virus orthologous groups, Universitiy of Vienna, http://vogdb.org, Accessed Dec 2021.

[3] Structural revolution? *Nature Ecology & Evolution*, 5(1):1–1, Jan 2021.

[4] A. M. Altenhoff, N. M. Glover, C.-M. Train, K. Kaleb, A. Warwick Vesztrocy, D. Dylus, T. M. de Farias, K. Zile, C. Stevenson, J. Long, H. Redestig, G. H. Gonnet, and C. Dessimoz. 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, Jan. 2018.

[5] J. A. Campillo-Balderas, A. Lazcano, and A. Becerra. Viral genome size distribution does not correlate with the antiquity of the host lineages. *Frontiers in Ecology and Evolution*, 3:143, 2015.

[6] M. N. Davies, A. Secker, A. A. Freitas, M. Mendao, J. Timmis, and D. R. Flower. On the hierarchical classification of G protein-coupled receptors. *Bioinformatics*, 23(23):3113–3118, Dec. 2007.

[7] E. S. Deutekom, B. Snel, and T. J. P. van Dam. Benchmarking orthology methods using phylogenetic patterns defined at the base of eukaryotes. *Brief. Bioinform.*, 22(3), May 2021.

[8] S. Duffy. Why are RNA virus mutation rates so damn high? *PLoS Biol.*, 16(8):e3000003, Aug. 2018.

[9] R. A. Edwards and F. Rohwer. Viral metagenomics. *Nat. Rev. Microbiol.*, 3(6):504–510, June 2005.

[10] G. F. Ejigu and J. Jung. Review on the computational genome annotation of sequences obtained by next-generation sequencing. *Biology (Basel)*, 9(9), Sept. 2020.

[11] S. El-Gebali, J. Mistry, A. Bateman, S. R. Eddy, A. Luciani, S. C. Potter, M. Qureshi, L. J. Richardson, G. A. Salazar, A. Smart, E. L. L. Sonnhammer, L. Hirsh, L. Paladin, D. Piovesan, S. C. E. Tosatto, and R. D. Finn. The pfam protein families database in 2019. *Nucleic Acids Res.*, 47(D1):D427–D432, Jan. 2019.

[12] H. ElAbd, Y. Bromberg, A. Hoarfrost, T. Lenz, A. Franke, and M. Wendorff. Amino acid encoding for deep learning applications. *BMC bioinformatics*, 21(1):1–14, 2020.

[13] G. Fang, N. Bhardwaj, R. Robilotto, and M. B. Gerstein. Getting started in gene orthology and functional analysis. *PLoS Comput. Biol.*, 6(3):e1000703, Mar. 2010.

[14] R. Feldbauer, L. Gosch, L. Lüftinger, P. Hyden, A. Flexer, and T. Rattei. DeepNOG: fast and accurate protein orthologous group assignment. *Bioinformatics*, 36(22-23):5304–5312, 12 2020.

[15] R. Feldbauer, F. Schulz, M. Horn, and T. Rattei. Prediction of microbial phenotypes based on comparative genomics. *BMC Bioinformatics*, 16 Suppl 14(S14):S1, Oct. 2015.

[16] R. D. Finn, J. Clements, and S. R. Eddy. HMMER web server: interactive sequence similarity searching. *Nucleic Acids Res.*, 39(Web Server issue):W29–37, July 2011.

[17] M. Y. Galperin, D. M. Kristensen, K. S. Makarova, Y. I. Wolf, and E. V. Koonin. Microbial genome analysis: the COG approach. *Brief. Bioinform.*, 20(4):1063–1070, July 2019.

[18] M. Y. Galperin, Y. I. Wolf, K. S. Makarova, R. Vera Alvarez, D. Landsman, and E. V. Koonin. COG database update: focus on microbial diversity, model organisms, and widespread pathogens. *Nucleic Acids Res.*, 49(D1):D274–D281, Jan. 2021.

[19] N. Glover, C. Dessimoz, I. Ebersberger, S. K. Forslund, T. Gabaldón, J. Huerta-Cepas, M.-J. Martin, M. Muffato, M. Patricio, C. Pereira, A. S. da Silva, Y. Wang, E. Sonnhammer, and P. D. Thomas. Advances and applications in the quest for orthologs. *Mol. Biol. Evol.*, 36(10):2157–2164, Oct. 2019.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] A. L. Grazziotin, E. V. Koonin, and D. M. Kristensen. Prokaryotic Virus Orthologous Groups (pVOGs): a resource for comparative genomics and protein family annotation. *Nucleic Acids Research*, 45(D1):D491–D498, 10 2016.

[22] A. Harish, A. Abroi, J. Gough, and C. Kurland. Did viruses evolve as a distinct supergroup from common ancestors of cells? *Genome Biol. Evol.*, 8(8):2474–2481, Aug. 2016.

[23] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[24] J. Huerta-Cepas, D. Szklarczyk, D. Heller, A. Hernández-Plaza, S. K. Forslund, H. Cook, D. R. Mende, I. Letunic, T. Rattei, L. Jensen, C. von Mering, and P. Bork. eggNOG 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic Acids Research*, 47(D1):D309–D314, 11 2018.

[25] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[26] G. Jurman, S. Riccadonna, and C. Furlanello. A comparison of mcc and cen error measures in multi-class prediction. *PLOS ONE*, 7(8):1–8, 08 2012.

[27] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *ArXiv*, abs/1706.02515, 2017.

[28] E. V. Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.*, 39(1):309–338, 2005.

[29] E. V. Koonin, M. Krupovic, and V. I. Agol. The baltimore classification of viruses 50 years later: How does it stand in the light of virus evolution? *Microbiol. Mol. Biol. Rev.*, 85(3):e0005321, Aug. 2021.

[30] E. V. Koonin and Y. I. Wolf. Genomics of bacteria and archaea: the emerging dynamic view of the prokaryotic world. *Nucleic Acids Res.*, 36(21):6688–6719, Dec. 2008.

[31] S. R. Krishnamurthy and D. Wang. Origins and challenges of viral dark matter. *Virus Res.*, 239:136–142, July 2017.

[32] P. W. Laffy, E. M. Wood-Charlson, D. Turaev, S. Jutz, C. Pascelli, E. S. Botté, S. C. Bell, T. E. Peirce, K. D. Weynberg, M. J. H. van Oppen, T. Rattei, and N. S. Webster. Reef invertebrate viromics: diversity, host specificity and functional capacity. *Environ. Microbiol.*, 20(6):2125–2141, June 2018.

[33] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, 2014.

[34] T. Lingner and P. Meinicke. Remote homology detection based on oligomer distances. *Bioinformatics*, 22(18):2224–2231, Sept. 2006.

[35] H. M. Oksanen and N. G. A. Abrescia. Membrane-containing icosahedral bacteriophage PRD1: The dawn of viral lineages. *Adv. Exp. Med. Biol.*, 1140:85–109, 2019.

[36] E. Pasolli, F. Asnicar, S. Manara, M. Zolfo, N. Karcher, F. Armanini, F. Beghini, P. Manghi, A. Tett, P. Ghensi, M. C. Collado, B. L. Rice, C. DuLong, X. C. Morgan, C. D. Golden, C. Quince, C. Huttenhower, and N. Segata. Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle. *Cell*, 176(3):649–662.e20, Jan 2019. 30661755[pmid].

[37] W. R. Pearson. An introduction to sequence similarity ("homology") searching. *Curr. Protoc. Bioinformatics*, Chapter 3(1):Unit3.1, June 2013.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[39] A. Possenti, M. Vendruscolo, C. Camilloni, and G. Tiana. A method for partitioning the information contained in a protein sequence between its structure and function. *Proteins: Structure, Function, and Bioinformatics*, 86(9):956–964, 2018.

[40] S. Roux, D. Páez-Espino, I.-M. A. Chen, K. Palaniappan, A. Ratner, K. Chu, T. B. K. Reddy, S. Nayfach, F. Schulz, L. Call, R. Y. Neches, T. Woyke, N. N. Ivanova, E. A. Eloe-Fadrosh, and N. C. Kyrpides. IMG/VR v3: an integrated ecological and evolutionary framework for interrogating genomes of uncultivated viruses. *Nucleic Acids Research*, 49(D1):D764–D775, 11 2020.

[41] S. Seo, M. Oh, Y. Park, and S. Kim. DeepFam: deep learning based alignment-free method for protein family modeling and prediction. *Bioinformatics*, 34(13):i254–i262, 06 2018.

[42] H. Shen, L. C. Price, T. Bahadori, and F. Seeger. Improving generalizability of protein sequence models with data augmentations. *bioRxiv*, 2021.

[43] J. Söding. Protein homology detection by HMM-HMM comparison. *Bioinformatics*, 21(7):951–960, Apr. 2005.

[44] H. Soueidan, L.-A. Schmitt, T. Candresse, and M. Nikolski. Finding and identifying the viral needle in the metagenomic haystack: trends and challenges. *Front. Microbiol.*, 5:739, 2014.

[45] Stanford University. CS231n: Convolutional Neural Networks for Visual Recognition, http://cs231n.stanford.edu/, Accesed Dec 2021.

[46] M. Steinegger and J. Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.

[47] R. L. Tatusov, E. V. Koonin, and D. J. Lipman. A genomic perspective on protein families. *Science*, 278(5338):631–637, Oct. 1997.

[48] J. Thannesberger, H.-J. Hellinger, I. Klymiuk, M.-T. Kastner, F. J. J. Rieder, M. Schneider, S. Fister, T. Lion, K. Kosulin, J. Laengle, M. Bergmann, T. Rattei, and C. Steininger. Viruses comprise an extensive pool of mobile genetic elements in eukaryote cell cultures and human clinical samples. *FASEB J.*, 31(5):1987–2000, May 2017.

[49] The UniProt Consortium. UniProt: the universal protein knowledgebase. *Nucleic Acids Res.*, 45(D1):D158–D169, Jan. 2017.

[50] The UniProt Consortium. UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research*, 49(D1):D480–D489, 11 2020.

[51] Towards Data Science. A Comprehensive Guide to Convolutional Neural Networks, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Accessed Dec 2021.

[52] K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Žídek, A. Bridgland, A. Cowie, C. Meyer, A. Laydon, S. Velankar, G. J. Kleywegt, A. Bateman, R. Evans, A. Pritzel, M. Figurnov, O. Ronneberger, R. Bates, S. A. A. Kohl, A. Potapenko, A. J. Ballard, B. Romera-Paredes, S. Nikolov, R. Jain, E. Clancy, D. Reiman, S. Petersen, A. W. Senior, K. Kavukcuoglu, E. Birney, P. Kohli, J. Jumper, and D. Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, Aug 2021.

[53] S. Vinga and J. Almeida. Alignment-free sequence comparison-a review. *Bioinformatics*, 19(4):513–523, Mar. 2003.

[54] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11:2837–2854, dec 2010.

[55] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1(4):337–348, 1994.

[56] E. M. Zdobnov, D. Kuznetsov, F. Tegenfeldt, M. Manni, M. Berkeley, and E. V. Kriventseva. OrthoDB in 2020: evolutionary and functional annotations of orthologs. *Nucleic Acids Res.*, 49(D1):D389–D393, Jan. 2021.

[57] A. Zielezinski, S. Vinga, J. Almeida, and W. M. Karlowski. Alignment-free sequence comparison: benefits, applications, and tools. *Genome Biol.*, 18(1):186, Oct. 2017.