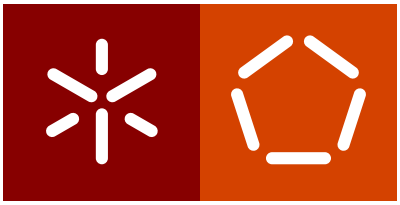
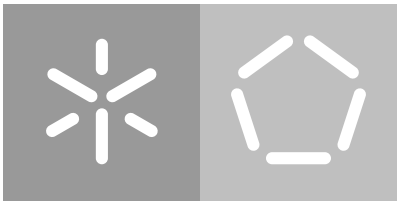


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Miguel Ângelo Pereira Barros

Development of a deep learning-based computational framework for the classification of protein sequences

December 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Miguel Ângelo Pereira Barros

Development of a deep learning-based computational framework for the classification of protein sequences

Master dissertation
Master Degree in Bioinformatics

Dissertation supervised by
Miguel Francisco Almeida Pereira da Rocha
Óscar Manuel Lima Dias

December 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

ACKNOWLEDGEMENTS

The submission of the dissertation symbolizes the conclusion of a major step in my academic journey. As I reach the end of this milestone, I would like to thank the contribution of those that supported me during this journey.

Firstly, I would like to address a special acknowledgement to my supervisors. To Dr. Miguel Rocha and Dr. Óscar Dias, for their advice, help, guidance and availability during the past year. Without their suggestions and knowledge, this thesis would not have been possible. For their trust in my work and the freedom to explore new areas of knowledge that contributed to my personal growth as a bioinformatician.

To the OmniumAI, for allowing me to develop this thesis in collaboration with the company, and also, to all the collaborators for sharing their knowledge. I would like to give a special acknowledgement to Ana Marta Sequeira, a PhD student and collaborator of OmniumAI. Thanks for all the patience, kindness, support and time that you dedicated to guiding me in this area of knowledge.

To my family, for their support, care and love. Without them, I couldn't complete my academic studies. To my girlfriend Alexandra, for all the patience, love and care. Thank you for always being there for me.

To my bioinformatician buddies at the University of Minho, in particular Mónica Fernandes, Tiago Machado and Tiago Silva. Thank you for keeping me sane and for the 16 hours calls. To my buddies from Biochemistry, in particular Inês and João. Thank you for remembering me that sometimes it's necessary to hit the brakes. And finally, to my longtime friends that are always ready for a cold one.

Thank you to all that in one way or another helped me during this journey!

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Proteins are one of the more important biological structures in living organisms, since they perform multiple biological functions. Each protein has different characteristics and properties, which can be employed in many industries, such as industrial biotechnology, clinical applications, among others, demonstrating a positive impact.

Modern high-throughput methods allow protein sequencing, which provides the protein sequence data. Machine learning methodologies are applied to characterize proteins using information of the protein sequence. However, a major problem associated with this method is how to properly encode the protein sequences without losing the biological relationship between the amino acid residues. The transformation of the protein sequence into a numeric representation is done by encoder methods. In this sense, the main objective of this project is to study different encoders and identify the methods which yield the best biological representation of the protein sequences, when used in machine learning (ML) models to predict different labels related to their function.

The methods were analyzed in two study cases. The first is related to enzymes, since they are a well-established case in the literature. The second used transporter sequences, a lesser studied case in the literature. In both cases, the data was collected from the curated database Swiss-Prot. The encoders that were tested include: calculated protein descriptors; matrix substitution methods; position-specific scoring matrices; and encoding by pre-trained transformer methods. The use of state-of-the-art pretrained transformers to encode protein sequences proved to be a good biological representation for subsequent application in state-of-the-art ML methods. Namely, the ESM-1b transformer achieved a Mathews correlation coefficient above 0.9 for any multiclassification task of the transporter classification system.

Keywords: Computational Biology, Protein Classification, Machine Learning, Deep Learning

RESUMO

As proteínas são estruturas biológicas importantes dos organismos vivos, uma vez que estas desempenham múltiplas funções biológicas. Cada proteína tem características e propriedades diferentes, que podem ser aplicadas em diversas indústrias, tais como a biotecnologia industrial, aplicações clínicas, entre outras, demonstrando um impacto positivo.

Os métodos modernos de alto rendimento permitem a sequenciação de proteínas, fornecendo dados da sequência proteica. Metodologias de aprendizagem de máquinas têm sido aplicadas para caracterizar as proteínas utilizando informação da sua sequência. Um problema associado a este método é como representar adequadamente as sequências proteicas sem perder a relação biológica entre os resíduos de aminoácidos. A transformação da sequência de proteínas numa representação numérica é feita por codificadores. Neste sentido, o principal objetivo deste projeto é estudar diferentes codificadores e identificar os métodos que produzem a melhor representação biológica das sequências proteicas, quando utilizados em modelos de aprendizagem mecânica para prever a classificação associada à sua função a sua função.

Os métodos foram analisados em dois casos de estudo. O primeiro caso foi baseado em enzimas, uma vez que são um caso bem estabelecido na literatura. O segundo, na utilização de proteínas de transportadores, um caso menos estudado na literatura. Em ambos os casos, os dados foram recolhidos a partir da base de dados curada Swiss-Prot. Os codificadores testados incluem: descritores de proteínas calculados; métodos de substituição por matrizes; matrizes de pontuação específicas da posição; e codificação por modelos de transformadores pré-treinados. A utilização de transformadores de última geração para codificar sequências de proteínas demonstrou ser uma boa representação biológica para aplicação subsequente em métodos ML de última geração. Nomeadamente, o transformador ESM-1b atingiu um coeficiente de correlação de Matthews acima de 0,9 para multiclassificação do sistema de classificação de proteínas transportadoras.

Palavras-chave: Biologia Computacional, Classificação de Proteínas, Aprendizagem de Máquina, Aprendizagem Profunda

CONTENTS

ACRONYMS

AE	Auto-encoders
AI	Artificial Intelligence
AutoML	Automated Machine Learning
ANN	Artificial Neural Network
BAcc	Balance Accuracy
Bi-RNN	Bidirectional Recurrent Neural Network
BLAST	Basic Local Alignment Search Tool
BLOSUM	Blocks Substitution Matrix
CART	Classification and Regression Trees
CHAID	Chi-square Automatic Interaction Detection
CNN	Convolutional Neural Network
CRF	Conditional Random Fields
DBN	Deep belief networks
DC	Domain Content
DL	Deep Learning
DNA	Deoxyribonucleic Acid
DNN	Deep Neural Network
DT	Decision Trees
EC	Enzyme Classification
ESM	Evolutionary Scale Modeling
FN	False Negatives
FP	False Positives

GNB	Gaussian Naive Bayes
GPUs	Graphics Processing Units
GRU	Gated Recurrent Unit
HE	Hot-Encoded Sequence
ICA	Independent Component Analysis
ID	Identification
ID3	Iterative Dichotomiser 3
KNN	k-Nearest Neighbors
LgR	Logistic Regression
LM	Language Modeling
LR	Linear Regression
LSTM	Long Short-Term Memory
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MLP	Multilayer Perceptron
NLF	Non-Linear Fisher
NLP	Natural Language Processing
NMF	Non-negative Matrix Factorization
PCA	Principal Component Analysis
PCF	Physicochemical Features
PPI	Protein-protein Interactions
Prec	Precison
PSI-BLAST	Position-Specific Iterative Basic Local Alignment Search Tool
PSFM	Position-Specific Frequency Matrix
PSSM	Position-Specific Scoring Matrices
RBF	Radial basis function

ReLU	rectified linear activation function
ResNet	Residual network
RF	Random Forest
RMSE	Root-mean-squared Error
RNA	Ribonucleic Acid
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SF	Structural Features
SMILES	Simplified Molecular-input Line-entry System
SSE	squared sum of errors
SVM	Support Vector Machine
TC	Transporter Classification
TCDB	Transporter Classification Database
TF	Transformer Features
TN	True Negatives
TP	True Positives
t-SNE	t-Distributed Stochastic Neighbor Embedding
W2V	Word2vec

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 MOTIVATION AND CONTEXT

Proteins are the main workhorses of living cells, since they carry different biological functions ranging from gene regulation, growth control, enzymatic catalysis, transport across membranes and many others. The knowledge of the protein characteristics and properties can have a wide implication in a variety of areas, which include industrial biotechnology, clinical applications, nutrition, and agriculture, among many others [1, 2].

The prevalence of high-throughput experimental methods has resulted in a large availability of biological data, such as protein sequences. The database's statistics reflect the increasing number of uncharacterized sequences, once experimental methods generally used to study these properties cannot scale up to the task due to associated cost and time [3, 4].

Although traditional machine learning methods encounter difficulties to learn complex data structures, they have been used in multiple applications, for example protein transcription factors, DNA-binding sites, protein function, etc. On the other hand, deep learning methods have shown to work well in several biological problems, with successful applications of these methods in learning representations from complex data. These motivated the usage of deep learning architectures to predict the properties and characterize proteins based on their sequences [5–8].

ProPythia is an open-source python package that was developed within the Biosystems group at CEB/ U. Minho, that allows the development of machine learning and deep learning approaches for protein sequence analysis and classification [9]. Still, the methods can still be improved as many recent approaches are still not implemented within this package.

1.2 OBJECTIVES

The main objective of this work is to develop a deep learning framework for sequence-based protein classification. The framework will be based on training and validating deep learning models based on protein sequences for the prediction of distinct biological activities.

The work will address the following scientific / technical objectives:

- Review relevant literature regarding deep learning models and their applications in protein classification tasks.
- Explore different deep learning frameworks for protein sequence representation and classification, assessing their performance in different case studies, including enzyme and transporter classification.
- Develop a framework that encompasses input representation/ encoding for protein classification, to be integrated within the ProPythia package.
- Develop a platform that allow to create deep learning based classifiers for protein sequences, including the features above, which will be integrated into OmniumAI's software platform.
- Writing the thesis with the main results of this work.

1.3 THESIS ORGANIZATION

This section is meant to explain the organization of this thesis. The thesis is organized in six chapters, a brief description of the chapters is given as follows:

- Chapter 1: it is composed by the motivation and main goals, which presents a brief description of the subject of this work.
- Chapter 2: this chapter consists of three main sections. The first section presents the theoretical foundation for the understanding of proteins and also a more detailed explanation of the study cases. The second section is composed by the theoretical foundations of machine and deep learning, where the main focus is to discuss existing algorithms and architectures. Lastly, the third section introduces the topic of protein sequence classification

by deep learning models. In this section is discuss the techniques for encoding the protein sequence, a review of relevant methods and computational tools for protein classification and relevant packages for the development of this thesis.

- Chapter 3: This chapter contains the methods used throughout the thesis. Along this chapter, it is explained the encoding methods used and the structure of the deep learning models.
- Chapter 4: The fourth chapter starts by stating the relationship between the developed work in thesis and the company. Then is explained the Automated Machine Learning (AutoML) pipeline created in the company. It is also described the implementation of protein encoders and the parameters used in deep learning methods used.
- Chapter 5: The fifth chapter presents the results from the methods used and the discussion of these results.
- Chapter 6: And lastly, conclusions taken from this dissertation are presented, as well as the future perspectives for this work.

2

STATE OF THE ART

2.1 PROTEINS

The improvement of high-throughput data acquisition technologies has been transforming biology into a data-rich science and, therefore, has caused an exponential growth in the available amount of biological data. Among the biological data available in databases is included protein sequences.

Proteins are polymeric macromolecules, which perform crucial functions across all domains of life. These macromolecules are composed of an evolution-derived combination of 23 amino acid residues linked together. In a cell, proteins can carry out different biological functions ranging from gene regulation, control of growth, enzymatic catalysis, and others. There are distinct families/groups of proteins based on their main function in the cell. Therefore, an opportunity for computational approaches was provided to develop methods to extrapolate the function of proteins based on the available data.

2.1.1 AMINO ACID RESIDUES

Proteins are polymeric macromolecules, which are composed of amino acid residues joined to their neighbor by a covalent bond. In total, there are twenty-six different amino acid residues, although only twenty are commonly found in proteins.

All twenty common amino acids (standard amino acids) are α-amino acids, meaning they are bonded to the α-carbon, specifically a carboxyl group and an amino group carbon. These monomeric molecules differ from each other by having different side chains bounded to the

α-carbon, or R group. These groups alter the amino acid properties, such as structure, size, and electric charge. The α-carbon is also connected to a hydrogen atom [10, 11].

Due to the tetrahedral arrangement of the four bounded groups, the α-carbon is a chiral center and, consequently, each one of them can only occupy two unique spatial arrangements, as represented in figure 1. The R group is the main responsible for the chemical properties of the common amino acid residues, being these properties essential for biochemistry understanding.

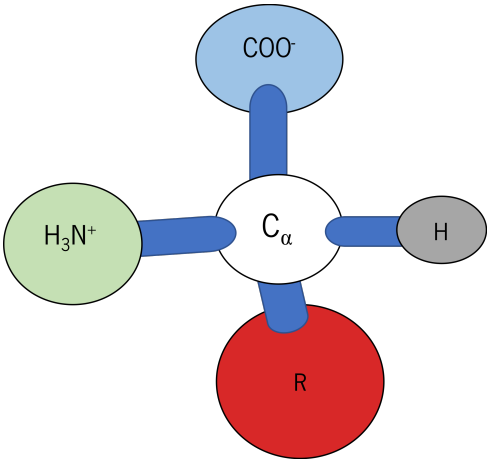


Figure 1 – Tetrahedral arrangement of an amino acid residue. A common structure to almost all α-amino acids. The side chain (red), R group, is attached to the α-carbon. Adapted from [10].

Based on the R group polarity or tendency to interact with water at neutral pH, each amino acid is categorized into one of five main classes: nonpolar and aliphatic, aromatic, positively charged, polar and uncharged, and negatively charged. Polarity can vary from nonpolar or hydrophobic to highly polar or hydrophilic.

In order to easily identify the amino acids in peptide or protein sequences, a three-letter code was created, also called abbreviation, generally consisting of the first three letters of the amino acid name. Also, a one-letter code or symbol was devised by Margaret Oakley Dayhoff, reflecting an attempt to reduce the size of data files. Both codes for the amino acid name can be seen in Table 1 [10, 11].

Table 1 – Amino acid residue and their codes. Representation of the amino acid residue name, abbreviation and symbol.Adapted from [10] and [11]

Amino acid	Abbreviation	Symbol
Nonpolar, Aliphatic R groups		
Glycine	Gly	G
Alanine	Ala	A
Proline	Pro	P
Valine	Val	V
Leucine	Leu	L
Isoleucine	Ile	I
Methionine	Met	M
Aromatic R groups		
Phenylalanine	Phe	F
Tyrosine	Tyr	Y
Tryptophan	Trp	W
Polar, Uncharged R groups		
Serine	Ser	S
Threonine	Thr	T
Cysteine	Cys	C
Asparagine	Asn	N
Glutamine	Gln	Q
Positively charged R groups		
Lysine	Lys	K
Histidine	His	H
Arginine	Arg	R
Negatively charged R groups		
Aspartate	Asp	D
Glutamate	Glu	E

2.1.2 From amino acids to proteins

To transform the monomers of amino acid residues into peptides or proteins, the amino acids need to be covalently joined through a substitute amide linkage, termed peptide bond. This linkage is formed by a condensation reaction that removes a water molecule from the α-carboxyl of one amino acid and the α-amino group of another, as shown in Figure 2.

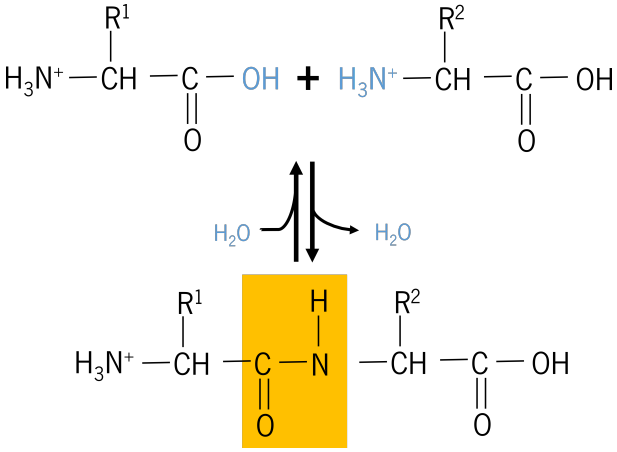


Figure 2 – Reaction of condensation to form a peptide. The α-amino group of one amino acid residue removes the hydroxyl group (OH-) from the other, forming the peptide bond. Adapted from [10].

When two amino acids are joined by a peptide bond, a dipeptide is formed, while if three amino acids are linked together by two peptide bonds, a tripeptide is formed. Similarly, four amino acids joined form a tetrapeptide, five amino acids joined form a pentapeptide, and so on. When a small number of amino acids are joined together the structure is nominated an oligopeptide, but if it is constituted by a greater number it is indicated as a polypeptide. To be classified as proteins, these molecules must have a molecular weight over 10,000 Dalton.

A protein or any molecule derived from the linkage of amino acids can be broken down by hydrolyzation of the peptide bond, which requires the replacement of the water element removed on the condensation reaction. Although the hydrolysis of a peptide bond is an exergonic reaction, it occurs slowly because of a high activation energy, making this bond a stable bond with a good average half-life.

Relatively to the protein function, there is not a universal rule that associates the molecular weight or length of the protein to their function. Biologically, proteins can have a wide length

range that varies from a few hundred to thousands of amino acid residues. Some small peptides can exert their effects in a very low concentration, such as vertebrate hormones, toxic mushroom poisons, and many antibiotics. Proteins can be composed of only a polypeptide chain or by more than one noncovalently associated polypeptides, denominated multi-subunit proteins.

Although some proteins only contain amino acid residues, some proteins are associated with other chemical components, being conjugated proteins. Conjugated proteins are classified based on the chemical group associated: lipoproteins contain lipids, metalloproteins contain a metal component, and glycoproteins contain sugar groups [10].

2.1.3 Proteins structure

The protein structure is usually divided into four levels of complexity, arranged in a kind of conceptual hierarchy. The primary structure is the sequence of amino acid residues, resulting from the covalent interactions and being the most important element of this structure. The secondary structure is obtained by the interaction of non-consecutive amino acids within a chain, describing particularly stable arrangements of amino acid residues, such as α-Helix, β-sheets, and nonregular coils [12, 13]. The combination of the local secondary structure elements along the protein chain originates a tertiary structure, describing all aspects of the three-dimensional folding of a polypeptide [14–16]. Finally, the protein chains can interact or assemble, forming protein complexes. The quaternary structure describes the arrangement in space of the chains of the protein complex in the same frame of reference [10, 17]. A visual representation of these 4 levels can be seen in figure 3.

The primary structure can be especially informative, since each protein has a distinct amino acid sequence and length. The primary structure of a protein also determines how the protein folds in the three-dimensional space, which in turn determines the function of the protein. A simple observation: proteins with different functions always have a different amino acid sequence, and many diseases have been traced to the production of defective proteins. This defect can range from a change of a single amino acid residue to a deletion of a larger portion of the amino acid sequence. Despite one change in the amino acid sequence can be linked to a certain disease, the protein sequence can be flexible having variations in sequence with little to no effect

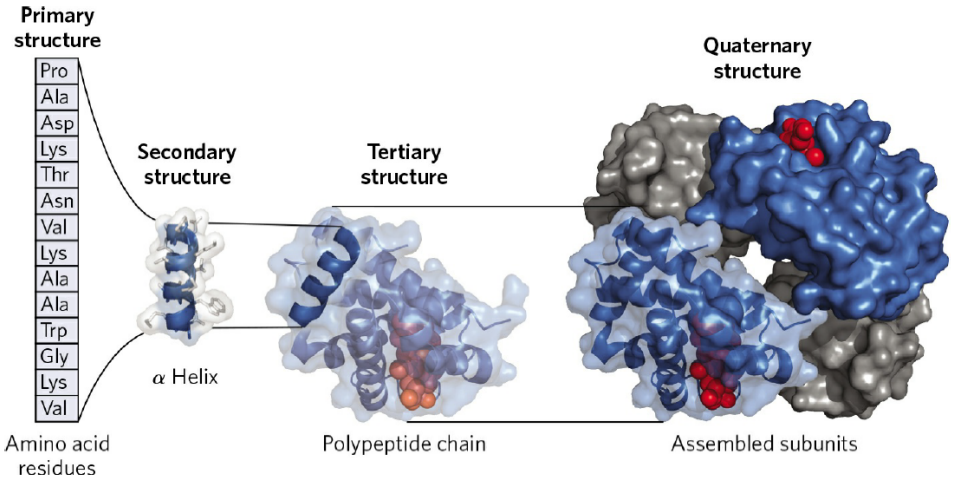


Figure 3 – The four levels of protein structure. The primary structure is represented by the sequence of amino acid residues. The secondary structure consists of particularly stable arrangements of amino acid residues, such as α-Helix, β-sheets. The tertiary structure is the combination of the stable arrangements along the amino acid sequence. The quaternary structure consists of the spatial association of all polypeptides. From source [10].

on the function of the protein, which is in this case called polymorphic. Furthermore, different sequences of amino acids exist in distinct species that carry out the same function [10].

Although some regions of the primary structure may vary considerably without affecting the protein function, most of the proteins contain crucial regions which are essential for the protein function, denominated conserved regions or domains. The conserved regions vary from protein to protein, making difficult the task of relating the sequence to the three-dimensional structure or the structure to their function. Much of the functional information encapsulated in protein sequences comes from consensus sequences [1].

2.1.4 Case studies

Enzymes

Enzymes are central to every biochemical process in a living cell. These proteins are highly specialized for their substrates and are capable of catalyzing chemical reactions. Multiple enzymes in a certain order carry out functions, such as degrading nutrient molecules, transforming chemical energy and making biological macromolecules from simple precursors. The catalytic

activity of enzymes depends on the integrity of their native protein conformation. Thus, the stability of the primary, secondary, tertiary, and quaternary structures is essential to their catalytic activity.

Just like any other protein, enzymes vary in length. While some enzymes just need their amino acid residues for activity, other enzymes can require additional chemical compounds such as cofactors or coenzymes. In certain cases, some enzymes are modified covalently by phosphorylation, glycosylation, and other processes [10].

Because of the increasing number of discovered enzymes, biochemists adopted an international system to classify and name new enzymes. The Enzyme Classification (EC) number, specifies the function of an enzyme by a 4 digits code. The first digit, which is the main node, classified them into 7 standard categories: Oxidoreductases, Transferases, Hydrolases, Lyases, Isomerases, Ligases, and Translocases [18, 19]. Each main class node extends for several subclass nodes, represented by the second digit, specifying the enzyme's subclass. In the same logic, the third digit represents the sub-subclasses, and the fourth digit represents the sub-sub-subclasses [20].

Transporters

Membrane transporters are proteins that mediate the transport of solutes through the cell membrane, which can range from ions to molecules and drugs. The transporters also mediate transports between the cells and the environment, being responsible to maintain the equilibrium for the survival of the cell. There are two types of transport: passive transport, and secondary active transport.

The passive transporters are also called uniporters, facilitators, or equilibrative transporters. The passive transport can occur via additional intermediate occluded states, facilitating substrate calibration across both sides of the membrane, or via channels where the diffusion of multiple substrates takes place upon the opening of the channel. In this type of transporter, the direction of substrate flow is determined only by electrochemical potential [21].

The secondary active transporters are used to concentrate the solutes on one side of a membrane. To achieve the movement of the substrate against the electrochemical potential, these proteins couple the movement of the substrate with a solute moving in favor of the

electrochemical potential. These transporters are called symporters or antiporters depending on if the substrates and solutes are moving in the same or opposite direction [21].

Transporters in the Transporter Classification Database (TCDB) are classified in a five-tier system, also known Transporter Classification (TC), following the structure of N1.L1.N2.N3.N4. The N represents numbers, and the L represents letters. The N1 is the class of the transporter. This system is a analogous to the EC number. Classes are divided into 9 classes: the classes from 1 to 5 define channels, secondary carriers, primary active transporters, group translocators, and transmembrane electron carriers, respectively; the classes 6 and 7 are empty, reserved for future classes; class 8 is constituted by accessory proteins, and class 9 by uncharacterized proteins. L1 represents the subclass, N2 is family or superfamily, N3 is the subfamily and N4 represents the sub-subfamily [22].

2.2 Machine Learning and Deep Learning

Artificial Intelligence (AI) is a general field which Machine Learning (ML) and Deep Learning (DL) are incorporated, and originated in the 1950s. AI intends to create systems to learn and extract knowledge from data. In the early stages, AI was achieved by handcrafting an explicit set of rules to extract knowledge from data. ML is a branch of AI that enables the discovery of underlying relationships within data by applying algorithms trained instead of being explicitly programmed. The backbone of ML is composed of statistical learning methods, allowing the system to find a statistical structure in the data in order to generate rules for automating the needed task. DL, which is a specific sub-field of ML based on networks that will be discussed in the section 2.2.3 [23, 24]. Figure 4 represents the relationship between AI, ML, and DL.

ML algorithms can be classified accordingly to the type of learning in supervised, unsupervised, semi-supervised, and reinforcement learning. Each type of learning is appropriate to address different tasks, requiring a type of input to train the models. Once supervised and unsupervised learning are the main paradigms, a brief description of each will be presented.

The goal of supervised learning is, based on input-output pairs, to discover relationships between the data (input data) and the output variables, such as labels, in order to make

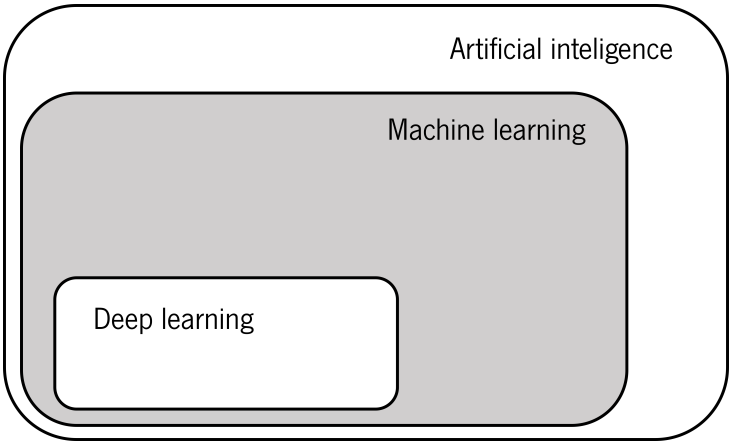


Figure 4 – Relationship between artificial intelligence, machine learning, and deep learning. Adapted from [24].

predictions for new and unseen data. Unsupervised learning only attempts to extract knowledge from the data, not requiring output variables [23, 25].

2.2.1 Unsupervised Learning

In unsupervised learning, the learning algorithm does not have access to a known output variable. In this approach, where the data does not contain labels, the algorithm intends to discover a hidden structure or data patterns to extract knowledge. Because of the frequent lack of labels associated with the input data, the main challenge related with this approach is evaluating if the unsupervised algorithm learned something valuable. Therefore, unsupervised methods are usually seen as an exploratory analysis technique, helping to understand better the data. These algorithms are frequently used as a pre-processing tool for supervised learning algorithms [26, 27].

This section will present two methods, *clustering algorithms* and *unsupervised transformations*.

Clustering algorithms

Clustering algorithms try to divide the input data into subgroups (clusters) without the knowledge of their group membership. Each cluster defines a set of inputs that share a certain degree

of similarity, being dissimilar to inputs associated to other clusters. Algorithms such as K-means (most-used), and hierarchical clustering fall into this category, often called as methods of ”unsupervised classification” [26, 28].

Unsupervised transformation

A common application of *unsupervised transformations* is dimension reduction, where it takes a high-dimensional representation of the input data, usually an input with many features, and creates a new representation of the original data to summarize the essential characteristics in a lower number of features.

The most remarkably used technique is Principal Component Analysis (PCA). PCA is a statistical procedure that uses an orthogonal transformation to perform the dimension reduction. The transformation is defined so that the components, the features created by this technique, are in descending order of explained variance. Other transformation algorithms include t-Distributed Stochastic Neighbor Embedding (t-SNE), Independent Component Analysis (ICA), or Non-negative Matrix Factorization (NMF) [23, 26, 28].

2.2.2 Supervised Learning

As already mentioned, the principal objective of a supervised learning algorithm is to infer relationships between the input data (observed data) and the variable that will be predicted. Therefore, it requires the separation of the input data into a training set and a test set. First, the training set is used to build the model fitting the data, based on the input data and output variables or labels. If the algorithm is flexible enough and data elements are coherent, iteration after iteration, it will adjust the parameters in the model to improve the prediction capacity. The objective is to make an accurate prediction for new, never-before-seen data.

Classification and regression are the two major types of supervised ML applications. In regression problems, the objective is to predict a continuous variable that is within a continuous set, based on the input data. Classification problems have the objective of predicting the label/class for the inputted data from a predefined list of possibilities. The classification problems

can be classified into binary problem (only two labels) or multiclass problem (more than two labels) [26].

The development of a ML workflow is composed of several steps. The first step is related to the collection of data and preparation. This is followed by a sampling of the data, training of the models, and evaluation of their prediction capabilities.

Data collection and preparation

The starting point in ML is the collection and preparation of data which is going to be used to build and train the model. This includes the dataset construction, pre-processing, feature extraction, and feature selection. Generally, a dataset is a matrix of data where each row represents an example and each column represents an attribute/feature. The columns must contain all the input attributes or features, and also the real output, being fixed-length vectors [29].

Depending on the collected data, processes such as feature extraction and pre-processing may be needed. The feature extraction transforms the initial collected data, raw data like text and images, into features suitable for modeling [26]. The pre-processing allows, for instance, the removal or substitution of corrupt and missing data, while it can also format the data, transform the data (e.g. log transform) or standardize the raw data if needed [30].

The last process that is normally used for the construction of the dataset is feature selection. Feature selection is a process that selects a subset of features that are sufficient to describe the target concept, from all inputted attributes [31]. Ideally, the feature selection allows removing redundant and irrelevant features and maintain only the relevant features [32].

Data sampling

Before feeding the data to a model, the dataset must be split into a train and test dataset. Usually, this division is randomly applied. While the training data is used to train the model, the test data is used only to evaluate the performance of the trained models. In most cases, a validation set may also be required, which is a third partition of the dataset, typically obtained by partitioning the training set, used to evaluate different model configurations [30].

Machine Learning traditional supervised algorithms

After following the previous steps, the training set can be applied to train the chosen algorithm. Some commonly used classes of models, with associated supervised learning algorithms, include Support Vector Machine (SVM), Decision Trees (DT), Random Forest (RF), and Artificial Neural Network (ANN), for which a brief description will be presented below. Other supervised algorithms include Logistic Regression (LgR), Linear Regression (LR), k-Nearest Neighbors (KNN), and Gaussian Naive Bayes (GNB).

Support Vector Machines

SVM are supervised learning models that can be used to analyze data and recognize patterns, for classification and regression applications, able to provide high accuracy in many applications [33]. A SVM training algorithm works over a representation of the data into a higher-dimensional space, in which the objective is to divide the input feature space, separating the two output classes, through a decision boundary, also called hyperplane, that is as wide as possible. For example, for a training set with the objective to predict labels, SVM will try to find the hyperplane which maximizes the distance between the labels [23, 25, 33].

The method can perform the objective by using a linear methodology or a nonlinear methodology. When dealing with nonlinear problems, it is useful projecting the original data transformed into a dimensional space and linearly separate the data. However, this increases the computational complexity. The use of *kernels* allows to efficiently perform a nonlinear method by reducing the computational complexity, allowing to benefit from the power of non-linear projections even in a very large number of dimensions [34].

A *kernel* is a function that the value of the kernel for two feature vectors is the dot product of the two projected vectors. The kernel methods aim to create a smooth separation based on nonlinear decision boundaries. Radial basis function (RBF), polynomial and sigmoid are examples of kernels that can be used in SVMs [25, 34].

Decision Trees

DT are statistical models used in classification and regression applications, being one of the simplest ML methods [35]. DTs can handle different types of data (nominal, textual, numeric), redundant data, and missing values, and can provide high accuracy with a low computational

effort. However, dealing with high dimensional data with DT can be difficult [23, 33]. This type of model, as the name suggests, breaks the data by making decisions based on a series of questions. A DT is started with a root and assigning it to a label according to majority votes obtained from the training set, becoming a leaf. After that, the model will perform a series of iterations and evaluate at each iteration the gain of splitting a single leaf. Then, the model will choose the split that maximizes the gain and performing the best option, learning the series of questions that are most informative to predict the output. The gain of splitting a leaf is determined by the algorithm used [26, 36]. A decision tree can be built based on different algorithms, such as Classification and Regression Trees (CART), Chi-square Automatic Interaction Detection (CHAID), and Iterative Dichotomiser 3 (ID3) [25].

Random Forest

RF is an ensemble method that operates by training several decision trees and returning the predicted class based on the majority of all trees trained. In some applications, RF models are slightly ahead of SVMs because they are fast and scalable, do not overfit, are robust to noise, and are easy to interpret and visualize with no additional parameters to manage. However, as the number of trees is increased, the algorithm can become slow for real-time prediction [33].

Artificial neural network

ANN was originally inspired by the idea of mimicking the functioning of the human brain with artificial neurons (processing units). The neurons or nodes, the basic building block of this method, are responsible for computing output from the inputted data.

The output is obtained by multiplying the data inputted to the neuron by the *weights* of the connections to other neurons, and by adding the *bias*. Followed by the application of an activation function that determines the output that the neuron will pass to the next layer. The *weights* are parameters associated with the linkage between neurons, which reflect the relative importance of each connection for the output of the receiving neuron. The *bias* is a parameter associated with the neuron to help the model to better fit the received data. Figure 5 represents the process of calculating the output of a neuron, with the inputs (χ), *weights* (ω), *bias* (b), weighted sum function, and the activation function (f) and output (y).

The activation functions more used are the sigmoid function, TanH function and the rectified linear activation function (ReLU). The sigmoid function returns a value between 0 and 1, while

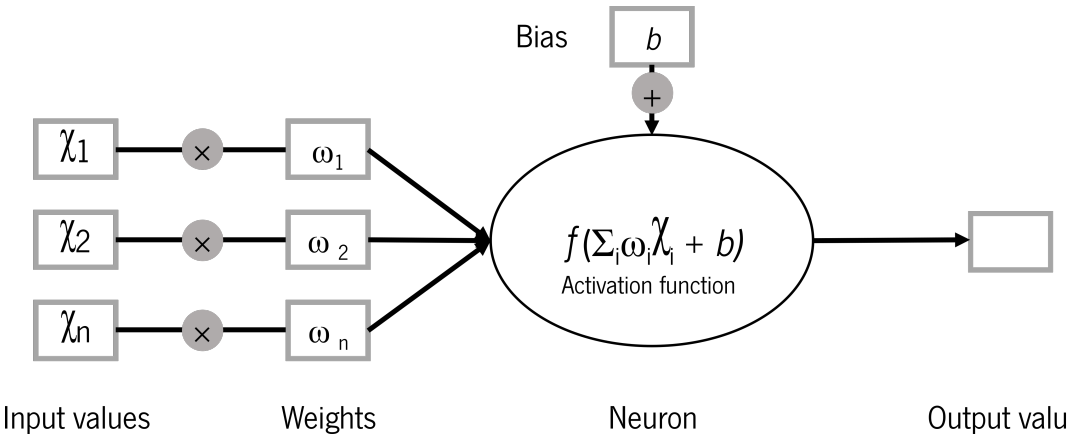


Figure 5 – Scheme of an artificial neuron. The neuron receives the input χ, *weights* ω, and bias *b*. The output produced by the neuron results from applying the nonlinear transformations of linear combinations and an activation function f. Adapted from [37].

the TanH returns a value between -1 and 1. However, the ReLU function is more widely used because it can learn a lot more quickly in networks with multiple layers. This function returns the input value, if it is positive, or 0 if the input is negative [38, 39].

A feedforward ANN is constituted by neurons disposed in layers and connections between the neurons. The ANN models have 3 types of layers: an input layer, hidden layers, and an output layer as represented in figure 6. The input layer receives the data fed to the model, each neuron represented by a feature of the data. In the hidden layers, each neuron receives input signals from the previous layer. The connections between the layers represent the learnable parameters, *weights*, and *bias*, which are integrated in the neurons to compute the output as described previously. Since each neuron has a different parameter associated, the inputted data is transformed differently in each neuron, followed by a non-linear activation function. In the end, the output layer receives the values from the last hidden layer and transforms them to output values [24].

The learning process is the adjustment of the learnable parameters to fit the training data, which are set randomly in the first moment. In order to learn from the data, it is required an *objective* function and an *optimizer*. The *objective function*, also called *loss* function, measures the difference between the predicted variable and true variable. It captures how well the network has performed in the prediction of training data, represented by a *loss* score. The *optimizer* is responsible for adjusting the learnable parameters utilizing a learning algorithm, based on gradient descent. The *backpropagation* algorithm was one of the first to be used, and is still a

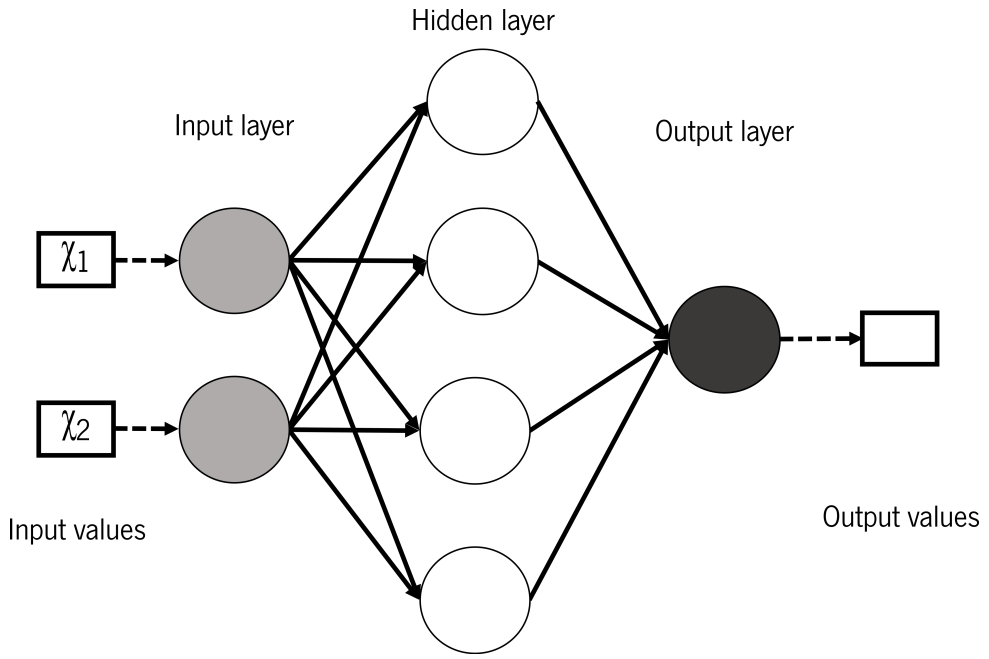


Figure 6 – Artificial neural network. It is composed by an inputted layer of two neurons, a single hidden layer with four neurons, and an output layer with only one neuron. Adapted from [37].

central algorithm in this field, which has been improved in the last years. The *optimizer* receives a score as feedback to perform the adjustments, usually the *loss* score [24, 38].

The *Adam* algorithm is one of the most popular *optimizers*. This algorithm determines the gradient and squared gradient by exponential moving average. The decay rates of these moving averages are controlled by the parameters β_1 and β_2. The *Adam* optimizer maintains the learning rate while updating the weights, and the learning rate does not change during training [40].

At the beginning of the training process, the *loss* score is usually very high, meaning that the predictions are far from the true label. During the training process, occurs the forward propagation where the inputs are processed through the ANN and generate the output. Then, they are compared to the real outputs to calculate the *loss* score by the *loss function*. The next step is to modify the learnable parameters, *weights* and *bias*, to minimize the *loss* score. This process is done by the backward propagation that uses the chosen *optimizer* to perform the adjustments. Figure 7 illustrates the learning procedure [24, 38].

This training loop is repeated multiple times to obtain the minimum *loss* function score. The network is considered trained when it achieves the minimal *loss* score, implying that the

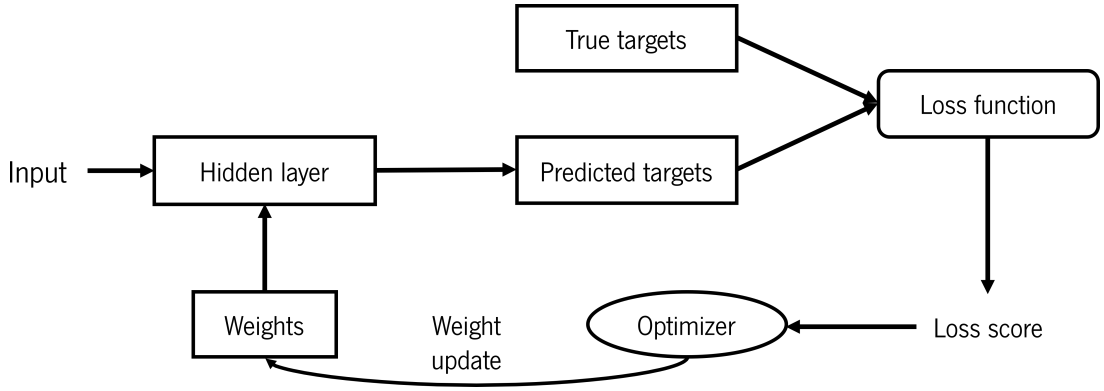


Figure 7 – The learning process of an artificial neural network. The learning loop starts by feeding the input to the network, calculate the *loss function* and update the *weights* of the network. Adapted from [24].

predicted outputs are most similar to the true labels as they can be, enabling the network to predict labels from unseen data [24].

Evaluation and application of the model

After the train of the chosen algorithm with the training data, the trained model can be evaluated using error metrics. This evaluation is made by exposing the trained model to the test dataset. The error metrics compare the outputs given by the model to the real values. The error metrics used depend on if it is a regression or a classification problem.

When dealing with a classification problem, a confusion matrix is often used. The confusion matrix consists of a matrix in which the rows represent the real classes and the columns the classes predicted by the trained model. In the binary case, this allows an easy representation of True Positives (TP), False Negatives (FN), True Negatives (TN), and False Positives (FP) cases. Accuracy is an error metric that produces a value related to how often the classifier makes the correct prediction. Based on the accuracy metric, the Balance Accuracy (BAcc) mitigates the limitation of the traditional accuracy for imbalanced data. Precison (Prec) is also often used, representing the samples predicted correctly as positive. Another frequently used metric is recall, which represents the number of positive labels amongst all positive labels [26, 28, 41].

The Matthews Correlation Coefficient (MCC) is another important classification metric. While other metrics only consider part of the predictions, MCC produces a more informative and balanced score by taking into account true and false positives and negatives. It returns a value

between -1, disagreement, and 1, perfect prediction [42]. Other relevant classification metrics include F_1 score, Receiver Operating Characteristic (ROC), and precision-recall curves [26, 28]. The equations of these metrics are represented in figure 8.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$F_1 \text{ score} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

$$\text{Balanced Accuracy} = \frac{\text{Recall} + \text{Specificity}}{2}$$

$$\text{Matthews Correlation Coefficient} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

		Predicted label	
		P	N
Real label	P	TP	FN
	N	FP	TN

Figure 8 – Error metrics for classification tasks. A confusion matrix is presented for a binary classification of positive (P) or negative (N). The equations to calculate the accuracy, precision, recall, specifity, F_1 score, balanced accuracy and Matthews correlation coefficient are presented considering the true positives (TP), false negatives (FN), true negatives (TN), and false positives (FP). Adapted from [42, 43].

For regression problems, the error metrics are based on the difference between real and predicted values in all examples. They can be calculated by functions such as the squared sum of errors (SSE), Root-mean-squared Error (RMSE), or R-squared [44].

Sometimes the chosen model can be too complex or too simple to correctly learn from the data. A model can be underfitting, when the model is too simple, not capturing the variability in data. The model can also overfit, when the model is too complex, resulting in a high performance of the model over the training data and a low performance for the test data. The objective is to obtain a trained model that generalizes as correctly as possible [26].

After the validation of the trained model, usually comes the model deployment. This corresponds to the application of the model in an actual task of prediction.

2.2.3 Deep learning models

Many methods labeled as ML rely heavily on feature engineering to make the right transformation of the raw data, facilitating the ML algorithm to solve the desired task. The main focus of a DL approach is not the handcraft of representations but the manipulation of a mathematical entity to allow the model to discover representations from the training data autonomously, which makes DL a powerful tool in data mining.

DL methods are a ML subset based on ANNs, which use a cascade of multiple layers to extract increasingly meaningful representations from the input stacked on top of each other. Typically, these methods have several hidden layers to transform the data [45].

DL techniques include fully connected deep neural networks or Deep Neural Network (DNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), transformers, Auto-encoders (AE), Deep belief networks (DBN), and deep reinforcement learning [46]. Below, we address a description of the main architectures: DNN, CNN, RNN and transformers.

Dense Neural Networks

The DNN is the simplest DL model, being the principle of this method the same as described in the traditional ANNs. While an ANN only contains one hidden layer, a DNN can contain multiple layers, leading to a higher level of complexity. They are also feedforward networks, meaning that the data flows in a single direction [27]. Figure 9 shows an example of a DNN with two hidden layers, containing 4 neurons in each layer.

Convolutional Neural Networks

CNNs were firstly introduced in 1998 by LeCun et al., as an architecture directed to digital recognition, belonging to the feedforward neural network group. CNN is mostly used in computer vision, image recognition, and video recognition. Other areas such as Natural Language Processing (NLP), drug discovery, and others often use this technique for different purposes [27]. CNNs are usually composed of an input layer, convolutional layers, pooling layers, fully-connected layers (including the output layer).

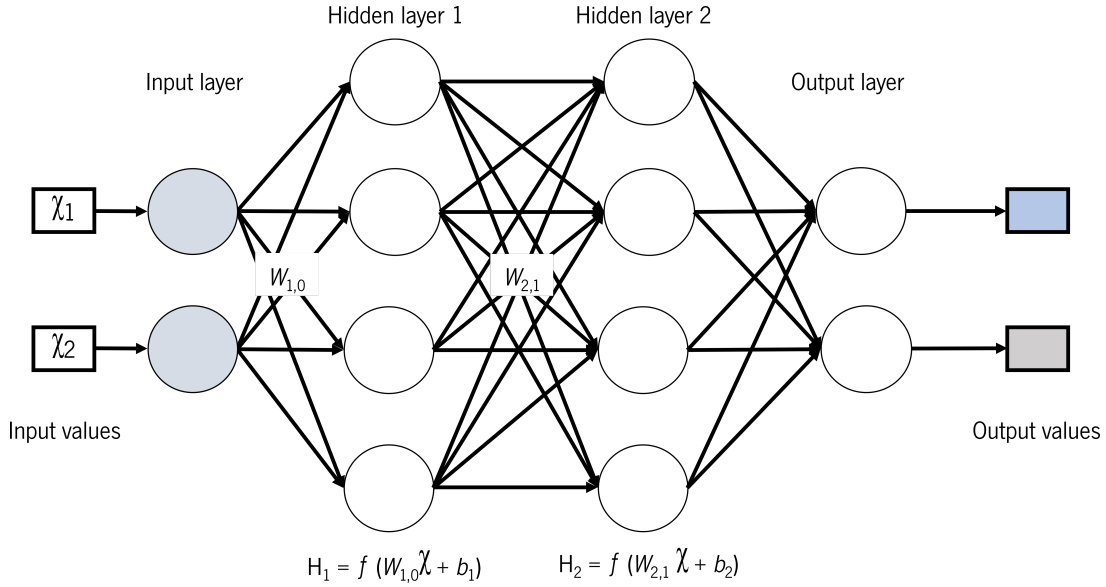


Figure 9 – Representation of a dense neural network. It consists of two input neurons that receive the input values, two hidden layers with 4 neurons and an output layer that produces the output values. Adapted from [27].

As mentioned, the input is given without any additional transformation, avoiding any complex feature extraction and data reconstruction process. Therefore, every layer in this network has a distinct function to transform the inputted data.

A convolutional layer is a feature extractor that identifies important features. It is responsible for learning local patterns instead of global patterns. The convolutions operate over feature maps with two spatial dimensions, height and width, and a depth axis, used for the channels for the color in the input image, but also to provide for different feature types in hidden layers.

A new feature map is obtained by multiplying the inputted feature map with a kernel, or filter, along the width [24]. Figure 10 show a convolutional operation. Consequently, the convolution extracts patches from the inputted feature map and applies the same transformation to all matches to produce the new feature map. After the convolution, the activation function is applied element wise. The outputted feature map tends to represent the presence of a local pattern or concept over the different tiles. A combination of multiple convolutional layers is able to progressively perform more refined feature extraction from the inputted data [27, 47, 48].

The pooling layers, also denominated subsampling or downsampling layers, usually come after convolution layers and are capable to perform a spatial dimension reduction of the inputted data in width and height, not altering the depth axis dimension. This allows to reduce the

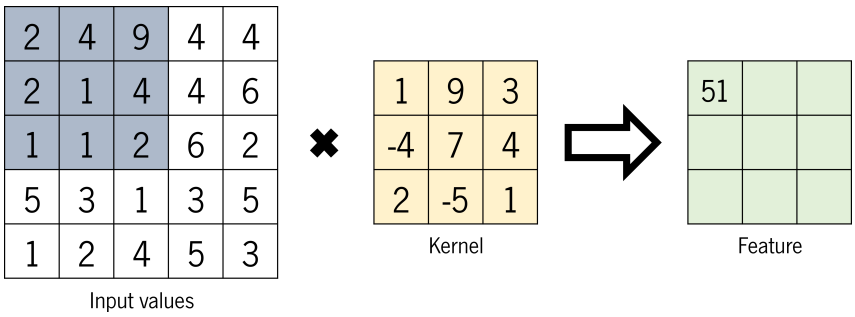


Figure 10 – Representation of a convolutional operation. The defined kernel dimensions is of 3×3.

dimension of the feature maps to process [49]. The average and max pooling are the most used strategies, the average pooling calculates the average value for the defined window, while max-pooling extracts only the maximum value of the window. In a detailed comparative analysis of max and average pooling performance, it was shown that max-pooling leads to a faster convergence, improved generalization, and selected superior invariant features [24, 27, 48, 49]. Figure 11 represents the result obtained by a pooling layer using a max pooling strategy.

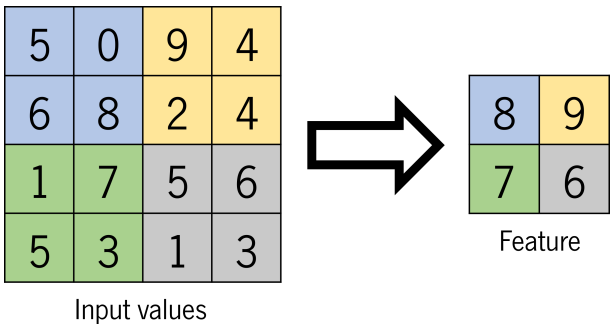


Figure 11 – Representation of a max-pooling layer. The dimensions defined for the window is 2×2. Adapted from [45].

After the multiple convolutions and pooling layers, usually there is one or more fully-connected layers. The neurons in a fully connected layer are connected to all neurons in the previous layer. The output of convolutional layers has to be flattened to feed the dense layer as a 1D feature vector. This feature vector can be used to feedforward for a classification task based on the extracted features or used as a feature vector for further processing [47–49]. Figure 12 represents an example of a CNN architecture for an object detection task.

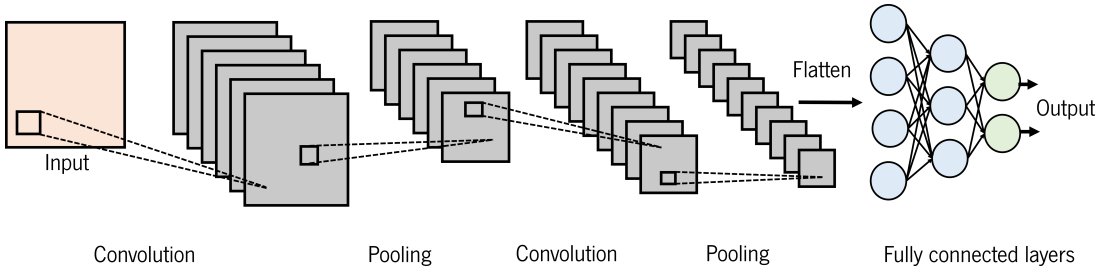


Figure 12 – An example of a complete convolutional neural network. The fed input passes through two convolution layers and two pooling layers, flattened and fed to a fully-connected layers. Adapted from [27].

Recurrent Neural Network

In feed forward networks, such as DNN or CNN, the neurons are connected between adjacent layers in the forward direction. This means that each layer processes the signals received by the previous layer independently and then propagates them to the next layer. On the other hand, RNNs can produce an output by comprising a latent vector, which contains the information relative to what the network has seen so far [47].

RNNs are a type of neural network designed to handle spatial-temporal structures, the layers have recurrent feedback neurons that are capable to merge the current incoming inputs with previous information, as represented in figure 13. This type of neural network has memory, having exploitable *bias* related to the ”past” to make informed decisions with the new data. RNNs are useful when working with sequences such as text, audio, or multiple sets of images [38].

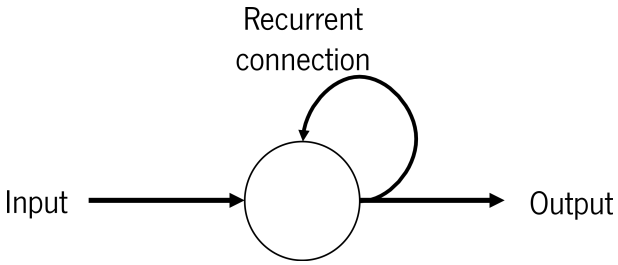


Figure 13 – Loop associated to a recurrent neural network neuron. Adapted from [24].

The biggest limitation of a RNN network is that it is not capable of dealing with long-term dependencies. This problem is known as the vanishing gradient problem, which causes the RNNs to ”forget” what they have seen in longer sequences, having a short-term memory. Therefore, to address the limitation of the RNN, variants surged such as Long Short-Term

Memory (LSTM) and Gated Recurrent Unit (GRU) which have internal gates to regulate the flow of information to the cell state and deal with the vanishing gradient problem [24, 50, 51].

An LSTM is usually composed of a cell state and three gates. The cell state, also called latent vector, is responsible for the memory which resets between the processing of two different and independent sequences. The cell state transfers relative information down the sequence chain. As the neural networks process the sequence, information gets added to the cell state.

The three internal gates are the forgot gate, the input gate, and the output gate. The input gate decides which information from the input is relevant, the output gate decides what information can be output based on the cell state, and the forget gate can decide which information will be discarded from the cell state. Each gate decision is based on activation functions, sigmoid and TanH functions. Each gate has a different combination of activation functions suited for its purpose, as can be seen in figure 14 [52].

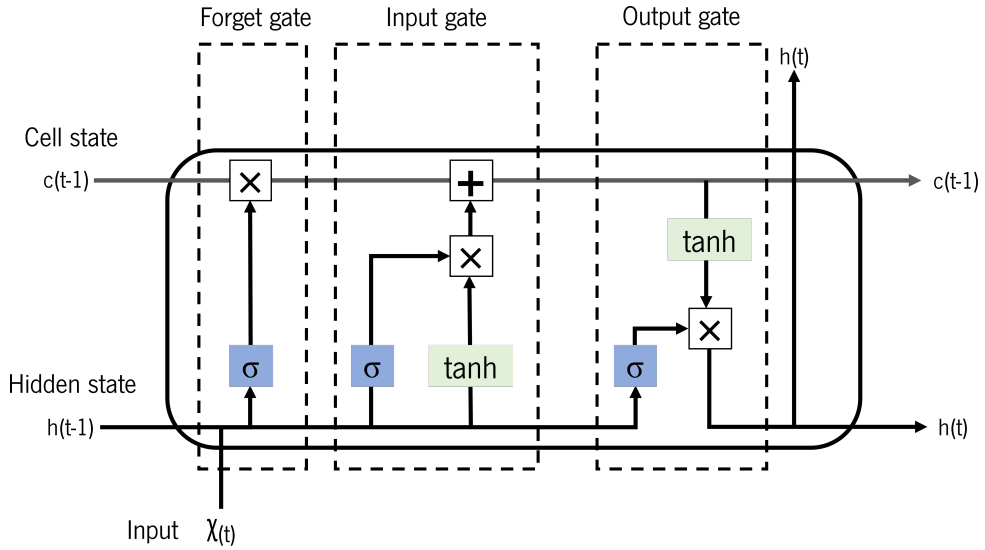


Figure 14 – Long short-term memory network cell. A cell state and a hidden state passes through the cell with the input. Firstly, it passes in the forgot gate, followed by the input gate and lastly the output gate. The present active functions are sigmoid function (σ) and TanH function (TanH). It also has operation of multiplication ($\times$) and addition ($+$). Adapted from [52].

For sequences as biological ones, a model can perform better if it processes the sequence in both directions, in forward and in the reversed direction. The bidirectional LSTMs are composed by two LSTMs, allowing to process the sequence in both directions. The first LSTM is trained with the input in a given order, and the second one is trained with the reversed input [24].

The GRU layers are simpler than the LSTM layers. GRU usually are composed of only two gates: reset and update gates. While the reset gate decides how much past information to forget, the update gate decides what information to throw away and the new information to add. The update gate can be seen as a combination between the forget and input gates in a LSTM network. GRU only has a hidden state, combining the cell state and the hidden state present in the LSTM. Because of the higher complexity in the LSTM layers, their training is slower than the GRU layers [24, 52].

However, the more exciting results based on RNNs were achieved by LSTMs, because of their powerful capacity of learning from the data. LSTMs were used in a wide variety of applications such as speech recognition, acoustic modeling, trajectory prediction, correlation analysis and sentence embedding, becoming a focus technique in DL [38, 50].

Transformers

The first models in the Language Modeling (LM) were often models based on the RNN architecture. Despite the improvements of the LSTM and GRU relative to RNN, the lack of comprehension in bigger contexts has proven to be an obstacle in these fields [53].

Transformers were initially introduced by Vaswani et al. [54] in 2017. Transformers are attention-based architectures, composed of encoder and decoder blocks. The attention mechanism is responsible for processing the data in parallel with no sequential dependency. This allows to overcome the LSTM context limitation, and capturing long-range sequence features [53, 55]. While, in the LSTM layer, all tokens of the sequence are represented by only one hidden state, the encoder produces a hidden state for each inputted token in the sequence. The encoder converts the inputted tokens into the called hidden state or context, embedding vectors.

The attention mechanism is between the encoder and the decoder. As the increased number of hidden states fed to the decoder can create an enormous input, the attention mechanism allows the decoder to assign more importance to certain hidden states, allowing to process the hidden states and generate an output sequence. Figure 15 is a representation of the procedure [56].

Despite the fact that this architecture was designed for sequence-to-sequence tasks, like translation and summarizing tasks, it was rapidly adapted to be explored for other tasks. The

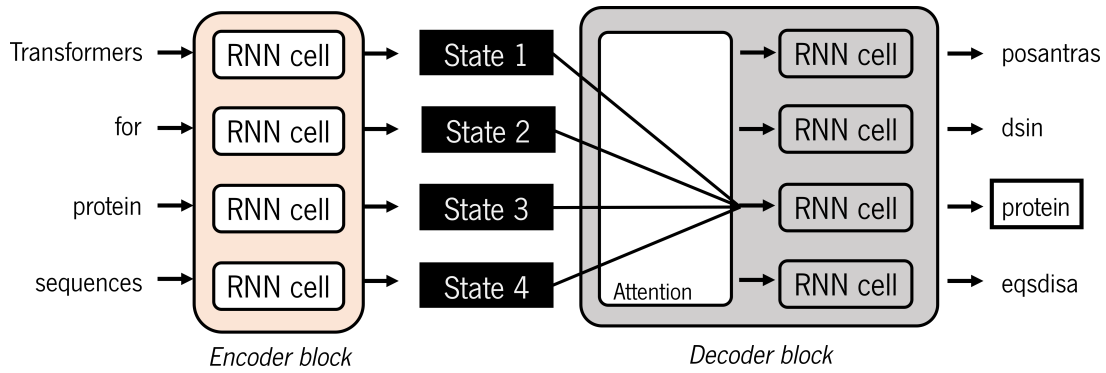


Figure 15 – An encoder-decoder architecture in an attention mechanism. Adapted from [56].

transformers rapidly become dominant in sequential tasked fields, such as NLP [57], speech processing [58, 59], computer vision [60], and gradually extended to handle non-sequential problems [61–63].

Nowadays, there are variations from the original transformers. Encoder-only are models designed to supply a rich representation of the inputted sequences, including models such as BERT [57], RoBERTa [64] and DistilBERT [65]. Decoder-only or *autoregressive attention* models were idealized to predict the next token in the sequence context, as it is the case with the family of GPT [66, 67] models. Lastly, encoder-decoder models are appropriate for sequence-to-sequence tasks, for example text summarization, BART [68] and T5 [69] are examples of this type.

A major drawback is the complexity of such models because they require a considerable amount of computation resources to train. However, the usage of pre-trained models allows the usage of the trained models with a generic collection of data. Therefore, pre-trained models are often used as feature extraction methods. These pre-trained models can then be loaded and used with lower computation resources, providing an accessible and efficient means to create representations of the inputted tokens [53, 56].

2.2.4 Automated Machine Learning

AutoML is an automatic pipeline, completing the steps usually created by the ML scientist. This pipeline is usually segmented into the multiple steps. In a normal ML pipeline, to achieve good results, all data, features, and the model must be manually prepared and fine-tuned through trial

and error by the ML scientist. However, the time required to manually complete this pipeline increases with the complexity of the models [70].

AutoML can be divided into four different stages, which one compassing different steps of the typical pipeline [71]. These stages are as follows:

- **Data Preparation**: The acquisition of new data to create and/or augment a dataset, along with the cleaning of data to remove noise.
- **Feature Engineering**: The process that uses feature extraction methods to transform existing features, feature construction to create new features from data, and feature selection to reduce dimensionality by selecting the most important features.
- **Model Generation**: Testing different ML algorithms and the parameters associated to the ML algorithms are optimized to achieve the best model architecture.
- **Model Evaluation**: Process where the trained models are evaluated in the performance of each model.

AutoML can encompass all steps of the pipeline, or only a group of selected steps. Tools that allow to realize one or more of these stages are : Auto-PyTorch [72], Autogluon [73], Auto-Keras [74] and many others.

2.3 Deep Learning applied to Protein sequence classification

High-throughput data acquisition technologies have been transforming biology into a data-rich science. The increasing capabilities of the technologies used by laboratory personnel have caused an exponential growth in the available amount of biological sequence data, such as Deoxyribonucleic Acid (DNA), Ribonucleic Acid (RNA), and protein sequences. However, due to the limitations in the wet lab experiments, the gap between the sequence data and characterized data is increasing.

To fulfill the need for methods capable of analyzing large-scale data, bioinformatics has found in data mining an excellent and mature ally. Bioinformatics recurs to ML methods not only to analyze sequence data in order to understand the structure and the function but also to drive the development of applications to obtain knowledge in a fast, inexpensive and efficient way.

As mentioned, proteins are responsible for most of the cell activity. Therefore, many models of ML have been applied to predict the function, the various protein structure, or even the interactions of a protein. The majority of tasks related with protein sequence analysis are classification tasks, which classify the protein sequence into defined classification systems, i.e. EC and TCDB.

An objective of this work is the development of a DL framework capable of performing the classification of proteins based on their primary structure. To evaluate the performance of the framework, two case studies of protein sequences will be used, enzymes and transporters. However, the major step is the encoding the amino acid sequence.

2.3.1 Sequence encoding

The amino acid sequences, proteins, are accessible in databases represented by the one-letter code, as explained in the section 2.1.1. Therefore, protein sequences are strings composed of letters and each letter represents an amino acid residue. Even though DL models can take the input from raw data, raw text can not be inputted into the model. Consequently, the raw text has to be transformed into a numeric tensor, a process called *vectorizing* text. A good representation of the protein in the mathematical vector is essential to provide insights into distinct characteristics of different proteins.

Substitution matrix

A common technique to encode sequences is the orthogonal encoding, also known by one-hot encoding, which transforms the tokens into a binary vector with the length of the vocabulary. The text is split into tokens, where each token can represent a word or a single character. The vector is composed of all zeros except for an index which represents each token. An example can be seen in figure 16, showing the binary vector for each character [24, 75].

Other relevant techniques are the use of a substitution matrix like Blocks Substitution Matrix (BLOSUM) representing accepted mutations between amino acid pairs [76]. The BLOSUM is a family of substitution matrices, often used for protein sequence alignment. Here, the score assigned to two residues reflects the likelihood of those being aligned in a homologous sequence.

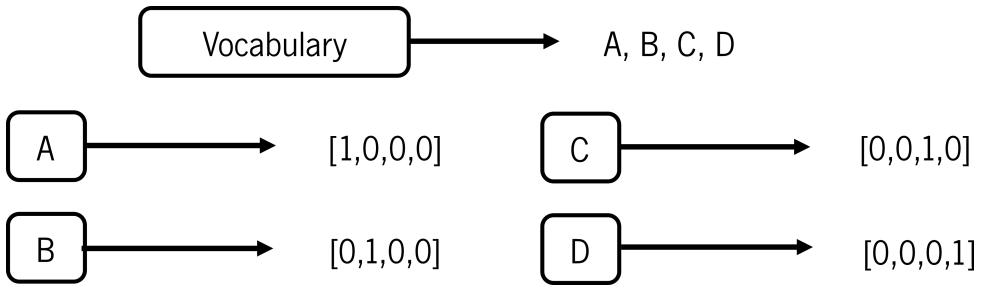


Figure 16 – Representation of orthogonal encoding. One-hot-encoding into binary vectors of a vocabulary of 4 letters.

These substitutions were found from studying protein sequence conservation in large databases of related proteins. The number included in each BLOSUM matrix name refers to the percentage identity at which sequences were clustered in their definition. A lower number is better for aligning sequences of lower identity, while higher values are better to find highly conserved regions [77, 78].

Z-scales and Non-Linear Fisher (NLF) matrices are substitution matrices for protein sequences. The Z-scale encodes each amino acid for a numeric representation of lipophilic, steric, and electronic properties of the amino acid residues [79]. The NLF matrix is the representation of supervised feature transformation based on non-linear Fisher transformation, which tries to preserve the distances between amino acid residues by eighteen numeric features [80].

Position Score Matrix

Also, an option is the use of Position-Specific Scoring Matrices (PSSM) or Position-Specific Frequency Matrix (PSFM) capturing evolutionary constrains on a protein family, assuming that each amino acid is highly conserved in a protein family [76, 81, 82]. PSSM is frequently referenced as profiles or hidden Markov models. The intuition behind this method is that a multiple sequence alignment (MSA) of related proteins can identify position-specific residues, which may benefit in the identification of homology (conserved regions). This position-specific residues are identified by a high positive score related to the preservation of the amino acid residue [83]. Also, this PSSM based feature descriptors have improved the prediction performance in a large spectrum of bioinformatics applications [84, 85]. Methods such as Position-Specific Iterative Basic Local Alignment Search Tool (PSI-BLAST) allow an iterative search for a more sensitive search of biologically relevant sequence similarities [86].

Embedding

The use of embedding layers can be seen as an alternative to the commonly used one-hot encoding, which is used after the input layer. The embedding layer compresses the input feature space into a smaller one, finding the optimal mapping of each of the unique tokens to a vector of real numbers. These can be obtained by word embedding techniques, such as word2vec and protVec, which perform a distribution encoding to characterize the relationship between the tokens. While word2vec is a technique related to word expression including the meaning and context of words in a document, protVec is designed to be applied to a wide range of problems in bioinformatics. This tool allows representing co-occurrence relationships between blocks of amino acids instead of between individual amino acids [87, 88].

Transformer

As referenced in section 2.2.3, the transformer or LM is a dominant architecture in sequential tasks by using attention. Despite, the transformers were initially developed for NLP, LM were successfully applied to protein sequences [63, 89].

ProtTXL, ProtBert, ProtXLNet, ProtAlbert, ProtElectra, ProtT5-X, and ProtT5-XXL are examples of usage successfully transformers in NLP trained with protein sequence data. ProtBert, ProtAlbert, ProtElectra, ProtT5-X and ProtT5-XXL belong to encoder-only type of transformers, while ProtXLNet and ProtTXL belong to the *autoregressive models* [63]. DistilProtBert [90] is another example of an adaptation from the NLP field to a biological field. The Evolutionary Scale Modeling (ESM) is a transformer protein language model from Facebook AI developed for tasks such as predicting variant effects, inverse folding, and contact prediction [91–95].

The usage of these transformers as feature extractors to represent the protein sequence was shown to have a better performance in multiple prediction tasks related to protein sequences [96, 97]. The data often chosen to train such transformers belong to UniRef [63, 90].

Protein Descriptors

Another encoding technique for sequences is obtained through the combination of the representations of proteins by physicochemical properties, such as pseudo-acid and amino acid

composition and physical properties, such as charge and hydrophobicity. There are also auto-correlation descriptors that consider the physicochemical properties of amino acids for specific positions and related in a higher dimensional protein space [81]. The feature encoding assumes the function/activity of the protein based on these properties computed through the knowledge of the amino acid sequence [98].

In correlation with the physicochemical properties, structural descriptors like hydrophobicity scale, average flexibility index, polarizability parameter, the free energy of amino acid solution in water, residue accessible surface area, amino acid residue volume, Van der Waals volume, and relative mutability can also be used [99]. Structural representations based on torsion angles, secondary structure elements and other structural elements are another option [100–102].

Binary profiles based on the absence or presence of sequence motifs (e.g. protein motifs from databases as PFAM) may also provide important information [9, 20, 82].

2.3.2 Relevant work on Protein Classification

Previously, numerous classification tasks associated with proteins have been conceived by using different frameworks. Some of the frameworks are trained with different transformation of the protein sequence, while some only use the encoded protein sequence, others use features obtained from the sequence.

The first method widely used to predict protein function is the Basic Local Alignment Search Tool (BLAST). BLAST searches in the database for sequences that are homologous to the protein query, and then uses the information of the homologous sequence found to predict the function of the query protein, relying on high sequence similarity [103]. Despite the value that this method has shown, it suffers from limitations, such as depending on homologous protein sequences to predict the function of the query protein [104].

Table 2 lists some of the previous works based on DL architectures. The table refers to the prediction task, the year of the publication, the type of architecture used, and the inputted data into the model for each study.

Previous works have shown the potential of DL for prediction tasks associated with proteins. The most frequent method to encode the sequences is the Hot-Encoded Sequence (HE), that few

studies combined to Structural Features (SF), Protein-protein Interactions (PPI) derived features, Domain Content (DC), Physicochemical Features (PCF) based on the protein sequence, amino acid composition, and others. Another approach is the use of PSSM to capture the evolutionary information in the sequences. The DeepPPF [6] is the only study which uses the sequence encoded by the embedding Word2vec (W2V) [87]. DeepConv-DTI [105] associates the hot encoded sequence to the Simplified Molecular-input Line-entry System (SMILES) to predict drug interactions. More models in the latest years use Transformer Features (TF) to encode the protein sequence.

Table 2 – Review of studies conceived for protein tasks. It is presented the prediction task, the year of the publication and the method used to input the sequence into the model. The inputted data is abbreviated as follows: hot-encoded sequence (HE), BLOSUM, sequence encoded by word2vec (W2V), structural features (SF), protein-protein interaction derived features (PPI), simplified molecular-input line-entry system (SMILES), domain content (DC), physicochemical features (PCF), residue–residue contacts (RRC) and transformer features (TF)

Name	Prediction task	Year	Data
DeepLoc 2.0 [96]	Protein subcellular localization	2022	TF
Tranception [106]	Protein Fitness Prediction	2022	TF
SPOT-Contact-LM [107]	Protein Contact Map	2022	HE, TF
A-Prot [108]	Protein structure	2022	TF
NetSurfP-3.0 [109]	Protein structural features	2022	TF
DTITR [110]	Drug-target interactions	2022	TF
LM-GVP [111]	Protein property	2022	TF
SPOT-Contact-Single [112]	Protein Contact Map	2021	HE, TF
DeepPPF [6]	Protein family	2021	W2V
DeepDTAF [113]	Protein–ligand binding affinity	2021	HE, PCF, SF, SMILES
DeepTFactor [114]	Transcription factors	2021	HE
DeEPn [115]	Enzyme EC number	2021	HE, PCF

Continued on next page

Table 2 – continued from previous page

Name	Prediction task	Year	Data
DeepSVM-fold [5]	Protein fold recognition	2020	RRC, PSSM, PSFM
SDN2GO [116]	Protein Function	2020	HE, DC, PPI
MusiteDeep [117]	Protein post-translational modification site	2020	HE
UDSMProt [118]	Protein classification	2020	HE
ProtTrans [63]	Per-residue secondary structure, Protein localization	2020	TF
DeepConv-DTI [105]	Drug-target interactions	2019	HE, SMILES
DeepFunc [119]	Protein function	2019	HE, PPI
DeepGOPlus [120]	Protein function	2019	HE
DeepEC [121]	Enzyme EC number	2019	HE
DEEPred [122]	Protein Function	2019	HE, PCF
DeepIon [123]	Ion Transporters and Ion Channels	2019	PSSM
ET-GRU [124]	Identify electron transporters	2019	PSSM
DeepFam [125]	Protein family modeling	2018	HE
DeepSol [7]	Protein solubility	2018	HE, PCF, SF
DeepCrystal [8]	Protein crystallization	2018	HE
DPPI [126]	Protein–protein interactions	2018	HE
DEEPre [20]	Enzyme EC number	2017	HE, SF, PSSM
DeepLoc [127]	Protein subcellular localization	2017	BLOSUM

The CNN architecture was applicable in the majority of the works seen in the research in the literature. However, models previously to 2019 also use the architectures of DNNs and LSTMs for some prediction tasks. Models in the literature after the year 2021 often introduce attention

mechanisms into the model structure. Another notable fact is that in more recent prepositions of models for protein prediction tasks, the usage of pretrained transformers as features extractors has increased. The usage of the transformers has allowed to surpass previously state-of-art in some tasks.

The most relevant studies to this thesis are related to the classification of the function of proteins, universal approaches for classification of protein into classes or families. Works such as DeEPn [115], DeepEC [121], and DEEPre [20] are models that predict the enzyme classification, which were capable to achieve a good prediction performance. However, the enzymes sequences are often used in other prediction tasks. For the classification of transporter sequences, it was found that most of the studies isolates the prediction task to small subgroups of transporters. The DeepIon [123] and ET-GRU [124] were the models which encompasses a larger number of transporter sequences. However, these models do not include all the transporter class from the TC system.

2.3.3 Tools for building protein classification algorithms

The development of a complete ML framework for data analysis has become essential in biological studies, which can be a hurdle to biologists and bioinformatics with no experience in ML. Therefore, some tools and web servers have been developed. The solutions include SPiCE [128], ASAP [129], [130], modlAMP [131], PyFeat [132], BioSeq-Analysis2.0 [133], iLearn [134], iLearnPlus [135] and ProPythia [9] which allow the feature calculation integrated with the functionality for further sequence analysis data and perform ML pipelines. A description of the more recent tools can be seen below.

BioSeq-Analysis2.0 [133] is available as a web server but also as a package for protein, DNA, and RNA sequences analysis based on ML approaches. This package performs feature reduction and feature calculation but also incorporates classification algorithms like SVM (RBF kernel), RF, and a sequence labeling algorithm: Conditional Random Fields (CRF). This package does not give a lot of freedom in terms of algorithms and choice of parameters.

iLearnPlus [134, 135] is an updated version of the iLearn. The iLearnPlus is available in a web server and a Graphical User Interface (GUI) version. Even though the code can not be

manipulated as a package, a ML pipeline for analysis can be built, and prediction using protein, DNA, or RNA sequences. The iLearnPlus concedes the sequence-based feature extraction and analysis. It allows the construction and deployment of models, assessment of predictive performance, statistical analysis, and data visualization without programming. This package allows the calculation of 37 feature descriptors, to use 5 feature selection methods, 10 types of clustering algorithms, 12 conventional ML algorithms but also implement DL architectures such as CNN, RNN, Bidirectional Recurrent Neural Network (Bi-RNN), Residual network (ResNet), AE and Multilayer Perceptron (MLP).

ProPythia [9, 136] is a generic Python-based free tool to apply ML and DL pipelines to protein classification problems. It has a modular nature and flexibility that makes the package able to adapt to different problems and fit the needs of the user. It offers a user-friendly possibility to implement ML/DL pipelines for protein/ peptide classification. It is a tool suitable for beginners because it gives the user control of the whole process without the need of writing extensive code. It is composed by 3 main groups, each one divided into independent modules. The first group is associated with the generation of the dataset and calculation of features, the second group is composed of unsupervised methods, and the third one is related to supervised methods, ML and DL. To the best of my knowledge, the ProPythia is the only package that allows performing extensive feature calculation and selection. It also allows the application of unsupervised and supervised algorithms, including DL, with an extensive set of options of visualization to analyze the results.

2.3.4 Relevant packages and tools

The code was implemented using the Python language. Taking this into account, the packages relevant to the completion of this thesis were Pytorch, TensorFlow, ProPythia, Pandas, NumPy, Autogluon and Fair-esm and Transformers.

- **Pytorch** is an open source high-level ML framework, which allows the implementation of DL models and the manipulation of numbers, vectors, matrices, or arrays in general. This package provides an array-based programming model accelerated by Graphics Processing Units (GPUs). Most of this package is written in C++ achieving a high performance. It is

also characterized as a library with optimization support for scientific computing in Python [45, 137].

- **TensorFlow** is an open source platform for ML frameworks, which was originally developed to conduct ML and DL research by the Google Brain team. This platform has a comprehensive, flexible ecosystem of libraries, tools, and community resources that allows researchers and developers to build and deploy ML frameworks [138].
- **Scikit-learn** is an open source platform for ML frameworks. However, this package only supports processes of traditional ML, covering supervised and unsupervised learning. Scikit-learn also provides methods for data preprocessing, model selection, model evaluation, and many other utilities [39, 139].
- **ProPythia** is a generic Python-based free tool to apply ML and DL pipelines to protein classification problems. As mentioned previously, it allows performing a extensive feature calculation and selection to protein sequences [9, 136].
- **Pandas** is a high-level building block for real-data analysis package in Python, which allows dealing with series, 1-dimensional data, and dataFrame, 2-dimensional data. Therefore, this package allows to implement a fast, flexible, and expressive data structures to work with “relational” or “labeled” data [140].
- **NumPy** is a package designed for scientific computing in Python. This package allows manipulating multidimensional structures, like matrices or arrays objects, masked arrays, and apply an assortment of routines on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. NumPy is also known by being a efficient and having a user-friendly package [141].
- **Autogluon** is a package from Amazon Web Services that automates ML tasks, enabling strong predictive performance in ML applications. The package enables easy-to-use and easy-to-extend AutoML with a focus on automated stack ensemble, DL, and real-world applications spanning image, text, and tabular data [73].
- **Fair-esm** is a package provided by Facebook AI which allows the download and loading of pre-trained transformer protein language models from Facebook AI Research. The Fair-

esm package allows the access to state-of-the-art ESM-2, ESM-1 and MSA Transformer to extract per-residue representation based on the pre-trained transformer [142].

- **Transformers** is an API provided by HuggingFace which allows access to pre-trained transformer models on a given generic dataset, also allowing the fine-tuning of pre-trained models with specific datasets. This API includes models from different fields such as NLP, Computer Vision, Audio, and Multimodal [55].

3

METHODS AND SOFTWARE DEVELOPMENT

3.1 PROCESSING AND ENCODING OF PROTEIN SEQUENCES

In order to apply the ML methods to protein sequences, it is necessary to convert the protein sequences into numeric values, as mentioned in the section 2.3.

The transformation of the protein sequences can be performed by different methods. Protein descriptors are biochemical properties that can be calculated from the sequence. Other techniques use encoding methods, such as one-hot encoding, BLOSUM, PSSM, and attention-based transformations that encode each amino acid residue into a numeric vector. These methods range from a simple matrix substitution to the use of transformers to encode the protein sequence.

The nature of the sequence may be related to some of the limitations of those methods. A common issue is that the transformation method cannot be executed in sequences that contain non-standard amino acids. Therefore, the uniformization of the protein sequences is usually required. The non-standard amino acids like Aspartic acid (B), Glutamic acid (Z), Selenocysteine (U) and Pyrrolysine (O) are frequently removed from the sequences, or replaced for the closest amino acid residue such as Asparagine (N), Glutamine (Q), Cysteine(C) and Lysine (K), respectively. Ambiguous amino acids, an amino acid letter (X and J) that can represent more than one amino acid residue, are also often removed.

While psycho-chemical descriptors are independent of the protein length, the output produced by encoders is dependent on the protein length, which may be a concern as the ML approaches cannot receive data with different dimensions. Therefore, it's necessary to scale the sequences in terms of length to produce a numerical representation equal in dimension that

can be fed to ML methods. In order to achieve this, amino acid residues are added to scale all sequences to the same length, N, amino acid residues. Generally, an ambiguous value is added at the end when the sequence is shorter than the selected length. The ambiguous value is either 0 or an ambiguous amino acid residue (X). The sequence can also be larger than the defined length. In such cases, a part of the sequence is deleted, thereby shortening the sequence by deletion of the amino acid residues.

3.1.1 Protein descriptors

The protein descriptors are features calculated based on the protein sequence. These methods calculate biochemical properties based on the amino acid residues that compose the protein sequence. The function for the calculation of the protein descriptors is available in open-source packages like ProPythia [9, 136]. This package allows calculating features such as physicochemical descriptors, amino acid composition descriptors, pseudo amino acid composition descriptors, auto-correlation descriptors, sequence order descriptors and others. For each sequence, a vector of chosen descriptors is produced, which can have a maximum of 9596 descriptor values.

3.1.2 Substitution matrix

One-hot encoding

The one-hot encoding replaces the amino acid residue into the binary vector as shown in figure 16. Uniformized sequences are composed of the 20 standard amino acids plus an ambiguous amino acid residue (X). Therefore, the binary vector generated for each amino acid residue (token) has a length of 21 tokens. A protein sequence encoded by this encoder will generate a matrix with 21 columns and a number of lines equal to the length of the sequence, in this case, a matrix of 21 columns and N lines.

BLOSUM encoding

As explained in section 2.3.1, the BLOSUM62 is the default substitution matrix for protein sequence alignment tools, such as BLAST. The BLOSUM50 is a matrix also used to identify the evolutionary conservation of protein sequences with a lower identity percentage.

The usage of the BLOSUM62 matrix allows encoding the standard and nonstandard amino acid residues, only excluding the ambiguous amino acid J. Each amino acid residue is encoded to a vector with a length of 24, producing a matrix of 24 columns and N lines for each sequence.

On the other hand, BLOSUM50 matrix allows the encoding of all amino acid residues. The amino acid residue is encoded in a vector with a length of 25, producing a matrix of 25 columns and N lines for each sequence. The columns represent the scores assigned to the likelihood of amino-acid substitution to other amino acid residues.

Non-Linear Fisher encoding

The NLF encoding [80] can only be executed in uniformized protein sequences, once any non-standard amino acid residues do not have a corresponding representation in this encoder. This method produces a vector of 18 numeric features for amino acid residue, therefore, the encoding of the protein sequence will generate a matrix with 18 columns and N lines.

Z-scale encoding

Similarly to the NLF encoding, the Z-scale [79] encoding can only be executed for uniformized sequences. In this method, each amino acid residue is represented by 5 attributes:

- Z1: Lipophilicity
- Z2: Steric properties (Steric bulk/Polarizability)
- Z3: Electronic properties (Polarity / Charge)
- Z4 and Z5: Properties relate electronegativity, the heat of formation, electrophilicity and hardness.

Therefore, the encoding of a protein sequence will generate a matrix with 5 columns and a number of lines equal to N.

3.1.3 Position Score matrix

The PSSM profile contains the evolutionary information of a sequence. This profile is generated by PSI-BLAST. The PSSM has shown to be a highly informative representation of the protein sequence [84].

In the first iteration of the PSI-BLAST program, a protein sequence is used as a seed to search and align to homologous sequences in the chosen database, similar to the BLASTp process. The second iteration is realized to calculate a profile or PSSM, that can capture a conservation pattern in the multiple alignment of sequences. The captured patterns are stored as a matrix of scores for each position, where the highly conserved patterns receive higher scores and the weakly conserved positions receive scores near zero.

For this application, the Swiss-Prot database was used (available in https://ftp.ncbi.nlm.nih.gov/blast/db/swissprot.tar.gz), a curated database of protein sequences. Each sequence is processed against the database, which can easily become a time-consuming method with the increment of the number of sequences to process.

Thus, for each protein sequence, a PSSM is generated, a log-odds matrix of the size of 20 columns, 20 standard amino acids, and N lines, the length of the sequence. Each amino acid residue is represented by the score of being mutated to other amino acid residues [84, 86].

3.1.4 Transformer encoding

As mentioned in section 2.2.3, the transformer is an attention-based architecture. These transformers were used as feature extraction methods. The models process the sequence, returning the last hidden-state. The transformers used include models from ProtTrans [63] and ESM [91–95].

ProtBert transformation

ProtBert is a transformer derived from BERT [57]. ProtBert was trained with around 216M proteins from the UniRef100 dataset [63] instead of NLP data. This pre-trained model can

be accessed from the package *transformers* provided by HuggingFace. The usage of this transformer allows for the extraction of features that captured biophysical properties related to protein shape. An amino acid residue is transformed into a numeric vector with a dimension of 1024. Therefore, a protein sequence is encoded into a representation with 1024 columns and N lines, the length of the sequence.

Evolutionary Scale Modeling

ESM are a series of transformer-model protein language models from Facebook AI Research, which are accessible with the package Fair-esm. This set includes models from the state-of-the-art ESM-2, ESM-1b and MSA Transformer (ESM-MSA-1), as well as the model ESM-IF1 for inverse folding prediction and ESM-1v for predicting variant effects. These models use the attention mechanism to learn the interaction patterns between pairs of amino acids.

The ESM models can produce different dimensions of transformation. Models with a lower number of trainable parameters, such as ESM2 with 8 million parameters, generate a vector of 320 features for amino acid residue. While models with a greater number of parameters, like the ESM2 with 15 billion trainable parameters, generate a vector of 5120 features for each amino acid residue. During this thesis, we tested the models ESM-MSA-1b, ESM-MSA-1, ESM-1b and ESM2. In the end, only the models ESM-1b, ESM2-150 and ESM2-650 were fully used and explored.

The ESM-1b model is composed of 33 layers with a total of 650 million trainable parameters, which produces a transformation of the amino acid residue into a vector of 1280 features, so each sequence is encoded into a representation with 1280 columns and N lines.

The ESM2-650, a newer version of the original model ESM-1b, is a model composed of 33 layers with a total of 650 million trainable parameters. This produces a representation with the same dimensions as the ESM-1b model. This model differs from the ESM-1b by being an updated version, trained with the new data from the reference databases.

The ESM2-150 model is composed of 30 layers with a total of 150 million trainable parameters. Producing a transformation of a vector of 640 features for the amino acid residue, so each sequence is encoded into a representation with 640 columns and N lines.

3.2 Deep Learning models

Deep Learning models from the literature were implemented to test each encoder's influence on the predictive ability of different models. Firstly, different models were tested with the PSSM encoder to assert the best possible models from the literature. Therefore, four models were selected for the tasks at hand. These were selected because of the better performance, the different types of architecture and different prediction tasks that were designed to do. The selected models were DeepPPF [6], DeepLoc [127], ET-GRU [124] and UDSMProt [118]. However, all models were adapted to be able to fit and train the encoders used. In all models, the activation function used for the neurons was the ReLU, for inner layers, while for the output layer softmax was used on multiclass tasks and sigmoid was adopted on binary classification tasks. The loss function used was categorical crossentropy on multiclass tasks and binary crossentropy on binary ones.

DeepPPF

DeepPPF [6] was designed to predict the functional family of a protein sequence, using a CNN architecture to identify and extract rich motifs that can be used to perform a prediction. The model receives as input the protein sequence, which is encoded by a trained embedding of W2V.

This model was adapted to receive the encoded sequence, where a dense layer was added, once this was shown to improve the model prediction capability. Figure 17 represents the main structure of the model used.

The encoded sequences are passed to three parallel CNN layers with 250 filters each, where the kernel size is defined as 20, 18 and 9, respectively, and the weights are initialized by the Glorot uniform. Each layer is max pooled to extract the maximum value of the convolution of each neuron. Thus, each kernel size for the max pooled operation is equal to the length of the convolution output. The results of the max pooling operation are added into a single matrix and regularized by dropout with a rate of 0.35. This output is then concatenated with the result of the max-pooling of the CNN layer with kernel 20 and regularized with a dropout rate of 0.35. The results from the convolution methods are then passed to a dense layer with 2000 units, then to

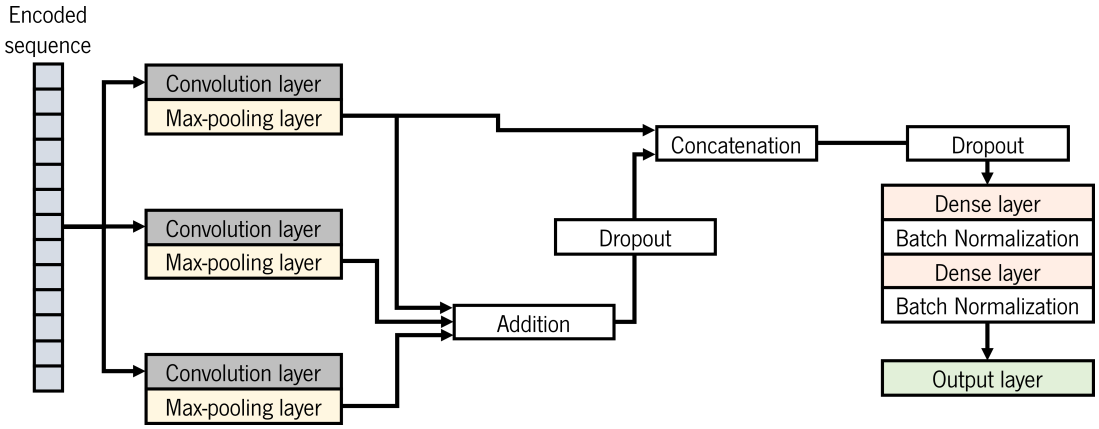


Figure 17 – Adapted DeepPPF architeture.

another dense layer with 500 units, with batch normalization after each dense layer. Lastly, the outcome is fed to the output layer.

DeepLoc

The **DeepLoc** [127] model was designed to predict the subcellular position localization of protein sequences. The proposed pipeline uses a set of CNNs to extract motif information from the sequence, followed by a bidirectional LSTM, and an attention decoder layer. This model takes as input the protein sequence encoded by the BLOSUM62 matrix. The attention decoder layer was removed from the original model once this layer could not be adapted from the original model. The adapted architecture can be seen in figure 18.

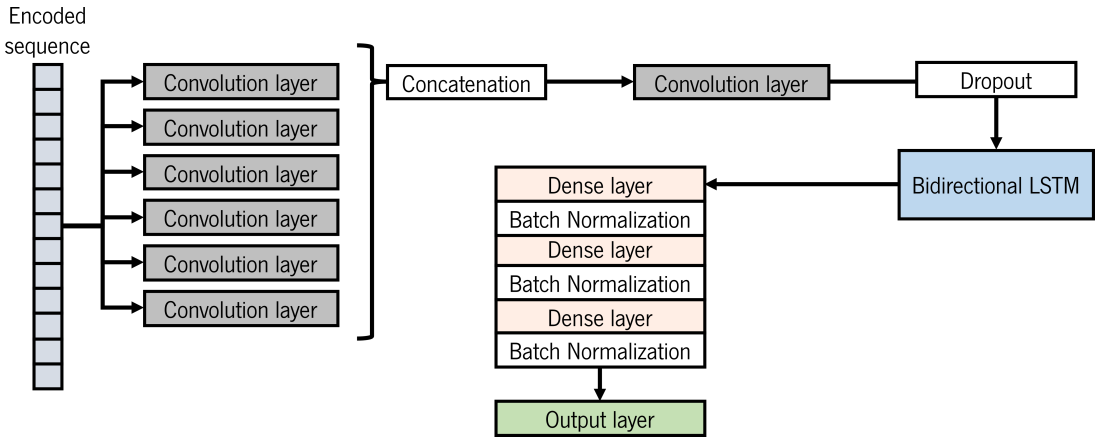


Figure 18 – Adapted DeepLoc architecture.

The encoded sequences are fed to six parallel CNN layers with 32 filters each, the kernel size is defined as 1, 3, 5, 9, 15 and 21, respectively, all with padding. The outputs from the CNN layers are concatenated and passed to a CNN layer with 64 neurons with a kernel size of 3 and padding. The output is regularized using dropout with a rate of 0.5 and passed to the bidirectional LSTM with 512 neurons. The result from the LSTM is received by a sequence of 3 dense layers with 500, 250 and 100 neurons, respectively, with batch normalization after each dense layer. Lastly, the outcome is fed to the output layer.

UDSMProt

UDSMProt [118] is a universal deep learning model for the classification of proteins based on the sequence of amino acids alone, which is based on an LSTM architecture. The UDSMProt uses an approach based on transfer learning, similar to the presented in ULMFit [143], to train the initial embedding layer. Therefore, the model was trained in the data present in the Swiss-prot at the time (2018) to obtain the representation of the proteins. Figure 19 represents the adapted model that was used.

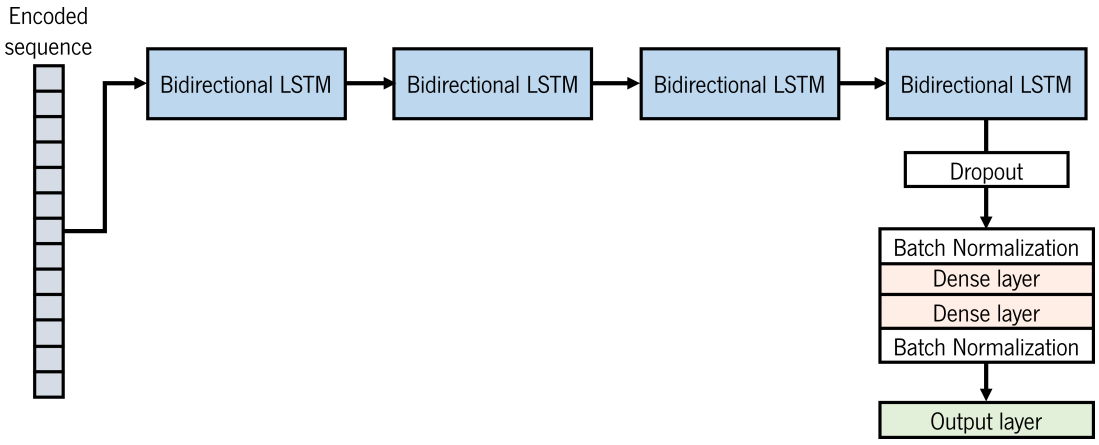


Figure 19 – Adapted UDSMProt architeture.

In UDSMProt, the encoded sequence is fed to four sequential bidirectional LSTMs with 128 neurons each. The output given by the last bidirectional LSTM is regularized with dropout with a rate of 0.25 and batch normalization. The normalized output is then fed to two sequential dense layers with 128 and 64 neurons, respectively. Then is normalized again by batch normalization and passed to the output layer.

ET-GRU

ET-GRU [124] is a multi-layer gated recurrent unit to identify electron transport proteins. ET-GRU is one of the few models designed to discriminate electron transporter proteins from other transporter proteins. The pipeline proposed by the ET-GRU is composed of a sequential CNN, followed by a GRU layer. This model originally receives as an input a PSSM profile of the proteins. Figure 20 represents the adapted model used.

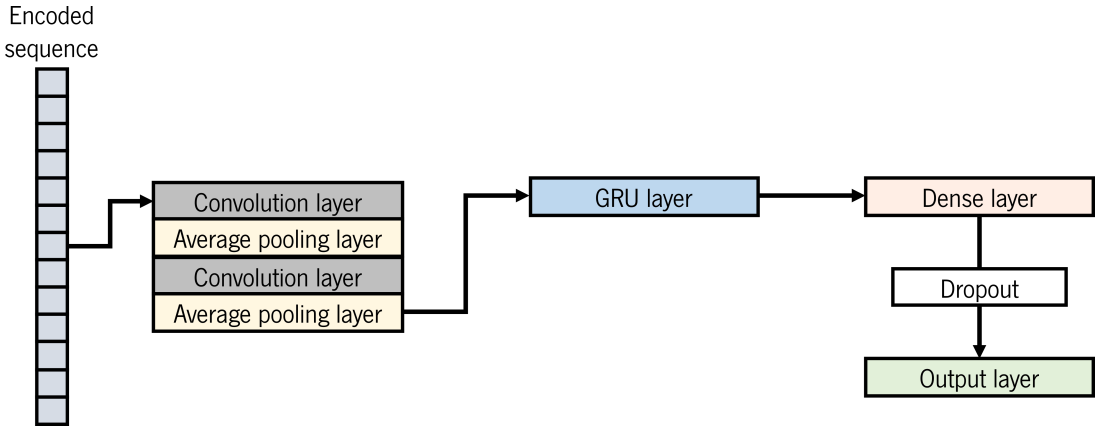


Figure 20 – Adapted ET-GRU architeture.

The encoded sequences are passed to a sequence of two CNN layers, with 250 filters and a kernel size of 3, and two average pools, with a pool size of 3. The result from this operation is passed to a GRU layer with 150 neurons and a dropout of 0.01. Finally, it is fed to a dense layer with 32 neurons, normalized with a dropout rate of 0.5 and fed to the output layer.

4

DEVELOPMENT

4.1 OMNIUMAI

The thesis was accomplished in collaboration with OmniumAI, a company related to the bioinformatics and artificial intelligence fields. OmniumAI provides solutions in the fields of AI and Data Sciences, including software development, consulting and tailored training. The main core of the company is the AI field. Therefore, OmniumAI focuses on offering enriched solutions for biological and biomedical data processing, analysis, mining, and integration. This company was founded in 2021 as a spin-off company from the Center of Biological Engineering, University of Minho.

The OmniumAI platform consists of a set of methods for processing and analyzing biological data. At the moment, it includes the following Python sub-packages:

- **Generics**: a generic sub-package designed to load data, transform data, and training AI models. Therefore, this sub-package is responsible for providing the methods to handle the data and its transformation. All the packages listed below depend on Generics to apply the methods specific to the data.
- **Compounds**: this sub-package contains specific methods for compound data. This sub-package is composed of methods capable of performing compound feature extraction, compound standardization, and molecular splitting.
- **Genes**: this sub-package contains specific methods to deal with DNA sequences. It is composed of methods to perform DNA feature extraction.

- **Metabolomics**: this sub-package contains specific methods for metabolomics data.
- **Proteins**: this sub-package contains specific methods to handle protein sequences. In the sub-package, a set of methods are implemented to perform protein standardization, calculate protein descriptors and perform the encoding of protein sequences.
- **Transcriptomics**: this sub-package contains specific methods for RNA-seq data. It is composed of methods capable of performing RNA-seq data parsing, RNA-seq data processing, and RNA-seq data feature selection.
- **Text mining**: this sub-package contains specific methods for textual data. The methods included in this sub-package allow text feature extraction and text processing for NLP approaches.

A full view of the Ominia architecture can be seen in figure 21.

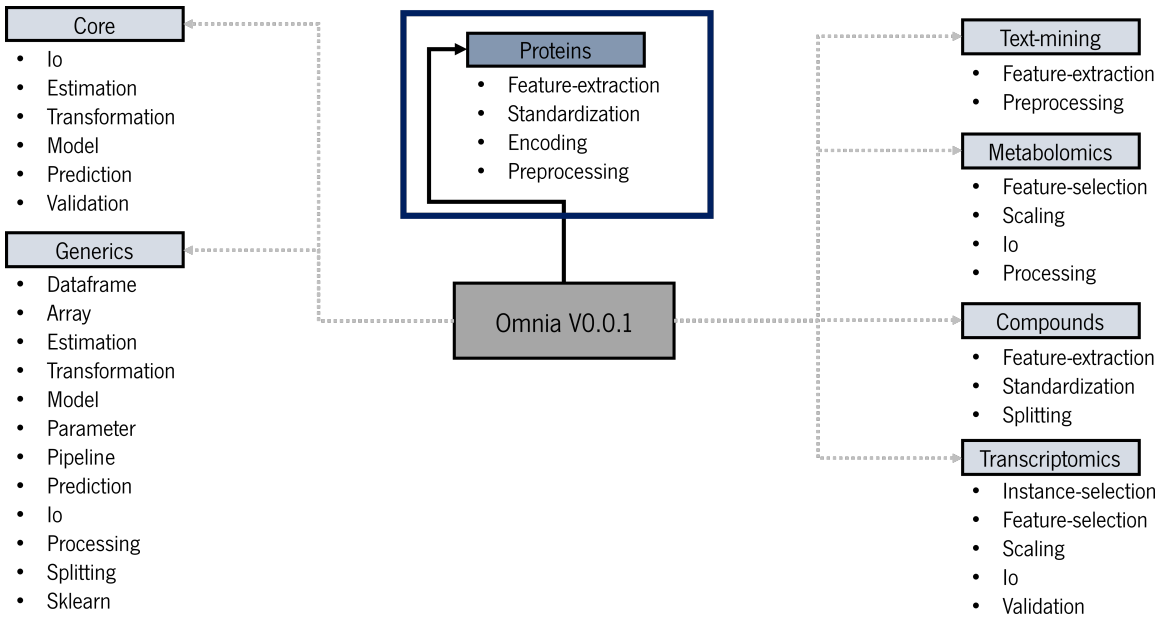


Figure 21 – Architecture of Omnia package and the typology of methods implemented in each sub-package.

The main purpose of this work is to further develop the Proteins sub-package of the OmniumAI platform, through implementation of different methods and evaluation of their ability to process protein sequences. These methods were then implemented into the AutoML platform of OmniumAI.

4.2 OmniumAI and ProPythia methods implementation

4.2.1 Feature extractors

Firstly, we addressed the implementation of the methods of processing and feature extraction for protein sequences in the sub-package Proteins. The methods were implemented in the OmniumAI platform, more specifically in the Omnia package.

As mentioned in section 2.3.3, the open-source ProPythia package already contained methods for feature extraction of protein sequences, some of which mentioned in the section 3.1. The feature extracting methods include the protein descriptors and encoders such as BLOSUM, one-hot, NLF and the Z-scale. The initial step was the improvement of these methods in the ProPythia package.

During that phase, the first stage consisted of fixing some bugs and removal of features at the request of the main author of the package. Then, code optimization was carried out and the addition of the parallelization methods was done to allow the execution of feature extraction in a multicore setting. The next step was to adapt these methods to be compatible with the OmniumAI platform and assure that the features generated were equal to the original source code.

Later in the development of this thesis, transformers were also implemented as feature extractor methods. These methods were implemented from scratch using pretrained transformers. Despite multiple pretrained transformers from the ProtTrans [63] and ESM [142] being tested, only 4 of these transformers were implemented into the OmniumAI platform as well as into the ProPythia package. Figure 22 and 23 represent the methods implemented on the OmniumAI platform and ProPythia package, respectively.

However, the position score matrix method depends on a local installation of the PSI-BLAST program. Therefore, this method was not implemented in either of the platforms. Finally, it was necessary to validate the functionality of the methods with the AutoML architecture implemented in the company platform.

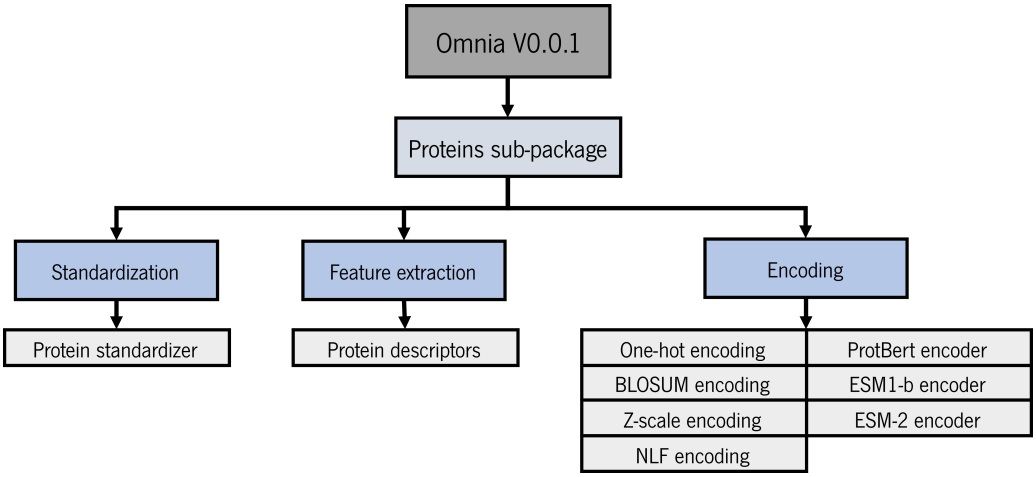


Figure 22 – Proteins sub-package in the OmniumAI platform. The methods implemented in the Omnia package.

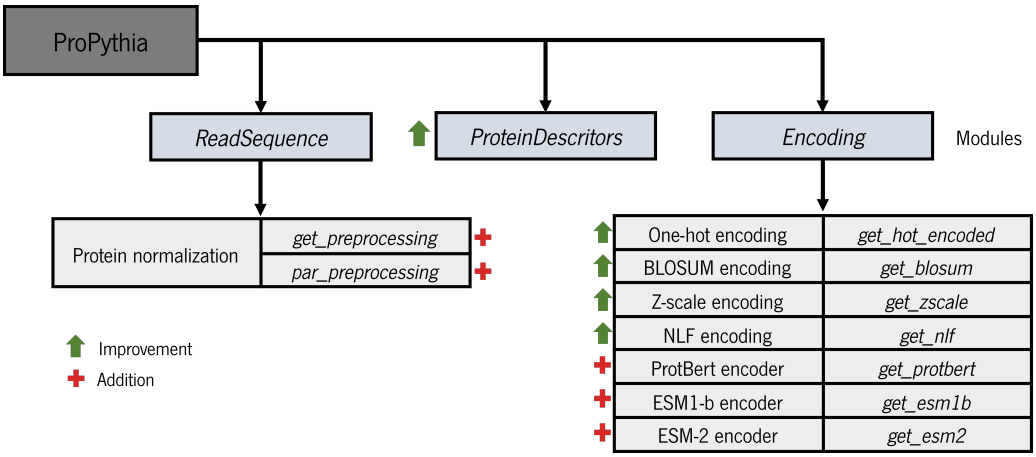


Figure 23 – Development in the ProPythia package. The alterations of the main source code with a green arrow the methods that were fixed or optimized and with a red cross the added methods to ProPythia package.

4.2.2 OmniumAI pipeline

The Generics sub-package of the company contains ML models, such as RF, SVMs, KNN, and linear models, and also contains simple neural networks. An AutoML pipeline that encompasses the feature engineering to the model evaluation was created. The pipeline extracted the protein descriptors features from the protein sequences, exploring different combinations of features. This pipeline fully processed the selected datasets and trained the models available in the Generics sub-package. This process is controlled by the Autogluon package.

The current AutoML method in the OmniumAI platform requires the definition of the dataset, feature extractor method and the ML models to test by the user. Therefore, the pipeline receives a dataset with the protein sequences and the labels required for the classification task. From this point, the AutoML tool does the feature engineering, model generation and model evaluation. Subsequently, it displays, in a table format, the results achieved by the trained models, sorted by the best-performing model to the least-performing model. Therefore, it is only required the user define the dataset, the feature extractor method and the ML models to train, while the remaining process is controlled autonomously by Autogluon. Figure 24 represents the chaining of events during the execution of the AutoML pipeline created. It also contains the different elements required to execute the pipeline.

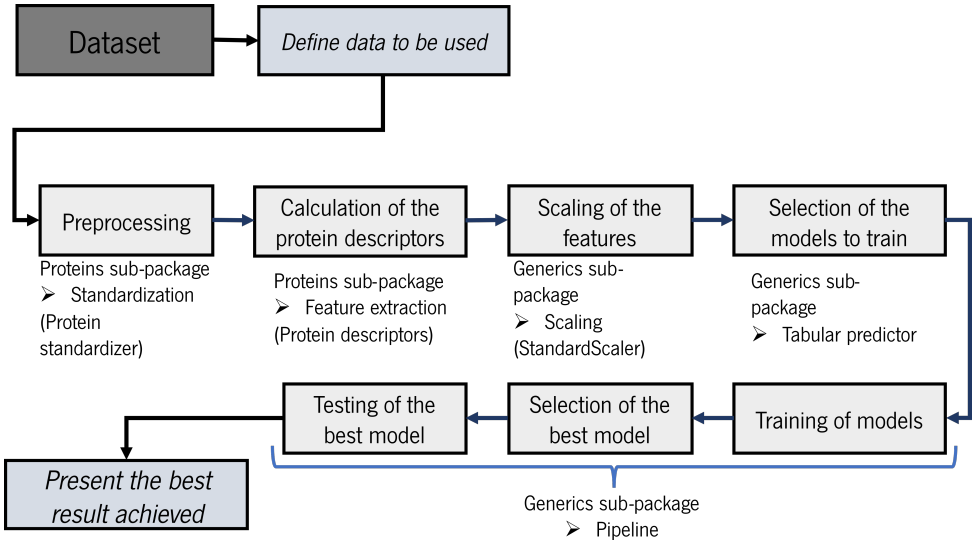


Figure 24 – An overview of the OmniumAI pipeline. It is indicated the steps for the completion of the pipeline and the sub-packages/modules required.

The encoders that produce the two-dimensional representations of proteins are often used to train DL architectures. However, the lack of support for more advanced deep learning structures, such as CNN, LSTM and others, led to research of encoders studied externally to the OmniumAI platform. The main intention of this external study was to verify the performance of these representations. The DL models used are explained in detail in section 3.2 and the training procedure in section 4.3.

4.3 Training and evaluation of the encoders

Each dataset was split into three data groups. From the original dataset, 80 % was used as training data, employed to train the model. 14 % was used as validation data, which is also used during the training of the model. The remaining 6 % of the dataset was used as the test data for evaluating the performance achieved by the model. This split of the dataset was realized by stratified sampling, allowing for an equal representation of the different classes in all data groups.

During the training of the models, callback functions were used. These functions allow to the application of routine or action inside the outer function of training the model. The callbacks used were *ModelCheckpoint*, *EarlyStopping* and *ReduceLROnPlateau*.

ModelCheckpoint is a function that allows saving the model during the training phase, with a frequency that can be defined by the user. Therefore, the model can be saved in a fixed number of epochs or by monitoring the performance of the model.

EarlyStopping is a function used to monitor the fitting metric and stop the training phase when no improvement occurs on a monitored metric during more than the defined value of epochs.

ReduceLROnPlateau is a function used to reduce the learning rate during the training of a model. If during the training process, a monitored metric does not improve for the defined value of epochs, the function will reduce the learning rate by multiplying it by a factor.

The improvement of the models was monitored by the metric of loss value for validation data. *ModelCheckpoint* was used to save the model based on the performance achieved, by monitoring the loss value for the validation data and saving the model with a lower value. The parameters used for *ReduceLROnPlateau* was a reduction factor of 0.2, when the loss value for the validation data did not improve for longer than 15 epochs. The training phase was stopped by the *EarlyStopping* when the loss value stopped improving for more than 80 epochs, except for the ET-GRU, which was stopped at 90 epochs.

The encoders used for the DL approaches include one-hot encoding, BLOSUM encoding, PSSM encoding, ProtBert transformation and ESM transformation (ESM-1b, ESM2-150 and ESM2-650). The batch size used for the encoders was 32 sequences with the optimizer

Adam. The initial learning rate used was 0.01, 0.0001, 0.001, 0.0001 for the adapted DeepPPF, DeepLoc, UDSMProt and ET-GRU, respectively.

The evaluation of the trained models was realized by loading the model with the best performance saved by the *ModelCheckpoint* callback. Then, the loaded model predicts the class of the sequences of the test data and compared them to the label associated with each sequence (real label). The metrics used to analyze the performance were BAcc, MCC, Prec and F_1 score.

5

RESULTS AND DISCUSSION

This chapter will describe the process used for the collection of protein data, creation of datasets and the results obtained with different combinations of encodings and DL models for two case studies. The models DeepPPF, DeepLoc, UDSMProt and ET-GRU referenced during this chapter corresponds to the adapted version implemented in the section 3.2 and trained/ evaluated by the procedure explained in the section 4.3.

The case studies chosen were enzyme and transporter classification. Enzyme classification is an important research topic with several applications, therefore, some ML and DL methods have been developed. The state-of-the-art literature have shown models with a good performance in tackling this problem, therefore, were used as a proof of concept. The classification of transporters is also an important topic to address. However, the results that have been reported are still far from being considered of good quality. Therefore, this will be the case study that we are going to focus on more on this work, seeking to obtain well performing models for this challenging task.

The databases Swiss-Prot and Uniref were used to collect the data. The choice was based on the fact that Swiss-Prot is a database composed only by curated protein sequences. The Uniref database allows the removal of sequences with a similarity above a given threshold. In this case, sequences with more than 90 % similarity were not considered.

5.1 Enzymes case study

5.1.1 Collection of enzyme sequences

The enzyme sequences were retrieved based on datasets described in the literature. The datasets from ECPRED [144] and Amidi et al. [145] articles were used as the starting point for the enzyme data. These articles were published in 2017, thereby it was appropriate to update the dataset to the current information. Only sequences with Uniprot Identification (ID) belonging to Swiss-Prot and Uniref90 were maintained. This was accomplished by using the ID mapping tool of Uniprot. This data was accessed on June 21, 2022.

The sequences were filtered, removing sequences that had associated more than one EC number from different main class, the first digit as referenced in section 2.1.4. For example, a enzyme with EC numbers of 1.2 and 1.3 was kept as 1, whereas a enzyme with EC number 2.5 and 1.2 was deleted. Sequences with a length smaller than 50 and greater than 2000 were also excluded.

This process resulted in a collection of 74791 enzyme sequences.The distribution of the enzyme sequences by EC main class can be seen in table 3.

Table 3 – Distribution of the enzymes sequences. The enzymes are distributed by the associated EC main class

	EC main class						
	1	2	3	4	5	6	7
Number of enzymes	8779	27822	14180	7208	4627	10441	1734

The dataset was uniformized, by replacing all non-standard amino acid residues with the closest amino acid residue, as mentioned previously.

In order to handle enzyme sequences with different lengths, a length threshold was defined and sequence padding and truncation were applied. The values of 1000, 700 and 500 amino acid residues were tested for this threshold. The threshold of 500 amino acid residues allowed to reduce computational costs without affecting the performance of the classification, and thereby

was the proposed length. Proteins longer than 500 amino acids were truncated at the end, while shorter sequences were padded, adding an ambiguous amino acid residue (X) also at the end of the sequence. It is important to note that most of the enzyme sequences had a length between 250 and 500 amino acid residues.

Finally, the homogenized sequences were encoded using one-hot, BLOSUM and PSSM encodings as described in section 3.1. The encoders are a deterministic transformation. Therefore, a sequence will always produce an equal representation for the encoder used.

5.1.2 Classification of Enzymes

As previously mentioned, the task of enzyme classification was used as a proof of concept. In order to assess the prediction capability of a predictor, BLOSUM, PSSM and one-hot encoders were fed into the models described in section 3.2. Initially, only these three encoders were executed. The metrics achieved for the prediction of the test data can be seen in table 4.

Table 4 – Performance of all models for EC class prediction through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between models are indicated in bold

Models	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
DeepPPF	MCC	**0.905**	0.873	**0.978**
	BAcc	**0.912**	0.876	**0.978**
	Prec	**0.927**	0.905	**0.983**
DeepLoc	MCC	0.840	0.845	0.954
	BAcc	0.843	0.844	0.958
	Prec	0.876	0.880	0.965
UDSMProt	MCC	0.842	**0.891**	0.967
	BAcc	0.844	**0.890**	0.972
	Prec	0.880	**0.915**	0.975
ET-GRU	MCC	0.777	0.750	0.937
	BAcc	0.762	0.745	0.936
	Prec	0.829	0.809	0.951

The encoders tested were capable of generating an informative representation of the amino acid residues to achieve a good performance. The performance achieved by the BLOSUM encoder and the one-hot encoder are similar within the same model, and both are outperformed by the PSSM encoder.

The models achieved different performances using the same encoders. The DeepPPF achieved the best overall performance. The model of ET-GRU was the model that obtained the lower performance among the models. The models originated from the models DeepLoc and UDSMProt achieved similar results for all the encoders.

PSSM encoder had a better performance than the BLOSUM and one-hot encoding. As the literature indicates, this method generates a informative representation of the protein using the multi alignment of sequences to generate a consensus scoring of the sequence. The use of this encoder seems the most important factor to improve prediction ability of the models in these experiments.

5.2 TRANSPORTERS CASE STUDY

The main interest of this thesis is to analyze the capability of different encoders to generate an informative representation of protein sequences for more complex problems, such as the classification of transporter sequences.

5.2.1 COLLECTION OF TRANSPORTER SEQUENCE

There is no dataset available from literature for the classification of transporter proteins in the complete range of the TC, and so the first task was to build one that would fit the purpose. The positive cases, ID and transporter sequences, were retrieved from the TCDB database downloading a FASTA file. The FASTA file contains a total of 21803 protein sequences. The protein sequences were first filtered by mapping the sequences present in the Swiss-Prot and Uniref90 databases. This was implemented to retain only the curated data without identical sequences.

Usually, protein sequences with less than 50 amino acid residues are associated to fragments of virus proteins. Therefore, sequences with less than 50 amino acid residues were excluded. Sequences with TC classes of 6, 7, 8, and 9 were also excluded as they have no interest for this work, as explained in the section 2.1.4. The classes 6 and 7 are considered empty classes, reserved for future classes. Class 8 and class 9 are constituted by accessory proteins and uncharacterized proteins, respectively.

The sequences considered as negative examples were obtained from the Swiss-prot database using the conditions: “NOT transporter AND NOT transporter activity”. A total of 550334 protein sequences were retrieved. All sequences with a length greater than 2000 amino acids and smaller than 50 amino acids were excluded. The information, for both cases, was collected on March, 18th 2022.

After the filtering process, positive cases were composed of 6225 transporter sequences and the negative cases were composed of 547462 protein sequences.

Given the unbalanced numbers in the two classes, the datasets were obtained by taking all transporter sequences (positive cases) with a selected number of randomly chosen negative cases. The classification of transporters in different TC classes was studied with and without the inclusion of non-transporter sequences. The number of non-transporter sequences added corresponds to the mean of the transporter sequences by class. The classification was executed to different levels of the TC system referenced in section 2.1.4. The levels of the TC system used as classification tasks are the following:

- Classification of the TC class of the transporter sequence.
- Classification of the TC subclass of the transporter sequence.
- Classification of the TC family of the transporter sequence.
- Classification of the TC superfamily of the transporter sequence.

Table 5 represents the distribution of sequences for the classification of the main class associated to the transporter sequence. This dataset has a total of 6225 transporter sequences and 1245 non-transporter sequences. The class represents if the protein is a channel, carrier/porter, primary active transporter or group translocator.

For the remaining classification tasks, the transporter sequences were firstly grouped by the TC level of interest (subclass, family, or superfamily), and the groups with less than 50 transporter sequences were excluded.

Table 5 – Distribution of the transporter sequences based on the TC class

TC class	Number of sequences
Non-transporter	1245
TC 1	1929
TC 2	2079
TC 3	1968
TC 4	153
TC 5	96

Table 6 shows the distribution of the sequences accordingly to their transporters subclass. This dataset has a total of 5835 transporter sequences and 583 non-transporter sequences. The grouping of the transporter sequences by class and subclass was based on identical transporter class and energy source used to drive the transport.

Table 6 – Distribution of the transporter sequences based on the TC subclass

TC subclass	Number of sequences
Non-transporter	583
TC 1.A	881
TC 1.B	271
TC 1.C	352
TC 1.F	60
TC 1.I	166
TC 2.A	2071
TC 3.A	1507
TC 3.D	383
TC 4.A	93

Continued on next page

TC subclass	Number of sequences
TC 5.A	51

The classification by transporter family consists of 2808 transporter sequences and 165 non-transporter sequences. The dataset is represented in the table 7. In this method, each group includes sequences that have the same class and subclass, and belong to the same transporter family.

Table 7 – Distribution of the transporter sequences based on the TC family

TC family	Number of sequences
Non-transporter	165
TC 1.A.1	166
TC 1.A.8	56
TC 1.A.9	51
TC 1.I.1	152
TC 2.A.1	457
TC 2.A.3	105
TC 2.A.6	53
TC 2.A.7	153
TC 2.A.29	122
TC 3.A.1	752
TC 3.A.2	168
TC 3.A.3	102
TC 3.A.5	77
TC 3.A.16	65
TC 3.A.31	53
TC 3.D.1	203
TC 3.D.4	73

Alternatively to the TC classification system, a dataset using the TC superfamily classification was made. This dataset includes 3292 transporter sequences and 164 non-transporter sequences. The distribution of the dataset is shown in table 8. This grouping differ from the previous TC categorization of the transporter sequences, as two sequences with different class or subclass can belong to the same superfamily. Instead of classifying the transporter sequence based on the physical attributes, the superfamily classifies them based on common ancestry. The superfamilies are defined in https://tcdb.org/superfamily.php.

Table 8 – Distribution of the transporter sequences based on the TC superfamilies

TC superfamily	Number of sequences
Non-transporte	164
AAA-ATPase	111
Cation Diffusion Facilitator	74
Cation:Proton Antiporter	96
Drug/Metabolite Transporter	153
Endomembrane Protein-Translocon	66
Iron-Sulfur Protein	98
Ion Transporter	98
Major Facilitator	603
Major Intrinsic Protein	62
Mercuric Ion Pore	122
Mitochondrial Carrier	309
Outer Membrane Pore-forming Protein I	229
Protein Kinase	233
P-type ATPase	102
Phosphotransferase (or PTS-GFL)	67
Resistance-Nodulation-Cell Division	53
Tetraspan Junctional Complex Protein	66
Acid-polyamine-organocation	403
Transporter-Opsin-G protein-coupled receptor	73
Voltage-gated Ion Channel	274

In this work, the model ability to distinguish transporter proteins from non-transporters was also evaluated. To this end, a dataset for binary classification with different ratios of the number of examples for positive and negative cases was created. For example, a 1 to 1 ratio means transporter sequence to 1 enzyme sequence ratio. The number of examples in each class, by ratio, can be seen in the table 9.

Table 9 – Distribution of the transporter and non-transporter sequences. The A to B ratio equals to A transporter sequences to B enzyme sequences ratio

	Number of sequences				
Transporter to Non-transporter ratio	1 to 1	2 to 1	3 to 1	1 to 2	1 to 3
Transporter	6283	6283	6283	6283	6283
Non-transporter	6283	3141	2094	12566	18849

Similar to what was made to the enzyme sequences, non-standard amino acid residues were replaced by the closest standard amino acid residue. A threshold length was defined to truncate and pad the sequences to obtain same length sequences. The length with the best compromise between computational resources required, and classification performance was 600 amino acid residues. Padding and truncation were made at the end of sequence.

First, we will discuss the results for the multi-classification of transporters without non-transporter sequences. Afterwards, the multi-classification of transporters with non-transporter sequences and finally a binary classification of transporter and non-transporters.

5.2.2 Classification of transporters

The initial encoders tested in the enzymes were used to encode the transporter sequences without the inclusion of non-transporter sequences. The aim is to predict the main class (N1) of the transporter sequences. The performance of the by different models in the prediction of the test data can be seen in table 10.

In the classification of the transporter class, the BLOSUM encoder and the one-hot encoder were not capable of supplying an equal representation of the sequence compared to the PSSM.

Table 10 – Performance of all models on TC main class prediction through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between models are indicated in bold

Models	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
DeepPPF	MCC	**0.616**	0.613	**0.869**
	BAcc	0.326	0.322	**0.759**
	Prec	**0.711**	0.702	**0.911**
DeepLoc	MCC	0.486	0.628	0.788
	BAcc	0.280	**0.395**	0.674
	Prec	0.632	**0.747**	0.854
UDSMProt	MCC	0.535	0.536	0.853
	BAcc	**0.357**	0.367	0.659
	Prec	0.667	0.698	0.897
ET-GRU	MCC	0.599	**0.640**	0.778
	BAcc	0.317	0.338	0.528
	Prec	0.698	0.721	0.853

The one-hot encoder either achieved a prediction capability similar or lower than the BLOSUM encoder in the trained models. However, there's not a fixed trend between the performance achieved and the model used. Due to the imbalance of the dataset, classes 1, 2 and 3 are overrepresented relative to classes 4 and 5. The balanced accuracy achieved in these encoders are usually lower than 0.4, while the weighted precision is around 0.7. This indicates that the predictions are not too accurate, but still most of the test sequences from the overrepresented classes were properly predicted. The best performance was achieved by the PSSM encoder, that, similarly to what happened in the enzyme dataset, outperformed the matrix substitution methods.

The DeepPPF was responsible for generating most of the highest prediction scores. This model achieved a MCC of 0.86 for the transporter sequences.

Then, the classification tasks for the TC subclass, family and superfamily were evaluated. The results achieved for the DeepPPF, DeepLoc, UDSMProt and ET-GRU models are presented in the tables 11, 12, 13 and 14, respectively, for the multiclass classification of transporters.

Table 11 – Performance of DeepPPF on TC subclass, family and superfamily classification through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.562	0.552	0.853
	BAcc	0.336	0.350	0.758
	Prec	0.686	0.663	0.893
TC family	MCC	**0.782**	**0.803**	**0.967**
	BAcc	**0.679**	**0.705**	**0.951**
	Prec	**0.803**	**0.821**	**0.976**
TC superfamily	MCC	0.657	0.696	0.918
	BAcc	0.530	0.586	0.910
	Prec	0.655	0.745	0.928

Table 12 – Performance of DeepLoc on TC subclass, family and superfamily classification through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.541	0.680	0.807
	BAcc	0.311	0.609	0.723
	Prec	0.619	0.753	0.857
TC family	MCC	**0.652**	**0.734**	**0.933**
	BAcc	**0.532**	**0.598**	**0.873**
	Prec	**0.686**	**0.769**	**0.953**
TC superfamily	MCC	0.601	0.718	0.846
	BAcc	0.522	0.612	0.830
	Prec	0.650	0.736	0.848

Table 13 – Performance of UDSMProt on TC subclass, family and superfamily classification through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.595	0.587	0.785
	BAcc	0.405	0.378	0.646
	Prec	0.677	0.672	0.828
TC family	MCC	**0.837**	**0.775**	**0.899**
	BAcc	**0.749**	**0.663**	**0.870**
	Prec	**0.863**	**0.820**	**0.924**
TC superfamily	MCC	0.707	0.729	0.890
	BAcc	0.663	0.695	0.846
	Prec	0.743	0.752	0.897

Table 14 – Performance of ET-GRU on TC subclass, family and superfamily classification through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.672	0.639	0.790
	BAcc	0.529	0.403	0.569
	Prec	0.744	0.705	0.832
TC family	MCC	**0.764**	**0.710**	**0.906**
	BAcc	**0.635**	**0.555**	**0.856**
	Prec	**0.805**	**0.714**	**0.930**
TC superfamily	MCC	0.714	0.696	0.879
	BAcc	0.612	0.556	0.851
	Prec	0.740	0.703	0.886

The decrease in the biological variety of the transporter sequences by grouping them by subclass, family or superfamily, caused the increase of the prediction capability of all models. In general, the prediction of the subclass had a better performance than the prediction of the class. The prediction of the superfamily achieved better metrics than the prediction of the TC subclass and class in all the models, but is often surpassed by the classification of family. Despite the TC system being analogous to the EC system, it was verified that some of the main classes had a larger number of families. For example, the class 1, channels/pores, contained more than 700 TC families. This includes transporter proteins with multiple subcellular localization and organisms. On the other hand, the class 5, transmembrane electron carriers, only contained 18 TC families.

Similar to the previous results, the PSSM was the best encoder for the classification on any task. When comparing the one-hot and BLOSUM encoding, none of these methods was superior across all models. While the BLOSUM encoding performed better for models such as DeepLoc and DeepPPF, the one-hot encoding performed better combined with the UDSMProt and ET-GRU models.

Regarding the used models, the best-performing was the DeepPPF achieving close to 0.97 of MCC with the PSSM encoder without the non-transporter sequences for the TC superfamily task. This model frequently achieved the best performance for a task with the encoder PSSM.

Less convincing results in the learning phase of the models using the BLOSUM and one-hot encoder were verified. The performance achieved by these encoders varied substantially based on the random sample at the start of the model. This instability caused some of the early models not to be able to adjusting the model weights properly. The MCC scores ranged from -0.1 to the presented results. Therefore, the representation of the protein sequences is not highly informative. It is important to notice that these encoders require low computational resources, producing a representation of the protein sequences in less than 0.01 seconds.

PSSM profiles are capable of generating an informative representation of the protein sequences. This allowed for the models to learn the task at hand. However, PSSM profiles have some restrictions and disadvantages. PSSM requires running the PSI-BLAST algorithm for each protein sequence against a database. This means that in order to obtain a PSSM profile, the query sequence needs to have a hit in the database. In the datasets used, two sequences were not able to produce a PSSM profile, even though all sequences belong to a

database composed of only curated sequences. Therefore, for newly discovered sequences, this method can be ineffective. Another drawback of this method is that it may be time-consuming when used on a larger scale. In average, each PSSM profile took around 12 seconds to be generated from a FASTA file and saved into a PSSM file. The disk space required to save the PSSM files can also be a limiting factor for larger-scale applications. For example, the PSSM files for all the sequences used in this thesis use a total of 15 Gigabytes to store. Because of these disadvantages, more exploration was performed to obtain other effective options for the representation of protein sequences.

Recently, the pre-trained transformers with protein sequence data have received a spotlight for classification tasks. Therefore, the usage of transformers to generate a representation of the protein was analyzed. Different transformer encoders were selected from the ProtTrans article [63] and from the ESM [142]. However, not all tested transformers are presented. The results from the best four performing transformers are given in tables 15, 16, 17 and 18 for the models DeepPPF, DeepLoc, UDSMProt and ET-GRU, respectively. Only the TC family and TC superfamily tasks were used in this phase.

Table 15 – Performance of DeepPPF on TC family and superfamily classification through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	**0.960**	**0.980**	**0.960**	**0.993**
	BAcc	**0.939**	**0.957**	**0.946**	**0.979**
	Prec	**0.968**	**0.987**	**0.974**	**0.995**
TC superfamily	MCC	0.917	0.967	0.956	0.956
	BAcc	0.890	0.951	0.940	0.943
	Prec	0.931	0.973	0.964	0.965

The DeepPPF model achieved the best performance for the majority of the encoders in the classification tasks. It was capable of achieving a MCC score of 0.99 in the ESM2-650 encoder for the classification of TC family. Between the ET-GRU, DeepLoc and UDSMProt models, none was capable of achieving a better performance in all encoders in comparison to the others. In other words, the prediction capability of these models depends on the encoder used.

Table 16 – Performance of DeepLoc on TC family and superfamily classification through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	**0.892**	**0.966**	**0.898**	**0.940**
	BAcc	**0.838**	**0.936**	**0.824**	**0.910**
	Prec	**0.913**	**0.976**	**0.913**	**0.958**
TC superfamily	MCC	0.840	0.929	0.924	0.918
	BAcc	0.768	0.903	0.917	0.883
	Prec	0.848	0.939	0.937	0.932

Table 17 – Performance of UDSMProt on TC family and superfamily classification through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	0.872	**0.960**	**0.966**	0.933
	BAcc	0.804	0.919	**0.926**	0.896
	Prec	**0.897**	**0.970**	**0.972**	0.949
TC superfamily	MCC	**0.885**	0.951	0.913	**0.945**
	BAcc	**0.838**	**0.935**	0.899	**0.934**
	Prec	0.893	0.959	0.932	**0.951**

Table 18 – Performance of ET-GRU on TC family and superfamily classification through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	0.878	**0.953**	**0.919**	0.926
	BAcc	0.773	**0.938**	0.863	0.878
	Prec	**0.886**	**0.961**	**0.938**	0.942
TC superfamily	MCC	**0.890**	0.945	0.896	**0.945**
	BAcc	**0.834**	0.922	**0.897**	**0.934**
	Prec	0.885	0.955	0.910	**0.959**

Despite the best performance achieved by the ESM2-650 encoder, the ESM-1b pretrained transformer achieved the best overall performance when considering all trained models. The ESM-1b was able to slightly outperform the ESM2-650 and the ESM2-150. While the ESM-1b obtained metrics over 0.93 in all the tests, the ESM2-650 and ESM2-150 frequently showed metrics below 0.90. The authors claimed that the ESM transformers of second generation (ESM-2) were more capable to represent the protein sequence than the ESM-1b transformer. However, this claim was not confirmed in this scenario.

The pretrained transformers based on the NLP approaches and adapted for protein data were the least capable of generating a good representation of protein sequences when compared to the ESM transformers. The ProtBert transformer was the best adapted NLP approach. However, these methods were able to outperform the BLOSUM and one-hot encoder. Protbert also achieved similar or better performance than the PSSM profile previously tested.

The transformers are faster in obtaining the representation than the PSSM encoders. On average, the transformers took around 2 seconds to obtain the representation for each protein. This process was accomplished in CPU, which increases the time to obtain the representation for each protein. The transformers can be executed in GPU, if these resources are available, which allows to obtain the protein representation in the same time frame as the BLOSUM and one-hot encoding. However, the transformers require more computation resources to obtain the

protein representations than the previously tested encoders, due to the computation requirement to load and process the pre-trained transformers.

Additionally, the higher complexity of the protein representation caused an increase in the time required for training the DL models. For context, the encoders previously tested only required a GPU to train a model, while the transformer encoders require 2 GPUs. It also took the double of the time to converge in the optimal model. However, this computational cost was proven to be worth it in recent works for the prediction of more complex tasks [107, 142, 146].

In the work developed along this thesis, the ESM transformers produced the most informative representation of the proteins, providing quality data to the models. Despite the disadvantages related to the computational resources of these methods, they solve the major drawbacks of the PSSM encoders. They are not dependent on a database, therefore, they always produce a representation. They are also capable of generating a representation informative that can be applied to different studies, which the encoders BLOSUM and one-hot could not perform.

Considering the models trained, and the performance achieved, the DeepPPF model achieved a better overall performance in the prediction of the transporters TC. However, the prediction of the TC class and subclass with transformers was not tested during the previous trials. Therefore, the model DeepPPF and the ESM-1b encoder were trained for these tasks, due to be the best overall combination so far. The performance achieved is presented in the table 19.

Table 19 – Performance of DeepPPF on TC class and subclass classification through ESM-1b encoder. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec)

Model	Metrics	TC class	TC subclass
DeepPPF	MCC	0.908	0.960
	BAcc	0.910	0.913
	Prec	0.938	0.970

For the TC class, the prediction achieved to 0.91 MCC while obtaining 0.96 MCC for TC subclass. With all the results obtained, it is possible to confirm that the combination of the DeepPPF with the ESM-1b can achieve high quality results for the prediction of the TC class, subclass, family and superfamily.

5.2.3 Classification of transporters with non-transporters sequences

Next, the classification with the addition of non-transporter sequences was also evaluated. This was made to assess if the models were able to classify transporter sequences and also identity non-transporter sequences. The approach used was similar to the approach followed in the multi-classification of transporters without non-transporters sequences.

The performance achieved for the same tests realized in the previous section can be seen in the appendix A.1. The analysis of the multi-classification of transporters with and without non-transporter sequences, shows that the tendency related to the performance between models or the usage of different encoders remains the same. The best representation continues to be obtained by the ESM-1b encoder, and the best prediction with the DeepPPF.

Nonetheless, a decrease in performance was verified between these case studies. The PSSM encoder with the DeepPPF for the TC class dropped from a MCC of 0.86 to a MCC of 0.81. For the classification of the TC superfamily with non-transporter sequences, the DeepPPF dropped from a 0.97 to 0.94 MCC with PSSM encoder and from 0.99 to 0.95 with ESM-1b encoder. In average, the models dropped around 0.05 in all metrics.

Other aspect verified was the decrease of the difference between the classification of the TC superfamily and TC family. In particular, when using the transformer encoders, these classifications became more volatile.

5.2.4 Binary classification of transporters and non-transporters

Because of the decrease in the prediction capability of models for the classification of transporters with non-transporter sequences, we tested the usage of a two stage classification system for the classification of the transporters. First, a binary representation was used to predict if the protein was a transporter, and then we would do to predict the TC system of the transporter.

Firstly, we tried the use of the PSSM encoder with different ratios, to find the influence of the ratio in the binary classification. Here, the number of transporter sequences remains the same, only fluctuating the number of non-transporter sequences. Also, all models were used to verify if DeepPPF is able to provide better results as before.

The table 20 shows the performance achieved by the DeepPPF model. Once again, this model was capable of achieving slightly better results than the other 3 models. The tables associated with the remaining 3 models are presented in table 32 in the appendix B.

Table 20 – Performance of DeepPPF on binary classification through PSSM encoder. The A to B ratio equals to A transporter sequences to B enzyme sequences ratio. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and F_1 score. The highest metric values are in bold type

Models	Metrics	3 to 1	2 to 1	1 to 1	1 to 2	1 to 3
DeepPPF	MCC	0.741	0.731	**0.760**	0.737	0.733
	BAcc	0.737	0.687	**0.757**	0.737	0.750
	F_1 score	**0.937**	0.916	0.873	0.825	0.802

In the binary classification, when analyzing the metrics of MCC and BAcc, the best prediction was achieved when the number of transporter sequences was equal to the number of non-transporter sequences. Interestingly, the F_1 score improves with the addition of non-transporter sequences. The F_1 score gives emphasis to the correct classification of the positive cases (transporter sequences), which is verified in the confusion matrix of these classifications shown in figure 25.

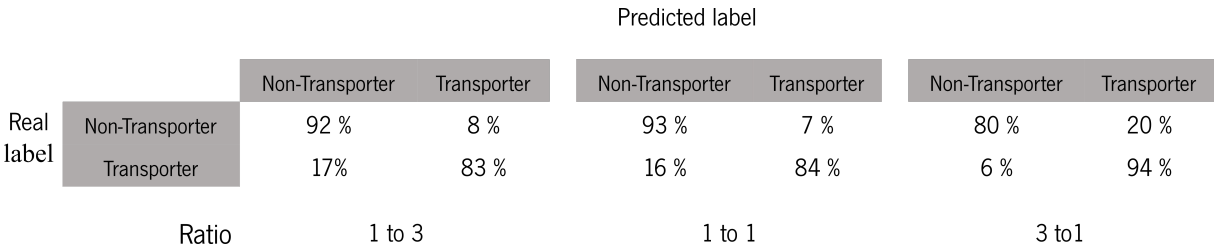


Figure 25 – Confusion matrices for binary classification with the PSSM encoder. These confusion matrices were obtained using the DeepPPF model. The A to B ratio equals to A transporter sequences to B enzyme sequences ratio.

The difference between ratios was not considered substantial in terms of the metrics generated, thereby, we used the ratio of one to one for the binary classification with the encoder ESM-1b and ESM2-650. The table 21 shows the result of the DeepPPF for these encoders.

Contrarily to the previous results, the use of the pretrained transformers does not cause a significant increase in the metrics obtained for the prediction capability of the models. Figure 26 shows that the confusion matrix is similar to the previous results, with similar numbers of false positives and false negatives.

Table 21 – Performance of DeepPPF on binary classification through ESM-1b and ESM2-650 encoders for 1 transporter sequence to 1 enzyme sequence ratio. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and F_1 score

Models	Metrics	ESM-1b encoder	ESM2-650 encoder
DeepPPF	MCC	0.764	0.770
	BAcc	0.762	0.770
	F_1 score	0.878	0.887

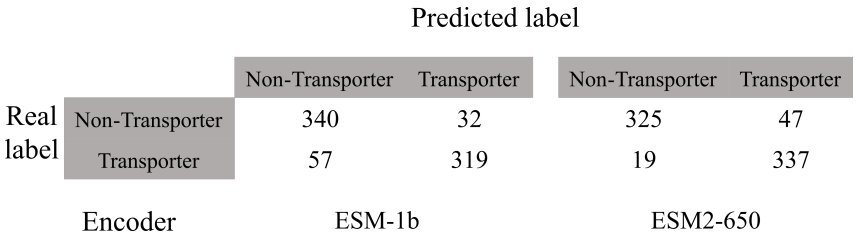


Figure 26 – Confusion matrixes for binary classification with the ESM-1b and ESM2-650 encoders. These confusion matrixes were obtained using the DeepPPF model for the ratio 1 transporter to 1 enzyme.

This indicates that the unremarkable results achieved can be associated more with the quality of the data instead of the representation capability of the encoder. In other words, the negative cases were selected randomly from a dataset that contains a very large spectrum of protein sequences (enzymes, structural proteins and others). The high biological variability of the non-transporter sequences associated to a low number of sequences used to represent the possible non-transporter groups could be the leading factor for the decreased performance.

6

CONCLUSION

6.1 MAIN RESULTS AND CONTRIBUTIONS

This work consisted in a study of different methods to obtain a informative representation of protein sequences for the prediction of protein properties and functions. As case studies, enzymes and transporter sequences were used.

Firstly, the calculation of protein descriptors was implemented into the Omnium AI platform and improved in the Propythia package, which was then used in a AutoML pipeline constructed in the company platform. Afterwards, encoders for the transformation of each amino acid residue into a vector representation were tested. In total, seven of those encoders were tested and presented in this thesis. The encoders include: One-hot encoder; BLOSUM encoder; NLF encoder; Z-scale encoder; PSSM encoder; ProtBert encoder; ESM-1b encoder; ESM2-150 encoder; ESM2-650 encoder. Apart from the PSSM encoder, all others were implemented in the Omnium AI platform. In the ProPythia package, the encoders One-hot, BLOSUM, NLF and Z-scale encoders and the protein descriptors were improved or fixed, while the encoders ESM-1b, ESM2-150 and ESM2-650 were implemented from scratch.

The encoders were tested in deep learning models adapted from the literature, in particular from DeepPPF, DeepLoc, UDSMProt and ET-GRU. When considering the models used, the architecture of the adapted DeepPPF achieved most of the best scores across the encoders tested. Among the remaining adapted models, none stood out.

The one-hot and BLOSUM encoders obtained a lower performance than the remaining. However, these encoders were capable of generating a reasonable performance in some simpler tasks, such as the enzyme classification of the EC main class or the classification of TC family.

The one-hot encoding achieved 0.91 MCC and 0.84 MCC for the EC class classification and TC family classification, respectively. The BLOSUM encoding achieved 0.89 MCC and 0.80 MCC for the EC class classification and TC family classification, respectively.

The PSSM encoder was able to achieve a MCC of 0.98 for the EC class classification and a MCC of 0.97 for the TC family classification. However, the major disadvantages associated to this encoder, such as being time-consuming and not always being effective in obtaining a profile, which lead to the usage of the transformers as features extractors.

In the best result, the ProtBert, ESM-1b, ESM2-150 and ESM2-650 achieved an MCC of 0.96, 0.98, 0.96 and 0.99 for the prediction of the TC family classification. Both ESM-1b and ESM2-650 achieved the best performance, followed by the PSSM and ESM2-150. Lastly, the ProtBert was only better than the orthogonal and BLOSUM encoder.

The best overall combination to achieve the best performance was the usage of the ESM-1b encoder with the model adapted of DeepPPF. This combination was able to do a prediction with a MCC of 0.91, 0.96, 0.97 and 0.98 for prediction of TC class, subclass, family and superfamily. The transformers from ESM showed the best potential to generate informative representations of protein sequences.

The transporter binary prediction achieved more limited results, with only 0.77 MCC and BAcc by using the adapted DeepPPF model combined with the ESM2-650 encoder. This limited result is probably a consequence of the generated dataset, which may be related to the variety of the negative cases.

The goals that have been set for this thesis were achieved successfully. During this thesis, relevant literature regarding DL to protein tasks was reviewed. Methods and deep learning frameworks for protein representation were tested, and their performance was assessed. Simultaneously, the frameworks were developed in the frameworks in the OminiumAI platform and ProPythia package.

6.2 Future perspectives

The great potential for the ESM models as features extractors to represent proteins was shown during the thesis. This encoder achieved a performance in the transporter classification as has never been demonstrated in the literature. However, some aspects of this work can be improved.

In future work, the evaluation of different models and more fine-tuning of the models used can lead to an improvement in the functional classification of proteins. Dimension reduction methods could also be tested to try to reduce the computer resources required in the usage of the pretrained transformers. Despite the informative representation that these methods produce, the computation resources can be a limitation if applied to a larger scale if the computational resources are not available.

Finally, in the binary classification, a more sophisticated sampling method should be used to select the negative sequences. The random selection used to select the negative case was likely a major cause for the not optimal results achieved in this thesis.

BIBLIOGRAPHY

[1] Laskowski, R. A., Watson, J. D., & Thornton, J. M. (2003). From protein structure to biochemical function? *Journal of Structural and Functional Genomics*, *4*(2-3), 167–177.

[2] Löchel, H. F., Eger, D., Sperlea, T., & Heider, D. (2020). Deep learning on chaos game representation for proteins. *Bioinformatics*, *36*(1), 272–279.

[3] Nauman, M., Ur Rehman, H., Politano, G., & Benso, A. (2019). Beyond Homology Transfer: Deep Learning for Automated Annotation of Proteins. *Journal of Grid Computing*, *17*(2), 225–237.

[4] Gligorijević, V., Barot, M., & Bonneau, R. (2018). DeepNF: Deep network fusion for protein function prediction. *Bioinformatics*, *34*(22), 3873–3881.

[5] Liu, B., Li, C.-C., & Yan, K. (2020). DeepSVM-fold: protein fold recognition by combining support vector machines and pairwise sequence similarity scores generated by deep learning networks. *Briefings in Bioinformatics*, *21*(5), 1733–1741.

[6] Yusuf, S. M., Zhang, F., Zeng, M., & Li, M. (2021). DeepPPF: A deep learning framework for predicting protein family. *Neurocomputing*, *428*, 19–29.

[7] Khurana, S., Rawi, R., Kunji, K., Chuang, G. Y., Bensmail, H., & Mall, R. (2018). DeepSol: A deep learning framework for sequence-based protein solubility prediction. *Bioinformatics*, *34*(15), 2605–2613.

[8] Elbasir, A., Moovarkumudalvan, B., Kunji, K., Kolatkar, P. R., Mall, R., & Bensmail, H. (2019). Deepcrystal: A deep learning framework for sequence-based protein crystallization prediction. *Bioinformatics*, *35*(13), 2216–2225.

[9] Sequeira, A. M., Lousa, D., & Rocha, M. (2021a). Propythia: A python package for protein classification based on machine and deep learning. *Neurocomputing*.

[10] Nelson, D. L., & Cox, M. M. (2017). *Lehninger principles of biochemistry, seventh edition* (Freeman, Ed.; Seventh Edition).

[11] Chandra, N., & Tyagi, V. K. (2013). Synthesis, properties, and applications of amino acids based surfactants: A review. *Journal of Dispersion Science and Technology*, *34*(6), 800–808.

[12] Pauling, L., Corey, R. B., & Branson, H. R. (1951). The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain. *Proceedings of the National Academy of Sciences*, *37*(4), 205–211.

[13] Pauling, L., & Corey, R. B. (1951). The Pleated Sheet, A New Layer Configuration of Polypeptide Chains. *Proceedings of the National Academy of Sciences*, *37*(5), 251–256.

[14] PERUTZ, M. F., ROSSMANN, M. G., CULLIS, A. F., MUIRHEAD, H., WILL, G., & NORTH, A. C. T. (1960). Structure of Hæmoglobin: A Three-Dimensional Fourier Synthesis at 5.5-Å. Resolution, Obtained by X-Ray Analysis. *Nature*, *185*(4711), 416–422.

[15] KENDREW, J. C., DICKERSON, R. E., STRANDBERG, B. E., HART, R. G., DAVIES, D. R., PHILLIPS, D. C., & SHORE, V. C. (1960). Structure of Myoglobin: A Three-Dimensional Fourier Synthesis at 2 Å. Resolution. *Nature*, *185*(4711), 422–427.

[16] Travers, A. A. (1989). DNA CONFORMATION AND PROTEIN BINDING. *Annual Review of Biochemistry*, *58*(1), 427–452.

[17] Jianlin Cheng, Tegge, A., & Baldi, P. (2008). Machine Learning Methods for Protein Structure Prediction. *IEEE Reviews in Biomedical Engineering*, *1*, 41–49.

[18] Ortuño, F., & Rojas, I. (Eds.). (2016). *Bioinformatics and Biomedical Engineering* (Vol. 965 6). Springer International Publishing.

[19] Jeske, L., Placzek, S., Schomburg, I., Chang, A., & Schomburg, D. (2019). BRENDA in 2019: a European ELIXIR core data resource. *Nucleic Acids Research*, *47*(D1), D542–D549.

[20] Li, Y., Wang, S., Umarov, R., Xie, B., Fan, M., Li, L., & Gao, X. (2018). DEEPre: sequence-based enzyme EC number prediction by deep learning (J. Hancock, Ed.). *Bioinformatics*, *34*(5), 760–769.

[21] Drew, D., & Boudker, O. (2016). Shared Molecular Mechanisms of Membrane Transporters. *Annual Review of Biochemistry*, *85*(1), 543–572.

[22] Saier, M. H., Reddy, V. S., Tamang, D. G., & Västermark, Å. (2014). The Transporter Classification Database. *Nucleic Acids Research*, *42*(D1), D251–D258.

[23] Mohammed, M., Khan, M. B., & Bashier, E. B. M. (2016). *Machine Learning: Algorithms and Applications* (1st Edition). CRC Press.

[24] Chollet, F. (2017). *Deep learning with python* (1st). Manning Publications Co.

[25] Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2016). A review of machine learning techniques using decision tree and support vector machine. *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 1–7.

[26] Christoph, M. A., & Guido, S. (2016). *Introduction to machine learning with python: A guide for data scientists*. Oreilly et Associates Inc.

[27] Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, *7*, 53040–53065.

[28] Raschka, S. (2015). *Python machine learning*. Packt Publishing.

[29] Guyon, I., & Elisseeff, A. (2006). An introduction to feature extraction. In I. Guyon, M. Nikravesh, S. Gunn, & L. A. Zadeh (Eds.), *Feature extraction: Foundations and applications* (pp. 1–25). Springer Berlin Heidelberg.

[30] Awad, M., & Khanna, R. (2015). *Efficient learning machines: Theories, concepts, and applications for engineers and system designers* (1st). Apress.

[31] Piramuthu, S. (2004). Evaluating feature selection methods for learning in data mining applications. *European Journal of Operational Research*, *156*(2), 483–494.

[32] Saeys, Y., Inza, I., & Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, *23*(19), 2507–2517.

[33] Singh, A., Thakur, N., & Sharma, A. (2016). A review of supervised machine learning algorithms. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 1310–1315.

[34] Bonaccorso, G. (2017). *Machine learning algorithms: A reference guide to popular algorithms for data science and machine learning*. Packt Publishing.

[35] Segaran, T. (2007). *Programming collective intelligence: Building smart web 2.0 applications*. O'Reilly Media.

[36] Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms* (Vol. 978110 7057). Cambridge University Press.

[37] Pérez-Enciso, M., & Zingaretti, L. M. (2019). A guide on deep learning for complex trait genomic prediction. *Genes*, *10*(7).

[38] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

[39] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, *12*, 2825–2830.

[40] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization.

[41] Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. In A. Sattar & B.-h. Kang (Eds.), *Ai 2006: Advances in artificial intelligence* (pp. 1015–1021). Springer Berlin Heidelberg.

[42] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1), 6.

[43] Horak, K. (2017). Classification of surf image features by selected machine learning algorithms.

[44] Mitchell, T. M. (1997). *Machine learning* (1st ed.). McGraw-Hill, Inc.

[45] Stevens, E., Antiga, L., & Viehmann, T. (2020). *Deep learning with pytorch*.

[46] Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–6.

[47] Mu, R., & Zeng, X. (2019). A Review of Deep Learning Research. *KSII Transactions on Internet and Information Systems*, *13*(4), 1738–1764.

[48] Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, *2018*, 1–13.

[49] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, 1–6.

[50] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019a). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, *31*(7), 1235–1270.

[51] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

[52] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019b). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, *31*(7), 1235–1270.

[53] Singh, S., & Mahmood, A. (2021). The nlp cookbook: Modern recipes for transformer based deep learning architectures. *IEEE Access*, *9*, 68675–68702.

[54] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need.

[55] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., . . . Rush, A. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45.

[56] Tunstall, L., von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers: Building language applications with hugging face*. O'Reilly Media, Incorporated.

[57] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[58] Lüscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schlüter, R., & Ney, H. (2019). Rwth asr systems for librispeech: Hybrid vs attention. *Interspeech 2019*.

[59] Synnaeve, G., Xu, Q., Kahn, J., Likhomanenko, T., Grave, E., Pratap, V., Sriram, A., Liptchinsky, V., & Collobert, R. (2020). End-to-end asr: From supervised to semi- supervised learning with modern architectures.

[60] Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, Ł., Shazeer, N., Ku, A., & Tran, D. (2018). Image transformer.

[61] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers.

[62] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale.

[63] Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., Bhowmik, D., & Rost, B. (2021). Prottrans: Towards cracking the language of life's code through self-supervised learning. *bioRxiv*.

[64] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

[65] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter.

[66] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

[67] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners.

[68] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

[69] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

[70] Zöller, M. A., & Huber, M. F. (2021). Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, *70*, 409–472.

[71] He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, 106622.

[72] Zimmer, L., Lindauer, M., & Hutter, F. (2021). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl [also available under https://arxiv.org/abs/2006.13799]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3079–3090.

[73] Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., & Smola, A. (2020). Autogluon-tabular: Robust and accurate automl for structured data.

[74] Jin, H., Song, Q., & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1946–1956.

[75] LIN, K., MAY, A. C., & TAYLOR, W. R. (2002). Amino acid encoding schemes from protein structure alignments: Multi-dimensional vectors to describe residue types. *Journal of Theoretical Biology*, *216*(3), 361–365.

[76] Jurtz, V. I., Johansen, A. R., Nielsen, M., Almagro Armenteros, J. J., Nielsen, H., Sønderby, C. K., Winther, O., & Sønderby, S. K. (2017). An introduction to deep learning on biological sequence data: examples and solutions. *Bioinformatics*, *33*(22), 3685–3690.

[77] Eddy, S. R. (2004). Where did the blosum62 alignment score matrix come from? *Nature Biotechnology*, *22*(8), 1035–1036.

[78] Pearson, W. R. (2013). Selecting the right similarity-scoring matrix. *Current Protocols in Bioinformatics*, *43*(1), 3.5.1–3.5.9.

[79] Sandberg, M., Eriksson, L., Jonsson, J., Sjöström, M., & Wold, S. (1998). New Chemical Descriptors Relevant for the Design of Biologically Active Peptides. A Multivariate Characterization of 87 Amino Acids. *Journal of Medicinal Chemistry*, *41*(14), 2481–2491.

[80] Nanni, L., & Lumini, A. (2011). A new encoding technique for peptide classification. *Expert Systems with Applications*, *38*(4), 3185–3191.

[81] van den Berg, B. A., Reinders, M. J., Roubos, J. A., & de Ridder, D. (2014a). SPiCE: a web-based tool for sequence-based protein classification and exploration. *BMC Bioinformatics*, *15*(1), 93.

[82] Bonetta, R., & Valentino, G. (2020). Machine learning techniques for protein function prediction. *Proteins: Structure, Function, and Bioinformatics*, *88*(3), 397–413.

[83] A.Schäffer, A., I.Wolf, Y., P.Ponting, C., V.Koonin, E., Aravind, L., & F.Altschul, S. (1999). IMPALA: matching a protein sequence against a collection of PSI-BLAST-constructed position-specific score matrices. *Bioinformatics*, *15*(12), 1000–1011.

[84] Wang, J., Yang, B., Revote, J., Leier, A., Marquez-Lago, T. T., Webb, G., Song, J., Chou, K.-C., & Lithgow, T. (2017). POSSUM: a bioinformatics toolkit for generating numerical sequence feature descriptors based on PSSM profiles. *Bioinformatics*, *33*(17), 2756–2758.

[85] Ding, S., Li, Y., Shi, Z., & Yan, S. (2014). A protein structural classes prediction method based on predicted secondary structure and psi-blast profile. *Biochimie*, *97*, 60–65.

[86] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, *25*(17), 3389–3402.

[87] Jang, B., Kim, I., & Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. *PLOS ONE*, *14*(8), 1–20.

[88] Asgari, E., & Mofrad, M. R. K. (2015). Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLOS ONE*, *10*(11), 1–15.

[89] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

[90] Geffen, Y., Ofran, Y., & Unger, R. (2022). Distilprotbert: A distilled protein language model used to distinguish between real proteins and their randomly shuffled counterparts. *bioRxiv*.

[91] Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., & Fergus, R. (2019). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*.

[92] Rao, R. M., Meier, J., Sercu, T., Ovchinnikov, S., & Rives, A. (2020). Transformer protein language models are unsupervised structure learners. *bioRxiv*.

[93] Rao, R., Liu, J., Verkuil, R., Meier, J., Canny, J. F., Abbeel, P., Sercu, T., & Rives, A. (2021). Msa transformer. *bioRxiv*.

[94] Meier, J., Rao, R., Verkuil, R., Liu, J., Sercu, T., & Rives, A. (2021). Language models enable zero-shot prediction of the effects of mutations on protein function. *bioRxiv*.

[95] Hsu, C., Verkuil, R., Liu, J., Lin, Z., Hie, B., Sercu, T., Lerer, A., & Rives, A. (2022). Learning inverse folding from millions of predicted structures. *bioRxiv*.

[96] Thumuluri, V., Almagro Armenteros, J. J., Johansen, A. R., Nielsen, H., & Winther, O. (2022). DeepLoc 2.0: multi-label subcellular localization prediction using protein language models. *Nucleic Acids Research*, *50*(W1), W228–W234.

[97] Singh, J., Litfin, T., Singh, J., Paliwal, K., & Zhou, Y. (2022a). SPOT-Contact-LM: improving single-sequence-based prediction of protein contact map using a transformer language model. *Bioinformatics*, *38*(7), 1888–1894.

[98] Lee, E. Y., Fulan, B. M., Wong, G. C. L., & Ferguson, A. L. (2016). Mapping membrane activity in undiscovered peptide sequence space using machine learning. *Proceedings of the National Academy of Sciences*, *113*(48), 13588–13593.

[99] Boareto, M., Yamagishi, M. E., Caticha, N., & Leite, V. B. (2012). Relationship between global structural parameters and Enzyme Commission hierarchy: Implications for function prediction. *Computational Biology and Chemistry*, *40*, 15–19.

[100] Ganapathiraju, M., Klein-Seetharaman, J., Balakrishnan, N., & Reddy, R. (2004). Characterization of protein secondary structure. *IEEE Signal Processing Magazine*, *21*(3), 78–87.

[101] Cheng, J., Randall, A. Z., Sweredoski, M. J., & Baldi, P. (2005). SCRATCH: a protein structure and structural feature prediction server. *Nucleic Acids Research*, *33*(suppl_2), W72–W76.

[102] Amidi, A., Amidi, S., Vlachakis, D., Paragios, N., & Zacharaki, E. I. (2016). A machine learning methodology for enzyme functional classification combining structural and protein sequence descriptors. *Bioinformatics and Biomedical Engineering*, 728–738.

[103] Hawkins, T., Chitale, M., Luban, S., & Kihara, D. (2009). PFP: Automated prediction of gene ontology functional annotations with confidence scores using protein sequence data. *Proteins: Structure, Function, and Bioinformatics*, *74*(3), 566–582.

[104] Gao, R., Wang, M., Zhou, J., Fu, Y., Liang, M., Guo, D., & Nie, J. (2019). Prediction of Enzyme Function Based on Three Parallel Deep CNN and Amino Acid Mutation. *International Journal of Molecular Sciences*, *20*(11), 2845.

[105] Lee, I., Keum, J., & Nam, H. (2019). DeepConv-DTI: Prediction of drug-target interactions via deep learning with convolution on protein sequences (J. M. Briggs, Ed.). *PLOS Computational Biology*, *15*(6), 1–21.

[106] Notin, P., Dias, M., Frazer, J., Hurtado, J. M., Gomez, A. N., Marks, D., & Gal, Y. (2022). Tranception: Protein fitness prediction with autoregressive transformers and inference-time retrieval. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, & S. Sabato (Eds.), *Proceedings of the 39th international conference on machine learning* (pp. 16990–17017, Vol. 162). PMLR.

[107] Singh, J., Litfin, T., Singh, J., Paliwal, K., & Zhou, Y. (2022b). SPOT-Contact-LM: improving single-sequence-based prediction of protein contact map using a transformer language model. *Bioinformatics*, *38*(7), 1888–1894.

[108] Hong, Y., Lee, J., & Ko, J. (2022). A-Prot: protein structure modeling using MSA transformer. *BMC Bioinformatics*, *23*(1), 93.

[109] Høie, M. H., Kiehl, E. N., Petersen, B., Nielsen, M., Winther, O., Nielsen, H., Hallgren, J., & Marcatili, P. (2022). NetSurfP-3.0: accurate and fast prediction of protein structural features by protein language models and deep learning. *Nucleic Acids Research*, *50*(W1), W510–W515.

[110] Monteiro, N. R., Oliveira, J. L., & Arrais, J. P. (2022). Dtitr: End-to-end drug–target binding affinity prediction with transformers. *Computers in Biology and Medicine*, *147*, 105772.

[111] Wang, Z., Combs, S. A., Brand, R., Calvo, M. R., Xu, P., Price, G., Golovach, N., Salawu, E. O., Wise, C. J., Ponnapalli, S. P., & Clark, P. M. (2022). LM-GVP: an extensible sequence and structure informed deep learning framework for protein property prediction. *Scientific Reports*, *12*(1), 6832.

[112] Singh, J., Litfin, T., Singh, J., Paliwal, K., & Zhou, Y. (2021). Spot-contact-single: Improving single-sequence-based prediction of protein contact map using a transformer language model. *bioRxiv*.

[113] Wang, K., Zhou, R., Li, Y., & Li, M. (2021). DeepDTAF: a deep learning method to predict protein–ligand binding affinity. *Briefings in Bioinformatics*, *22*(5).

[114] Kim, G. B., Gao, Y., Palsson, B. O., & Lee, S. Y. (2021). DeepTFactor: A deep learning-based tool for the prediction of transcription factors. *Proceedings of the National Academy of Sciences*, *118*(2), e2021171118.

[115] Semwal, R., Aier, I., Tyagi, P., & Varadwaj, P. K. (2021). DeEPn: a deep neural network based tool for enzyme functional annotation. *Journal of Biomolecular Structure and Dynamics*, *39*(8), 2733–2743.

[116] Cai, Y., Wang, J., & Deng, L. (2020). SDN2GO: An Integrated Deep Learning Model for Protein Function Prediction. *Frontiers in Bioengineering and Biotechnology*, *8*.

[117] Wang, D., Liu, D., Yuchi, J., He, F., Jiang, Y., Cai, S., Li, J., & Xu, D. (2020). MusiteDeep: a deep-learning based webserver for protein post-translational modification site prediction and visualization. *Nucleic Acids Research*, *48*(W1), W140–W146.

[118] Strodthoff, N., Wagner, P., Wenzel, M., & Samek, W. (2020). UDSMProt: universal deep sequence models for protein classification (Y. Ponty, Ed.). *Bioinformatics*, *36*(8), 2401–2409.

[119] Zhang, F., Song, H., Zeng, M., Li, Y., Kurgan, L., & Li, M. (2019). DeepFunc: A Deep Learning Framework for Accurate Prediction of Protein Functions from Protein Sequences and Interactions. *PROTEOMICS*, *19*(12).

[120] Kulmanov, M., & Hoehndorf, R. (2019). DeepGOPlus: improved protein function prediction from sequence (L. Cowen, Ed.). *Bioinformatics*, *36*(2), 422–429.

[121] Ryu, J. Y., Kim, H. U., & Lee, S. Y. (2019). Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. *Proceedings of the National Academy of Sciences*, *116*(28), 13996–14001.

[122] Sureyya Rifaioglu, A., Doğan, T., Jesus Martin, M., Cetin-Atalay, R., & Atalay, V. (2019). DEEPred: Automated Protein Function Prediction with Multi-task Feed-forward Deep Neural Networks. *Scientific Reports*, *9*(7344).

[123] Taju, S. W., & Ou, Y.-Y. (2019). DeepIon: Deep learning approach for classifying ion transporters and ion channels from membrane proteins. *Journal of Computational Chemistry*, *40*(15), 1521–1529.

[124] Le, N. Q. K., Yapp, E. K. Y., & Yeh, H.-Y. (2019). ET-GRU: using multi-layer gated recurrent units to identify electron transport proteins. *BMC Bioinformatics*, *20*(1), 377.

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

[126] Hashemifar, S., Neyshabur, B., Khan, A. A., & Xu, J. (2018). Predicting protein–protein interactions through sequence-based deep learning. *Bioinformatics*, *34*(17), i802–i810.

[127] Almagro Armenteros, J. J., Sønderby, C. K., Sønderby, S. K., Nielsen, H., & Winther, O. (2017). DeepLoc: prediction of protein subcellular localization using deep learning (J. Hancock, Ed.). *Bioinformatics*, *33*(21), 3387–3395.

[128] van den Berg, B. A., Reinders, M. J., Roubos, J. A., & de Ridder, D. (2014b). SPiCE: a web-based tool for sequence-based protein classification and exploration. *BMC Bioinformatics*, *15*(1), 93.

[129] Brandes, N., Ofer, D., & Linial, M. (2016). ASAP: a machine learning framework for local protein properties [baw133]. *Database*, *2016*.

[130] Liu, B., Wu, H., Zhang, D., Wang, X., & Chou, K.-C. (2017). Pse-analysis: A python package for dna/rna and protein/ peptide sequence analysis based on pseudo components and kernel methods. *Oncotarget*, *8*(8), 13338–13343.

[131] Müller, A. T., Gabernet, G., Hiss, J. A., & Schneider, G. (2017). modlAMP: Python for antimicrobial peptides. *Bioinformatics*, *33*(17), 2753–2755.

[132] Muhammod, R., Ahmed, S., Md Farid, D., Shatabda, S., Sharma, A., & Dehzangi, A. (2019). PyFeat: a Python-based effective feature generation tool for DNA, RNA and protein sequences. *Bioinformatics*, *35*(19), 3831–3833.

[133] Liu, B., Gao, X., & Zhang, H. (2019). BioSeq-Analysis2.0: an updated platform for analyzing DNA, RNA and protein sequences at sequence level and residue level based on machine learning approaches. *Nucleic Acids Research*, *47*(20), e127–e127.

[134] Chen, Z., Zhao, P., Li, F., Marquez-Lago, T. T., Leier, A., Revote, J., Zhu, Y., Powell, D. R., Akutsu, T., Webb, G. I., Chou, K.-C., Smith, A. I., Daly, R. J., Li, J., & Song, J. (2019). iLearn: an integrated platform and meta-learner for feature engineering, machine-learning analysis and modeling of DNA, RNA and protein sequence data. *Briefings in Bioinformatics*, *21*(3), 1047–1057.

[135] Chen, Z., Zhao, P., Li, C., Li, F., Xiang, D., Chen, Y.-Z., Akutsu, T., Daly, R., Webb, G., Zhao, Q., Kurgan, L., & Song, J. (2021). iLearnPlus: a comprehensive and automated

machine-learning platform for nucleic acid and protein sequence analysis, prediction and visualization. *Nucleic Acids Research*, *49*(10), e60–e60.

[136] Sequeira, A. M., Lousa, D., & Rocha, M. (2021b). ProPythia: A Python Automated Platform for the Classification of Proteins Using Machine Learning. *Advances in Intelligent Systems and Computing*, *1240 AISC*, 32–41.

[137] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

[138] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). *TensorFlow, Large-scale machine learning on heterogeneous systems*.

[139] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: Experiences from the scikit-learn project. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122.

[140] The pandas development team. (n.d.). *pandas-dev/pandas: Pandas*.

[141] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362.

[142] Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., & Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, *118*(15), e2016239118.

[143] Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification.

[144] Dalkiran, A., Rifaioglu, A. S., Martin, M. J., Cetin-Atalay, R., Atalay, V., & Doğan, T. (2018). ECPred: a tool for the prediction of the enzymatic functions of protein sequences based on the EC nomenclature. *BMC Bioinformatics*, *19*(1), 334.

[145] Amidi, S., Amidi, A., Vlachakis, D., Paragios, N., & Zacharaki, E. I. (2017). Automatic single- and multi-label enzymatic function prediction by machine learning. *PeerJ*, *5*(3), 16.

[146] Wang, W., Peng, Z., & Yang, J. (2022). Single-sequence protein structure prediction using supervised transformer protein language models. *bioRxiv*.

A

SUPPORT WORK

A.1 MULTI-CLASSIFICATION OF TRANSPORTERS WITH NON-TRANSPORTERS

Table 22 – Performance of all models for TC main class prediction (with non-transporters) through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the models are indicated in bold

Models	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
DeepPPF	MCC	0.509	0.544	**0.809**
	BAcc	0.315	0.337	**0.761**
	Prec	0.617	0.633	**0.858**
DeepLoc	MCC	0.428	**0.548**	0.750
	BAcc	0.281	**0.432**	0.626
	Prec	0.560	0.653	0.816
UDSMProt	MCC	**0.561**	0.520	0.763
	BAcc	**0.451**	0.407	0.714
	Prec	**0.677**	0.651	0.823
ET-GRU	MCC	0.558	0.544	0.781
	BAcc	0.411	0.386	0.578
	Prec	0.668	**0.662**	0.839

Table 23 – Performance of DeepPPF on TC subclass, family and superfamily classification (with non-transporters) through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.500	0.550	0.805
	BAcc	0.297	0.296	0.601
	Prec	0.580	0.621	0.843
TC family	MCC	**0.746**	**0.715**	**0.943**
	BAcc	**0.631**	**0.603**	**0.900**
	Prec	**0.772**	**0.736**	**0.956**
TC superfamily	MCC	0.645	0.676	0.865
	BAcc	0.499	0.564	0.838
	Prec	0.682	0.698	0.889

Table 24 – Performance of DeepLoc on TC subclass, family and superfamily classification (with non-transporters) through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.485	0.561	0.746
	BAcc	0.300	0.374	0.602
	Prec	0.566	0.637	0.796
TC family	MCC	**0.589**	0.721	0.868
	BAcc	**0.471**	0.625	0.783
	Prec	**0.632**	0.728	0.884
TC superfamily	MCC	0.512	**0.729**	**0.875**
	BAcc	0.444	**0.653**	**0.848**
	Prec	0.559	**0.731**	**0.903**

Table 25 – Performance of UDSMProt on TC subclass, family and superfamily classification (with non-transporters) through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.522	0.508	0.772
	BAcc	0.336	0.379	0.651
	Prec	0.605	0.614	0.814
TC family	MCC	0.637	**0.689**	**0.893**
	BAcc	0.508	**0.604**	**0.819**
	Prec	0.688	**0.733**	**0.917**
TC superfamily	MCC	**0.647**	0.562	0.849
	BAcc	**0.603**	0.483	0.830
	Prec	**0.707**	0.600	0.859

Table 26 – Performance of ET-GRU on TC subclass, family and superfamily classification (with non-transporters) through one-hot, BLOSUM and PSSM encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	One-hot encoder	BLOSUM encoder	PSSM encoder
TC subclass	MCC	0.556	0.542	0.756
	BAcc	0.348	0.301	0.500
	Prec	0.625	0.607	0.794
TC family	MCC	**0.652**	0.678	**0.892**
	BAcc	0.513	0.563	**0.832**
	Prec	0.652	0.688	**0.913**
TC superfamily	MCC	0.650	**0.692**	0.844
	BAcc	**0.530**	**0.567**	0.793
	Prec	**0.678**	**0.703**	0.856

Table 27 – Performance of DeepPPF on TC family and superfamily classification (with non-transporters) through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	0.893	**0.950**	0.906	**0.943**
	BAcc	0.814	0.900	0.827	0.888
	Prec	0.914	**0.962**	**0.936**	**0.954**
TC superfamily	MCC	**0.922**	0.932	**0.917**	0.922
	BAcc	**0.892**	**0.934**	**0.913**	**0.904**
	Prec	**0.936**	0.943	0.927	0.937

Table 28 – Performance of DeepLoc on TC family and superfamily classification (with non-transporters) through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	**0.836**	**0.924**	**0.867**	**0.925**
	BAcc	0.745	**0.869**	**0.812**	0.857
	Prec	**0.879**	**0.937**	**0.891**	**0.936**
TC superfamily	MCC	0.823	0.886	0.859	0.912
	BAcc	**0.768**	0.867	0.806	**0.878**
	Prec	0.838	0.900	0.854	0.925

Table 29 – Performance of UDSMProt on TC family and superfamily classification (with non-transporters) through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	0.824	0.918	0.880	**0.905**
	BAcc	0.748	0.845	0.812	0.840
	Prec	0.868	0.929	0.905	**0.921**
TC superfamily	MCC	**0.880**	**0.933**	**0.906**	0.891
	BAcc	**0.824**	**0.932**	**0.895**	**0.876**
	Prec	**0.879**	**0.943**	**0.919**	0.910

Table 30 – Performance of ET-GRU on TC family and superfamily classification (with non-transporters) through ProtBert, ESM-1b, ESM2-150 and ESM2-650 encoders. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec). The highest metric values determined by each encoder and between the classification tasks are indicated in bold

Classification task	Metrics	ProtBert encoder	ESM-1b encoder	ESM2-150 encoder	ESM2-650 encoder
TC family	MCC	**0.856**	**0.905**	0.874	0.893
	BAcc	0.761	0.838	0.800	0.822
	Prec	**0.872**	0.916	0.905	0.915
TC superfamily	MCC	0.849	0.902	**0.901**	**0.922**
	BAcc	**0.791**	**0.862**	**0.866**	**0.870**
	Prec	0.866	**0.918**	**0.911**	**0.936**

Table 31 – Performance of DeepPPF on TC class and subclass classification through ESM-1b encoder. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and precision (Prec).

Model	Metrics	TC class	TC subclass
DeepPPF	MCC	0.903	0.887
	BAcc	0.903	0.813
	Prec	0.927	0.910

B

DETAILS OF RESULTS

B.1 BINARY CLASSIFICATION OF TRANSPORTERS AND NON-TRANSPORTERS

Table 32 – Performance of DeepLoc, UDSMProt and ET-GRU on binary classification through PSSM encoder. The A to B ratio equals to A transporter sequences to B enzyme sequences ratio. The performance is represented by the metrics Matthews correlation coefficient (MCC), balanced accuracy (BAcc) and F_1 score

Models	Metrics	3 to 1	2 to 1	1 to 1	1 to 2	1 to 3
DeepLoc	MCC	0.738	0.676	0.709	0.700	0.683
	BAcc	0.686	0.650	0.708	0.696	0.672
	F_1 score	0.940	0.898	0.859	0.799	0.760
UDSMProt	MCC	0.693	0.670	0.712	0.705	0.736
	BAcc	0.654	0.653	0.711	0.686	0.721
	F_1 score	0.929	0.895	0.860	0.798	0.800
ET-GRU	MCC	0.721	0.632	0.744	0.694	0.724
	BAcc	0.678	0.586	0.743	0.688	0.726
	F_1 score	0.935	0.888	0.874	0.795	0.794