



# DL4Phylo

## A Deep Learning Approach to Phylogenetic Analysis

Miguel Raposo  
Gonalo Silva

Supervisors: Ctia Vaz, ISEL

Final report written for Project and Seminary  
BSc in Computer Science and Computer Engineering

July 2024



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

# DL4Phylo

## A Deep Learning Approach to Phylogenetic Analysis

49456 Miguel dos Santos Raposo

---

49451 Gonalo Cardoso e Silva

---

Supervisors: Ctia Vaz, ISEL

---

Final report written for Project and Seminary  
BSc in Computer Science and Computer Engineering

July 2024



# Abstract

Phylogenetic inference stands as a cornerstone in bioinformatics, having an important role in reconstructing the evolutionary relationships among species through their genetic information. By meticulously analyzing DNA sequences, scientists can construct phylogenetic trees that depict these relationships, offering insights into the evolutionary history of life.

Advanced computational methods, including deep learning, are leveraged to significantly improve the accuracy and efficiency of tree construction. These advancements make phylogenetic inference an indispensable tool in understanding the complexity of biological diversity and evolution.

Several tools are designed to enhance the performance of this process through the use of deep learning methods, such as Fusang and Phyloformer. However, both of these tools are limited to working with DNA sequences.

There are numerous databases with typing data and various approaches for performing the inference process using this kind of data instead of DNA sequences. Therefore, it would be ideal to have a tool that supports this type of data. With this problem in mind, we propose DL4Phylo.

DL4Phylo is a tool designed to support typing data by enhancing the accuracy and efficiency of the inference process through deep learning techniques, building upon Phyloformer. By adapting, extending, and generalizing Phyloformer, DL4Phylo avoids the need to create a new solution from scratch and provides opportunities to explore and study the field of artificial intelligence.

Phyloformer has demonstrated its effectiveness with MSAs, and now, by incorporating typing data via DL4Phylo, we can assess its advantages for the inference process. Specifically, this evaluation aims to determine if it can deliver faster, yet equally accurate, results compared to using DNA sequences directly.

**Keywords:** DL4Phylo; phylogenetic analysis; typing data; tree inference; Phyloformer; deep learning; artificial intelligence.



# Resumo

A inferência filogenética é um processo crucial em bioinformática, utilizada para reconstruir as relações evolutivas entre espécies com base na sua informação genética. Ao analisar sequências de ADN, os cientistas podem construir árvores filogenéticas que ilustram essas relações, fornecendo informações sobre a história evolutiva da vida.

Alguns métodos computacionais, como a aprendizagem profunda, são utilizados para melhorar significativamente a precisão e eficiência da construção das árvores. Esses avanços tornam a inferência filogenética uma ferramenta indispensável para a compreensão da complexidade na diversidade biológica e na evolução.

Várias ferramentas foram desenvolvidas para melhorar o desempenho deste processo através do uso de métodos de aprendizagem profunda, como o Fusang e o Phyloformer. No entanto, ambas estas ferramentas estão limitadas a trabalhar apenas com sequências de ADN.

Como existem várias bases de dados que possuem *typing data* e várias ferramentas que realizam o processo de inferência sob estes dados, seria ideal ter uma ferramenta que fosse capaz de suportar este tipo de dados. Com este problema em mente, nós propomos o DL4Phylo.

O DL4Phylo é uma ferramenta projetada para suportar *typing data*, melhorando a precisão e eficiência do processo de inferência através de técnicas de aprendizagem profunda, baseando-se no Phyloformer. Ao adaptar, estender e generalizar o Phyloformer, o DL4Phylo evita a necessidade de criar uma nova solução do zero e oferece oportunidades para explorar e estudar o campo da inteligência artificial.

O Phyloformer demonstrou a sua eficácia com as MSAs e agora, incorporando a *typing data* através do DL4Phylo, podemos avaliar as suas vantagens para o processo de inferência. Esta análise tem por objetivo determinar se o DL4Phylo é capaz de entregar rapidamente e com igual precisão resultados quando comparados ao uso direto de sequências de ADN.

**Palavras-chave:** DL4Phylo; análise filogenética; *typing data*; inferência de árvores; Phyloformer; aprendizagem profunda; inteligência artificial.





# Contents

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Acronyms</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Outline . . . . .	2
<b>2 Requirements</b>	<b>5</b>
<b>3 Background</b>	<b>7</b>
3.1 Phylogenetic Analysis . . . . .	7
3.1.1 Sequence Data . . . . .	8
3.1.2 Typing Data . . . . .	8
3.1.3 Inference Process . . . . .	10
3.2 Neural Networks . . . . .	12
3.2.1 Convolutional Neural Network . . . . .	13
<b>4 DL4Phylo</b>	<b>21</b>
4.1 Overview . . . . .	21
4.2 Neural Network Architecture . . . . .	22
4.3 Training Phase . . . . .	23
4.3.1 Pre-processing . . . . .	25
4.3.2 Training deep neural network procedure . . . . .	28
4.4 Prediction Phase . . . . .	32

4.4.1	Inference Process . . . . .	33
4.5	DL4Phylo vs Phyloformer . . . . .	34
4.5.1	Typing Data Extension . . . . .	34
4.5.2	Alignment Blocks Extension . . . . .	38
4.5.3	Code Refactoring . . . . .	40
4.6	Implementation Details . . . . .	40
<b>5</b>	<b>Experimental Evaluation</b>	<b>43</b>
<b>6</b>	<b>Final Notes</b>	<b>47</b>
	<b>References</b>	<b>48</b>

# List of Figures

3.1	Example of two sequences in FASTA format labeled as taxon0 and taxon1.	9
3.2	Example of six sequences in MLST format.	9
3.3	Example of two sequences in SNP format.	10
3.4	Example of distance matrix, resultant from applying the Hamming distance.	11
3.5	Application of Robinson-Foulds metric for two rooted trees.	12
3.6	Example of an input matrix and kernel of a convolution.	13
3.7	Convolution operation.	14
3.8	Input with 5 channels and the kernel matching in depth.	14
3.9	Convolutional operation with 3 kernels in parallel giving 3 unique feature maps.	15
3.10	Representation of the effects of a dropout layer.	16
3.11	Fully-Connected layer representation.	17
3.12	Self-Attention mechanism.	19
4.1	Neural Model Architecture Layout.	23
4.2	Detailed Neural Model Architecture.	24
4.3	Detailed Axial Attention Block.	25
4.4	Neural model training process pipeline.	26
4.5	Tensor obtained after applying one-hot encoding to sequence “ATCG-CAC”.	27
4.6	One-hot encoded tensor transposed.	27
4.7	One-hot encoded tensor reshaped to three dimensions.	27
4.8	Tensor obtained after concatenating every one-hot encoded tensor.	28

4.9	Tensor obtained at the end of pre-processing phase. . . . .	28
4.10	Pair representation matrix populated with zeros. . . . .	29
4.11	Final pair representation matrix . . . . .	29
4.12	Site-level attention. . . . .	30
4.13	Pair-level attention. . . . .	31
4.14	Prediction process pipeline. . . . .	33
4.15	Distance Matrix Example - using Hamming distance. . . . .	34
4.16	Phylogenetic Tree Example. . . . .	34
4.17	Binary encoding for one sequence. . . . .	35
4.18	Binary encoded tensor reshaped. . . . .	36
4.19	Tensor obtained after concatenating every binary encoded tensor. . . .	36
4.20	Final tensor obtained in the pre-processing phase. . . . .	37
4.21	Partitioning of sequences into different blocks. . . . .	38
4.22	Typing Data file obtained. . . . .	39
4.23	Sequence separated by blocks with the '-' as the separator. . . . .	39
5.1	Training loss over epochs for the top models trained on typing data. . .	46

# List of Tables

4.1	Example of sequence alignments used for testing the pre-processing phase.	26
4.2	Example of a sequence alignment used in the partition testing. . . . .	37
5.1	Results of the best models trained on sequence data and typing data datasets. . . . .	44
5.2	Average results of the best 10 models for each dataset, whenever possible.	45



# List of Acronyms

<i>DNA</i>	Deoxyribonucleic Acid
<i>MSA</i>	Multiple Sequence Alignment
<i>MLST</i>	Multi Locus Sequence Typing
<i>MLVA</i>	Multi Locus Variable Number Tandem Repeat Analysis
<i>ST</i>	Sequence Type
<i>SNP</i>	Single Nucleotide Polymorphism
<i>RF</i>	Robinson–Foulds
<i>CNN</i>	Convolutional Neural Network
<i>ReLU</i>	Rectified Linear Unit
<i>GELU</i>	Gaussian Error Linear Unit
<i>MRE</i>	Mean Relative Error
<i>MAE</i>	Mean Absolute Error
<i>NJ</i>	Neighbor Joining
<i>eBURST</i>	electronic Based Upon Related Sequence Type
<i>goeBURST</i>	global optimal eBURST
<i>UPGMA</i>	Unweighted Pair Group Method with Arithmetic Mean
<i>GPU</i>	Graphics Processing Unit





# Chapter 1

## Introduction

Phylogenetics is the study of the evolutionary history and relationships among individuals, groups of organisms, or other biological entities with evolutionary histories [1]. Through phylogenetic analysis, a field of biomedical research focused on studying these relationships, we can gain a better understanding of the evolution of bacterial and viral epidemics [2].

The task of inferring evolutionary history, known as phylogenetic inference, can be accomplished using various methods that result in an estimation called a phylogenetic tree. The most computationally efficient methods are based on distance measurements. These methods infer the relationship between individuals as the number of genetic differences between pairs of sequences, which become represented in the form of a squared matrix of dimension 'n', where 'n' equals the number of sequences and sequences could be of DNA or allelic profiles. To infer the tree, it is necessary to traverse, at most, the upper triangular part of this matrix, implying that these algorithms have a complexity that is at least quadratic. Neighbor Joining (NJ), UPGMA, and goeBURST are some traditional methods used for this purpose [1]. NJ has a time complexity of  $O(n^3)$ ; UPGMA, depending on its implementation, has a time complexity of  $O(n^3)$ ,  $O(n^2 \log n)$ , or  $O(n^2)$ ; and goeBURST has a time complexity of  $O(n^2)$ .

There are also alternative approaches based on deep learning that aim to speed up the process and improve the accuracy of the inferred trees. Examples include Phyloformer [3], which generates a distance matrix, and Fusang [4], which generates quartets, i.e., 4-leaf trees. Both tools assist the inferring process by receiving as input a multiple sequence alignment (MSA), which is an aggregate of multiple DNA sequences.

Numerous databases contain typing data, and various methods exist for performing the inference process using this kind of data instead of DNA sequences. Consequently, it would be advantageous to have a tool that supports this type of data. With this need in mind, we propose our solution, DL4Phylo, a deep learning tool designed to support

typing data as an input that enhances both the accuracy and efficiency of the inference process, similarly to those that Phyloformer provides for DNA sequences. It builds upon the Phyloformer tool, adapting, extending, and generalizing it to avoid the need to develop an entirely new solution from scratch. Phyloformer has already proven to produce good results when tested with MSA's, and now by enabling the use of typing data through DL4Phylo, we can evaluate its potential benefits to the overall inference process, particularly in terms of achieving faster yet still accurate results compared to the direct use of DNA sequences.

This project also provides an opportunity to delve into the field of artificial intelligence, allowing us to explore a new area of expertise beyond our current knowledge and become more familiar with the general theories and techniques used in deep learning.

Summarizing, in our project we study all the work put into building Phyloformer and how from it we got to adapt and extend that work into what is DL4Phylo. With DL4Phylo, we enable the use of typing data in the context of a phylogenetic inference tool that tries to increase the inference process's efficiency while still getting accurate results.

Finally, DL4Phylo will be made available as a PyPi package, providing a tool capable of supporting typing data for anyone who wishes to use it, and it is submitted, as an article, on the **Inforum** platform.

## 1.1 Outline

The remainder of this document consists of four chapters that collectively provide an understanding of the DL4Phylo tool:

- Chapter 2 - describes the goals and requirements of this project.
- Chapter 3 - introduces the domain of phylogenetic analysis, explaining key concepts and techniques. It also embraces and explains all the Neural Networks concepts related to our project.
- Chapter 4 - contains a brief explanation of the network's architecture and all processes involved in the training pipelines of the neural model and the prediction of pairwise distances. Additionally, it describes all the modifications made to Phyloformer, including the adaptation of input data and its encoding.
- Chapter 5 - describes the evaluation conducted between the use of sequences or typing data to predict the pairwise distances.

- Chapter ?? - presents the final notes on the current state of the project and outlines our objectives for the future of the DL4Phylo tool.



# Chapter 2

## Requirements

In the context of our project, we propose to build a solution that incorporates all the functionality of Phyloformer, allowing the use of sequences as input to produce phylogenetic trees as output. However, we will extend this approach to include Typing Data, allowing the reception of both types of data.

The software is available for use as a Python library or through command-line tools that are installed with the package. It is designed with the purpose of being used by people in the field, such as students and specialists.

The following obligatory requirements were defined for this project:

1. Replicate the results obtained by the Phyloformer team.
2. Test Phyloformer, evaluating experimentally its accuracy and efficiency.
3. Implement the DL4Phylo solution, adapting from Phyloformer's.
  - Input can be Typing Data instead of MSA like used in Phyloformer.
  - It must be implemented or adapted an encoder for Typing Data.
  - It must be implemented or adapted a decoder for Typing Data, if needed.
4. Evaluate the results obtained by the DL4Phylo solution.
5. Release DL4Phylo as a PyPi package.

As an optional requisite it is presented the following one:

1. Compare the generated tree against other phylogenetic inference algorithms not available in Phyloformer, like goeBURST [5].

In the evaluation part of the results produced by our solution, the goal is to use the inferred phylogenetic trees, which use the predicted distances, to compare with others whose distances were obtained through traditional methods, such as Maximum Likelihood. This comparison can be measured using the Robinson-Foulds metric [6], which is an integer value that quantifies the topological differences between pairs of trees.

# Chapter 3

## Background

This chapter provides an overview of the phylogenetic analysis domain as well as the concepts related to neural networks.

### 3.1 Phylogenetic Analysis

Phylogenetics is the study of evolutionary history and relationships among individuals, groups of organisms or other biological entities with evolutionary histories [7]. These relationships are found using phylogenetic inference techniques, which analyze reported heritable features like DNA sequences, protein amino acid sequences or morphology using an evolutionary model.

Phylogenetic analysis [8] aims at uncovering the evolutionary relationships between different species, or even between individuals of the same species, to obtain an understanding of their evolution. The result of this analysis is a phylogeny, which can be a phylogenetic tree or network. In this project, our focus is on the inference and comparison of these trees.

A phylogenetic tree is characterized by a series of branching points expanding from the last common ancestor (root) of all operational taxonomic units up to the most recent organisms. The branches represent the passage of genetic information between subsequent generations, and branch lengths denote genetic change or divergence.

A common pipeline to analyze phylogenetic data is:

- Sequencing of isolated data
- Assembly of sequences, comparing the draft genome with a database of gene alleles
- Given the assembly results, obtain the allelic profile that characterizes the strain

- Phylogenetic inference

### 3.1.1 Sequence Data

The phylogenetic analysis includes the alignment of genetic sequences [9], also known as **multiple sequence alignment** (MSA). This involves arranging the sequences to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships. It allows to map genomes of new organisms, finish the ones of previously known organisms, or to compare across multiple samples.

The sequences assembled in the alignment process may occupy a given position of a locus and define distinct alleles of that locus. A **locus** (plural: loci) in genetics is a fixed position on a chromosome, like the position of a gene. Each chromosome carries many genes, for example, humans' estimated *haploid* protein coding genes are 19,000-20,000, on the 23 different chromosomes.

The variant of the similar DNA sequence located at a given locus is called an **allele**. Alleles can come in different versions, with small differences in the sequence of the DNA at that locus. Different alleles can result in different traits. In phylogenetics, the distribution of different alleles can provide information about the evolutionary history.

The alleles are usually represented through files which usually follow the **FASTA** format. FASTA is a text-based format for representing either nucleotide sequences or amino acid sequences, where these are represented using single-letter codes.

Here's a basic outline of the FASTA format:

- A single-line description, also called the '*sequence identifier*', begins with a greater-than symbol (">"). It's often used to include useful information about the sequence;
- Following the description line, the actual sequence is represented. It can span multiple lines, but there are no spaces within the sequence;
- A new sequence begins with a new description line. There can be any number of sequences in a FASTA formatted file.

The Figure 3.1 shows an example of a FASTA file with two sequences.

### 3.1.2 Typing Data

After the alignment phase, a typing methodology is applied to identify each organism based on the genes that are present in almost all organisms. This process refers to



```

>taxon0
GKCSPKVWCTGLLNPLASYCTLKGPLASHIIGTVKMKNAQQHNPRQKDFTPNFFVEVALLNMKSVQAVVLG
IPQAVSNYPDYLFKFDVLSKQQRWHVGAYGGPDLDTFKRDQPDADVSPSNDGSLFELNYALAEALQDNQSP
GVQTINDQANRIKKVIFGFFVHNNGLAHKPEKGFGAPPTLGADSPYKDVEVNAGVM
>taxon1
AACCDGLWCTVLLKKPLTSFYTLVGPLSEEEYIATVKIDAASEHEPRPKKFSTMIFLDTALYSEQLTQSLAVG
APQAVTQYLTTRLAYSVHCDQCPLNIEKVTGPDLELFEKGQPLTTLNSVTGTHELFALVYSIGEHLSNDSNQ
GAKTVEDDSASIARMLLSQFVHRPAPSQKCERGYKKAGALEGDGTYGRITIAGGIK

```

Figure 3.1: Example of two sequences in FASTA format labeled as taxon0 and taxon1. The sequences are stored in a text file and follow the conventions of the FASTA format, which uses single letter codes to represent nucleotides or amino acids.

the genetic information of an organism that is used to identify and classify organisms based on their evolutionary relationships [10].

The typing data is usually the result of a molecular typing technique, such as **Multi Locus Sequence Typing** (MLST) or **Multi Locus Variable Number Tandem Repeat Analysis** (MLVA). For instance, MLST characterizes isolates of bacterial species using the sequences of internal fragments of usually seven house-keeping genes.

The comparison of the profiles is done between lines, analyzing whether the identifier between them is equal or not. In this way, if the numbers, for the same locus, are different, it means that they present at least a different allele at that locus.

It is usually stored in a tabular format, where each row represents a profile and each column represents a locus. The values in the table are the allele number for each locus in each profile.

<i>ST</i>	<i>adk</i>	<i>atpG</i>	<i>frdB</i>	<i>fucK</i>	<i>mdh</i>	<i>pgi</i>	<i>recA</i>
1	1	1	1	14	15	1	5
2	14	7	1	15	16	4	1
3	1	1	1	1	1	1	5
4	4	17	4	1	2	9	6
5	12	5	5	2	3	11	7
6	10	14	4	5	4	7	8

Figure 3.2: Example of six sequences in MLST format. Each row corresponds to a profile, and each column represents a locus. The profiles are identified by a Sequence Type (ST) followed by the identifiers for each locus. In the example, the profiles are associated with specific combinations of alleles for the loci *adk*, *atpG*, *frdB*, *fucK*, *mdh*, *pgi*, and *recA*. The numbers present in each locus indicate the allele identifiers.

And nowadays, with the mainstream use of High Throughput Sequencing, it is common to have profile having thousands of loci, using schemas like cgMLST and wgMLST [11].

**Single nucleotide polymorphisms** (SNPs) [12] are polymorphisms that are caused

by point mutations that give rise to different alleles containing alternative bases at a given position of nucleotide within a locus. It enables the comparison of genetic relatedness between bacteria even on a sub-species level.

SNP analysis is a genomic typing method based on single nucleotide loci, which has been widely used in pathogenic classification, tracking and prevention. Bacterial SNP analysis may adopt its core genome or whole genome. Compared with MLST, cgMLST and wgMLST, SNP has the highest resolution in analysis of gene sequence differences between strains [13].

The SNP format represents each sequence by a line with a sequence of 1's and 0's preceded by a number that identifies the sequence. A value of 0 represents the character state that was mostly found on that location, while a 1 represents any other possible character state.

```
1 0100000111101010001000101010101001010100011101011000101010
2 11110100101011001001010101010010000010100100010101010100
```

Figure 3.3: Example of two sequences in SNP format. The sequences identifiers are the first number on each sequence: '1' and '2'. Each sequence exhibits variations at multiple nucleotide positions within the locus.

### 3.1.3 Inference Process

Succeeding the typing process follows the inference of a phylogenetic tree. The tree results from applying a phylogenetic inference method [14] to phylogenetic data, such as DNA, protein sequences or typing data, with the objective of predicting the evolutionary history and relationships among a group of organisms. There are several types of phylogenetic inference methods, some of which depend on the calculation of a distance matrix, a table that quantifies the genetic divergence between pairs of species or sequences [1].

#### 3.1.3.1 Distances

The distance matrix methods of phylogenetic analysis explicitly rely on the genetic distance between the sequences being classified. The distances are often defined as the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches.

The Figure 3.4 shows an example of a distance matrix, resultant from applying the Hamming distance between each profile. Sometimes, a distance correction is applied to the Hamming distance according to some model of evolution.

$$D = \begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 2 & 0 & 2 & 4 & 6 \\ 3 & 2 & 0 & 3 & 5 \\ 4 & 4 & 3 & 0 & 6 \\ 5 & 6 & 5 & 6 & 0 \end{bmatrix}$$

Figure 3.4: Example of distance matrix, resultant from applying the Hamming distance. Each row and column correspond to a specific sequence profile. The values in the matrix indicate the genetic distances between the profiles. For example, the value at row 1 and column 3 represents the distance between profile 1 and profile 3. In this case, they differ by a distance of 2, indicating two mismatches between the sequences.

### 3.1.3.2 Inference Algorithms

An inference algorithm is then executed to compute the inference based on the distance matrix previously calculated.

There are several methods used for phylogenetic inference, but the ones that will be addressed and used in our project are the **Distance-based** ones. These methods aim to construct a tree that corresponds to a matrix of pairwise genetic distances. For every two sequences, the distance is a single value based on the fraction of positions in which the two sequences differ.

There are several distance based inference algorithms, however in the scope of our project we will use the Neighbour Joining algorithm [15].

### 3.1.3.3 Metrics

Multiple inference methods are available to infer phylogenetic trees, with resulting trees not being unique. Therefore it is necessary to compare it with other trees that have been obtained by other traditional methods, as a way to compare if the obtained tree is close to these. Along with this comes the concept of metrics, which aim to measure their dissimilarity.

There are several measures for assessing differences between two phylogenetic trees, with the **Robinson–Foulds** distance being one of the most used, and which can be computed in linear time and space. This metric is designed to count the number of clusters that are not shared between the trees [6]. A cluster is defined as a group of biological sequences that share a recent common ancestor, often due to their phylogenetic relationships [16]. Each cluster in a tree is characterized by the set of leaves (or nodes in fully labelled trees) that descend from a specific node.

According to the trees present in Figure 3.5, we can calculate this metric for each tree,

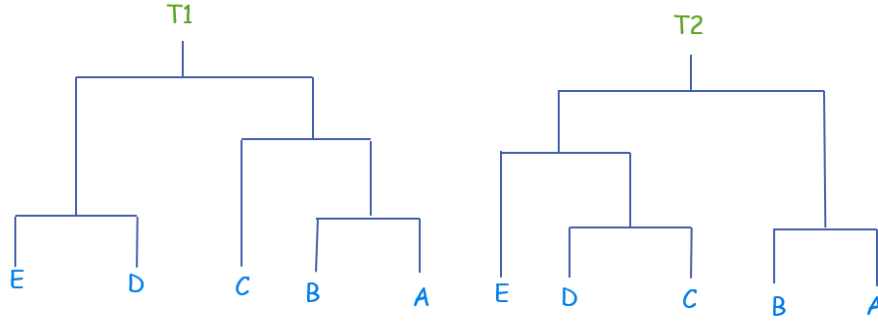


Figure 3.5: Application of Robinson-Foulds metric for two rooted trees.

and since the comparison is made between tree pairs, both values are summed and then divided by 2:

$$C(T1) = \{ \{E\}, \{D\}, \{C\}, \{B\}, \{A\}, \{E,D\}, \{B,A\}, \{C,B,A\}, \{E,D,C,B,A\} \}$$

$$C(T2) = \{ \{E\}, \{D\}, \{C\}, \{B\}, \{A\}, \{D,C\}, \{B,A\}, \{E,D,C\}, \{E,D,C,B,A\} \}$$

$DRF(T1,T2) = (2 + 2)/2 = 2$ , value that indicates that both the trees have 2 different clusters.

## 3.2 Neural Networks

A **neural network** is a machine learning model that makes decisions similarly to the human brain, using processes that mimic how biological neurons work together to identify phenomena, weigh options, and arrive at conclusions [17].

A typical neural network architecture consists of at least three types of layers of nodes, or artificial neurons: an **input** layer, one or more **hidden** layers, and an **output** layer [18]. The hidden layer is the most crucial to the network's functionality.

Data received in the input layer passes through the hidden layers, enabling the network to learn patterns associated with the input. As more hidden layers are introduced in a network's architecture, it becomes capable of recognizing more specific features as the data traverses each layer [18].

This process underlies the "intelligence" of a neural network and forms the basis for deep learning.

Common neural network architectures include recurrent neural networks, transformers, autoencoders, graph neural networks, generative adversarial networks, and convolutional neural networks.

### 3.2.1 Convolutional Neural Network

**Convolutional neural networks** (CNNs) are a type of neural network specialized in processing two-dimensional data [19]. They are commonly used for image recognition, pattern recognition, and computer vision.

A CNN is Phyloformer’s main neural network architecture, making it one of our primary objectives to understand. Phyloformer’s CNN utilizes four types of hidden layers.

#### 3.2.1.1 Convolutional layer

Convolutional layers are the core of CNNs and perform an operation called convolution [19]. Through this operation, these layers can filter and detect specific traits in the given input.

A convolution involves a series of dot products between the input matrix and a matrix of weights, called a filter or **kernel**.

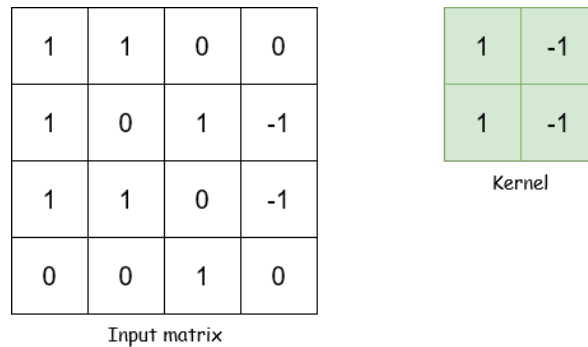


Figure 3.6: Example of an input matrix and kernel of a convolution.

The kernel is intentionally smaller than the input matrix, as it slides across the input matrix by a certain stride to produce a two-dimensional matrix called a **feature map**. The Figure 3.7 shows an example of the convolution operation.

Inputs to convolutional layers may have channels, or depth, and each kernel used in the convolution must match this depth. For example, if an input has 5 channels, each kernel must also have 5 channels, as seen in Figure 3.8, with each channel representing a different set of weights.

After performing the convolution operation channel by channel, the resulting values are summed to produce the feature map. Typically, multiple kernels operate in parallel with the input, with each kernel delivering a unique feature map. Thus, a convolution with **3 kernels** will produce **3 feature maps**, like shown in Figure 3.9.

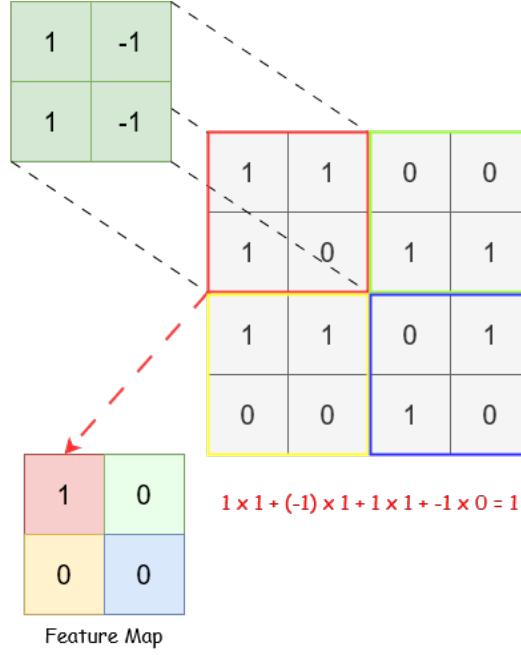


Figure 3.7: Convolution operation.

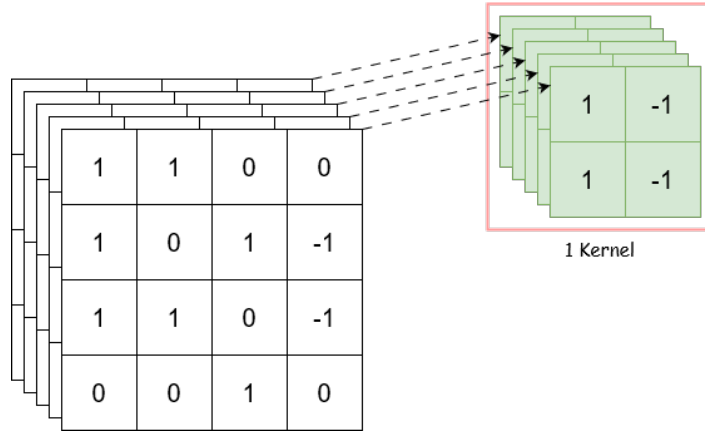


Figure 3.8: Input with 5 channels and the kernel matching in depth.

### 3.2.1.2 Normalization layer

Normalization has always been an important asset in deep learning, as it plays crucial role at stabilizing and decreasing the model training time by a huge factor while producing more reliable results [20].

There are various normalization methods, such as, batch normalization, weight normalization, instance normalization, group normalization, among others. In Phyloformer’s particular case there will be a major use of a normalization layer of type: **Layer Normalization**.

The layer normalization is a normalization technique created as an alternative to the

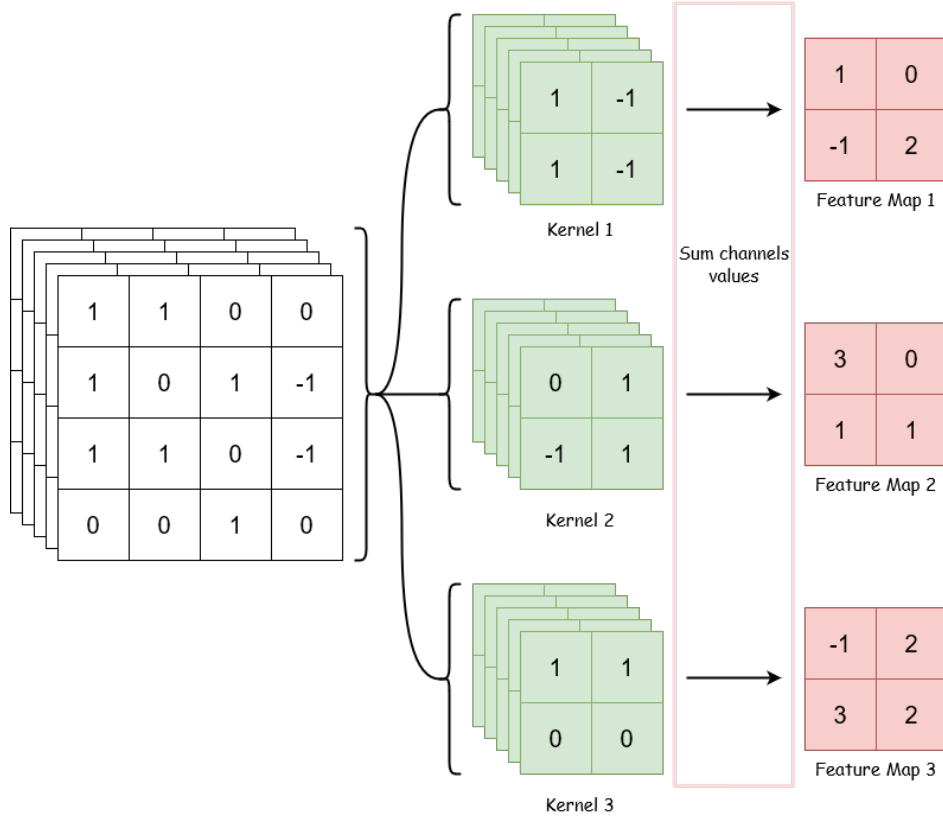


Figure 3.9: Convolutional operation with 3 kernels in parallel giving 3 unique feature maps.

batch normalization, as it overcomes some of its negative aspects by removing the dependency on batches, normalizing the activation's of each layer across the feature dimension [21].

A layer normalization therefore normalizes by computing the zero mean and unit variance for each feature independently [21].

### 3.2.1.3 Dropout layer

To reduce overfitting and improve generalization error in neural networks, a regularization technique called **dropout** can be used, applied through the dropout layer [22].

The problem of overfitting arises when large neural networks are trained on relatively small datasets, causing the model to learn the training data too well. This results in poor performance when the model is evaluated on new data [23].

Dropout is a method that approximates training a large number of neural networks with different architectures in parallel. It is computationally inexpensive and effective at regularizing deep neural networks [23]. Dropout works by "dropping out" (deactivating) a random subset of neurons during each forward and backward pass of training,

making the layer appear as if it has fewer nodes, as seen in Figure 3.10. This affects the connections between layers and forces nodes within a layer to probabilistically take on more or less responsibility for the inputs [22].

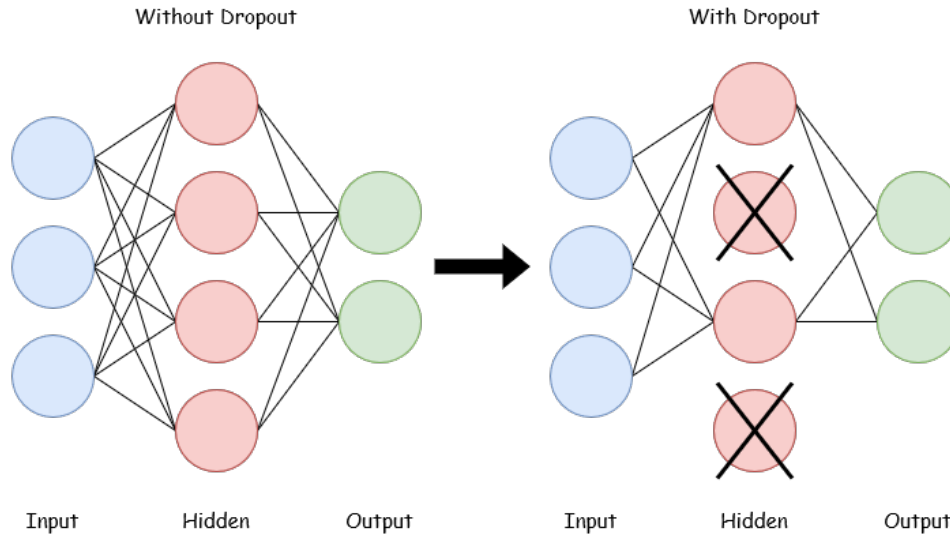


Figure 3.10: Representation of the effects of a dropout layer.

By using dropout, the method can break up situations where network layers co-adapt to correct mistakes from prior layers, making the model more robust [22].

#### 3.2.1.4 Fully Connected layer

Fully connected layers, also known as **linear layers**, apply a linear transformation to an input vector, changing it into a different vector through a matrix of weights [19].

They are called fully connected because all possible connections between layers are present, meaning that every input node influences every output node, as represented in Figure 3.11.

#### 3.2.1.5 Activation Functions

**Activation functions** are at the core of deep neural networks, enabling them to learn arbitrarily complex mappings. Without activation functions, neural networks would consist solely of linear operations, like matrix multiplication, without introducing any non-linearities [24, 25].

Activation functions allow neural networks to capture complex, non-linear relationships by incorporating non-linear dynamics. They determine the activation state of a neuron by computing the weighted sum of its inputs and adding a bias term [24, 25].



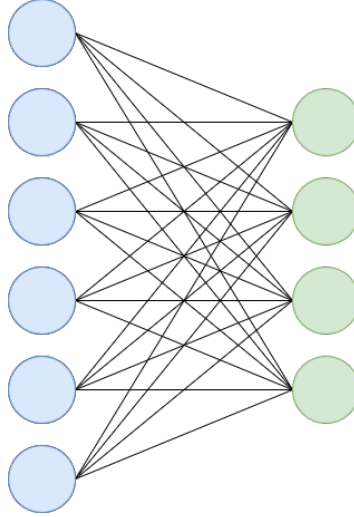


Figure 3.11: Fully-Connected layer representation.

There are many activation functions to choose from, and selecting the most appropriate one is crucial for training a neural network that generalizes well and provides accurate predictions [24].

In Phyloformer's case the used functions are:

- **Elu** or exponential linear unit, is a parameterized function with an 'alpha' parameter that upon receiving a number as input returns the number itself if it's positive, and gives 'alpha' multiplied by the exponentiated input subtracted by 1 if not.

$$elu(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases} \quad (3.1)$$

The ELU has the potential of getting better accuracy than the ReLU [25].

- **ReLU** or rectified linear unit simply eliminates a negative value by making its value zero.

$$ReLU(x) = \max(0, x) \quad (3.2)$$

Although having lesser performance than ELU, is highly computationally efficient [25].

- **Softmax** applies one-sum probabilities to individual components of a vector. It can be used for multi-class classification problems. [25].

$$Softmax(x_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, j = 1, \dots, K \quad (3.3)$$

- **Softplus** is the softer, or smoother version of the ReLU. When the ReLU gives zero gradients, the SoftPlus allows smooth gradients [26].

$$\text{SoftPlus}(x) = \frac{1}{\beta} * \log(1 + e^{\beta x}) \quad (3.4)$$

- **GELU** or Gaussian error linear unit function is  $x\Phi(x)$ , where  $\Phi(x)$  is the standard Gaussian cumulative distribution function. GELU exceeds ReLUs and ELUs across numerous tasks [27].

$$\text{GELU}(x) = 0.5x(1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{\pi}}} e^{-t^2} dt) \quad (3.5)$$

One can approximate GELU with:

$$\text{GELU}(x) = 0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}} \times (x + 0.044715 \times x^3))) \quad (3.6)$$

### 3.2.1.6 Transformer and attention

An important aspect of Phyloformer’s network architecture is that it is transformer-based. A **Transformer** is a model that uses Attention, a technique that allows the model to focus on relevant parts of the input as needed, enhancing the speed at which these models can be trained.

**Self-attention** is a mechanism used in machine learning, to capture dependencies and relationships within input sequences. It allows the model to identify and weigh the importance of different parts of the input sequence by attending to itself. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [28].

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the **query**, **keys**, **values**, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [28]. The Figure 3.12 depicts these calculations.

Finally, often what is used in practice is **multi-head self attention**, which conceptually allows to focus on several different features of the inputs. It is realized using ‘h’ so-called attention heads, that is ‘h’ different triplets of weight matrices for query, key and value, each computing a set of outputs with the aforementioned attention mechanism. The outputs corresponding to each input and the different attention heads are then concatenated and multiplied by a common weighted matrix [3].

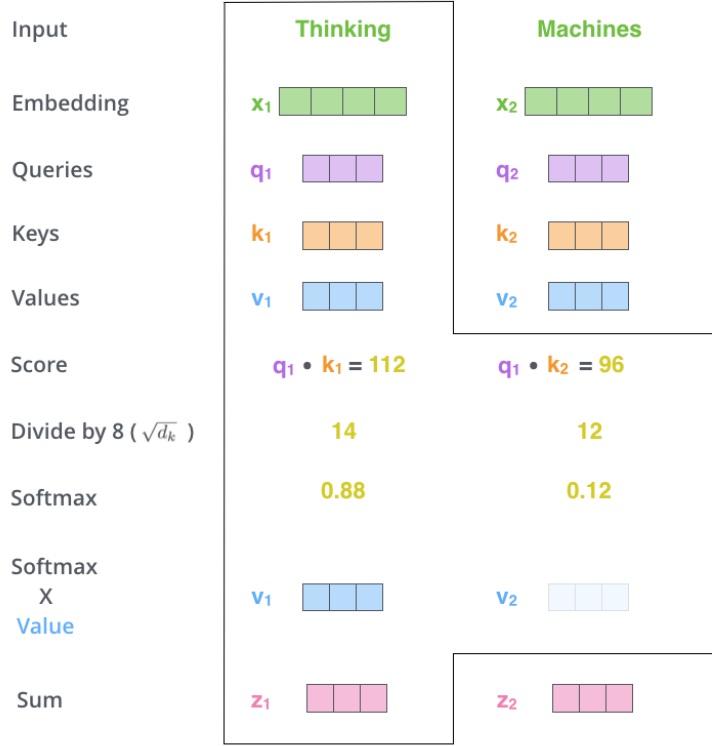


Figure 3.12: Self-Attention mechanism [29].

Phyloformer's transformer-like behavior is encapsulated in a component called **Axial-Attention** [30]. There can be many Axial-Attention blocks in Phyloformer's network, all of which are arranged in a stack and are identical in structure.

### 3.2.1.7 Metrics MAE and MRE

The **Mean Relative Error (MRE)** [31] and **Mean Absolute Error (MAE)** [32] are statistical metrics used to evaluate the accuracy of a model's predictions. MRE measures the average relative difference between predicted and actual values, normalizing the errors by the actual values, which is useful for comparing errors across different scales:

$$(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{|y_i - \hat{y}_i|}{|y_i|}, N = \text{Batchsize} \quad (3.7)$$

In contrast, MAE calculates the average absolute difference between predicted and actual values, providing a straightforward interpretation of the model's prediction accuracy without normalization:

$$(y, \hat{y}) = \frac{\sum_{i=0}^{N-1} |y_i - \hat{y}_i|}{N}, N = \text{Batchsize} \quad (3.8)$$

Both metrics help in assessing how well a model performs, with lower values indicating better accuracy.

# Chapter 4

## DL4Phylo

This chapter provides an overview and also a detailed description of the different components present in the DL4Phylo project. It also includes a comparison between DL4Phylo and Phyloformer [3], depicting their differences in Section 4.5.

### 4.1 Overview

DL4Phylo is a transformer-based Network that infers phylogenetic relationships. It takes as input either a multiple sequence alignment (MSA) or a typing data file and then infers a phylogenetic tree which is subsequently evaluated using the Robinson-Foulds metric. We chose to expand this tool to Typing Data to allow for comparison of the results produced using each type of data. It is a research question we aim to answer: 'Can we achieve good results using typing data?'

DL4Phylo includes two main execution flows, which we will refer to as pipelines. The first consists on the neural network training. The second aims to describe the entire process of distance prediction and inference of the trees. This involves the encoding of the input data, the application of the trained neural network from the previous pipeline, the generation of a distance matrix, the inference of the phylogenetic tree, and the application of metrics to compare with other trees generated from traditional methods.

The training of this model serves for DL4Phylo to be able to predict the pairwise distances, that is, the distances between pairs of sequences present in the alignment. With this training, the trained model produced in its output will be used for prediction in the second pipeline. This will apply this model to predict the pairwise distances, from alignment sequences or their typing data, both mentioned in Sections 3.1.1 and 3.1.2, respectively. From these, the distance matrix is constructed and subsequently the phylogenetic tree is inferred from this matrix. As mentioned in Chapter 3, the

algorithm used for the tree inference is Neighbour Joining, however our network is extendable to support any other distance-based method.

To define the neural network it was used a well-known library for this type of application, **Pytorch** [33]. With this library, it is possible to define various types of neural network models or even different activation functions, allowing the support of GPU acceleration for a faster computation. This library uses a data structure called **tensors**, which will be similar to a list or matrix depending on the dimension in which it is being used. Thus, they are responsible for encoding the input and output of neural models. As this data structure works in different dimensions, the notation used to represent the different dimensions is as follows:

- 2D - (number of line, number of rows)
- 3D - (depth, number of line, number of rows)
- 4D - (batch size, depth, number of line, number of rows)

## 4.2 Neural Network Architecture

DL4Phylo uses the same architecture as Phyloformer, which is a CNN architecture. As a result, both have an identical neuronal structure that is shown in Figure 4.1.

In a more detailed way, the neural model is represented in Figure 4.2.

As can be observed in Figure 4.2, it consists of an input layer, an output layer, and several hidden layers. The input is made up of as many nodes as the size of the alphabet used, and the output only presents a single node. In the case of the hidden layers, these are usually made up of many more nodes than those existing in the input and output layers, with a dimension of 64 being used. This dimension was the one being used by Phyloformer.

This architecture in Figure 4.2 uses **convolutional layers**, the red ones, and **normalization layers**, the yellow ones. Convolutional layers are used to represent the **position-wise fully connected** layers used before and after the axial attention blocks. The term ‘position-wise’ means that the position of a given value in the vector is taken into consideration. ‘Fully connected’ refers to the process where each neuron applies a linear transformation to the input vector using a weights matrix.

In addition to this, dropout mechanisms are also applied after some of the convolutional layers.

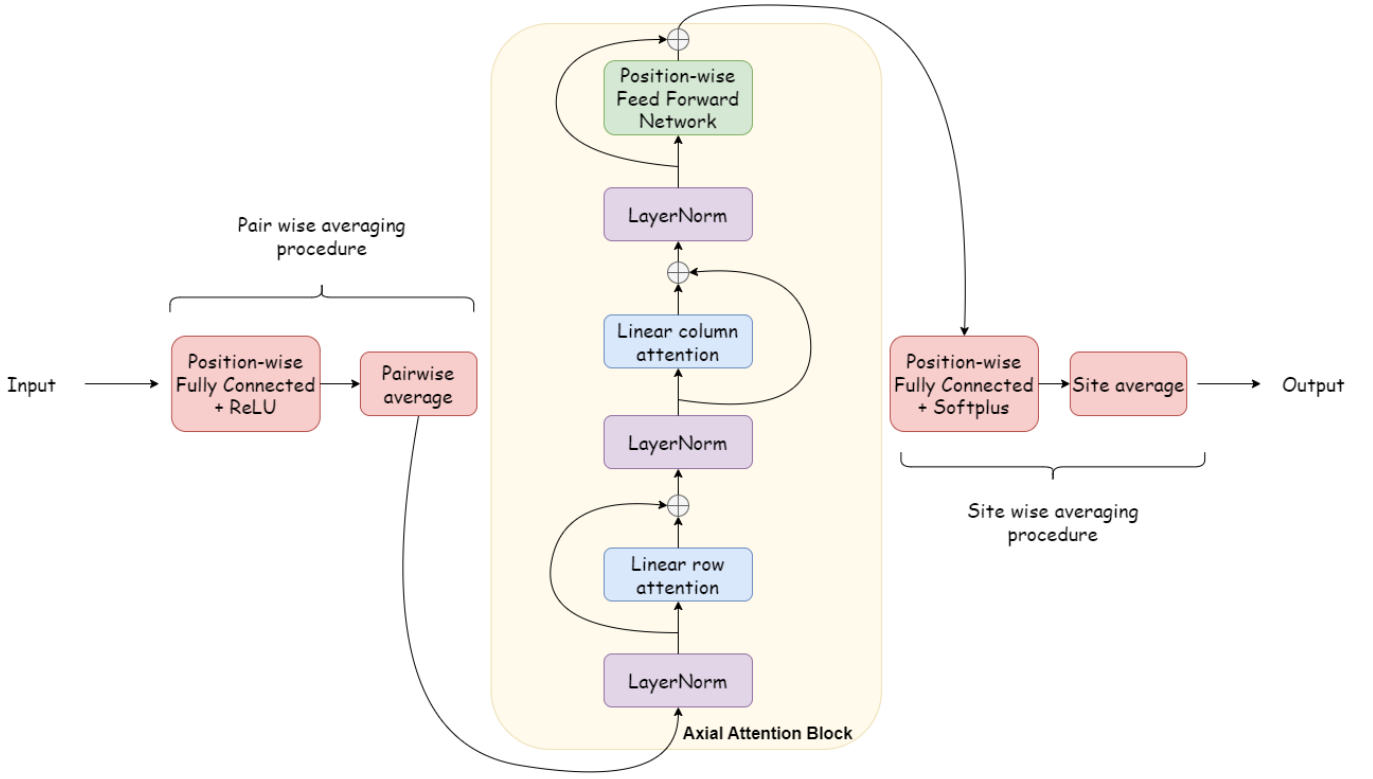


Figure 4.1: Neural Model Architecture Layout.

For the case of the axial attention blocks, they follow the mechanism of **linear self-attention**, using 3 auxiliary layers: **query**, **key**, and **value**. In this way, the interior of each one of the axial attention blocks will present **4 Linear layers** for each of the applications of linear self-attention, that is, for the **sites** and **pairs**. The **Position-wise Feed Forward Network** is the final layer in each axial attention block and will already be made up of two convolutional layers that will have a hidden dimension with four times 64, which refers to the value mentioned previously. This layers are all present in Figure 4.3.

### 4.3 Training Phase

The process that is responsible for training our neural model is represented in the training pipeline of Figure 4.4.

As depicted in Figure 4.4, the process begins with the simulation of phylogenetic trees, in the case of Phyloformer the **ETE3** simulator is used [34]. Through these trees, it is possible to generate the corresponding input data, whether they are alignments or typing data. Both, the input data and the trees simulated earlier will be converted to tensors which will be passed as training input to the neural model. In the case of input

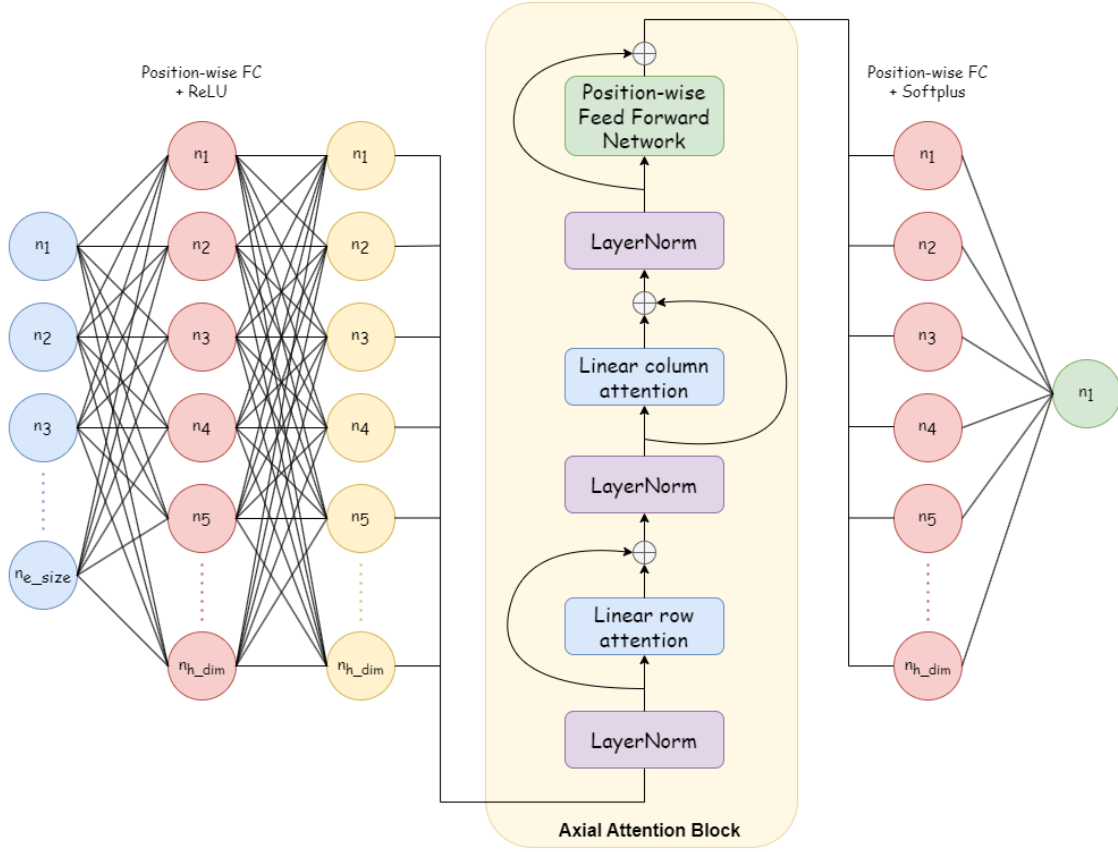


Figure 4.2: Detailed Neural Model Architecture - Represents each layer of the neural network. Each 'n' represents a node in the network, where in the first layer the 'e\_size' represents the dimension of the data encoding and in the remaining layers the 'h\_dim' represents the hidden dimension. The red nodes belong to Convolutional layers, the yellow ones belong to Normalization layers, the blue ones belong to the input, and the green one belongs to the output.

data, they are passed to tensors through the pre-processing phase, while for the trees the tensors represent the distance matrices of each one. After the conversion, these are passed to the neural model, initiating the deep neural network training procedure. It is at this stage that the sequences are paired, the neural model is trained, and all the learned information is condensed to produce a trained model. During this training procedure, one of the available objective functions is used:

- **L1 Loss** – it measures the mean absolute error (MAE) between each element in the target input  $x$  and output  $y$ :

$$l(x, y) = L = \{l_1, \dots, l_N\}^T, l_n = |x_n - y_n|, N = Batchsize. \quad (4.1)$$

- **MSE Loss** – it measures the mean squared error (squared  $L_2$  norm) between each



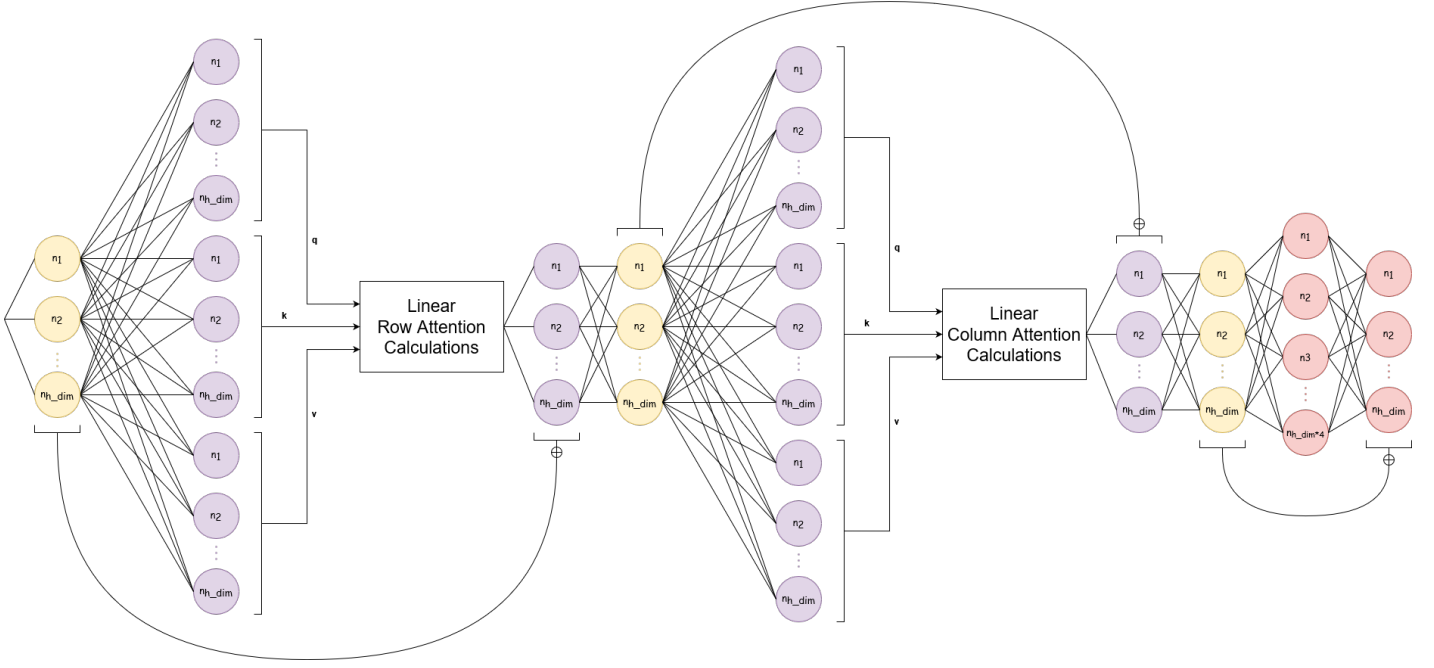


Figure 4.3: Detailed Axial Attention Block - Block that uses linear layers, the purple nodes, normalization layers and convolutional layers. The initial normalization layer is the same as the one represented in Figure 4.2.

element in the target input  $x$  and output  $y$ :

$$l(x, y) = L = \{l_1, \dots, l_N\}^T, l_n = (x_n - y_n)^2, N = \text{Batchsize}. \quad (4.2)$$

### 4.3.1 Pre-processing

The initial step in the training pipeline, would be what was designated as ***Pre-processing***, where there's a need in transforming the non processed data into a binary representation.

Each site of these sequences, with a site being a position in the sequence, will consist of its corresponding binary representation. Therefore, the chosen data representation will have to take into account the number of existing sequences, the size of each sequence, and the encoding of each site of the sequences. The best data representation for this scenario is a three-dimensional tensor.

The depth size of this tensor will depend on the encoding of each site of the sequences. If nucleotide encoding is being done, then since the alphabet,

$$\text{NUCLEOTIDES} = \{\mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{G}\} \quad (4.3)$$

has only 4 characters, the final tensor will also have a depth of 4. However, if it's amino

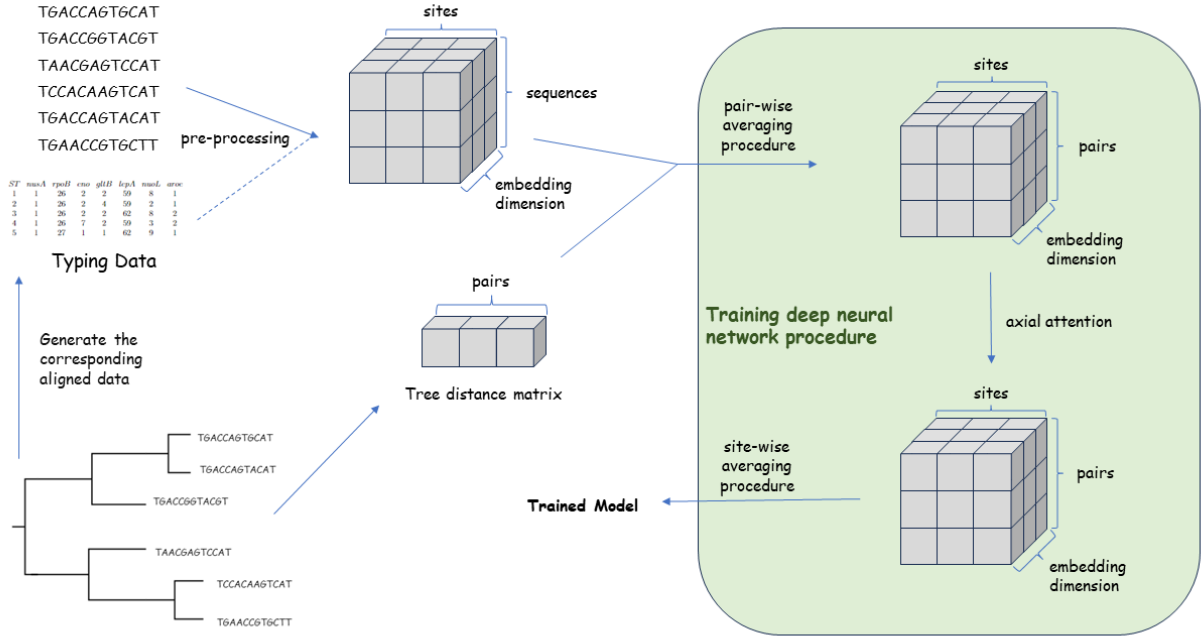


Figure 4.4: Neural model training process pipeline - This involves the simulation of phylogenetic trees and the generation of corresponding input data, whether it is alignment sequences or alternatively their corresponding typing data. Subsequently, both the input data and the trees are converted into tensors and passed to the model, with the entire internal process of this model being called the training deep neural network procedure. At the end of its learning process, a trained model is obtained which is used for the prediction phase.

acid encoding, then the alphabet expands to 22 characters,

$$AMINO\ ACIDS = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V, X, -\} \quad (4.4)$$

producing a tensor with a depth of 22. In the case of our adaptation for typing data, the depth size will be 32, which is the value corresponding to the number of bits needed to encode any integer.

To better understand this process, a small example using the nucleotide alphabet will be used. This consists of 3 sequences, defined by taxons, each with a length of 7.

taxon0	ATCGCAC
taxon1	ATGGCAT
taxon2	GCACCTC

Table 4.1: Example of sequence alignments used for testing the pre-processing phase.

Initially all the “.fasta” files (sequence alignments) provided will be parsed and go through a process of **one-hot encoding**, an algorithm that converts categorical variables into numerical ones, so that each nucleotide present in the sequence is codified

into a binary code that identifies that respective nucleotide. For instance, as depicted in Figure 4.5 and considering the alphabet of nucleotides, a tensor with shape (sequence length \* 4) is obtained.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.5: Tensor obtained after applying one-hot encoding to sequence “ATCGCAC”.

The transpose is then applied to the tensor of Figure 4.5, being crucial for the PyTorch processing that is taking place.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.6: One-hot encoded tensor transposed.

Upon obtaining the tensor of Figure 4.6, the next step is to reshape it into a three-dimensional one. Therefore, instead of a tensor with 4 rows and 7 columns, the reshaped one has the same 4 rows but in depth, resulting in a shape of (4 \* 1 \* sequence length). This reshaping can be observed in Figure 4.7.

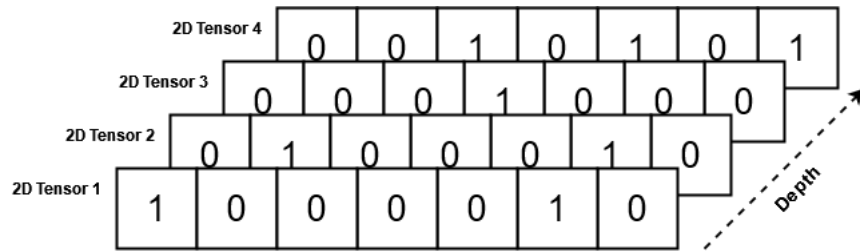
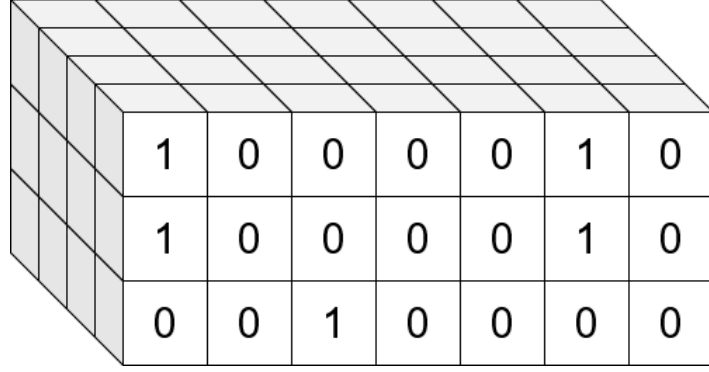


Figure 4.7: One-hot encoded tensor reshaped to three dimensions.

The process of encoding is completed for one sequence and then carried out for each remaining one. Once each sequence is encoded, they are all concatenated into a single tensor as shown in Figure 4.8.

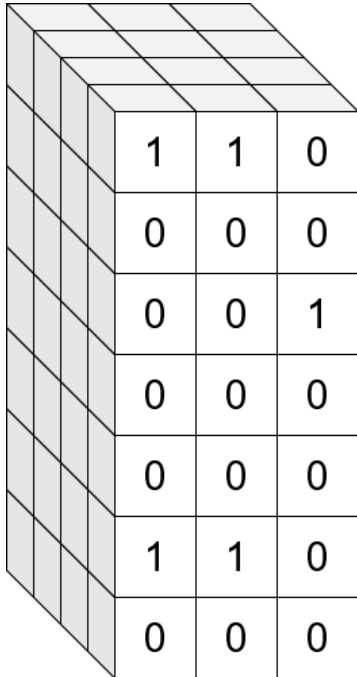
Now we have a tensor with shape (4 \* number of sequences \* sequence length), and to finish the pre-processing phase it's simply transposed the number of sequences with the sequence length.

The final tensor embodies all of the encoded sequences in the alignment.



1	0	0	0	0	1	0
1	0	0	0	0	1	0
0	0	1	0	0	0	0

Figure 4.8: Tensor obtained after concatenating every one-hot encoded tensor.



1	1	0
0	0	0
0	0	1
0	0	0
0	0	0
1	1	0
0	0	0

Figure 4.9: Tensor obtained at the end of pre-processing phase.

### 4.3.2 Training deep neural network procedure

The training deep neural network procedure phase is the one responsible for all the mechanisms and behaviors of our tool, **DL4Phylo**, that follows the architecture mentioned in Section 4.2.

To better divide and organize this layout of Figure 4.1, the training deep neural network procedure was divided into three large phases:

- **Pair Wise Averaging Procedure** - encompasses the first position-wise fully connected layer along with the pair average block;
- **Axial Attention** - encompasses all the normalization, linear self attention and position wise feed forward network layers;

- **Site Wise Averaging Procedure** - encompasses the last position-wise fully connected layer along with the site average.

All these phases only serve as a form of organization for a better understanding of the neural model.

#### 4.3.2.1 Pair Wise Averaging Procedure

This first block is responsible for handling the construction of the sequence pair representation, passing the information from the encoder discussed in the subsection 4.3.1 through a **2D convolutional layer**. As mentioned in the Phyloformer article, the position-wise fully connected layers are represented as 2D convolutional layers.

Starting with the representation of sequence pairs, a matrix is constructed with the representation of which sequences are present in each pair. Initially, this matrix is populated only with zeros, with shape (number of pairs, number of sequences). Referring again to the example used in the scope of this chapter, the initial tensor is present in Figure 4.10.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Figure 4.10: Pair representation matrix populated with zeros.

The number of pairs is calculated through 2 by 2 combinations among the different sequences. This matrix is iterated and updated to represent the constitution of each pair.

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Figure 4.11: Final pair representation matrix, where each pair of 1's in a line represents the sequences that are paired up, shows three pairs: the first between sequence 1 and sequence 2, the second between sequence 1 and sequence 3, and the third between sequence 2 and sequence 3.

With this matrix built, the tensor, coming from the encoding phase, is then passed to a layer consisting of a 2D convolutional layer and a ReLU activation function, resulting in a tensor with shape (4, 64, sequence length, number of sequences).

Next, the multiplication is performed between the tensor obtained from the previous layer and the tensor generated with the representation of the pairs. This multiplication will allow to select the information only for the sequences present in each pair, producing the tensor of shape (4, 64, number of pairs, sequence length).

#### 4.3.2.2 Axial Attention

This block is where all the logic for the neural model learning is applied, namely the linear self-attention mechanism. The dimensions of the tensor will not be affected during the learning process.

The learning process will involve the application of linear self-attention among the different axial attention blocks. Each one of these blocks is made up of the layers present in the Figure 4.1.

The learning of the neural network alternates between updating the representation of each pair, separately, by sharing information across sites, and updating the representation of each site, separately, by sharing information across pairs. With this, it is possible to apply the self-attention mechanism to any MSA with an arbitrary number of sequences and dimensions of each sequence.

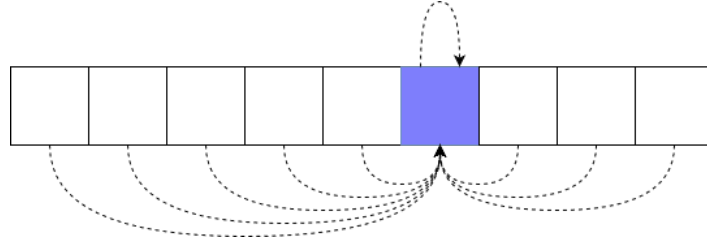


Figure 4.12: Site-level attention.

To update the representation of each pair, a **site-level attention mechanism** is used, which for each pair of sequences will update all the information between the sites in the same way as it is represented in Figure 4.12.

On the other hand, in order to update the representation of each site, a **pair-level attention mechanism** is used. This will fix a certain site in all present sequence pairs and update the information between them.

With this, both of these mechanisms will go through all the rows and columns of the matrix so that the information is updated by the entirety of it. It's important to mention that the row attention and the column attention represent the site-level and pair-level attentions, respectively.

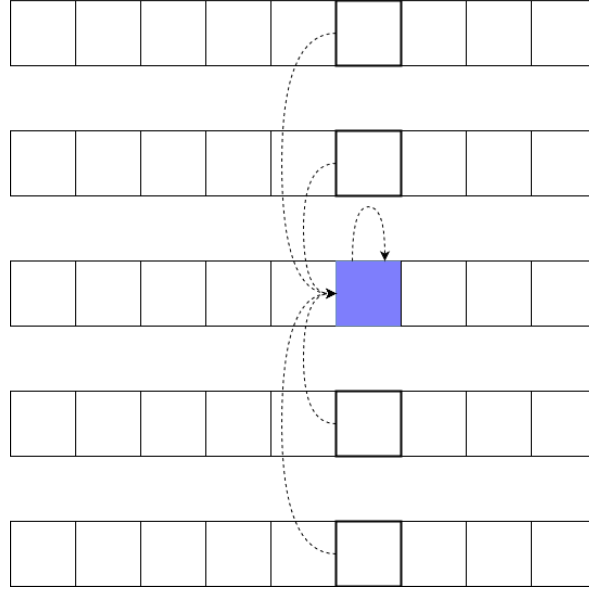


Figure 4.13: Pair-level attention.

The learning logic in the axial attention blocks follows the linear self-attention mechanism. This was chosen due to its **scalability**, something that self-attention did not present. To apply this mechanism, a linear layer and the application of the ELU activation function were used for each of the matrices, query, key, and value. Subsequently, the separation of the different channels used by the different attention heads is carried out. It's important to mention that these channels were adapted to the alphabet size currently used.

Between each learning layer of the neural model, it is verified that the sum between the output and input of that same layer is made so that there is more information to learn in the next layer. This allows the neural model to always have the maximum amount of information possible in each layer for the learning process.

The last layer of each transformer block, position-wise feed forward network, is made up of two concatenated 2D convolutional layers, with a hidden dimension of 256, and the GELU activation function is applied between them along with a dropout layer. The block is finished with the application of another dropout layer on the output of the convolutional layers.

During this block all the transpose/permute allow to apply normalization and attention over the desired dimensions and are then followed by the inverse transposition/permutation.

#### 4.3.2.3 Site-Wise Averaging Procedure

The last block of the neural model will handle condensing all the learned information into a smaller tensor, through which the trained model will be obtained. This is made up of a 2D convolutional layer followed by a dropout layer and a softplus activation function.

This process begins with the application of the mentioned layers on the tensor obtained from the output of the axial attention blocks, obtaining a shape of (4, 1, number of pairs, sequence length). The feature dimension goes from 64 to 1 because the data has just passed through the last layer of the neural model.

On this tensor, the mean method is applied and then the squeeze to condense the learned information, resulting in a shape of (4, number of pairs), where 4 is the batch size.

Subsequently, the tensor obtained from this process will serve as the basis for the trained model that is used for the prediction phase.

## 4.4 Prediction Phase

With a trained model, it is possible to move on to the **prediction phase**, which is responsible for predicting and inferring phylogenetic trees from a dataset, both in sequences and in typing data.

As depicted in Figure 4.14, this process begins with the input data going through the pre-processing phase, obtaining the tensor of the encoding of these values. This is passed to the **pre-trained model** in order to obtain the corresponding distance matrix. From this matrix, the tree is inferred using a common distance-based method, for example, **neighbour joining**.

With the inferred tree, it is necessary to confirm whether this process was indeed successful or not. For this, we resort to comparing with other trees inferred by known algorithms, as they can use the same one that we used or others. However, the matrices that give rise to these trees are obtained through other common methods.

To carry out this comparison, a metric is used to compute a value that will represent the comparison of the structure of both trees. In this case, the default metric to be used is **Robinson-Foulds(RF)**, which, when comparing the trees, will output a measure that has the objective to quantify the dissimilarity between the two.



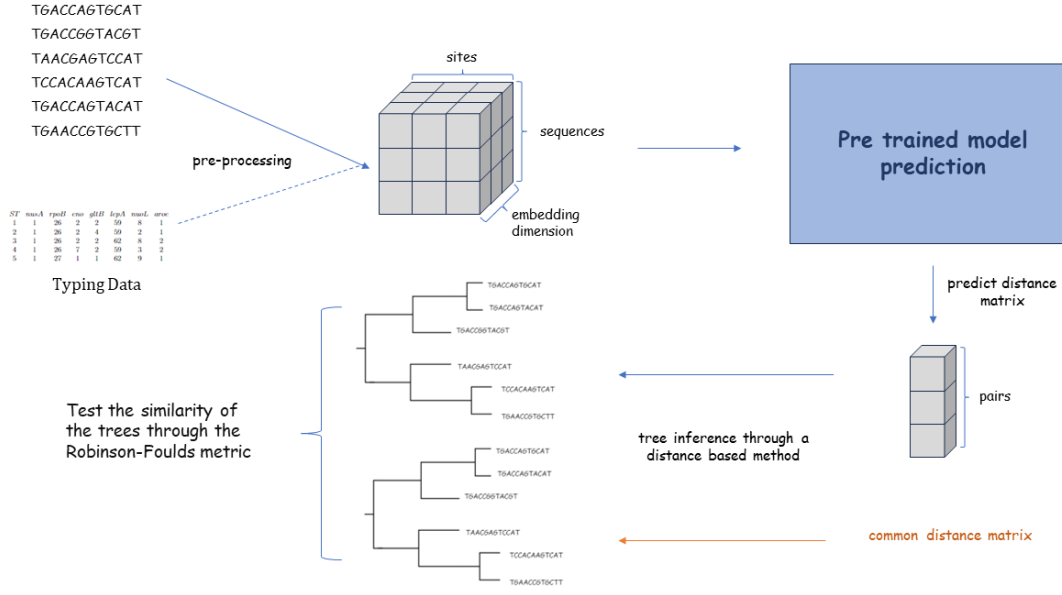


Figure 4.14: Prediction process pipeline - The data is submitted to the pre-processing phase and in turn passed to the pre-trained model. With this, the distance matrix is obtained from which it is possible to infer the phylogenetic tree. Subsequently, it is possible to compare the inferred tree with one whose distance matrix was obtained using a traditional method.

#### 4.4.1 Inference Process

The inference process, for the distance based methods, is divided into two different parts: one that predicts the **distance matrix** and one that infers the **phylogenetic tree**.

Starting with the one that predicts the distance matrix, this is constructed through the distances predicted by the pre-trained model. The process consists of iterating over an empty matrix and for each position inserting the value calculated by the model. This iterated matrix will have a shape of (number of sequences, number of sequences).

To evaluate the distances between the different sequences, an iteration is made with two indices, each corresponding to a different sequence. In this way, whenever these indices correspond to the same sequences, then the matrix is populated with 0 and all the values present in the upper triangular of the matrix will be mirrored to the lower triangular, as the distances between these two sequences are the same regardless of the order in which they appear. For example: the distance between [ACCT, ACCG] is equal to the distance between [ACCG, ACCT].

It's important to mention that the values in Figure 4.15 are from our example men-

$$\begin{bmatrix} 0 & 2 & 5 \\ 2 & 0 & 6 \\ 5 & 6 & 0 \end{bmatrix}$$

Figure 4.15: Distance Matrix Example - using Hamming distance.

tioned at the beginning of this chapter. The distance matrix was generated using Hamming distance calculations for illustrative purposes, but we could have also provided an example with a distance matrix that was predicted by any other traditional method.

To conclude the inference process, we take the distance matrix produced earlier and use neighbour joining to infer the **phylogenetic tree**.

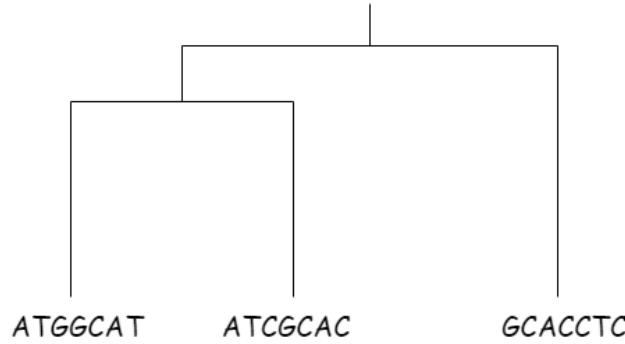


Figure 4.16: Phylogenetic Tree Example.

## 4.5 DL4Phylo vs Phyloformer

This section focuses on describing all the changes and adaptations made to Phyloformer. The three major changes are: the adaptation of the input data to receive typing data, allowing a new sequence simulator to be used and the refactoring of certain areas of the analyzed code. Besides these changes, it was also analyzed that the illustration of the neural network architecture did not match the implemented one. It was therefore modified.

### 4.5.1 Typing Data Extension

As mentioned in the Chapter 3, the Typing Data format consists of the identifiers of genes corresponding to a certain area of the sequence. This type of data present a set of numbers for each sequence that represent the identifiers, with their comparison being made between lines.

Therefore, it will be necessary to reformulate the Pre-processing phase and the process of generating input data for the training of the neural model.

#### 4.5.1.1 Pre-processing

In Pre-processing, the goal remains the same: to encode the input data into a binary representation, so that the machine can process it. Since the data will no longer be sequences of amino acids or nucleotides, the encoding method through the one-hot encoding algorithm no longer applies. Thus, in this case, the encoding of the gene identifiers will be done through the binary encoding of these same numbers.

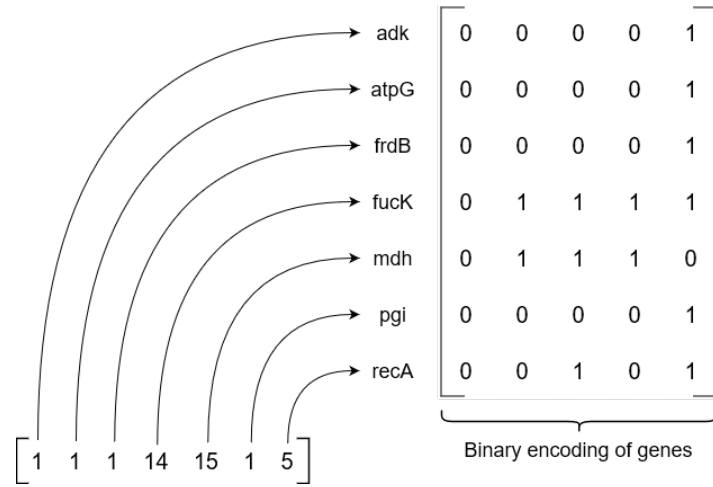


Figure 4.17: Binary encoding for one sequence, originating a matrix with shape (genome length \* 32). The encoding size of 32 is omitted from the figure, and instead, a size of 5 is used for simplicity.

At this moment, the encoding size has a fixed value of 32 bits to allow the encoding of all possible integers, but in the future one of the goals is to generalize this size to allow dynamic encoding, that is, the encoding size varies according to the file used. In this way, the matrix in Figure 4.17 will present a shape of (genome length \* 32).

Subsequently, the transpose and reshaping of this matrix is carried out, turning it into a 3-dimensional one, just as it happens in the encoding of nucleotide and amino acid sequences.

With all the encoded sequences, the concatenation is performed, forming a tensor with shape (32 \* number of sequences \* genome length) represented in Figure 4.19.

To finish this phase it's simply transposed the number of sequences with the genome length.

With the final tensor in Figure 4.20, it is possible to verify that the same data structure used by the phyloformer was obtained, this time for another type of data. Therefore,

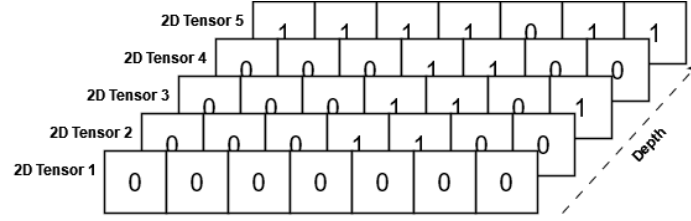


Figure 4.18: Binary encoded tensor reshaped to three dimensions - shape (32 \* 1 \* genome length).

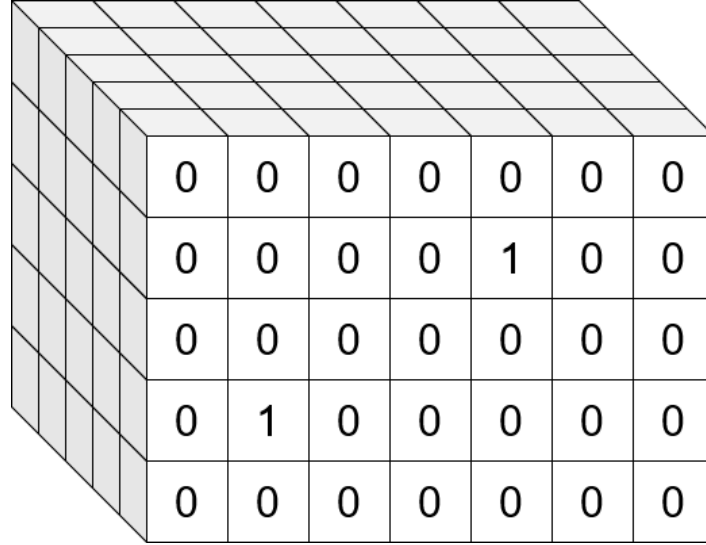


Figure 4.19: Tensor obtained after concatenating every binary encoded tensor.

both the training of the neural model and the prediction made of the pairwise distances do not need to undergo big changes about this adaptation.

#### 4.5.1.2 Training Phase

One of the necessary changes is the passage of typing data files to the neural model instead of continuing to use the sequences. In this way, the phase starts by simulating the phylogenetic trees, then generate the corresponding alignments and finally convert the sequences to typing data, which are then converted into tensors and passed to the model. Due to the unavailability of a typing data simulator, ones logic was implemented based on the approach used for GrapeTree [35].

The entire process up to the generation of sequences is similar to that defined in section 4.3, however, it will be necessary to convert these sequences into typing data. Since each gene is made up of blocks of nucleotides or amino acids, a possible approach is to break down the sequences generated from the trees and assign identifiers to each partition.

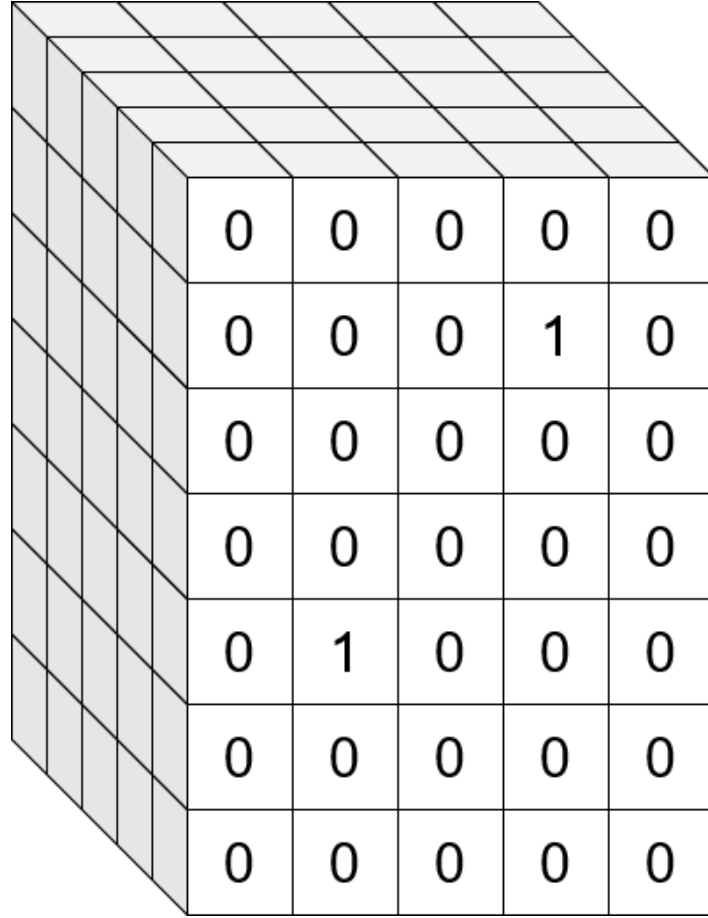


Figure 4.20: Final tensor obtained in the pre-processing phase, similar to the one obtained by the phyloformer.

This partitioning process will be illustrated using the sequences from Table 4.2.

taxon0	ATGCTTAAGTCCGGCTAATGTA
taxon1	ATGCAAAAGTCCATTAAAGGTT
taxon2	ATGCTACCAGCCGCAACGTGTA

Table 4.2: Example of a sequence alignment used in the partition testing.

These sequences are broken down into blocks, which in this example will have a size of 4 for simplification. As the goal is to identify genes from a sequence that encodes them, it is necessary to take into account the intergenic regions. In this way, the partition of these sequences will have intervals between each block, with these intervals being in this illustration. In addition, it is important to mention that each column of blocks refers to a different gene, with the identifiers of that column being unique to that gene.

In Figure 4.21, it is possible to observe the partition process and the generation of the corresponding identifiers.

After this process, the typing data files are obtained, which in this case will generate the values present in Figure 4.22.

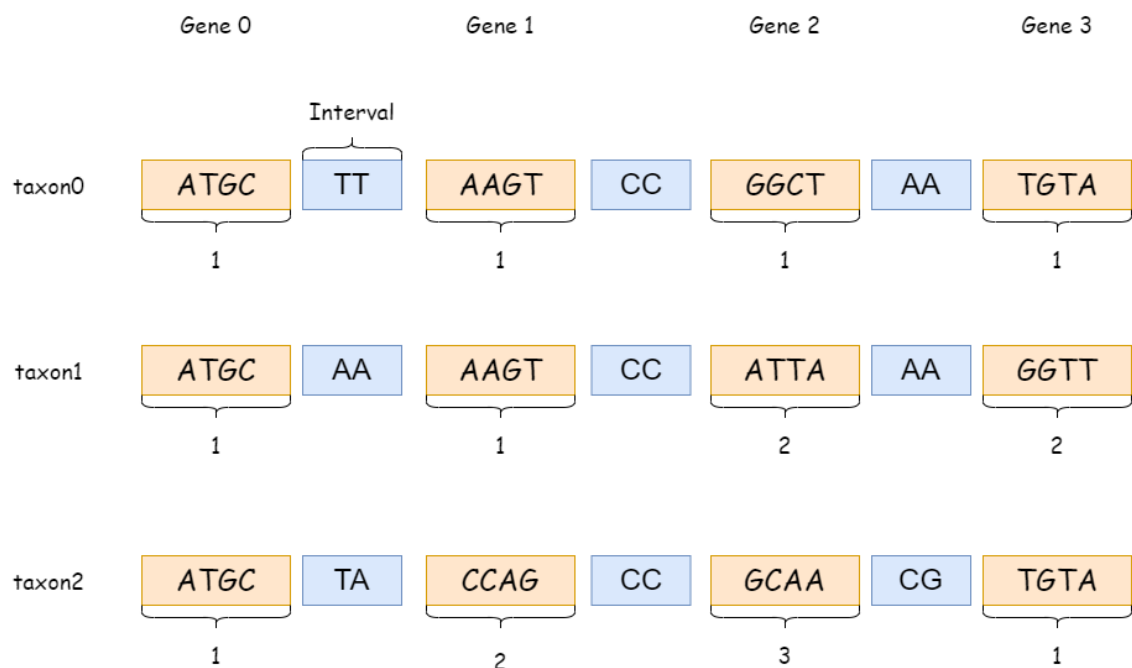


Figure 4.21: Partitioning of sequences into different blocks in order to convert these into typing data. In each column, each block represents a gene, and therefore different blocks will have different identifiers for the same gene. The intervals are referred to as intergenic zones. These zones mark the end of one gene's coding sequence and the beginning of the next, which is why they are discarded.

Finally, these are converted into tensors and used for training the neural model.

## 4.5.2 Alignment Blocks Extension

In addition to adapting Typing Data, it is also possible to use the same format, meaning partitioning the alignment sequences into blocks, but instead of using the corresponding gene identifier, the sequence associated with that identifier is used.

This way of handling the data comes with two different approaches:

- A '-' is introduced between each block so that the model can identify where one ends and the next begins;
- Each block is encoded according to its size to identify the separations, with all blocks having the same size.

The first approach is quite simple, as it only requires introducing a '-' during the separation of sequences into blocks whenever one is identified. Figure 4.23 shows an

ST	Gene 0	Gene 1	Gene 2	Gene 3
1	1	1	1	1
2	1	1	2	2
3	1	2	3	1

Figure 4.22: Typing Data file obtained.

example of the partitioning done using this approach on the same sequences shown in Table 4.2.

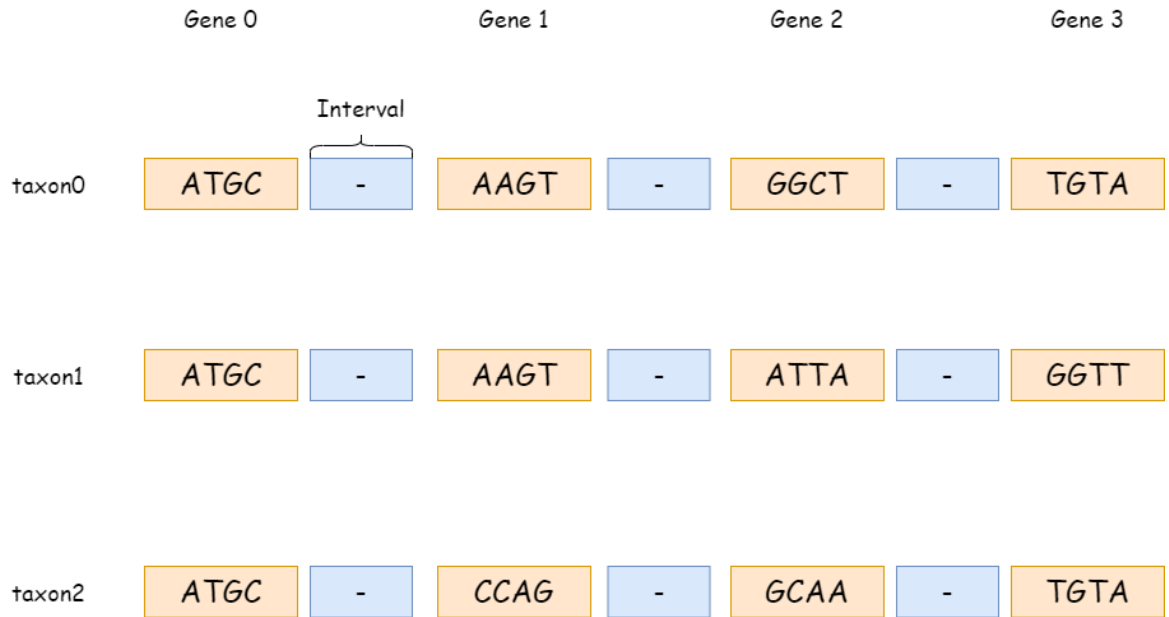


Figure 4.23: Sequence separated by blocks with the '-' as the separator.

In the second approach, it is necessary to change the way the associated data is encoded. For each block, the size must be considered, and the encoding must be done block by block. Thus, the Pre-processing phase will be similar to that used for encoding any type of data, except that the size used for encoding will be the depth of the tensor obtained. Following the example in Table 4.2, and considering that they are composed of nucleotides, if the separation is done in blocks of 8 with intervals of 3, then the depth of the produced tensor corresponds to:

$$AlphabetSize \times BlockSize = 4 \times 8 = 32 \quad (4.5)$$

### 4.5.3 Code Refactoring

In addition to adapting the type of input data, some changes were also made to the code provided by Phyloformer. These changes involve both parameterization of values that were fixed and removal of empty variables.

These variables were present in the file responsible for the structure and behavior of the neural model, where it was possible to observe a list, with None values, being returned by this model, never being used again.

In addition to this refactoring of the Phyloformer files, a generalization of the alphabet used for the pre-processing phase of the data was also made. Previously, everything was used exclusively for amino acids, but it was given the possibility to select between these and the nucleotides.

## 4.6 Implementation Details

This tool was being developed in **Visual Studio Code** using **Python**, with a minimum required version of 3.9. The dependencies used are as follows:

- scipy - version 1.13.0
- numpy - version 1.24.2
- ete3 - version 3.1.3
- biopython - version 1.83
- dendropy - version 5.0.0
- scikit-bio - version 0.6.0
- scikit-learn - version 1.5.0
- tqdm - version 4.66.2

In addition to these dependencies, the one that has the greatest impact on the development of the neural network, Pytorch, is also being used:

- torch - version 2.2.2



If the goal is to train the model using Pytorch [33] with CUDA [36], it is necessary to install:

- torch - version 2.2.2+cu121
- torchaudio - version 2.2.2+cu121
- torchvision - version 0.17.2+cu121

As simulators, **Seq-Gen** [37], **SimBac** [38], and **ETE3** [34] are used. Seq-Gen is responsible for generating random sequences based on the tree structure, SimBac can generate the complete dataset, meaning both the trees and the sequences, and ETE3 only simulates phylogenetic trees.

The project has several scripts with different functionalities implemented. The execution of the instructions and the dependencies are explained in the README file of the project repository. This repository is available at <https://github.com/phyloLearn/DL4Phylo>.



# Chapter 5

## Experimental Evaluation

The training approach followed was a mix between random and Bayes search through a set of alternative values for each hyperparameter that diverged slightly from the set that lead to the best results in the original Phyloformer. Furthermore, the datasets used also followed a similar architecture to the latter as much as possible where the differences mainly originated from the inherently different data at hand and the lack of clarity in the dataset generation process of the Phyloformer. Since we are interested in comparing our results with the Phyloformer, we also tried to replicate its results with sequence data datasets. Therefore, 8 datasets were used, namely, 5k-PAM-200, 10k-PAM-200 (200 amino acids sequence data generated by Seq-Gen with PAM model), 5k-F84-200, 10k-F84-200 (200 nucleotides sequence data generated by Seq-Gen with F84 model), 5k-SimBac-7-600, 10k-SimBac-7-600 (7 blocks of 600 nucleotides typing data generated by SimBac), 5k-SimBac-1000-600 and 10k-SimBac-1000-600 (1000 blocks of 600 nucleotides typing data). The dichotomy between 5k and 10k is justified by our further interest in understanding how the results evolve with the enlargement of the dataset used. Note that the original Phyloformer used 100k sized dataset during training. Due to storage and computational limitations we were not capable of training datasets of this size.

To pick the best models for the analysis of the average and best results we used the simple method of selecting the 10 models with the lowest training loss, whenever possible and after eliminating overfitted models. Table 5.2 shows the average results for each of these sets of picks. This method assumes a significant enough correlation between the training loss of each models and the other metrics tracked by us, namely, validation loss, MAE and MRE. Note that the validation MAE and MRE necessarily correlate with both the training and validation losses since the set of alternative values for the loss hyperparameter search is composed by the L1 (MAE) and the L2 (MRE) losses.

Through the analysis of Table 5.1, it is evident that the obtained results for the typing

Table 5.1: Results of the best models trained on sequence data (amino acids and nucleotides) and typing data datasets. Metrics include training time, loss, MAE and MRE. It also shows the normalized Robinson-Foulds (RF), on a test dataset never seen during training, for the distance-based methods: Neighbor Joining (NJ), Unweighted Pair Group Method with Arithmetic Mean (UPGMA), and Single Linkage (SL), indicating the accuracy of each method by comparing the inferred tree to the true tree (lower is better).

Dataset	Train		Validation			Test (500 samples)		
	Time (s)	Loss	Loss	MAE	MRE	NJ-RF	UPGMA-RF	SL-RF
Sequence (Amino Acids)								
5k-PAM-200	37741	0.28	0.28	0.39	0.11	0.19	0.17	0.26
10k-PAM-200	12088	0.13	0.15	0.29	0.08	<b>0.18</b>	<b>0.13</b>	<b>0.23</b>
Sequence (Nucleotides)								
5k-F84-200	26336	0.75	0.8	0.8	0.22	0.23	0.18	0.28
10k-F84-200	5496	0.81	0.84	0.84	0.25	0.23	0.19	0.28
Typing								
5k-SimBac-7-600	478	1.34	1.4	1.4	7.4	0.99	0.98	0.98
10k-SimBac-7-600	2583	1.34	1.32	1.32	10	0.98	0.94	0.93
5k-SimBac-1000-600	1909	1.38	1.42	1.42	12.5	0.98	0.99	0.99
10k-SimBac-1000-600	76095	2.99	3.38	1.36	96.34	0.96	0.97	0.97

data are significantly worse from the ones obtained for the sequence data and ultimately unusable in real-world scenarios. However, it is important to mention that a significant portion of the trained models for the typing data never stabilized their training losses whilst not overfitting. Nearly all the typing data trained models show a training loss graph with a fairly steep negative slope, i.e., exhibiting the potential to further decrease their losses, as shown in Figure 5.1.

Another aspect to consider is that of the site-specific rate of internal recombination and the site-specific mutation rate of the sequence generator used, SimBac [38], which were both set to 0.001. This may have lead to genome variation not adequate nor realistic for the task at hand. In further studies we recommend trying to increase and diverge them.

Furthermore, it is important to also consider the difference in complexity between the sequence data and the typing data used in order to not take the comparison at face value. Whilst they are inherently related to each other, we were interested in simulating chains of genomes which significantly increases the complexity of the sequences that precede the typings when compared with sequence lengths of 200 used in the Phyloformer. In further studies we recommend trying to compare sequence data with lengths similar to the number of blocks, i.e., genomes, in the typing data.

Regarding the sequence data results we were capable of attesting the Phyloformer

Table 5.2: Average results of the best 10 models for each dataset, whenever possible.

Dataset	Samples	Train		Validation		
		Time (s)	Loss	Loss	MAE	MRE
Sequence (Amino Acids)						
5k-PAM-200	7	20278.9	0.61	0.62	0.66	0.24
10k-PAM-200	6	33570.2	0.98	1.02	0.7	0.26
Sequence (Nucleotides)						
5k-F84-200	10	25967.1	1.45	1.47	1.1	1.42
10k-F84-200	10	16175.3	1.11	1.22	1.07	0.4
Typing						
5k-SimBac-7-600	10	2134.3	1.36	1.4	1.4	7.07
10k-SimBac-7-600	10	7144.7	1.34	1.32	1.32	9.46
5k-SimBac-1000-600	2	36987.5	2.66	2.77	1.46	14.89
10k-SimBac-1000-600	1	-	-	-	-	-

results since the amino acids produced the best results overall. They are necessarily lower in quality than the ones reported in the respective paper but this was expected since our datasets are smaller. The results for the nucleotides were also high in quality and close to the latter with a difference in the normalized Robinson-Foulds of about 0.05, 0.06 and 0.05 for the distance-based methods NJ, UPGMA and SL, respectively. Note that for the sequence data we used a different sequence generator, Seq-Gen [37].

When observing the difference between the results of the 5k datasets and the 10k datasets we can conclude that more data does lead to better results overall.

Finally, we report that, whilst sequence data tends to generate the best results using the  $L_2$  loss, typing data tends to generate the best results using  $L_1$  loss. The latter is somewhat expected due the increased complexity of the typing data in comparison to the sequence data used. It may be interesting to check whether this trend is observed when the complexity between the datasets used is not as divergent. Of all hyperparameters considered, the number of axial attention blocks was the one that correlated the most with the training loss, where by increasing the number of blocks the loss decreases with a correlation factor of 52.7%.

This evaluation conducted for typing data was also used for the submission of an article on the **Inforum**, aiming to describe and explain the purpose and functionalities of DL4Phylo.

In addition to these tests, attempts were also made to perform the same on data from sequences partitioned into blocks. However, the storage capacity is insufficient, and the hardware is not powerful enough to run such tests. Consequently, it was concluded that it would be necessary to reduce the size of the tests conducted on this type of data, which would defeat the purpose of comparing them with the others.

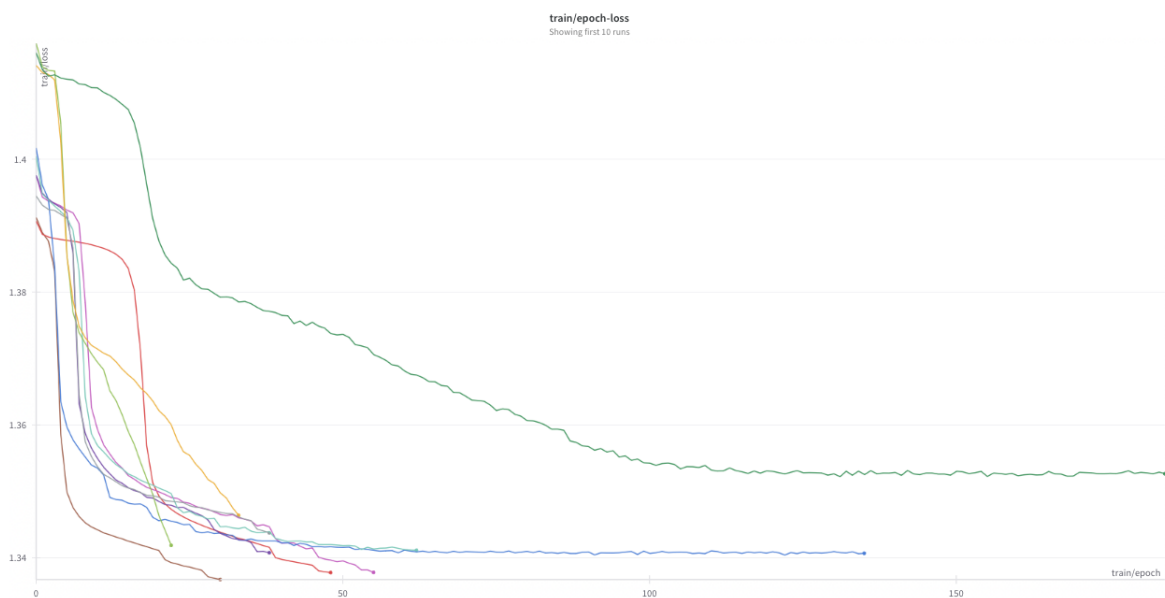


Figure 5.1: Training loss over epochs for the top models trained on typing data. Only the 10 models with the lowest final loss are displayed.

# Chapter 6

## Final Notes

Throughout this project, we have studied phylogenetic inference, analyzed Phyloformer, and replicated its team’s results, while also obtaining additional ones. We also explored the Python programming language and various libraries, with an emphasis on PyTorch. This allowed us to delve into the world of artificial intelligence, previously unfamiliar to us, and gain a general understanding of contemporary and potentially future deep learning techniques.

Having concluded the development of DL4Phylo, we made the use of Typing Data a possibility. Unfortunately, the results obtained were far from ideal. Since only minimal testing was conducted, further testing is necessary. We will continue working on this by trying different recombination and mutation rate values for various data sets, as little to no changes were observed in the sequences tested.

Regarding the decoder that was to be implemented, after thoroughly analyzing Phyloformer, we concluded that it was unnecessary, as Phyloformer did not utilize one to begin with, making its implementation pointless.

The insertion of the Alignment Blocks extension also presented some challenges. Hardware limitations made it impossible to test realistic sets of values, and only very small dimensions could be tested. Due to time constraints and the extensions not being our main priority, as they were optional work, further testing has not been done. However, their use remains possible within our tool, but additional testing should be conducted, either by us in the future or by someone with high-end hardware and storage capability. Taking into account our results, we plan also to incorporate a tree comparison metric in the objective loss function to be optimized in the model training.

Finally, DL4Phylo has been uploaded as a Python package to PyPi and is available at the following link: <https://pypi.org/project/DL4Phylo/>.





# References

- [1] Cátia Vaz, Marta Nascimento, João A Carriço, Tatiana Rocher, and Alexandre P Francisco. “Distance-based phylogenetic inference from typing data: a unifying view”. In: *Briefings in Bioinformatics* 22.3 (July 2020), bbaa147. ISSN: 1477-4054. DOI: 10.1093/bib/bbaa147. eprint: <https://academic.oup.com/bib/article-pdf/22/3/bbaa147/38165363/bbaa147.pdf>. URL: <https://doi.org/10.1093/bib/bbaa147>.
- [2] André Jesus, Nyckollas Brandão, and André Páscoa. *Phyloviz Web Platform*. Available: <https://github.com/phyloviz/phyloviz-web-platform/blob/master/docs/final-presentation.pdf>. Instituto Superior de Engenharia de Lisboa, 2023.
- [3] Luca Nesterenko, Bastien Boussau, and Laurent Jacob. “Phyloformer: towards fast and accurate phylogeny estimation with self-attention networks”. In: *bioRxiv* (2022), pp. 2022–06.
- [4] Zhicheng Wang et al. “Fusang: a framework for phylogenetic tree inference via deep learning”. In: *Nucleic Acids Research* 51.20 (2023), pp. 10909–10923.
- [5] Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João A Carriço. “Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach”. In: *BMC bioinformatics* 10 (2009), pp. 1–15. DOI: 10.1186/1471-2105-10-152.
- [6] David F Robinson and Leslie R Foulds. “Comparison of phylogenetic trees”. In: *Mathematical biosciences* 53.1-2 (1981), pp. 131–147.
- [7] Masatoshi Nei and Sudhir Kumar. *Molecular evolution and phylogenetics*. Oxford university press, 2000.
- [8] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.
- [9] D Ashley Robinson, Edward J Feil, and Daniel Falush. *Bacterial population genetics in infectious disease*. John Wiley & Sons, 2010.
- [10] Cátia Vaz et al. “TypOn: the microbial typing ontology”. In: *Journal of Biomedical Semantics* 5 (2014), pp. 1–11.
- [11] Zhemín Zhou et al. “The EnteroBase user’s guide, with case studies on Salmonella transmissions, Yersinia pestis phylogeny, and Escherichia core genomic diversity”. In: *Genome research* 30.1 (2020), pp. 138–152.

- [12] National Institutes of Health et al. “What are single nucleotide polymorphisms (SNPs)”. In: *Genetics Home Reference-NIH. US National Library of Medicine* (2019).
- [13] Santosh Kumar, Travis W Banks, and Sylvie Cloutier. “SNP discovery through next-generation sequencing and its applications”. In: *International journal of plant genomics* 2012 (2012).
- [14] Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013.
- [15] STUDIER JA. “A note on the neighbor-joining method of Saitou and Nei”. In: *Mol Biol Evol* 5 (1988), pp. 729–731.
- [16] Manon Ragonnet-Cronin et al. “Automated analysis of phylogenetic clusters”. In: *BMC bioinformatics* 14 (2013), pp. 1–10.
- [17] *What is a Neural Network? — IBM — ibm.com*. <https://www.ibm.com/topics/neural-networks>. [Accessed 02-06-2024]. 2021.
- [18] Coursera Staff. *What Is a Hidden Layer in a Neural Network? — coursera.org*. <https://www.coursera.org/articles/hidden-layer-neural-network>. [Accessed 03-06-2024]. 2024.
- [19] Keiron O’shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).
- [20] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).
- [21] Nilesch Vijayarania. *Different Normalization Layers in Deep Learning — towardsdatascience.com*. <https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6>. [Accessed 03-06-2024]. 2020.
- [22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [23] Jason Brownlee. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks - MachineLearningMastery.com — machinelearningmastery.com*. <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>. [Accessed 03-06-2024]. 2019.
- [24] Moez All. *Introduction to Activation Functions in Neural Networks — datacamp.com*. <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>. [Accessed 03-06-2024]. 2023.
- [25] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. “Activation functions in neural networks”. In: *Towards Data Sci* 6.12 (2017), pp. 310–316.
- [26] Kelei Sun, Jiaming Yu, Li Zhang, and Zhiheng Dong. “A convolutional neural network model based on improved softplus activation function”. In: *International Conference on Applications and Techniques in Cyber Intelligence ATCI 2019: Applications and Techniques in Cyber Intelligence 7*. Springer. 2020, pp. 1326–1335.
- [27] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

- [28] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [29] Jay Alammam. *The Illustrated Transformer* — [jalammam.github.io. https://jalammam.github.io/illustrated-transformer/](https://jalammam.github.io/illustrated-transformer/). [Accessed 03-06-2024]. 2018.
- [30] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. “Axial attention in multidimensional transformers”. In: *arXiv preprint arXiv:1912.12180* (2019).
- [31] Gavin R Finnie and Gerhard E Wittig. “AI tools for software development effort estimation”. In: *Proceedings 1996 International Conference Software Engineering: Education and Practice*. IEEE. 1996, pp. 346–353.
- [32] Jitendra Singh Kushwah, Atul Kumar, Subhash Patel, Rishi Soni, Amol Gawande, and Shyam Gupta. “Comparative study of regressor and classifier with decision tree using modern tools”. In: (2022).
- [33] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. “PyTorch”. In: *Programming with TensorFlow: Solution for Edge Computing Applications* (2021), pp. 87–104.
- [34] Jaime Huerta-Cepas, François Serra, and Peer Bork. “ETE 3: reconstruction, analysis, and visualization of phylogenomic data”. In: *Molecular biology and evolution* 33.6 (2016), pp. 1635–1638.
- [35] Zheming Zhou et al. “GrapeTree: visualization of core genomic relationships among 100,000 bacterial pathogens”. In: *Genome research* 28.9 (2018), pp. 1395–1404.
- [36] Rob Farber. *CUDA application design and development*. Elsevier, 2011.
- [37] Andrew Rambaut and Nicholas C Grass. “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees”. In: *Bioinformatics* 13.3 (1997), pp. 235–238.
- [38] Thomas Brown, Xavier Didelot, Daniel J Wilson, and Nicola De Maio. “SimBac: simulation of whole bacterial genomes with homologous recombination”. In: *Microbial genomics* 2.1 (2016), e000044.